Програмування мовою Java для дітей, батьків, дідусів та бабусь

Яків Файн

Java Programming for Kids, Parents and Grandparents

by Yakov Fain

Cover design and illustrations: Yuri Fain
Technical editor: Vitaliy Kostuchenko
Kid technical editor: David Fain

May 2004: First Electronic Edition (in English)

June 2005: Second Electronic Edition (in French) "Programmation Java pour les enfants, les parentset les grands-parents"

October 2011: Third Electronic Edition (in Russian) "Программирование на Java для детей, родителей, дедушек и бабушек"

May 2014: Fourth Electronic Edition (in Ukrainian) "Програмування мовою Java для дітей, батьків, дідусів та бабусь"

ISBN: 0-9718439-5-3

Передмова до українського видання

Коли 10 років тому я вирішив написати цю книгу англійською, я й уявити собі не міг, що її перекладуть на декілька мов. Тоді я звернувся до кількох американських видавців з проханням надрукувати її в кольорі, оскільки книга призначалася в основному для дітей. Мені всі відмовили. Справа в тому, що кольоровий друк коштує набагато дорожче чорно-білого. Тоді я вирішив, викласти в інтернеті електронну версію книжки безкоштовно, але в кольорі.

Через рік я отримав листа з Франції, в якому мене питали дозволу перекласти книжку на французьку. Я погодився, за умови, що і вони її викладуть в Інтернеті безкоштовно. Так вийшла версія книжки для франко-мовних читачів.

Потім з багатьох країн зі мною зв'язувалися люди з пропозицією перекласти її на свою мову. Я погоджувався, але після перекладу першого ж розділу у людей пропадав інтерес - дуже вже це великий проект, а не у всіх є на це вільний час.

У 2011 році кілька програмістів з Росії зробили переклад на російську.

А в 2013 році зі мною зв'язався чоловік з України на ім'я Віталій Костюченко з пропозицією перекласти книжку на українську. Я традиційно погодився, але собі сказав, що, принаймні, з'явиться переклад одного розділу. Але Віталій довів справу до кінця і переклав всю книгу. А це не просто переклад. За десять років вийшло багато нових версій мови Java. Тобто потрібно було написати нові інструкції з встановлення Java на вашому комп'ютері і перевірити ще раз всі приклади з книги.

Віталій виконав цю роботу. Мені це особливо приємно, тому що я сам родом з України. З 1992 року живу і працюю в Америці, хоча зв'язку з Україною не втрачаю і приїжджаю раз на рік до рідного Києва для участі в конференції з розробки програмного забезпечення на Java.

У 2014 році зі мною несподівано зв'язалися представники американського видавництва No Strarch Press і запропонували написати нову версію книжки з програмування на Java для дітей. Я знову сказав, що моя умова -

друк книги в кольорі. На мій подив, видавець погодився і в кінці року книга буде надрукована англійською. Більше того, я попросив видавця дозволити мені викладати чернетки розділів в Інтерет, тому, якщо ви читаєте англійською, то чернетки ви знайдете тут:

http://yfain.github.io/Java4Kids_NoStarchPress/

Ілюстрації до нової версії книги знову робить мій старший син Юрій, який працює мультиплікатором в Нью Йорку.

Я щиро радий, що люди, які розмовляють українською, тепер можуть вивчати програмування своєю рідною мовою! Під час читання цієї книжки вам можуть стати в нагоді мої відео уроки з програмування мовою Java. Ось їх російська версія: http://goo.gl/AeKdcN, а англійська версія тут: http://goo.gl/J9H8IG. Сподіваюся, що вам сподобається!

Яків Файн

Передмова

Одного дня, мій син Дэйв-пароход з'явився у мене в офісі, тримаючи мій підручник по Java для дорослих. Він попросив мене навчити його програмуванню, аби створювати комп'ютерні ігри. На той момент я вже написав декілька книг по мові Java і провів навчання програмуванню на комп'ютерах в декількох групах, але це були дорослі! У результатах пошуку на Amazon (найбільший американський інтернет-магазин) не було нічого, окрім книжок «для чайників», але Дейв не був «чайником»!

Після того, як я провів декілька годин пошуку в Google, мені вдалося знайти, або декілька не найвдаліших спроб створити курси Java для дітей, або декілька книг, написаних в стилі популярної в Америці дитячої серії Reader-Rabbit. Вгадайте, що я вирішив зробити? Я вирішив написати книгу по програмуванню для дітей. З метою розуміння дитячого образу думок, я попросив Дейва стати моїм першим учнем-дитиною.

Так з'явилася ця книга, яка підійде наступним групам людей:

- дітям у віці від 11 до 18 років;
- шкільним викладачам інформатики;
- батькам, що бажають навчити програмуванню своїх дітей;
- абсолютним новачкам в програмуванні (вік значення не має).

Не дивлячись на те, що при поясненні програмування я використовую просту мову, обіцяю шанобливе ставлення до моїх читачів. Я не планую писати, щось схоже на «Любі друзі! Ви збираєтеся почати нову і дивну подорож...». Так, звичайно! Просто візьмемося за справу.

Перші розділи книги завершаться невеликою програмою-грою, яка супроводиться детальними інструкціями про те, як зробити її робочою. Також ми створимо калькулятор, який виглядає і працює аналогічно калькулятору вашого комп'ютера. У другій частині книги ми разом створимо програми для гри в хрестики-нолики і пінг-понг.

Вам потрібно буде звикнути до мови професійних програмістів. Всі

важливі слова будуть надруковані *ось таким шрифтом*.

Елементи мови Java і програм також будуть виділені, наприклад, `String`.

Ця книга не охоплює всі елементи мови Java. Інакше, це зробило б її дуже товстою і нудною. Проте в кінці кожного розділу поміщені матеріали для додаткового читання, які містять посилання на англомовні веб-сайти з детальними відомостями про дану тему.

Крім того, в кінці кожного розділу ви знайдете завдання для самостійного виконання. Кожен читач повинен виконати завдання, які містяться в розділі *Практичні вправи*. Якщо ці завдання здадуться вам дуже легкими, то *спробуйте виконати завдання з розділу Практичні вправи підвищеної складності*. Насправді, якщо ви вирішили читати цю книгу, то ви напевно здібна людина і повинні спробувати виконати всі завдання.

Аби отримати максимум з цієї книги, прочитайте її від початку до кінця. Не слід рухатися далі, поки ви не зрозумієте вміст поточного розділу. Підлітки, батьки, дідусі і бабусі повинні впоратися з цією книгою, не вдаючись до сторонньої допомоги, проте маленькі діти повинні читати цю книгу разом з дорослими.

Подяки

Дякую всім архітекторам і розробникам, що безоплатно працюють над програмою Eclipse, яка є однією з найкращих із доступних середовищ інтегрованої розробки програм.

Особлива вдячність водіям міжміських автобусів компанії New Jersey Transit за плавне водіння — половина цієї книги була написана дорогою на роботу у автобусі № 139.

Дякую дружині Наташі за успішне управління бізнесом під назвою сім'я.

Особлива вдячність Юрію Гончарову, експерту в області програмування на Java з Торонто, Канада. Він виконав редагування книги, перевірив кожен приклад коду і надав цінний відгук, який дозволив покращити цю книгу.

Зміст

Розділ 1. Перша програма

Люди говорять один з одним, використовуючи для цього різні мови. Так само вони пишуть комп'ютерні програми, такі як ігри, калькулятори, текстові редактори, використовуючи для цього різні мови програмування. Без програм ваш комп'ютер буде даремний, а його екран завжди буде чорним. Компоненти комп'ютера називають ***апаратним забезпеченням***, а програми — ***програмним забезпеченням***. Найпопулярнішими комп'ютерними мовами є C# і Java. Чим мова Java відрізняється від безлічі інших мов?

По-перше, одна і та ж програма Java може бути *запущена* (працювати) без будь-яких змін на різних комп'ютерах, наприклад PC, Apple або інших платформах. Фактично програми, написані на Java навіть не знають, на якому комп'ютері вони виконуються, оскільки вони виконуються усередині спеціальної програмної оболонки, яка називається віртуальна машина JVM (Java Virtual Machine).

Якщо, наприклад, програмі Java потрібно надрукувати якісь повідомлення, вона просить зробити це віртуальну машину JVM, яка знає, як потрібно взаємодіяти з вашим принтером.

Друге, Java дозволяє створювати програмні елементи (*класи*), які представляють об'єкти з реального світу. Наприклад, можна створити клас Java з ім'ям `Car` (автомобіль) і задати властивості цього класу, такі як двері, колеса, подібно до тих, які є в справжніх автомобілів. Після цього, грунтуючись на цьому класі, можна створити інший клас, наприклад, `Ford`, який матиме всі властивості класу `Car` плюс ті властивості, які є лише в автомобілів марки Ford.

Третє, мова Java володіє величезною кількістю додаткових і безкоштовних примочок (програмних бібліотек), написаних тисячами програмістів з усього світу, і це робить середовище програмування Java набагато потужнішим в порівнянні з іншими мовами.

Четверте, мова Java поставляється безкоштовно! Ви можете знайти в Інтернеті все необхідне для створення програм на Java, не заплативши жодної

копійки за це!

Установка Середовища Java

Для того, щоб почати програмування на Java, необхідно завантажити спеціальне програмне забезпечення (ПЗ) з веб-сайту компанії Oracle. Мова Java була створена компанією Sun Microsystems. У 2009 році компанія Oracle придбала компанію Sun Microsystems. Повне найменування цього ПЗ — Java Development Kit (JDK). На момент написання книги останню версію цього ПЗ (JDK 8) можна завантажити на цьому веб-сайті:

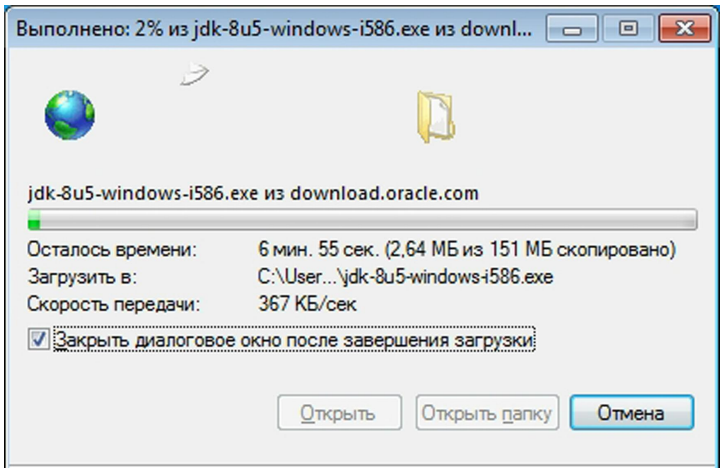
http://www.oracle.com/technetwork/java/javase/downloads/

Прийміть умови ліцензійної угоди. Виберіть пакет, відповідний операційній системі, яка встановлена на вашому комп'ютері. Якщо на вашому комп'ютері встановлена операційна система Windows, то це буде Windows x86 (32-розрядна версія) або Windows x64 (64-розрядна версія).

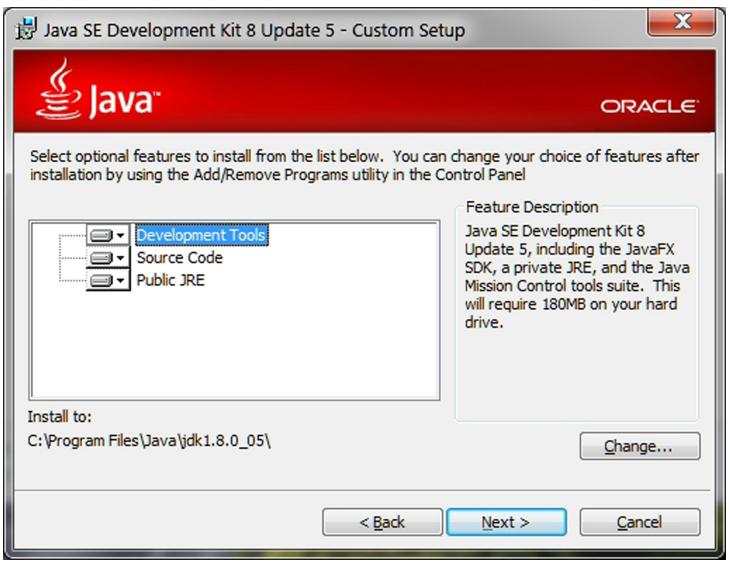
Якщо на вашому комп'ютері встановлена система Linux або Solaris, виберіть відповідний пакет. До недавнього часу комп'ютери Apple поставлялися з вже встановленим пакетом JDK. Якщо у вас комп'ютер Apple, запустіть програму Термінал, введіть там слово Java і натисніть клавішу Enter. Якщо ви побачите повідомлення Command not found, то це означає, що на вашому Маку Java не встановлена, і її потрібно завантажити звідси:

http://support.apple.com/downloads/#Java

Клацніть по посиланню з вибраним пакетом. Почнеться завантаження файлу. На комп'ютерах з операційною системою Windows це може виглядати так:



Після закінчення завантаження запустіть процес установки — двічі клацніть кнопкою миші по завантаженому файлу.

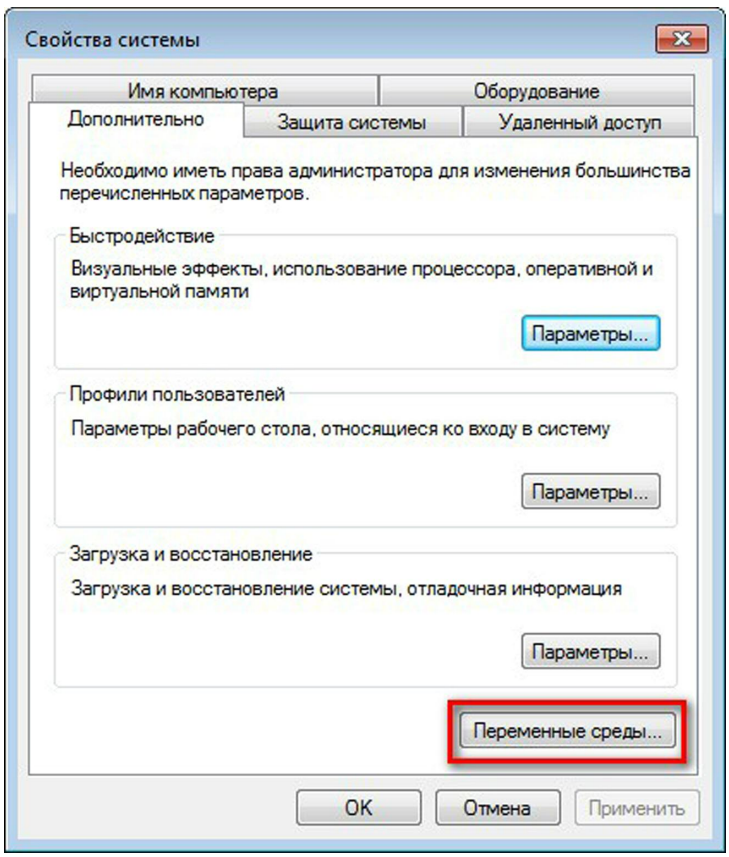


Відбудеться запуск майстра установки. Наприклад, на комп'ютері під управлінням Windows буде створений наступний каталог:

`C:\Program Files\Java\jdk1.8.0_05\`, де C: це ім'я жорсткого диска.

Якщо недостатньо місця на диску C:, то виберіть інший диск. Слідуйте вказівкам майстра установки – просто натискуйте кнопки *Next, Install* і *Finish* у вікнах, що з'являються на екрані. Протягом декількох хвилин установка Java на комп'ютер буде завершена.

На наступному етапі установки необхідно задати дві системні змінні. Наприклад, в Windows натисніть кнопку *Пуск*, далі клацніть правою кнопкою миші по пункту *Мій комп'ютер* і виберіть пункт *Властивості*.

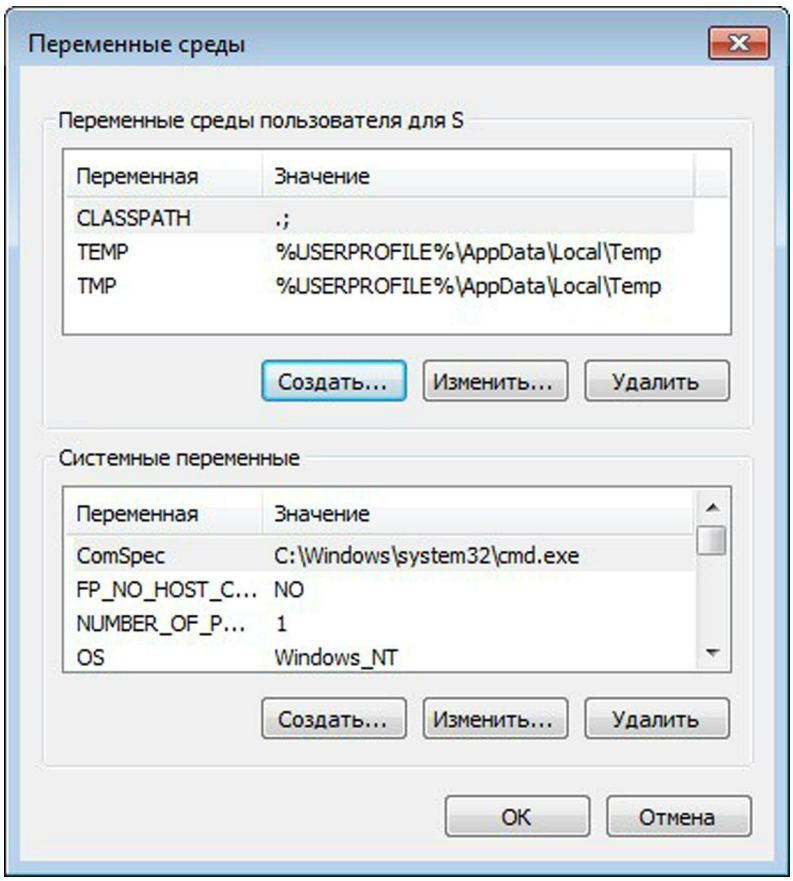


Далі перейдіть по посиланню *Додаткові параметри системи*, яке розташоване в лівій верхній частині панелі ліворуч. Перейдіть на вкладку *Додатково* і натисніть на кнопку *Змінні середовища...*

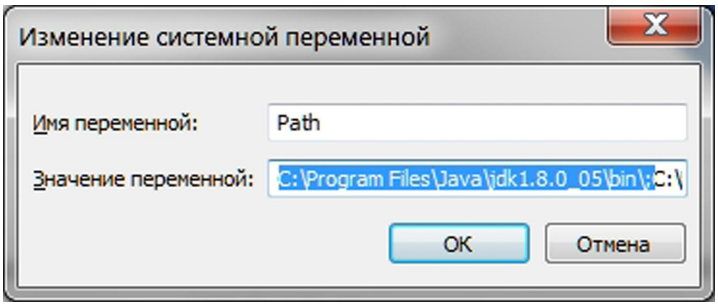
На наступній сторінці наведено зображення вікна з налаштуваннями

для Windows 7.

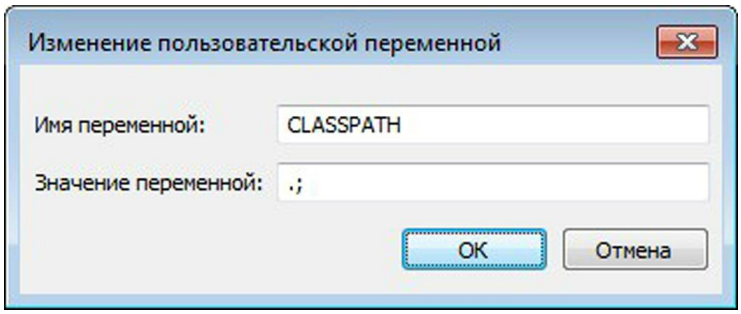
Так виглядає вікно, яке містить всі системні змінні, які є в системі:



Натисніть нижню кнопку Створити і оголосіть змінну `Path`, яка допоможе Windows (або Unix) знайти пакет JDK на вашому комп'ютері. Ретельно перевірте шлях до каталога, в який була проведена установка Java. Якщо змінна `Path` вже існує, просто додайте шлях до каталога Java і крапку з комою в самий початок поля *Значення змінної*:



Також оголосіть змінну CLASSPATH. Як її значення вкажіть крапку, за якою слідує крапка з комою (.;). Ця системна змінна допоможе Java знайти ваші програми. Крапка означає, що Java почне пошук ваших програм в поточному каталозі. Крапка з комою всього лише відіграє роль роздільника, за якими можуть слідувати інші каталоги.



Нарешті, установка пакету JDK завершена!

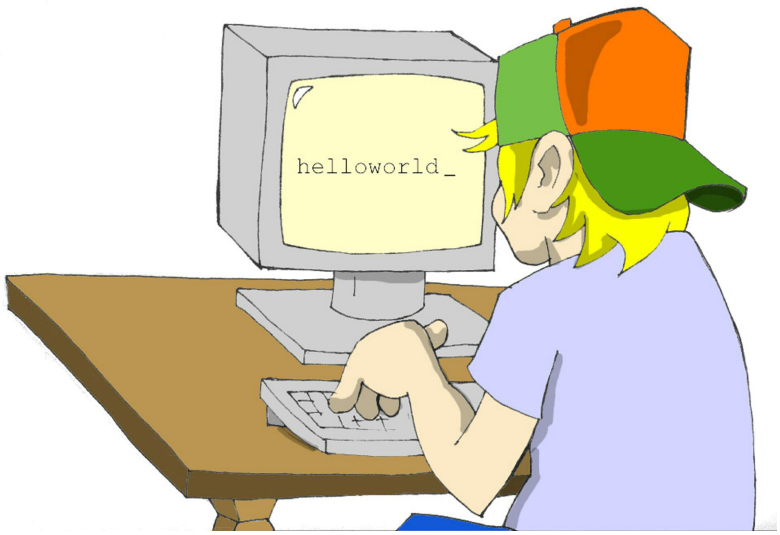
Три основні кроки в програмуванні

Для того, щоб створити працездатну програму на Java необхідно пройти через три наступні кроки.

- Написати програму на Java і зберегти її на диск.
- Виконати компіляцію програми, для того, щоб перекласти її з мови Java в спеціальний байт-код, який розуміє віртуальна машина JVM.
- Запустити програму.

Крок 1 - введення тексту програми

Для того, щоб написати програму на Java, ви можете використовувати будь-який текстовий редактор, наприклад, «Блокнот».



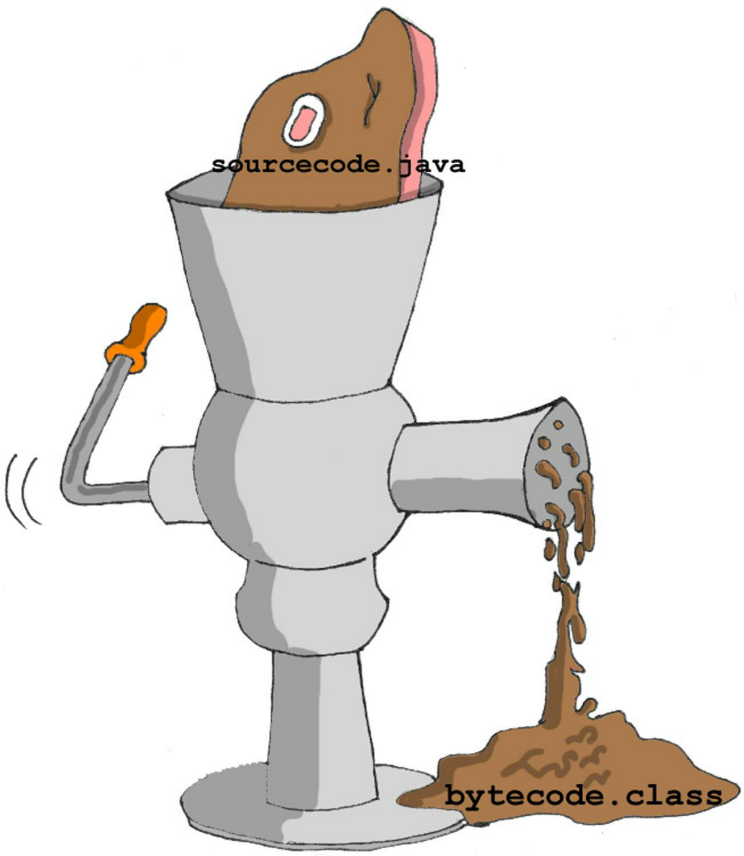
По-перше, потрібно буде надрукувати програму і зберегти її в текстовий файл, ім'я якого закінчуватиметься розширенням .java. Наприклад, якщо потрібно написати програму з назвою `HelloWorld`, надрукуйте її текст (цей текст ми називаємо *вихідний код*) в програмі «Блокнот» і збережіть його у файл з ім'ям `HelloWorld.java`. Увага, не використовуйте пропуски в іменах Java-файлів.

Нижче наведено програму, яка виводить на екран слова *Hello World*:

```
public class HelloWorld {

     public static void main(String[]   args) {

          System.out.println("Hello World");
     }
}
```

Я поясню роботу цієї програми трохи пізніше в цьому розділі, але зараз просто повірте мені – ця програма друкуватиме слова *Hello World* на третьому кроці.

Крок 2 – компіляція програми



Тепер необхідно відкомпілювати програму. Ви використовуватимете компілятор `javac`, який входить до складу пакету JDK.

Припустимо, ви зберегли програму в каталозі з ім'ям `C:\practice`. Натисніть кнопку *Пуск*, введіть в рядку пошуку команду `cmd` і натисніть клавішу ENTER. В результаті відкриється чорне командне вікно.

Аби переконатися в тому, що ви правильно встановили значення системних змінних `PATH` і `CLASSPATH`, введіть команду `set`. Знайдіть і перевірте ці значення в результатах виведення цієї команди.

Змініть поточний каталог на `C:\practice` і відкомпілюйте програму:

```
cd \practice

javac HelloWorld.java
```

Вам необов'язково називати каталог ім'ям *practice* – назвіть його так, як вам подобається.

Програма `javac` це компілятор мови Java. Ви не побачите ніякого підтвердження про те, що ваша програма `HelloWorld` успішно відкомпілювалася. Це саме той випадок, коли відсутність новин – це гарна новина!

Введіть команду `dir`. Результатом її виконання є виведення на екран всіх файлів, які існують в каталозі. Ви повинні побачити, що з'явився новий файл з ім'ям `HelloWorld.class`. Це є доказом того, що ваша програма успішно відкомпілювалася. Ваш вихідний файл `HelloWorld.java` також буде там. Ви можете змінити його пізніше так, щоб на екран виводилися слова *Привіт, мамо!* або щось ще.

Якщо в програмі є синтаксичні помилки, скажімо, ви забули надрукувати останню фігурну дужку, компілятор Java виведе повідомлення про помилку. В цьому випадку вам необхідно виправити помилку і відкомпілювати програму ще раз. Якщо є декілька помилок, то може потрібно повторення цих дій кілька разів, поки не буде створений файл `HelloWorld.class`.

Крок 3 – запуск програми

Ну а тепер, запустимо програму. У тому ж командному вікні введіть наступну команду:

```
java HelloWorld
```

Чи звернули ви увагу, що цього разу ви використовували програму `java`, а не `javac`? Дана програма входить в середовище виконання JRE (Java Run-time Environment) і вона запускає *JVM*, в яку завантажує вашу програму HelloWorld.

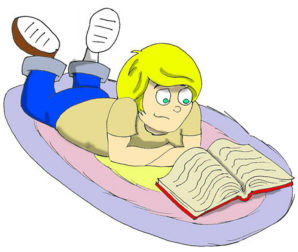
Пам'ятайте, що мова Java чутлива до регістру, це означає, що якщо ви назвали програму `HelloWorld` з великою літерою H і з великою літерою W, не намагайтеся запустити програму `helloworld` або `helloWorld` - JVM скаржитиметься, що, мовляв, не знаходжу файл.

Тепер можна розважитися – спробуйте здогадатися, як можна змінити цю програму. У наступному розділі я пояснюватиму, як працює ця програма. Проте спробуйте передбачити, як можна змінити її, аби сказати «Привіт!» своїй домашній тварині, другу або надрукувати свою адресу.

Пройдіть через всі три кроки, аби побачити, чи працює моя програма після ваших змін ☺.

У наступному розділі я покажу, як друкувати, компілювати і запускати ваші програми в привабливішому місці, ніж текстовий редактор і чорне командне вікно.

Матеріали для додаткового читання



Створення першої програми:
http://download.oracle.com/javase/tutorial/getStarted/cupojava/win32.html

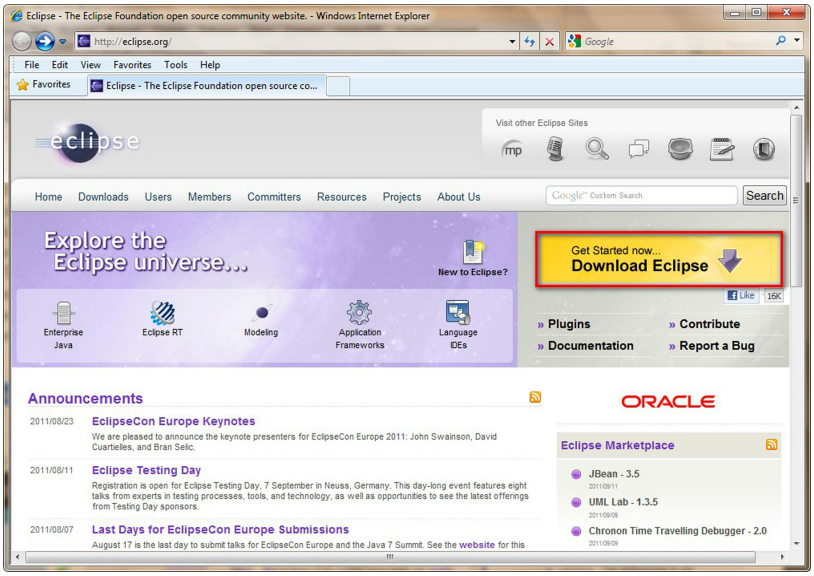
Інструкції щодо встановлення Java для Windows:
http://download.oracle.com/javase/7/docs/webnotes/install/windows/jdk-installation-windows.html

Розділ 2. Перехід до Eclipse IDE

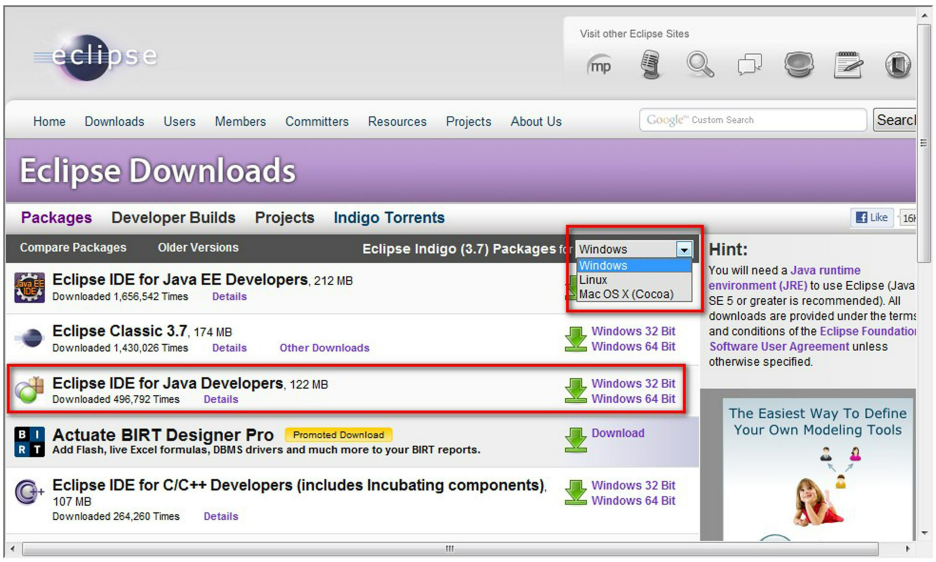
Програмісти зазвичай працюють з інструментом, який називається *інтегроване середовище розробки IDE* (Integrated Development Environment). Програми можна писати, компілювати і запускати прямо в цьому середовищі. У IDE також є така штука, як *Довідка*, що містить всі елементи мови, яка спрощує пошук і виправлення помилок в програмах. Деякі програми IDE мають високу вартість, проте існує чудова безкоштовна IDE під назвою Eclipse. Її можна завантажити з веб-сайту www.eclipse.org. У цьому розділі я допоможу вам завантажити і встановити Eclipse IDE на ваш комп'ютер і створити в цьому середовищі проект з назвою `Hello World`. Після цього всі наші програми ми створюватимемо в цьому середовищі. Влаштовуйтеся зручніше в Eclipse — це чудовий інструмент, який використовує безліч професійних програмістів на Java.

Установка Eclipse IDE

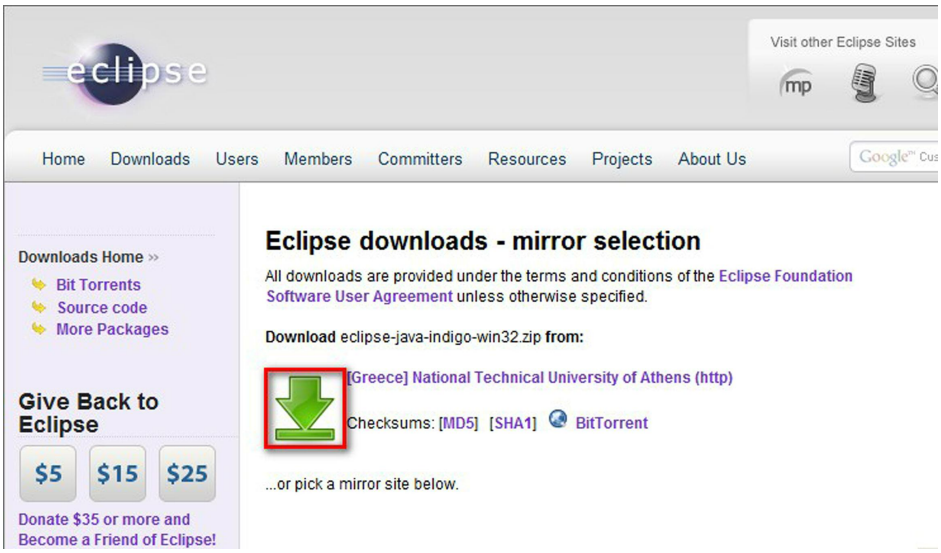
Відкрийте веб-сайт www.eclipse.org і натисніть у верхній правій частині сторінки кнопку *Download Eclipse* («Завантаження Eclipse»).



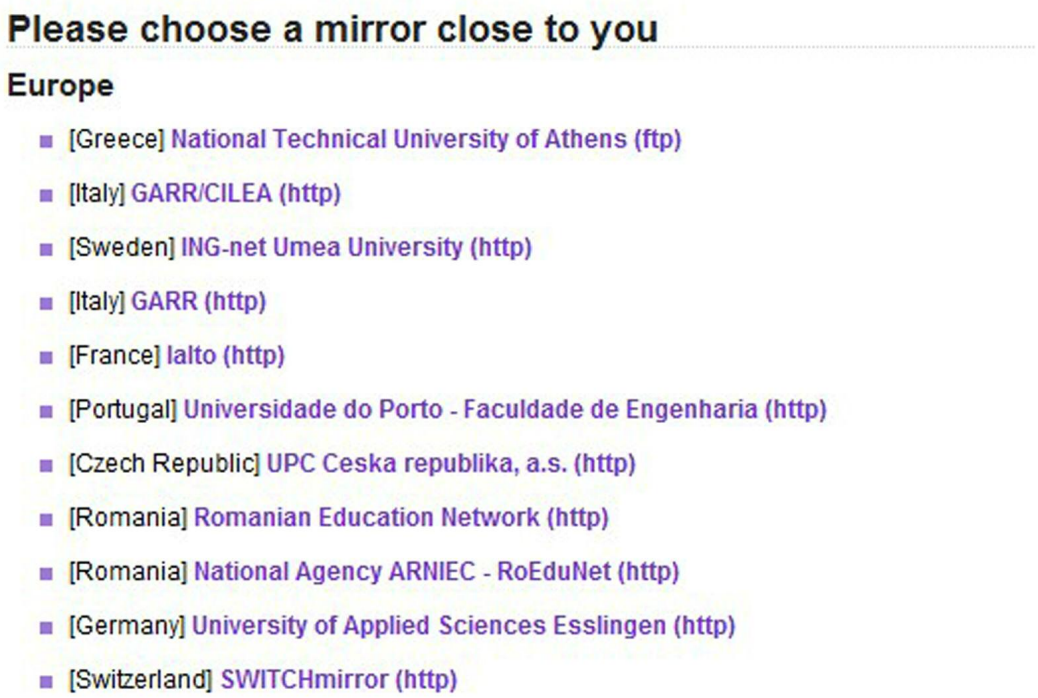
Після переходу на сторінку *Eclipse Downloads*, виберіть у випадному списку, що випадає, версію вашої операційної системи. Далі виберіть випуск *Eclipse IDE for Java Developers*. Щоб завантажити його, перейдіть по посиланню в правій частині, яка відповідає версії вашої системи. Якщо ви не впевнені яка версія Windows у вас встановлена, вибирайте 32-х розрядну версію.



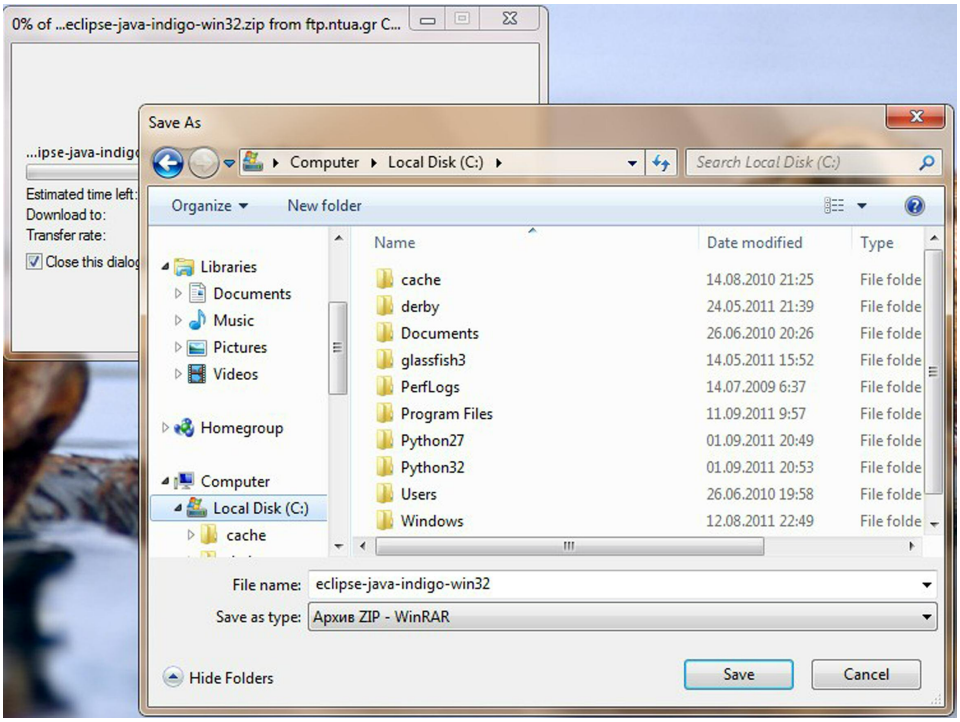
На сторінці, що відкрилася, для того, щоб почати завантаження із запропонованого системою сервера, клацніть по кнопці зі стрілкою.



Або, в нижній частині цієї сторінки, ви можете вибрати альтернативне джерело завантаження з найближчої до вас країни або міста. Це рекомендується зробити у разі, якщо завантаження із запропонованого джерела неможливе або виконується дуже повільно (також залежить від швидкості вашого підключення).

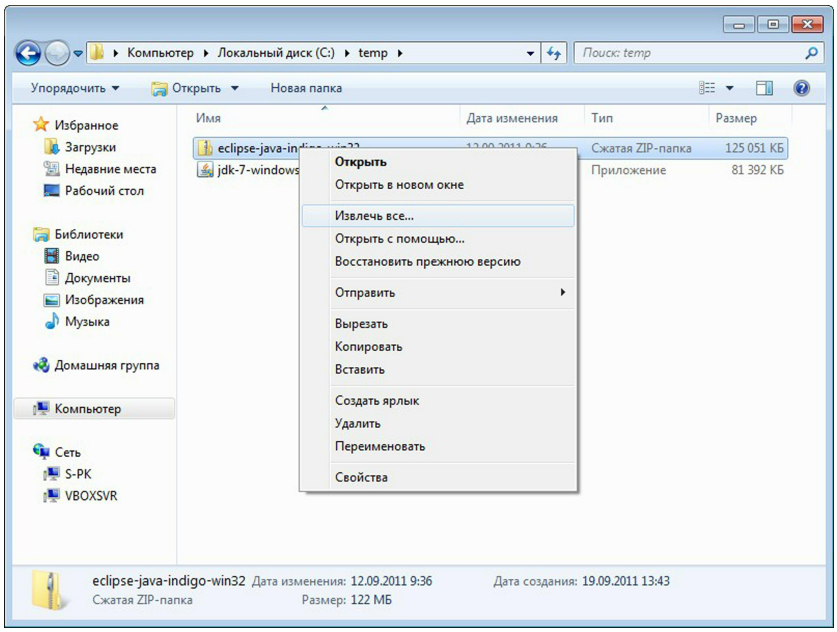


Збережіть файл на диск в будь-якій теці.



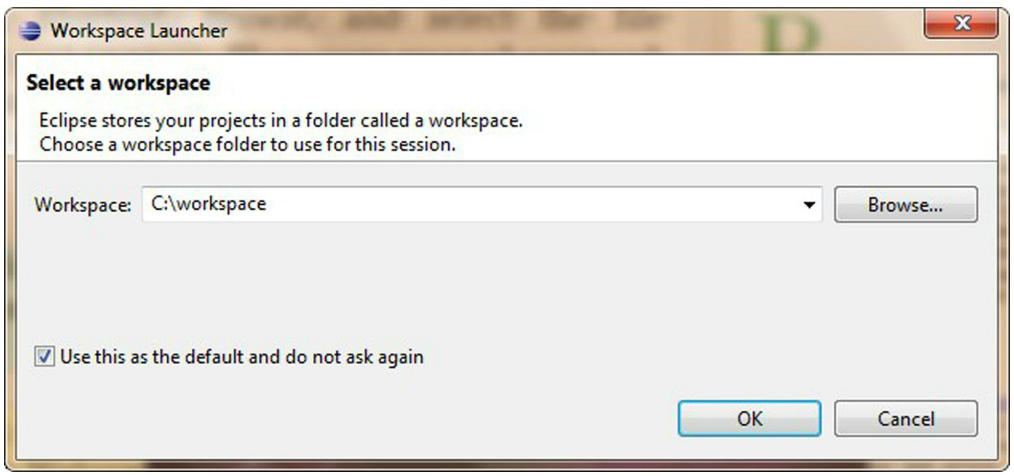
Тепер потрібно просто розпакувати цей файл на диск C:. Для цього клацніть правою кнопкою миші по архіву. Виберіть пункт *Розпакувати все...*

Файли, які мають розширення ZIP, — це архіви. Усередині себе вони можуть містити інші файли. Розпакувати файл означає витягнути вміст архіву на диск.

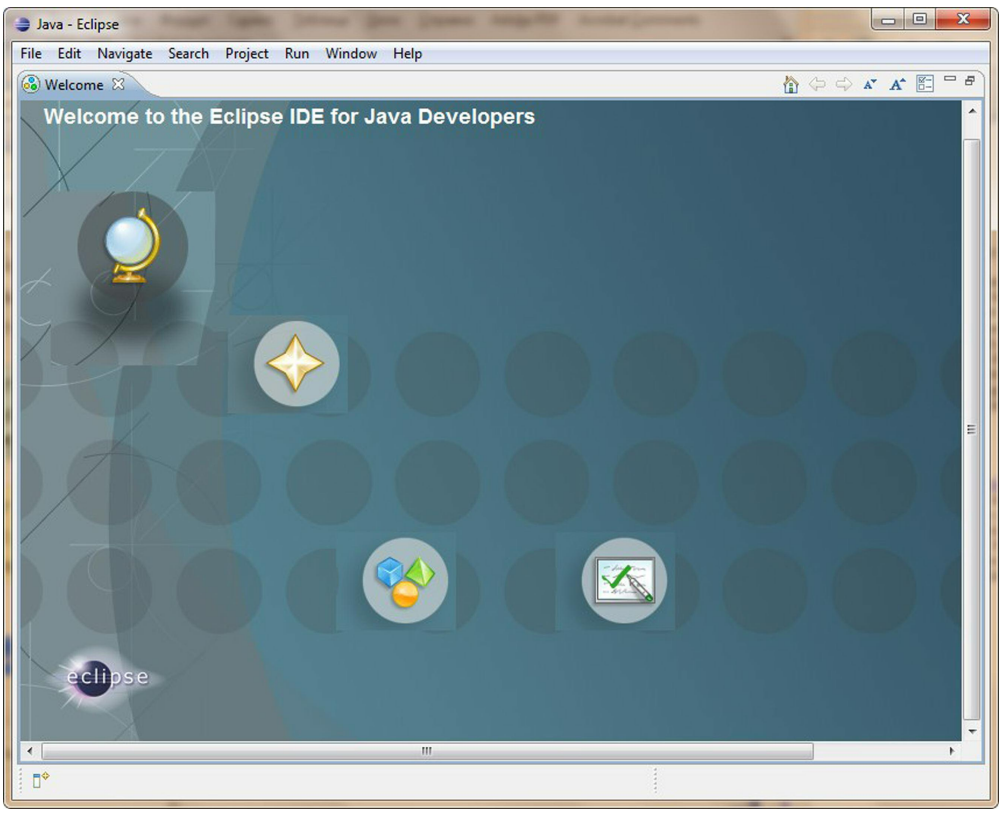


Інсталяцію Eclipse виконано! Для зручності створіть ярлик для Eclipse. Клацніть правою кнопкою по робочому столу. У меню, що з'явилося, послідовно виберіть пункти *Створити* > *Ярлик* > *Огляд* і виберіть в теці `C:\eclipse` файл `eclipse.exe`. Для того щоб запустити програму, двічі клацніть по синьому значку *Eclipse*.

При першому запуску Eclipse відкриється вікно з проханням вказати розташування робочої області (workspace) — каталога, в якому зберігатимуться файли ваших програм.



У полі *Workspace* («Робоча область») вкажіть зручне для вас місце на диску, наприклад `C:\workspace`. Якщо ви не хочете, щоб Eclipse кожного разу при запуску просила вас вказати робочу область, встановіть прапорець *Use this as the default and do not ask again* («Використовувати це значення за замовчуванням надалі»). Натисніть кнопку *OK*. Після запуску Eclipse відкривається початкова сторінка *Welcome*. Зовнішній вигляд цього вікна трохи міняється з кожною версією Eclipse.



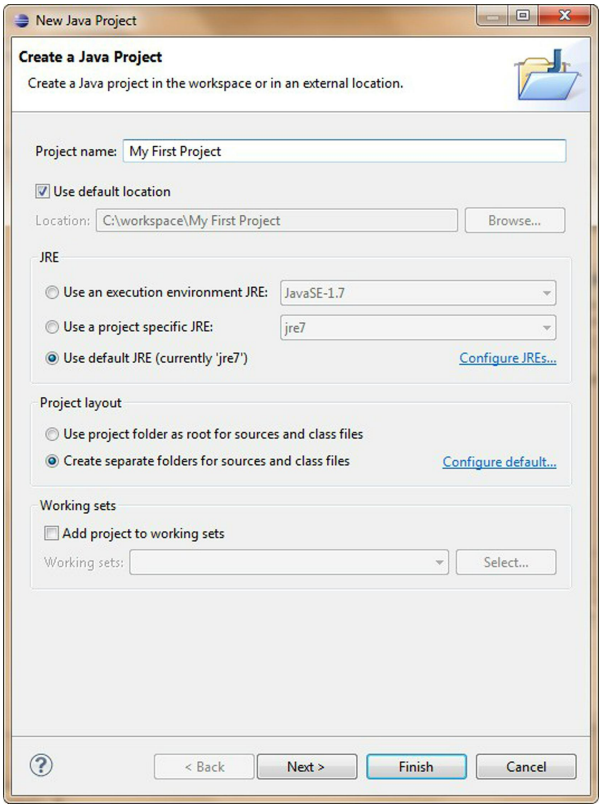
Тепер перейдіть в так зване середовище *Workbench* («Робоче середовище»), яке є робочим місцем для ваших проектів на Java. Для цього просто закрийте початкову сторінку, натисніть на хрестик після слова Welcome.

Приступаємо до роботи з Eclipse

У даному розділі я покажу вам, як можна швидко створити і запустити програми на Java в Eclipse. Ви також можете знайти непоганий підручник прямо в Eclipse. Для цього послідовно перейдіть меню *Help* («Довідка»), *Help Contents* («Зміст довідки»), далі перейдіть в розділ *Java Development User Guide* («Посібник користувача по розробці на Java»).

Щоб почати працювати над програмою, необхідно створити новий проект. Простий проект, як наш *My First Project*, міститиме всього один файл — `HelloWorld.java`. Пізніше ми створимо складніші проекти, які міститимуть декілька файлів.

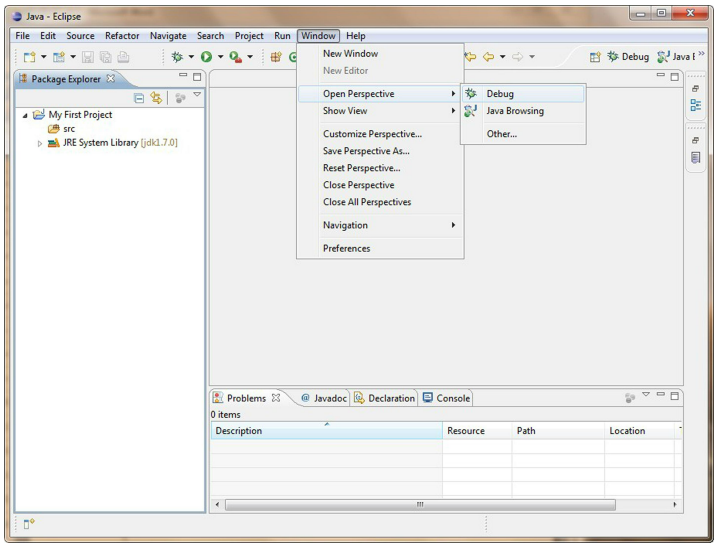
Щоб створити новий проект в Eclipse, виберіть наступні пункти меню *File* («Файл»), *New* («Створити»), *Java Project* («Проект Java»). У результаті відкриється вікно *New Java Project* («Створення проекту Java»). Тепер необхідно ввести ім'я нового проекту, наприклад, *My First Project*.



Зверніть увагу на поле *Location* («Розташування»). Воно вказує вам розташування на диску, в якому буде розміщений цей проект. За замовчуванням буде запропоновано зберегти проект в каталозі з ім'ям даного проекту. Цей каталог знаходитиметься в робочій області, яка була вами задана при запуску Eclipse. Пізніше ви створите окремі проекти для калькулятора, гри в хрестики-нулики і інших програм. До кінця цієї книги в робочій області знаходитиметься декілька проектів.

Робоча область Eclipse містить декілька панелей, певну компоновку яких називають проекцією (perspective).

Для того, щоб відкрити проекцію, потрібно вибрати в меню пункти *Window* («Вікно»), *Open Perspective* («Відкрити проекцію») і далі вибрати потрібну проекцію.



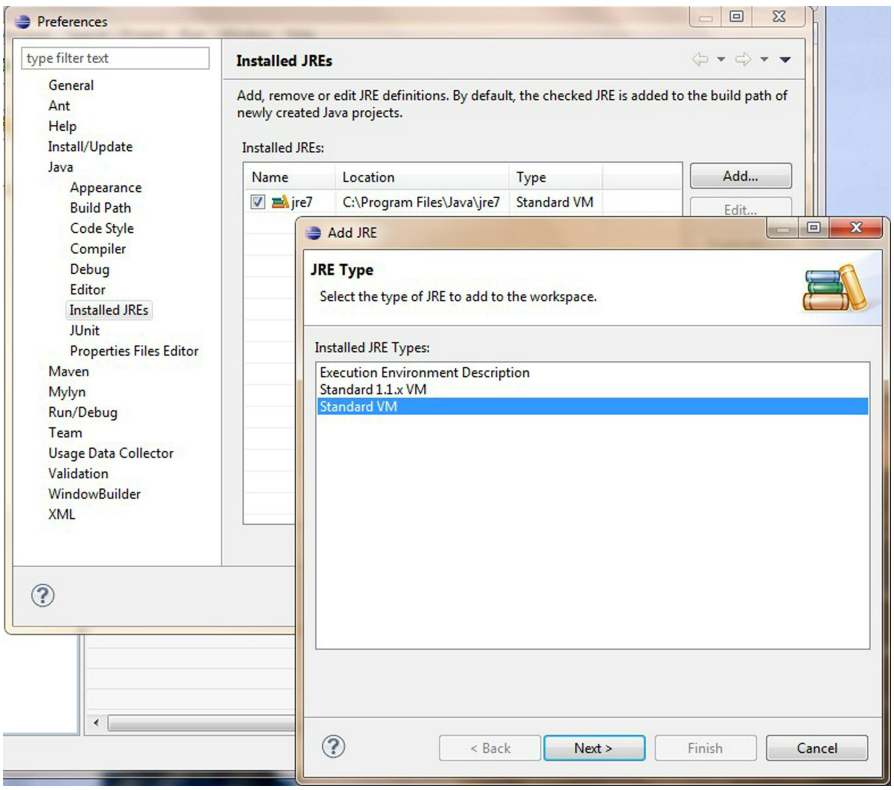
Кожна проекція містить набір панелей, які використовуються при вирішенні різних завдань. Наприклад, при розробці програм на Java ми використовуватимемо проекцію Java, а при налагодженні програм — проекцію Debug («Налагодження»).

Якщо клацнути по значку трикутника поряд з ім'ям проекту *My First Project*, при розкритті буде показаний елемент *JRE System Library (jdk 1.8.0)*. Цей елемент є частиною проекту. Якщо з будь-якої причини у вас немає цього елементу, додайте його. Для цього виберіть пункти меню *Windows* («Вікно»),

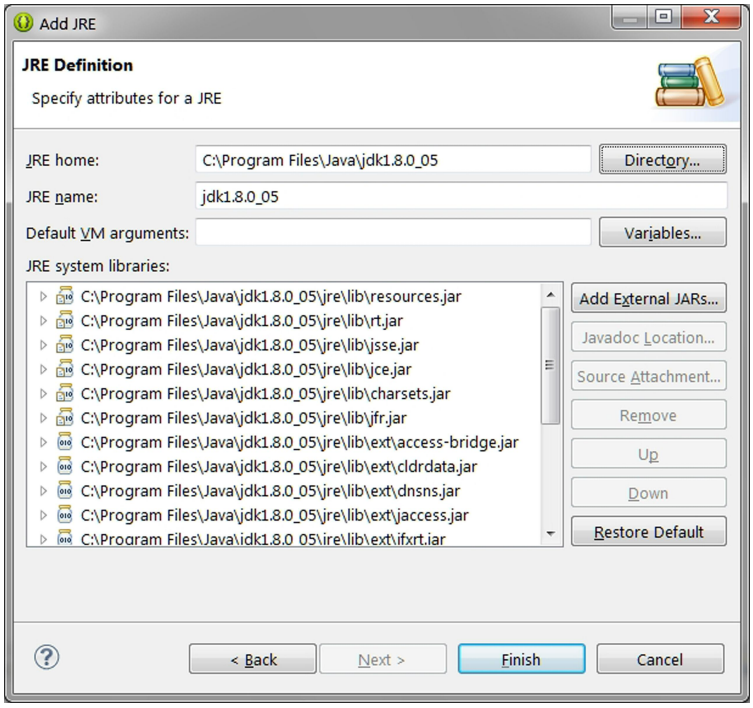
P*references* («Параметри»).

У вікні, що відкрилося, перейдіть в розділ *Java, Editor* («Редактор»), *Installed JREs* («Встановлені JRE»). Натисніть кнопку *Add* («Додати»).

У вікні, що відкрилося, виберіть елемент *Standard VM* («Стандартна віртуальна машина»). Натисніть кнопку *Next* («Далі»).



У вікні, що відкрилося, натисніть кнопку *Directory* і вкажіть директорію, в якій ви встановили Java, наприклад `C:\Program Files\Java\jdk1.8.0_05\`.

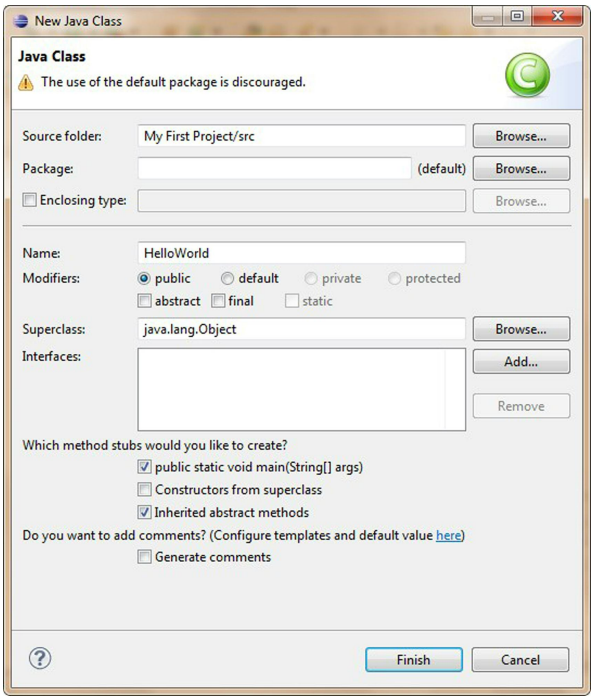


Створення програм в Eclipse IDE

Тепер повторимо створення програми `HelloWorld` з розділу 1 в Eclipse. Програми Java є класами, які представляють об'єкти з реального життя. Додаткові відомості про класи ви дізнаєтесь з наступного розділу.

Для того, щоб створити клас в Eclipse, виберіть в меню пункти *File*, *New*, *Class*. У вікні, що відкрилося, введіть `HelloWorld` в поле *Name* («Ім'я»). Також в розділі *Which methods stubs you would like to create* («Які заготовки для методів необхідно створити?») встановіть прапорець для методу main():

```
public static void main(String[] args)
```



Натисніть кнопку *Finish* і Eclipse створить клас `HelloWorld`, який міститиме порожній метод `main()`. Слово метод в даному випадку означає дію. Для того, щоб клас Java можна було запустити як програму, необхідно, щоб цей клас мав метод з ім'ям `main()`.

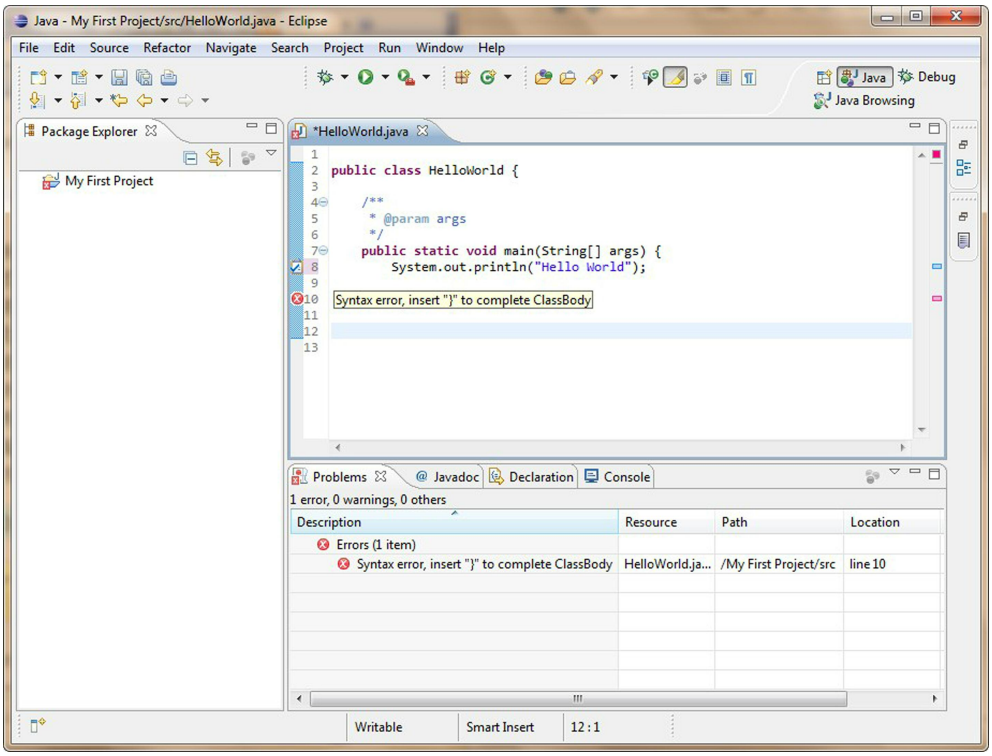
```
public class HelloWorld {

      public static void main(String[]   args) {

      }
}
```

Щоб завершити нашу програму, помістіть курсор після відкриваючої фігурної дужки в рядку з ім'ям `main`, натисніть клавішу *Enter* і на новому рядку надрукуйте наступний текст:

```
System.out.println("Hello World");
```

Для того, щоб зберегти програму на диск і одночасно відкомпілювати її, натисніть комбінацію клавіш *CTRL-S*. Якщо ви не зробили синтаксичних помилок, то жодних повідомлень про успішну компіляцію не буде, але програма відкомпілювалася. А давайте спеціально зробимо помилку, аби поглянути, що станеться.

Зітріть останню фігурну дужку і знову натисніть комбінацію клавіш *CTRL-S*. На вкладці *Problems* Eclipse виведе повідомлення про помилку *Syntax error, insert "}" to complete ClassBody* («Синтаксична помилка; вставте "}" для завершення ClassBody»). Також поряд з рядком, який містить помилку, з'явиться червона відмітка. При наведенні на неї покажчика миші, спливе повідомлення з аналогічним описом помилки.



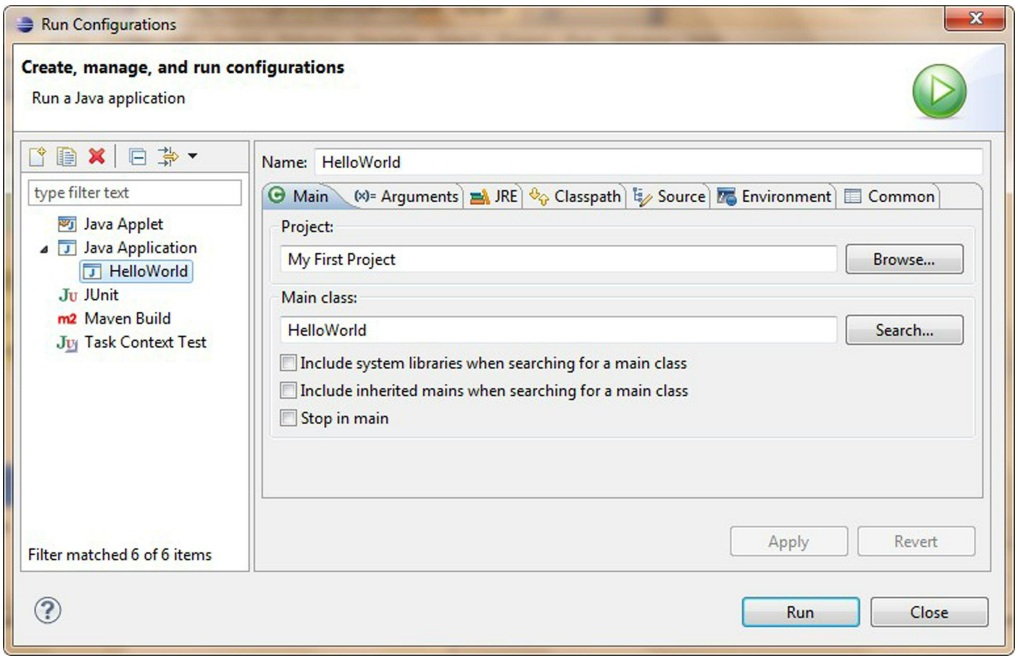
Зі збільшенням розміру ваших проектів, вони міститимуть декілька файлів, і компілятор може генерувати більшу кількість помилок. Ви навчитеся швидко знаходити і виправляти помилки синтаксису. Для цього двічі клацніть кнопкою миші по повідомленню про помилку в панелі *Problems*.

Поверніть фігурну дужку на місце і знову натисніть *CTRL-S* — вуаля, повідомлення про помилку зникло!

Запуск HelloWorld в Eclipse

Наш простий проект складається з одного класу, що не типово. Ось чому перед запуском проекту вперше, необхідно вказати Eclipse клас, який є головним в цьому проекті.

Виберіть в меню *Run* («Запуск»), далі *Run Configurations* («Виконати.»). У вікні, що відкрилося, перевірте, що в лівій частині вибраний пункт *Java Application* («Додаток Java»). На вкладці *Main* в полі *Project* («Проект») введіть ім'я проекту, в полі *Main class* («Головний клас») введіть ім'я головного класу.



Тепер, щоб запустити програму, натисніть кнопку *Run* («Запуск»). В результаті в панелі *Console* («Консоль») будуть надруковані слова *Hello World*, так само, як це було зроблено у розділі 1.

Тепер можна запускати проект на виконання за допомогою вибору в меню пунктів Run («Запуск»), *Run* («Запуск») або за допомогою натиснення клавіш *Ctrl-F11*.

Як працює програма HelloWorld

Давайте почнемо розбиратися – що ж фактично відбувається в програмі `HelloWorld`?

Клас HelloWorld містить лише один метод `main()`, який є точкою входу додатку на Java. На те, що `main` – це метод, вказують круглі дужки після слова `main`. Методи можуть викликати (використовувати) інші методи, наприклад, наш метод `main()`, щоб надрукувати на екрані текст `Hello World`, викликає метод `println()`.

Кожен метод починається з рядка оголошення, який називають сигнатурою методу:

```
public static void main(String[] args)
```

Сигнатура методу говорить нам про наступне.

- Хто має доступ до методу – `public`. Ключове слово `public` означає, що метод `main()` доступний для будь-якого іншого класу Java або самій JVM.
- Як викликати метод – `static`. Ключове слово `static` означає, що вам не потрібно створювати екземпляр (копію) об'єкту `HelloWorld` в пам'яті, щоб використовувати цей метод. Детальніше про екземпляри класу ми поговоримо в наступному розділі.
- Чи повертає метод якісь дані? Ключове слово `void` означає, що метод `main()` не повертає даних в програму, яка його викликала. В даному випадку це програма Eclipse. Проте, якщо метод повинен зробити якісь розрахунки, він може повертати отримані результати об'єкту, який його викликав.
- Ім'ям методу є слово перед круглими дужками – `main`.
- Список аргументів – деякі дані, які можуть бути передані методу – `String[] args`. У методі `main()` слова `String[]`

`args` означають, що цей метод може отримувати масив об'єктів з типом `String`, тобто текстові дані. Значення, які передаються методу, називаються аргументами або параметрами.

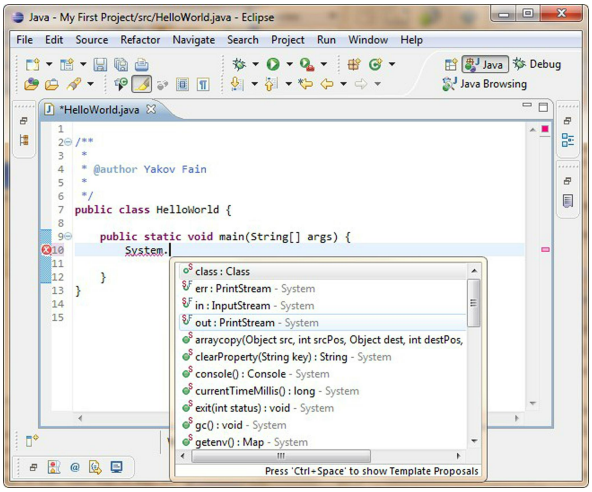
Як я вже кащав, програма може складатися з декількох класів, але лише один з них містить метод `main()`. Клас Java зазвичай містить декілька методів. Наприклад, клас `Game` може містити методи `startGame()`, `stopGame()`, `readScore()` і таке інше.

Тіло нашого методу `main()` містить лише один рядок:

```
System.out.println("Hello World");
```

Кожна команда або виклик методу повинен закінчуватися крапкою з комою (;). Метод `println()` знає як виводити дані в системну консоль (командну консоль). Після імені методів Java завжди йдуть круглі дужки. Якщо ви бачите метод з порожніми круглими дужками, це означає, що цей метод не має аргументів.

Слова `System.out` означають, що змінна `out` визначена усередині класу `System`, який поставляється разом з Java. Як же вам взнати, що в класі `System` є щось з ім'ям `out`? Eclipse допоможе вам з цим. Після того, як ви надрукуєте слово `System` і поставите крапку, Eclipse покаже вам все, що є в цьому класі. У будь-який момент ви можете помістити курсор після крапки і натискувати комбінацію клавіш *Ctrl-Space*, аби викликати вікно довідки, показане нижче.



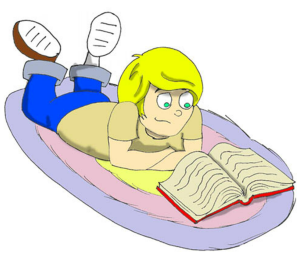
Слова `out.println()` говорять нам, що є об'єкт, який представлений змінною `out` і це «щось під назвою `out`» містить метод з ім'ям `println()`. Крапка, яка знаходиться між класом і ім'ям методу означає, що цей метод був оголошений всередині цього класу. Скажімо, у вас є клас `PingPongGame`, який містить метод `saveScore()` — зберігає рахунок в грі. Нижче наведений приклад, як ви можете викликати цей метод для Дейва, який виграв три гри:

```
PingPongGame.saveScore("Дейв", 3);
```

Нагадую, дані в круглих дужках називаються аргументи або параметри. Ці параметри даються методу, щоб він виконав над ними певну обробку, наприклад, зберіг дані на диск. Метод `saveScore()` має два аргументи — рядок тексту «Дейв», і число 3.

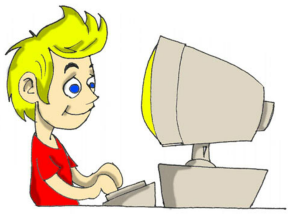
Створювати програми в Eclipse набагато приємніше, ніж в звичайному текстовому редакторі, правда? У Додатку Б є корисні поради, які дозволять вам прискорити процес програмування на Java в цьому чудовому IDE.

Матеріали для додаткового читання



Eclipse IDE Tutorial by Lars Vogel:
http://www.vogella.de/articles/Eclipse/article.html

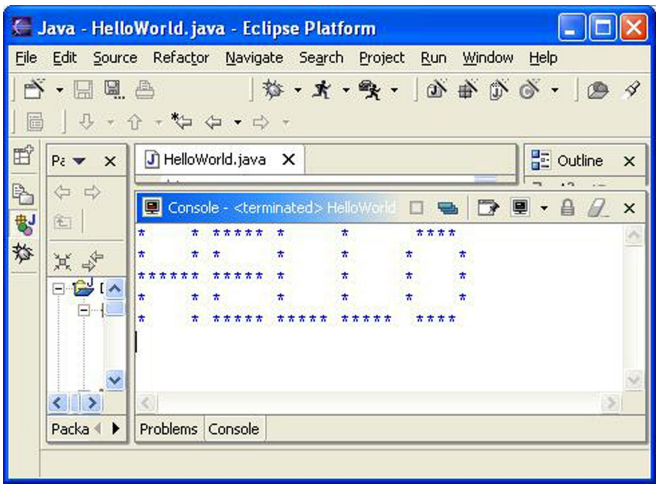
Практичні вправи



Змініть клас `HelloWorld`, щоб надрукувати свою адресу за допомогою декількох викликів методу `println()`.

Практичні вправи для розумників і розумниць

Змініть клас `HelloWorld`, щоб надрукувати слово *Hello* так, як показано нижче.



Розділ 3. Домашня Тварина і Риба Мовою Java

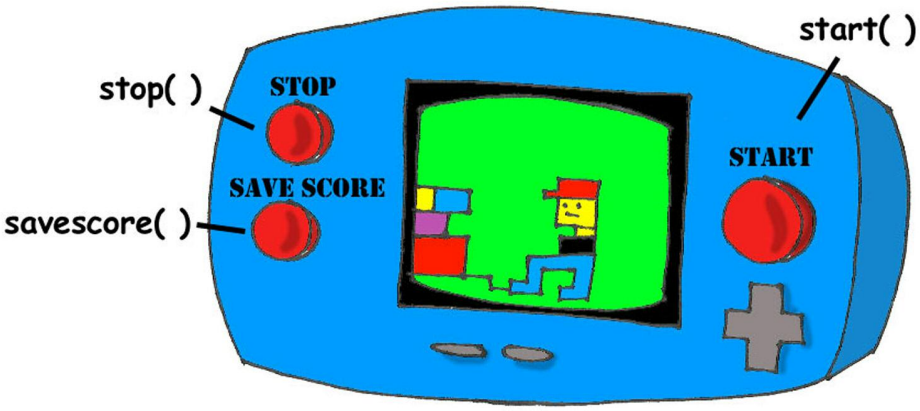
Програми мовою Java складаються з класів, що являють об'єкти реального світу. Люди розуміють по-різному, що значить добрий стиль програмування, проте більшість згодна, що потрібно прагнути програмувати в так званому об'єктно-орієнтованому стилі. Це означає, що хороші програмісти спочатку вирішують, які об'єкти включити в програму і які Java класи їх представлятимуть, а лише потім вже вони починають писати програму.

Класи і об'єкти

Давайте вигадаємо і обговоримо клас, який називатиметься VideoGame (відео гра). Цей клас може мати декілька методів, з яких буде ясно, що можуть робити об'єкти цього класу: почати гру, зупинити її, зберегти (запам'ятати) рахунок і так далі.

А ще цей клас може мати будь-які атрибути, наприклад: ціна, колір екрану, кількість ручок керування, і все таке інше.

- Класи Java можуть мати і методи, і атрибути.
- Методи визначають, що клас може зробити.
- Атрибути – це характеристики класу.



Наш клас може виглядати ось так:

```
class VideoGame  {
     int color; //колір
     String int price; //ціна

     void start () {
     }
     void stop () {
     }
     void saveScore(String playerName, int score) {
     }
}
```

Клас VideoGame буде схожий на інші класи, які представляють відео ігри, – у всіх у них є екрани, правда, різного кольору і розміру, всі вони виконують схожі дії, і всі вони коштують грошей.

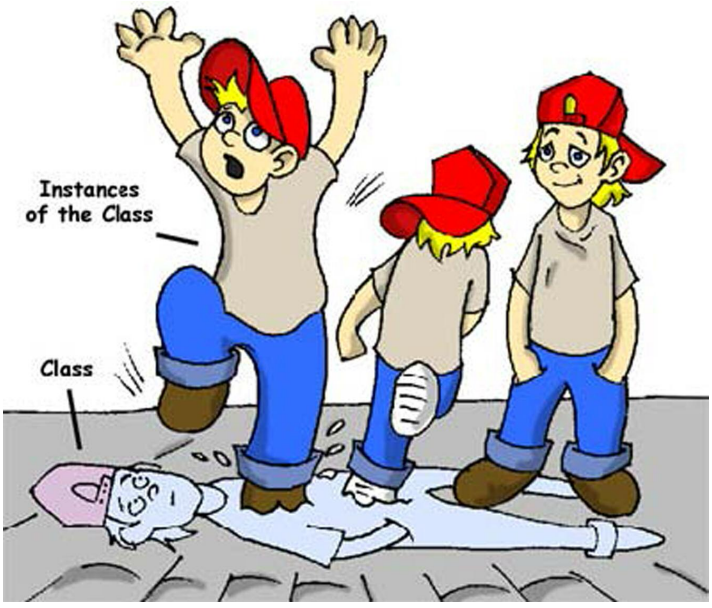
А тепер давайте додамо більше конкретних деталей і створимо інший клас з назвою GameBoyAdvance – колись популярна в Америці електронна гра. Цей клас теж належить сімейству відео ігор, проте він матиме деякі атрибути, які має лише гра GameBoy Advance, наприклад тип касети.

У даному прикладі клас GameBoyAdvance оголошує два атрибути cartridgeType and screenWidth і два методи – startGame() і stopGame(). Але ці методи ще не готові виконувати будь-які дії, тому що у них немає жодного

програмного коду між фігурними дужками.

```
class GameBoyAdvance {
      String cartridgeType; //тип касети
      int screenWidth; //ширина екрану
      void startGame() {
      }
      void stopGame() {
      }
}
```

Фабричний опис гри GameBoy Advance має таке ж відношення до вже зробленої гри, як Java-класс до його екземпляра в пам'яті комп'ютера. А процес виготовлення справжніх ігор на основі такого опису, схожий на процес створення екземплярів об'єктів GameBoyAdvance мовою Java.



У багатьох випадках програма може користуватися класом лише після створення його екземпляра. Адже виробники ігор теж створюють тисячі копій по тому ж самому опису. Не дивлячись на те, що ці копії представляють той самий клас, їхні атрибути можуть мати різні значення – якісь з них бла-

китні, якісь перламутрові, і тому подібне. Іншими словами, програма може створювати безліч екземплярів об'єктів GameBoyAdvance.

Типи Даних

Окрім слова клас вам доведеться звикати до ще одного нового значення слова об'єкт.

А фраза "створити екземпляр об'єкту" - просто означає створити копію об'єкту в пам'яті комп'ютера згідно опису цього класу.

Змінні є атрибутами класу, параметрами методу або просто можуть використовуватися для короткострокового зберігання будь-яких даних. Змінні спочатку мають бути оголошені, і лише після цього ними можна користуватися.

Пам'ятаєте рівняння типа y=x+2? У мові Java вам доведеться оголосити змінні x і y, використовуючи певний числовий тип даних, наприклад ціле число (integer або int) або число подвійної довжини (double):

```
int x;
int y;
```

Наступні дві строчки коду привласнюють значення цим змінним. Якщо ваша програма привласнить змінній ікс значення п'ять, змінна ігрек буде рівною семи:

```
x=5;
y=x+2;
```

Java дозволяє міняти значення змінної дещо незвичайним способом. Ось наприклад, як можна змінити значення змінної ігрек з п'яти на шість:

```
int y=5;
y++;
```

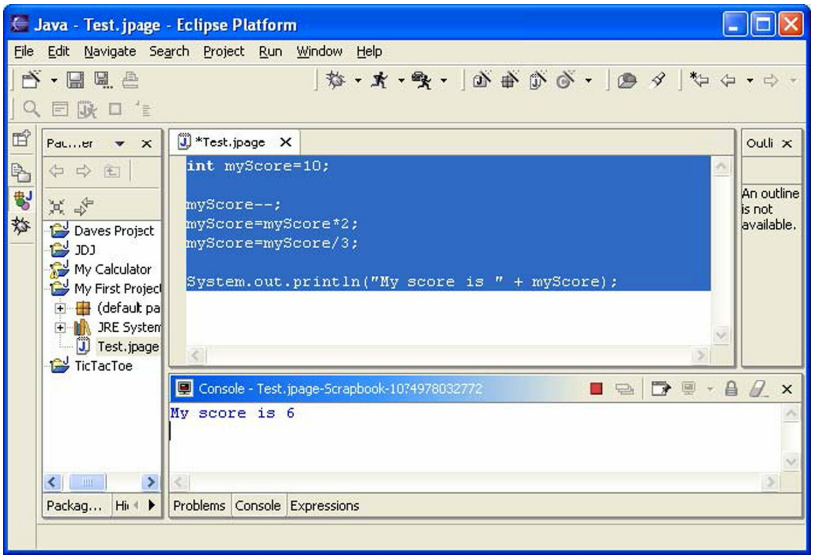
Не дивлячись на два знаки плюс, Java збільшить значення ігрека на одиницю. А після ось цього прикладу, значення змінної myScore теж шість:

```
int myScore=5;
myScore=myScore+1;
```

Так само можна робити множення, ділення і віднімання, ось дивіться:

```
int myScore=10;
myScore--;
myScore=myScore*2;
myScore=myScore/3;
System.out.println("My score is " + myScore);
```

Що ж надрукує цей код (до речі, "My score is" перекладається як "Мій рахунок")? У додатку Eclipse, де ми тепер пишемо програми, є класна штука під назвою чернетка (scrapbook) яка дозволяє вам легко тестувати будь-який шматочок коду навіть без створення класу. Виберіть меню File, New, Scrapbook Page і надрукуйте слово Test - це буде ім'я вашого файлу-чернетки. Тепер надрукуйте в чернетці п'ять рядків попереднього прикладу, висвітіть їх і натисніть на кнопку з маленьким збільшувальним склом:



Для того, щоб отримати результат обчислень, просто натисніть на закладку Console внизу екрану:

```
My score is 6
```

В даному прикладі параметр методу println() був склеєний з двох шматочків - тексту "My score is " і значення змінної myScore, яка була рівна шести. Таке "склеювання" рядків з шматочків називається к*онкатенація (concatenation)*. Не зважаючи на те, що змінна myScore зберігає не текст, а число: Java досить розумна, аби перетворити цю змінну в тип даних `String` і потім вже приклеїти її значення до тексту *"MyScore is"*.

Ось ще декілька прикладів того, як можна міняти значення змінних:

```
myScore=myScore*2; // те ж що myScore*=2;
myScore=myScore+2; // те ж що myScore+=2;
myScore=myScore-2; // те ж що myScore-=2;
myScore=myScore/2; // те ж що myScore/=2;
```

У мові Java є вісім простих (примітивних) типів даних, і вам вирішувати якими користуватись у вашій програмі. Це, звичайно, залежить від того, дані якого типу і розміру вам потрібно зберігати в цих змінних:

- Чотири типи даних для зберігання цілих чисел - byte, short, int, та long.
- Два типи даних для значень з десятковою крапкою - float і double.
- Один тип даних для зберігання однієї букви - char.
- Один логічний тип, званий boolean, який може мати лише два значення: true або false (істина і хиба).

Java дозволяє привласнювати початкове значення змінної під час її оголошення. У народі це називається ініціалізація змінних:

```
char grade =  'A';
int chairs = 12;
boolean playSound = false;
```

```
double nationalIncome = 23863494965745.78;
float gamePrice = 12.50f;
long totalCars =46372836483921;
```

У останніх двох рядках f означає float, а l означає long.

Якщо ви все-ж забудете ініціалізувати змінні, Java сама привласнить нуль числовим змінним, false змінних типа boolean, і спеціальний код '\u0000' змінним типа char.

А ще є спеціальне ключове слово final, і якщо воно присутнє в оголошенні змінної, вам буде дозволено привласнити значення цієї змінної лише один раз і ви не зможете вже змінити це значення. У деяких мовах програмування final-змінні називаються константами. Прийнято називати константи великими літерами.

```
final String STATE_CAPITAL="Вашингтон";
```

Окрім примітивних типів даних, ви можете використовувати класи для оголошення змінних. У кожного примітивного типа даних є відповідний клас-обгортка, наприклад `Integer`, `Double`, `Boolean` та інші. Ці класи мають багато корисних методів, щоб перетворювати дані з одного типа в інший.

Примітивний тип даних char може зберігати лише одну букву, але в мові Java існує клас `String` для роботи з довшим текстом:

```
String lastName="Сміт";
```

Імена змінних не можуть починатися з цифри і не можуть містити пропуски.

- Біт це найменша порція даних, яка може зберігатися в пам'яті. Ви можете зберігати в біті лише 1 або 0.
- Байт складається з восьми бітів.
- char займає два байти в пам'яті комп'ютера.
- int і float займають чотири байти пам'яті.
- Змінним long і double потрібно по вісім байтів.

Числові типи даних, які займають більше пам'яті, можуть зберігати ве-

ликі величини.

- 1 кілобайт (KB) - це 1024 байтів
- 1 мегабайт (MB) - це 1024 кілобайт
- 1 гігабайт (GB) має 1024 мегабайтів

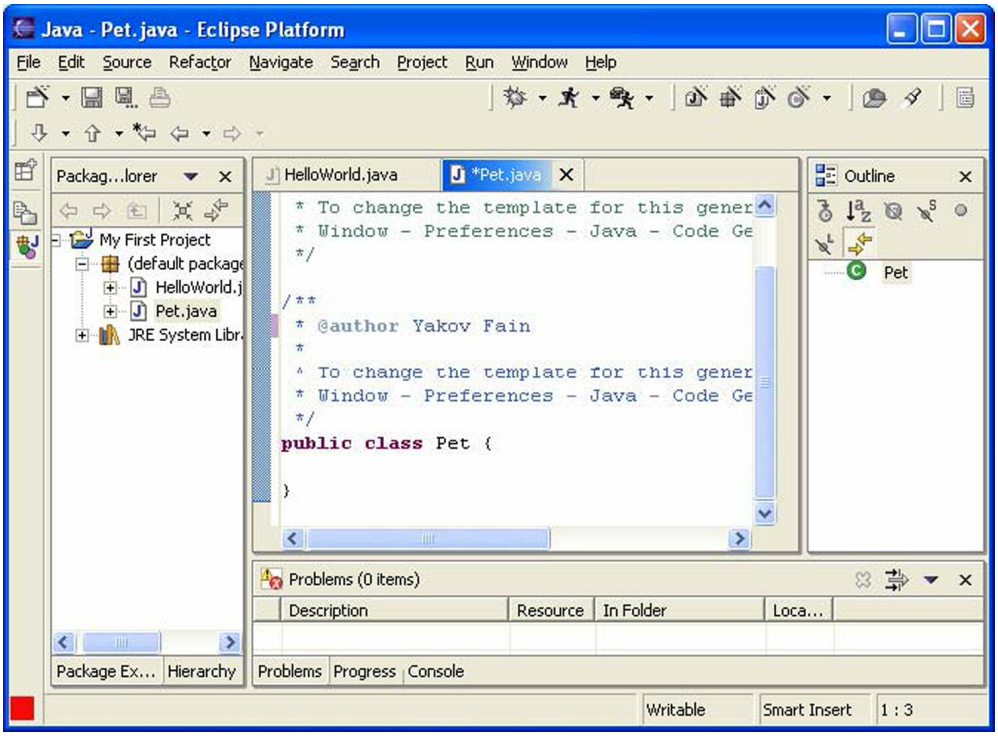
Створюємо Домашню Тварину

Давайте вигадаємо і створимо клас Домашня Тварина (англійською просто Pet). Спочатку потрібно вирішити, які дії наш Pet зможе виконувати. Як щодо їсти, спати і говорити (eat, sleep, say)?

Ми запрограмуємо ці дії в методах класу Pet. А ще ми дамо нашій домашній тварині такі атрибути: вік (age), зріст (height), вага (weight) і колір (color).

Почнемо зі створення нового класу на ім'я Pet в проекті MyFirstProject з другого розділу, але при цьому не ставте галочку, що вимагає створення методу `main()`.

Ваш екран виглядатиме приблизно так:



Тепер ми готові оголосити атрибути і методи в класі Pet. Код в класах і методах має бути оточений фігурними дужками. Кожна відриваюча дужка повинна мати свою закриваючу:

```
class Pet{
}
```

Давайте виберемо типи даних для атрибутів нашого класу. Я пропоную `int` для віку, `float` для ваги і зросту, `String` для кольору.

```
class Pet{
     int age;
     float weight;
     float height;
     String color;
}
```

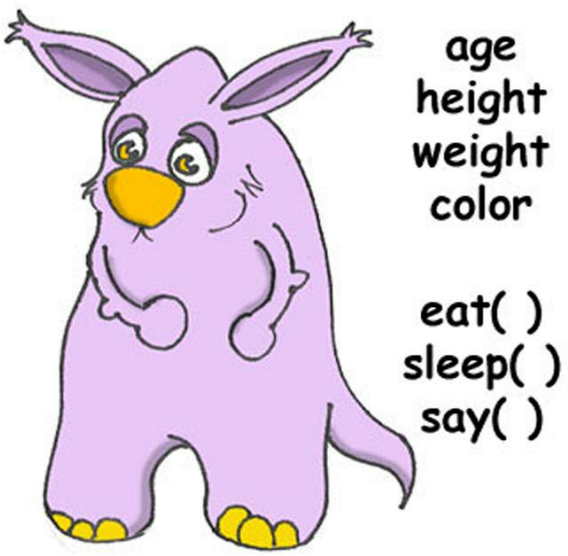
Додамо декілька методів в наш клас. Тут потрібно вирішити чи будуть ці методи мати параметри і повертати якісь дані:

- Метод sleep() просто друкуватиме фразу Добраніч, до завтра - йому не потрібні жодні параметри і він не повертатиме жодних значень.
- Те ж саме відноситься і до методу eat() - він друкуватиме повідомлення Я дуже голодний...давайте перекусимо чіпсами!
- Хоча метод say() теж друкуватиме повідомлення, але наша домашня тварина ще і "виголошуватиме" слово або фразу, яку ми йому дамо, як параметр. Цей метод будуватиме фразу, що містить значення цього параметра, після чого фраза повертатиметься програмі.

Нова версія класу Pet буде виглядати так:

```
public class Pet  {
      int age;
      float weight;
      float height;
      String color;

      public void sleep(){
            System.out.println("На добраніч! До завтра");
      }

      public void eat(){
            System.out.println("Я дуже голодний,
                  давайте перекусимо чіпсами!");
      }

      public String say(String aWord){
            String petResponse = "Ну гаразд!! " + aWord;
            return petResponse;
      }
}
```

Цей клас являє собою ось таку симпатичну істоту з реального світу:



Давайте поговоримо про заголовок (сигнатуру) методу sleep():

```
public void sleep()
```

Цей заголовок говорить нам, що метод sleep() можна викликати з будь-якого іншого класу (`public`) і що метод не повертає жодних даних (`void`). Порожні дужки означають, що цей метод не має параметрів - адже йому не потрібні жодні дані з довкілля, тому що він завжди друкує один і той самий текст. Заголовок методу say() виглядає так:

```
public String say(String aWord)
```

Цей метод теж можна викликати з будь-якого іншого класу, але він ще і повертає якийсь текст - це і є роль ключового слова `String`, що стоїть перед ім'ям методу. А крім того, цей метод чекає якісь текстові дані ззовні і для цього в заголовок методу включений параметр `String aWord`.

А як же визначити - чи повинен метод повертати дані? Якщо метод виконує якісь дії над даними і повинен передати результат цих дій классу, що його викликав, то він повинен повертати дані. Ви можете заперечити, що в класі Pet немає жодного викликаючого класу! Це так, тому ми зараз і ство-

римо клас PetMaster (Господар). У цього класу буде метод `main()`, що міститиме код для роботи з класом Pet.

Створіть новий клас PetMaster, але цього разу поставте галочку в Eclipse біля питання *чи створювати метод main()*. Не забувайте, що без цього методу ви не зможете стартувати програму. Додайте декілька рядків коду до класу, який зробив для вас Eclipse, аби він виглядав так:

```
public class PetMaster {
     public static void main(String[] args) {
          String petReaction;
          Pet myPet = new Pet();
          myPet.eat();
          petReaction = myPet.say("Цвінь!! Цвірінь!!");
          System.out.println(petReaction);
          myPet.sleep();
     }
}
```

Не забудьте натиснути *CTRL-S*, аби зберегти і відкомпілювати цей клас А для того, щоб стартувати програму PetMaster, виберіть наступні меню в Eclipse *Run, Run..., New* і надрукуйте ім'я головного класу - PetMaster. Натисніть на кнопку Run, і програма надрукує такий текст:

```
Я дуже голодний,  давайте перекусимо чіпсами!
Ну гаразд!! Цвінь!! Цвірінь!!
На добраніч,  до завтра
```

PetMaster - це викликаючий клас і він спочатку створює екземпляр об'єкту Pet. Він оголошує змінну myPet і використовує оператор *new*:

```
Pet myPet = new Pet();
```

Цей рядок оголошує змінну myPet типа Pet (так, ви можете використовувати будь-які класи, створені вами, як нові типи даних). Тепер змінна myPet

знає місце в пам'яті, де був створений екземпляр об'єкту Pet, і можна користуватися цією змінною, щоб викликати будь-які методи класу Pet, наприклад:

```
myPet.eat();
```

Якщо метод повертає якесь значення, його можна викликати інакше. Оголосіть змінну того-ж типу, що і значення, яке повертається, і викличте метод так, щоб привласнити це значення змінній, наприклад так:

```
String petReaction;
petReaction = myPet.say("Цвінь!!  Цвірінь!!");
```

Тепер повернене значення знаходиться в змінній petReaction і його дуже просто можна роздрукувати:

```
System.out.println(petReaction);
```

Наслідування - Рибка Теж Домашня Тварина

Наш клас Pet познайомить вас з ще однією важливою особливістю мови Java, яка називається *наслідування (inheritance)*. У реальному світі кожна людина успадковує щось від своїх батьків. У світі Java ви теж можете створити новий клас за типом, що вже існує.

Клас Pet, і поводиться, і має атрибути типові для багатьох домашніх тварин - вони їдять, сплять, деякі з них видають звуки, їх шкіра має колір і так далі. З іншого боку, домашні тварини відрізняються один від одного - собаки гавкають, рибки беззвучно плавають, папужки розмовляють краще, ніж собаки.

І все-ж, всі вони сплять, їдять, і мають зріст і вагу. Тому набагато легше створити клас Fish (риба) так, щоб він успадкував загальні риси і поведінку в класу Pet, ніж кожного разу створювати з початку класи для собак, папуг і риб.

Для цього і існує спеціальне ключове слово `extends`:

Тепер ви маєте повне право сказати що Fish - це *підклас* класу Pet, а клас Pet - це *супер-клас* класу Fish. Ми використовували клас Pet як своєрідний шаблон для створення класу Fish.

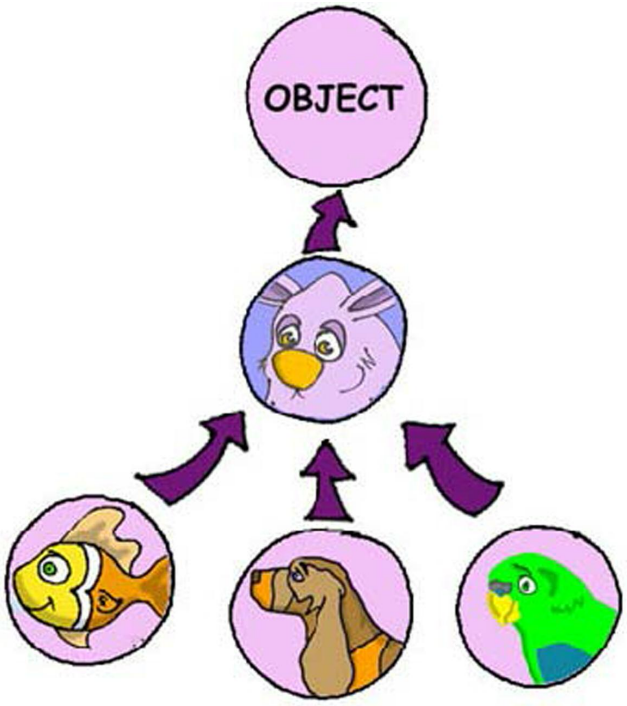
```
class Fish extends Pet{
}
```

Навіть якщо ви залишите клас Fish таким, як він є зараз, все одно вже можна використовувати кожен метод і атрибут, успадкований з класу Pet. Ось погляньте:

```
Fish myLittleFish = new Fish();
myLittleFish.sleep();
```

Хоч ми ще і не оголошували жодних методів в класі Fish, вже можна викликати метод sleep(), що знаходиться в його супер-класі!

Немає нічого легшого за створення підкласів в додатку Eclipse! Виберіть меню File, New, Class і надрукуйте слово Fish, як ім'я класу. Замініть в полі супер-клас java.lang.Object на слово Pet.



Не забувайте, що ми створюємо підклас класу Pet, щоб додати те, що властиво лише рибам, а загальний для всіх тварин код, оголошений в супер-класі, ми просто використовуємо.

Не всі домашні тварини можуть пірнати, але рибки, звичайно ж, можуть. Давайте додамо до класу Fish метод dive() - упірни.

У методу dive() є параметр howDeep, який "говорить" рибці, як глибоко вона повинна пірнути. А ще ми оголосили змінну currentDepth, куди поміщатимемо поточне значення глибини при кожному виклику методу dive(). Цей метод повертає значення змінної currentDepth викликаючому класу.

Зробіть, будь ласка, ось такий клас FishMaster:

Пора розповісти маленький секрет - всі класи в мові Java успадковані з супер-дупер класу Object, навіть якщо ви і не використали ключове слово extends.

На відміну від людей, Java-класи не можуть мати двох батьків.

А якби у нас це було як в мові Java, діти не були б "підкласами" своїх батьків, а всі хлопчики походили б від Адама, а дівчатка - від Єви ☺.

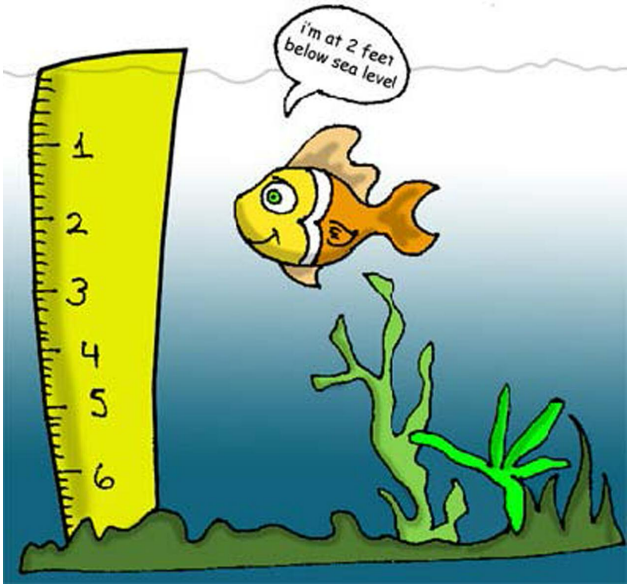
```
public class Fish extends Pet {
     int currentDepth=0;
     public int dive(int howDeep){
          currentDepth=currentDepth + howDeep;
          System.out.println("Занурююсь на глибину "
                              +  howDeep + " футів");
          System.out.println("Я на глибині "
                    + currentDepth + " футів нижче за
                    рівень моря");
          return currentDepth;
     }
}
```

Метод main() у класі FishMaster створює екземпляр об'єкту Fish і двічі викликає його метод dive() з різними параметрами. Після цього він викликає метод sleep().

```
public class FishMaster {
     public static void main(String[] args) {
          Fish myFish = new Fish();
          myFish.dive(2);
          myFish.dive(3);
          myFish.sleep();
     }
}
```

```
Під час виконання FishMaster надрукує наступне:
Занурююсь на глибину 2 футів
Я на глибині 2 футів нижче за рівень моря
Занурююсь на глибину 3 футів
Я на глибині 5 футів нижче за рівень моря
На добраніч,  до завтра
```

Ви помітили, що FishMaster викликає не лише методи оголошені в класі Fish, але також і метод sleep() його супер-класу Pet? Отож-бо! У цьому і є вся краса наслідування - вам не потрібно копіювати код з класу Pet. Просто напишіть слово extends і клас Fish зможе користуватися методами класу Pet



Так, іще одне, хоча метод dive() і повертає значення змінної currentDepth, наш FishMaster їм не користується. Це не біда, просто нашому класу FishMaster воно не потрібне. Але якимось іншим класам, які теж можуть працювати з класом Fish, це значення може бути дуже навіть корисно. Уявіть, наприклад, клас FishTrafficDispatcher (регулювальник руху риб), який повинен знати положення інших риб в морі, перш ніж дозволити пірнання щоб уникнути дорожньо-транспортних пригод ☺.

Перевизначення методів

Ви, звичайно, знаєте, що риби не говорять (принаймні вони не роблять це гучно). Але наш клас Fish був успадкований з класу Pet, в якого є метод say(). Це означає, що ви безперешкодно можете написати щось в цьому роді:

myFish.say();

Ну і ну, наші рибки заговорили... Щоб уникнути цього, в класі Fish потрібно перевизначити (override) метод say(), оголошений в класі Pet. Це працює так: якщо ви оголошуєте в під-класі метод, що має точно такий само заголовок, як в його супер-класі, Java виконає метод під-класу, замість методу супер-класу. Давайте додамо до класу Fish метод say().

```
public String say(String something){
     return "Ти що, не знаєш, що риби не розмовляють?";
}
```

А тепер викличемо метод say() з методу main() класу FishMaster:

```
myFish.say("Привіт");
```

Виконайте цю програму і вона надрукує наступне:

```
Ти що, не знаєш, що риби не розмовляють?
```

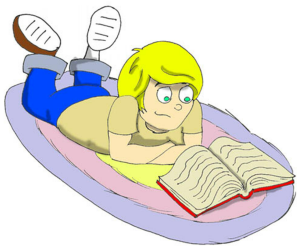
Це підтверджує, що метод say() класу Pet був перевизначений.

Оце так! Ми вивчили багато нового у цьому розділі - давайте відпочинемо.

Якщо заголовок методу включає ключове слово final, такий метод перевизначити не можна, наприклад:

```
final public void sleep(){...}
```

Додаткове читання



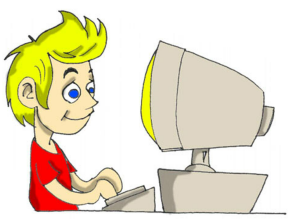
1.Java Data Types:
http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html

2.About inheritance:
http://java.sun.com/docs/books/tutorial/java/concepts/inheritance.html

Практичні вправи

1. Створіть новий клас Car (автомобіль) и введіть до нього такі методи:

```
public void start()
public void stop()
public int drive(int howlong)
```



Метод drive() (їдь) повинен повертати загальну відстань, що пройшов автомобіль за заданий час. Використовуйте наступну формулу дла розрахунку відстані:

```
distance = howlong*60;
```

2. Створіть ще один клас CarOwner (власник автомобиля), який буде створювати экземпляр об'єкту Car і викликати його методи. Результат кожного такого виклику повинен бути надрукований за допомогою `System.out.println()`.

Практичні вправи для розумників і розумниць

Зробіть підклас класу Car, назвіть його JamesBondCar (автомобіль Джеймса Бонда) і перевизначіть в ньому метод drive(). Використовуйте наступну формулу для розрахунку відстані:

```
distance = howlong*180;
```

Будьте винахідливі! Друкуйте смішні повідомлення.

Розділ 4. Основні конструкції мови Java

У програму, написану мовою Java, можна додавати будь-які текстові коментарі, щоб пояснити, для чого потрібен конкретний рядок коду, метод або клас. Через деякий час буває важко пригадати, чому ви написали програму саме так, а не інакше. Ще одна причина писати коментарі - це допомогти іншим програмістам зрозуміти ваш код.

Коментарі в програмі

Існує три типи коментарів:

1 Якщо ваш коментар укладається в один рядок, почніть його з двох косих рисок:

```
// Цей метод обчислює відстань
```

2 Довші багаторядкові коментарі повинні бути обмежені символами /* і */, наприклад:

```
/* наступні 3 рядки коду
   потрібні для збереження позиції риби.
*/
```

3 Разом з Java поставляється утиліта javadoc, яка дозволяє витягти коментарі з вашої програми в окремий файл довідки. Цей файл може бути використаний як технічна документація для ваших програм. Такі коментарі повинні знаходитися між символами /** і */. Лише найважливіші коментарі, такі як опис класу або методу, слід поміщати між цими символами.

```
/** Цей метод обчислює розмір знижки в
   залежності від ціни. Якщо ціна більше $100,
   знижка = 20%, в іншому випадку лише 10%.
*/
```

Далі я додаватиму коментарі в приклади коду, щоб дати краще уявлення про те, де і як їх використовувати.

Ухвалення рішень за допомогою оператора if

У нашому житті ми постійно приймаємо рішення: *якщо вона скаже мені так - то я відповім їй ось так, інакше я зроблю по-іншому*. У Java є оператор if, який перевіряє, чи є деякий вираз істинним (`true`) або помилковим (`false`).

На підставі результатів цього виразу виконання програми розгалужується, і лише одна відповідна частина коду виконуватиметься.

Наприклад, якщо умова *Чи хочу я піти до бабусі?* повертає true, ми повертаємо ліворуч, інакше йдемо праворуч.

Якщо вираз повертає істину, JVM виконуватиме код що знаходиться між першими фігурними дужками, інакше виконається код, що знаходиться в блоці else. Наприклад, якщо ціна більше 100 доларів, то зробити 20% знижки, інакше лише 10%.

```
// Більш дорогі товари продаються зі знижкою 20%
if (price > 100){
     price=price*0.8;
     System.out.println("Ваша знижка 20%");
}
else {
     price=price*0.9;
     System.out.println("Ваша знижка 10%");
}
```

Давайте змінимо метод dive() у класі Fish так, щоб обмежити сотнею метрів глибину, на якій може плавати наша рибка:

```
public class Fish extends Pet {
     int currentDepth=0;
     public int dive(int howDeep){
          currentDepth=currentDepth + howDeep;
          if (currentDepth > 100) {
            System.out.println("Я маленька рибка "+
               " і не можу плавати глибше 100 метрів");
            currentDepth=currentDepth - howDeep;
          }
          else{
            System.out.println("Занурюся ще на " +
                        howDeep + " метрів");
            System.out.println("Я на глибині " +
                        currentDepth + " метрів");
          }
          return currentDepth;
     }
     public String say(String something){
          return "То хіба ви не знаєте, що риби
                                   не говорять?";
     }
}
```

Тепер зробимо невелику зміну в класі FishMaster - давайте спробуємо занурити нашу рибку на глибину більше 100 метрів:

```
public class FishMaster {
     public static void main(String[] args) {
          Fish myFish = new Fish();
          //Спробуємо змусити рибу зануритися нижче 100 метрів
          myFish.dive(2);
          myFish.dive(97);
          myFish.dive(3);
          myFish.sleep ();
     }
}
```

Запустіть цю програму, і вона видасть наступне:

```
Занурююся ще на 2 метрів
Я на глибині 2 метрів
Занурююся ще на 97 метрів
Я на глибині 99 метрів
Я маленька рибка і не можу плавати глибше 100 метрів
На добраніч,  побачимося вранці
```

Логічні оператори

Інколи, аби прийняти рішення, необхідно перевірити більше одного умовного виразу. Наприклад, якщо назва штату Техас (Texas) або Каліфорнія (California), потрібно додати податок штату до ціни кожного товару в магазині. Це приклад *логічного або* - або Техас або Каліфорнія. Це працює так - якщо будь-яка з двох умов вірна (`true`), результат всього виразу - теж вірний (`true`).

В Java знак *логічного або* позначається однією або двома вертикальними лініями. У наступному прикладі я використовуватиму змінну типа String. В цього класу в Java є метод `equals()` для перевірки рядків на рівність і я використовуватиму його, щоб перевірити чи рівна змінна state (штат) *Техас* або *Каліфорнія*:

```
if (state.equals("Texas") | state.equals("California"))
```

Можна також використовувати дві вертикальні лінії в операторі `if`:

```
if (state.equals("Texas") || state.equals("California"))
```

Різниця між цими умовами в тому, що якщо ви використовуєте дві вертикальні лінії, і вираз, що стоїть зліва вірний (`true`), то вираз, що стоїть справа, навіть не перевіряється. Якщо використовується одна вертикальна лінія, JVM завжди обчислюватиме результат обох виразів.

Логічне і задається одним або двома амперсандами (&&). Увесь вираз істинний, якщо істинна кожна частина цього виразу. Наприклад, знімає по-

даток штату, лише якщо цей штат (state) - Нью-Йорк і ціна (price) більше 110 доларів. Обидві умови мають бути істинними одночасно:

```
if (state.equals("New York") && price>110)
```

або

```
if (state.equals("New York") & price>110)
```

Якщо використовується подвійний амперсанд і перший вираз помилковий (`false`), то другий вираз навіть не перевірятиметься, оскільки незалежно від нього, увесь вираз буде помилковим. При одному амперсанді будуть обчислюватись обидві умови.

Логічне не (negation) позначається знаком оклику. Воно міняє значення виразу на протилежне. Наприклад, якщо деяка дія може бути виконана лише якщо штат не Нью-Йорк, можна написати так:

```
if (!state.equals("New York"))
```

Інший приклад - наступні вирази абсолютно ідентичні:

```
if (price < 50)
if (!(price >= 50))
```

У другому випадку *логічне не* застосовується до результату обчислення виразу в дужках.

Умовний оператор

Існує ще один тип оператора if, який називається умовний оператор. Він використовується для того, щоб привласнити значення змінній, залежно від істинності виразу, що стоїть перед знаком питання. Якщо вираз достеменний, привласнюється значення що стоїть відразу після знаку питання,

інакше - значення що стоїть після двокрапки:

```
discount = price > 50? 10:5;
```

Якщо ціна більше п'ятдесяти, значення змінної discount (знижка) буде рівне десяти, інакше - п'яти. Це просто коротша форма запису звичайного оператора if:

```
if (price > 50){
     discount = 10;
} else {
     discount = 5;
}
```

Використання else if

Також можна створювати складені оператори `if`, що містять декілька `else if` блоків. Давайте створимо клас `ReportCard` (табель українською), в якого буде метод `main()`, а також метод з одним аргументом, в який передається чисельний результат деякого тесту. Залежно від цього числа, метод друкуватиме вашу оцінку за буквеною системою оцінок (A, B, C, D). Назвемо цей метод `convertGrades()`.

```
public class ReportCard {
  /**
  Метод convertGrades приймає один цілочисельний аргумент
  - результат тесту і повертає символ A,  B,   C or D
  залежно від цього аргументу.
  */
  public char convertGrades(int testResult){

     char grade;

     if (testResult >= 90){
```

```
        grade = 'A';
      }else if (testResult >= 80 && testResult < 90){
        grade = 'B';
      }else if (testResult >= 70 && testResult <  80){
        grade = 'C';
      }else {
        grade = 'D';
      }
      return grade;
  }

  public static void main(String[] args){

      ReportCard rc = new ReportCard();
      char yourGrade = rc.convertGrades(88);
      System.out.println("Ваша перша оцінка " + yourGrade);
      yourGrade = rc.convertGrades(79);
      System.out.println("Ваша друга оцінка " + yourGrade);
  }
}
```

Окрім використання `else if` в даному прикладі демонструється, як використовувати змінні типу `char`. Також можна побачити, що за допомогою оператора `&&` можна перевірити, чи потрапляє число у вказаний діапазон. Не можна написати "якщо testResult між 80 і 89", в Java потрібно писати, що testResult має бути більше або рівний 80, і (&) в той же час менший 89, ось таким чином:

```
testResult >= 80 && testResult < 89
```

Подумайте, чому ми тут не використовуємо оператор ||.

Оператор `switch` і ухвалення рішень

Оператор `switch` інколи використовується як альтернатива `if`. Значення змінної, що стоїть після оператора `switch` обчислюється, і програма переходить лише до одного з `case` блоків, аргумент якого збігається з результатом цього обчислення:

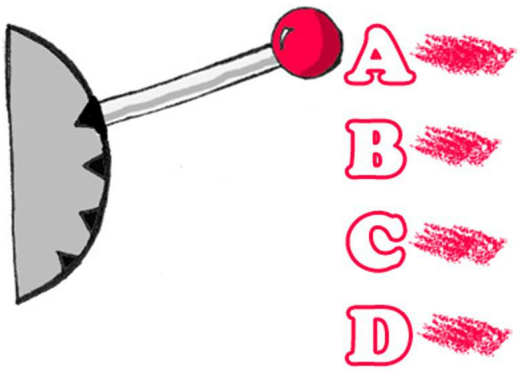
```
public static void main(String[] args){

  ReportCard rc = new ReportCard();

  char yourGrade = rc.convertGrades(88);

  switch (yourGrade) {

     case 'A':
       System.out.println("Чудова робота!");
       break;

     case 'B':
       System.out.println("Хороша робота!");
       break;

     case 'C':
       System.out.println("Треба підтягнути знання!");
       break;

     case 'D':
       System.out.println("Будь серйознішим!");
       break;
  }
}
```

Не забудьте написати ключове слово `break` в кінці кожного `case` блоку, щоб після завершення виконання його коду, стався вихід з оператора

switch. Якщо ви не напишете break, то друкуватимуться всі чотири рядки, не дивлячись на те, що значення змінної yourGrade має лише одне значення.

Раніше оператор switch мав обмеження - змінна, яка в нього передається, повинна була мати один з наступних типів:

```
char
int
byte
short
enum
```



Починаючи з Java 1.7, тип String так само може використовуватися у операторі switch. Крім того, завдяки автобоксингу, який з'явився в Java 1.5 можуть використовуватися типи обгортки: Character, Byte, Short, і Integer.

Як довго живуть змінні?

Усередині методу convertGrades() класу ReportCard оголошується змінна grade. Змінна, оголошена усередині будь-якого методу називається *локальною*. Це означає, що вона існує і доступна лише *усередині цього методу*. Після того, як метод виконався, локальна змінна автоматично видаляється з пам'яті.

Програмісти також використовують термін *зона дії (scope)*, щоб задати скільки часу та або інша змінна існуватиме.

Якщо змінна має бути використана декількома методами, то її потрібно оголосити поза всіма методами. У класі Fish, currentDepth це *атрибут класу (member variable)*. Термін життя цих змінних визначається терміном життя об'єкту Fish, тому вони ще називаються *атрибутами екземпляра класу (instance variables)*. Такі змінні можуть спільно використовуватися всіма методами класу, і, в деяких випадках, навіть бути доступними для інших класів.

Наприклад, у вислові `System.out.println()` використовується змінна out, яка оголошена в класі `System`.

Хвилиночку! А хіба можна використовувати атрибут класу System, якщо ми не створювали екземпляра цього класу? Та можемо, якщо змінна оголошена за допомогою ключового слова static (статичний). Якщо оголошення атрибуту класу або методу починається із слова static, то не обов'язково створювати екземпляр класу, аби їх використовувати. Статичні атрибути класу використовуються для зберігання значень загальних для всіх екземплярів класу.

Наприклад, метод convertGrades() може бути оголошений в класі ReportCard як статичний, тому що в реалізації цього методу для читання і запису не використовуються атрибути, специфічні для конкретного екземпляра класу. Статичний метод sqrt () з класу Math можна викликати ось так:

```
double squareRoot = Math.sqrt(4.0);
```

Спеціальні методи: конструктори

У Java для створення екземплярів класів і виділення під них пам'яті використовується оператор new, наприклад:

```
Fish myFish = new Fish();
```

Круглі дужки після слова Fish говорять про те, що в цього класу визначений метод Fish(). Так і є, існують спеціальні методи, які називаються *конструкторами (constructors)*, і в цих методів є наступні особливості:

- Конструктори викликаються лише один раз при створенні об'єкту в пам'яті.
- Вони повинні називатися так само, як називається клас.
- Вони нічого не повертають, не потрібно навіть писати слово void в сигнатурі цього методу.

У класу може бути декілька конструкторів. Якщо ви не написали жодного конструктора, під час компіляції Java автоматично створить за вас так

званий *порожній конструктор за замовченням (default no-argument constructor)*. Ось чому компілятор ніколи не "лаятиметься" на вираз new Fish(), навіть якщо в класі Fish ви не оголосили жодного конструктора.

В основному, конструктори використовуються для привласнення початкових значень атрибутам класу, наприклад, наступна версія класу Fish включає конструктор з одним аргументом, який задає початкове значення атрибуту currentDepth рівним значенню аргументу конструктора.

```
public class Fish extends Pet {

     int currentDepth;

     Fish(int startingPosition){
          currentDepth=startingPosition;
     }

}
```

Тепер клас FishMaster може створити екземпляр класу Fish і задати початкове положення рибки. Нижче створюється екземпляр класу Fish, який спочатку занурює рибку в море на глибину 20 метрів:

```
Fish myFish = new Fish(20);
```

Для класу, в якому був визначений конструктор з аргументами, конструктор за замовчуванням створюватися автоматично не буде. Якщо вам необхідний конструктор без аргументів - напишіть його.

Ключове слово `this`

Ключове слово this корисно, коли потрібно послатися на екземпляр класу усередині об'єкту цього класу. Розглянемо наступний приклад:

```
class Fish {
      int currentDepth ;

      Fish(int currentDepth){
            this.currentDepth = currentDepth;
      }
}
```

Тут ідентифікатор this допомогає уникнути конфлікту імен, наприклад, this.currentDepth посилається на атрибут класу currentDepth, у той час, як currentDepth посилається на значення аргументу конструктора.

Іншими словами, екземпляр класу Fish вказує на самого себе за допомогою слова this.

Інший важливий приклад використання ключового слова `this`, ви зустрінете у розділі 6 в секції *Як передавати дані між класами*.

Масиви

Передбачимо, програма повинна зберегти імена чотирьох гравців. Замість того, щоб оголошувати чотири змінні типу `String`, можна оголосити масив, який містить чотири елементи типу `String`. Масиви позначаються за допомогою квадратних дужок, поміщених після типу даних або після імені змінної:

```
String[] players;
```

або

```
String players[];
```

Ці інструкції повідомляють компілятору Java, що ви плануєте зберегти декілька рядків в масиві players. Кожен елемент масиву має свій індекс, починаючи з нуля. У наступному прикладі створюється масив, який може зберігати чотири об'єкти типа String, і потім елементам масиву привласнюються значення:

```
String players[] = new String [4];

players[0] = "David";
players[1] = "Daniel";
players[2] = "Anna";
players[3] = "Gregory";
```

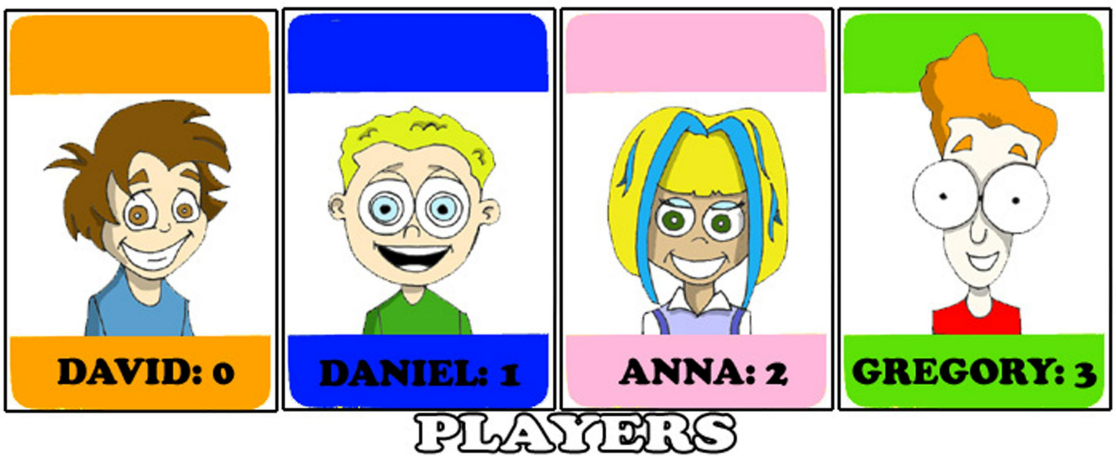
Необхідно знати розмір масиву, перед тим, як задавати значення для його елементів. Якщо кількість елементів заздалегідь невідома, масиви не можуть бути використані. У таких випадках, замість масивів використовують інші класи Java, наприклад, ArrayList, але давайте сконцентруємося на масивах.

У кожного масива є атрибут length, який "пам'ятає" кількість елементів у масиві, і ви завжди можете дізнатись, як багато елементів у вас є:

```
int totalPlayers = players.length;
```

Якщо під час ініціалізації масиву, ви знаєте значення всіх елементів, які в нім зберігатимуться, то Java дозволяє створити такий масив в один рядок:

```
String[] players = {"David", "Daniel", "Anna", "Gregory"};
```



Уявімо, що в нашій грі переміг другий учасник і хочеться його привітати. Якщо ім'я гравця збережене в масиві, потрібно витягувати другий елемент:

```
String theWinner = players[1];
System.out.println("Вітаємо, " + theWinner + "!");
```

Цей код виведе на екран наступне:

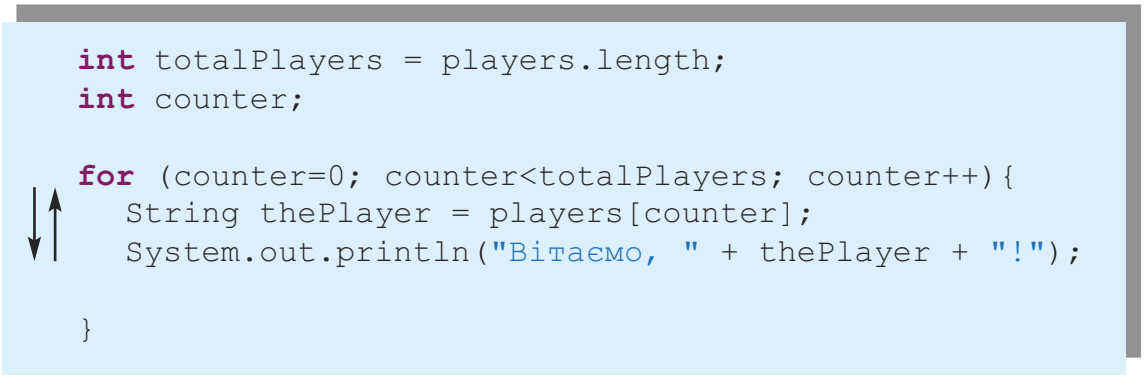
```
Вітаємо, Daniel!
```

Ви знаєте, чому другий елемент має індекс [1]? Звичайно, знаєте, тому що індекс першого елементу завжди [0].

Масив гравців в нашому прикладі *одновимірний (one-dimensional)*, тому що ми зберігаємо їх в ряд. Якщо ми хочемо зберегти значення у вигляді матриці, ми можемо використовувати двовимірний масив. Java дозволяє створювати *багатовимірні масиви (multi-dimensional arrays)*. Ви можете зберігати в масивах будь-які об'єкти, і я покажу, як це робиться у розділі 10.

Повторення дій за допомогою циклів

Цикли використовуються для того, щоб виконати будь-яку дію кілька разів. Наприклад, потрібно надрукувати поздоровлення для декількох переможців. Якщо ви заздалегідь знаєте скільки разів повторити дію - використовуйте цикл `for`:



```
int totalPlayers = players.length;
int counter;

for (counter=0; counter<totalPlayers; counter++){
   String thePlayer = players[counter];
   System.out.println("Вітаємо, " + thePlayer + "!");

}
```

JVM виконує кожен рядок між фігурними дужками, і потім повертається до першого рядка циклу для того, щоб збільшити значення counter (лічильника) і перевірити умову завершення циклу. Цей код означає наступне:

Надрукувати значення елементу масиву, чий номер індексу збігається із значенням лічильника. Почати з елементу з номером 0 (counter=0), і збільшувати значення counter на одиницю (`counter++`). Продовжувати до тих пір, поки counter менше totalPlayers (`counter<totalPlayers`).

Існує ще одне ключове слово для створення циклів - while. У таких циклах не потрібно точно знати, скільки разів повторюватиметься дія, але необхідно задати умову закінчення циклу. Давайте поглянемо, як можна привітати учасників гри за допомогою циклу while, він закінчиться, коли counter (лічильник) стане рівним totalPlayers:

```
int totalPlayers = players.length;
int counter=0;

while (counter<totalPlayers){
   String thePlayer = players[counter];
   System.out.println("Вітаємо, " + thePlayer + "!");
   counter++;

}
```

У розділі 9 розказано, як зберегти дані у файл на диску і як зчитати їх назад у пам'ять програми. Якщо ви читаєте результати гри з файлу, що знаходиться на диску, то заздалегідь невідомо, як багато записів було збережено у файл. Швидше за все, доведеться зчитувати такі записи за допомогою циклу `while`.

Також можна використовувати ще два оператори в циклах: `break` і `continue`.

Ключове слово `break` використовується, щоб вистрибнути з циклу, коли деяка умова стає достеменною (true). Скажімо, ми не хочемо друкувати більше трьох вітань, незалежно від того, як багато гравців брало участь.

У наступному прикладі, після виведення елементів масиву 0, 1 і 2, виконання програми вийде з циклу через інструкцію `break` і продовжиться з рядка після закриваючої фігурної дужки. Зверніть увагу на подвійний знак рівності в операторі if. Значення змінної counter порівнюється з числом 3. Одинарний знак рівності означав би, що значення 3 привласнюється змінній counter. Якщо переплутати == з = у операторі if, то програма працюватиме, але неправильно і таку помилку інколи дуже важко знайти:

```
int counter =0;

while (counter< totalPlayers){

  if (counter == 3){
     break; //Вистрибуємо з циклу
  }

  String thePlayer = players[counter];
  System.out.println("Вітаємо, "+thePlayer+ "!");
  counter++;

}
```

Оператор continue дозволяє пропустити виконання рядків, що стоять після нього і повернутися на початок циклу. Уявимо, що ми хочемо привітати всіх, окрім Девіда - continue поверне виконання програми на початок циклу.

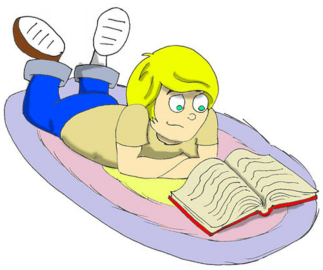
```
while (counter<totalPlayers){

     counter++;
     String thePlayer = players[counter];

     if (thePlayer.equals("David"){
          continue;
     }

     System.out.println("Вітаємо, "+ thePlayer+ "!");

}
```

Є ще один тип оператора `while`, який починається із слова `do`, наприклад:

```
do {
      // Тут ваш код, який підраховує
} while (counter<totalPlayers);
```

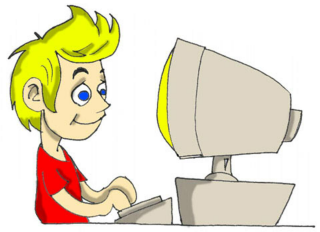
У таких циклах умова перевіряється після виконання коду, що стоїть між фігурними дужками. Таким чином, цей код виконається *хоча б один раз*. Цикли, в яких `while` стоїть спочатку, можуть не виконатися жодного разу, якщо умова помилкова (`false`).

Матеріали для додаткового читання



Офіційний підручник по Java:
http://download.oracle.com/javase/tutorial/java/index.html

Практичні вправи



1. Створіть новий клас і назвіть його TemperatureConverter. Додайте в нього метод для перетворення температур, з такою сигнатурою:

```
public String convertTemp
          (int temperature, char convertTo)
```

Якщо аргумент convertTo рівний F, то температура має бути перетворена у Фаренгейти, якщо C, то в Цельсії. Коли ви викликатимете цей метод, помістіть значення аргументу типу char в одинарні лапки.

2. Оголосіть метод convertGrades() класу ReportCard як статичний і видаліть рядок ініціалізації класу з методу main().

Практичні вправи для розумників і розумниць

Ви помітили, що в прикладі з ключовим словом `continue` ми перемістили рядок з counter++; в початок циклу?

Що б сталося, якби ми залишили цю строчку в кінці циклу, як це було в прикладі з `break`?

Розділ 5. Робимо Графічний Калькулятор

Java містить широкий набір класів, які дозволяють створювати графічні додатки. Існує дві основні групи класів для створення вікон в Java.

AWT і Swing

У першій версії мови Java для роботи з графікою була тільки бібліотека - AWT. Ця бібліотека - простий набір класів, таких, як `Button` (кнопка), `TextField` (текстове поле), `Label` (текстова мітка або іконка) та інші. Незабаром була створена більш досконала бібліотека, яку назвали Swing. Вона так само включає в себе кнопки, текстові поля та інші елементи управління графічними додатками. Назви компонентів цієї бібліотеки починається з букви J. Наприклад, `JButton`, `JTextField` і так далі.

Все в Swing трішки краще, швидше і зручніше, але в деяких випадках наші програми можуть бути запущені на комп'ютерах зі старою версією JVM, яка може не підтримувати класів Swing.

Пакети і ключове слово import

Java поставляється з великою кількістю корисних класів, які організовані в пакети (*packages*). Деякі пакети містять класи для малювання графіки, інші - класи для роботи з інтернетом і так далі. Наприклад, клас `String` знаходиться в пакеті з назвою java.lang і повне ім'я цього класу `java.lang.String`.

Компілятор Java знає, де знайти класи, що знаходяться в `java.lang`, тому я не вказував явно повне ім'я `String` в попередніх прикладах коду, але існує багато інших пакетів з корисними класами і ваше завдання повідомити

компілятору, в якому пакеті містяться класи, що використовуються в програмі. Наприклад, більшість класів бібліотеки Swing знаходяться в наступних двох пакетах:

```
javax.swing
javax.swing.event
```

Було б дуже незручно кожного разу, коли використовується клас, писати його повне ім'я. Щоб уникнути цього, ви можете написати ключове слово `import` всього один раз перед оголошенням класу, як показано в прикладі:

```
import javax.swing.JFrame;
import javax.swing.JButton;

class Calculator{

     JButton myButton = new JButton();
     JFrame myFrame = new JFrame();

}
```

Ключове слово `import` дозволяє використовувати короткі імена класів, такі як `JFrame` або `JButton` і повідомляє компілятору, де шукати ці класи.

Якщо потрібно використовувати декілька класів з одного пакету, немає необхідності перераховувати кожен з них у рядку з `import`, можна просто використовувати символ *. У наступному прикладі за допомогою зірочки, всі класи з `javax.swing` стають знаходжуваними:

```
import javax.swing.*;
```

Проте, краще використовувати окремі оператори import для кожного класу. Це дозволяє швидше бачити, який клас імпортується з якого пакету.

Тема пакетів буде висвітлена більш докладно в розділі 10.

Основні елементи Swing

Ось деякі основні об'єкти, з яких складаються Swing-додатки:

- Вікно або кадр (*frame*), який може бути створений за допомогою класу `JFrame`.
- Невидима панель (*panel*) або, як ще її називають, *pane* (віконне скло) містить всі кнопки, текстові поля, мітки та інші компоненти. Панелі створюються за допомогою класу `JPanel`.
- Віконні елементи управління, такі як кнопки `JButton`, текстові поля `JTextfield`, списки `JList`, і таке інше.
- Схеми розміщення (*layout managers*) компоненти, які допомагають організувати всі ці кнопки і поля на панелі.

Наприклад, можна створити екземпляр класу JPanel і призначити для нього схему розміщення. Потім створити різні графічні компоненти і додати їх на панель. Після цього додати панель на фрейм, задати його розмір і зробити його видимим.

Проте відображення кадру це тільки половина роботи. Потрібно ще додати обробку різних подій, наприклад натискань на кнопки.

У цьому розділі я розповім, як створювати вікна з компонентами, а в наступному - обробляти події (events), які можуть статися з компонентами вікна.

Наша основна мета у цьому розділі - написати калькулятор, який дозволяє скласти два числа і побачити результат. Створіть новий проект в Eclipse, назвіть його My Calculator і додайте в нього новий клас SimpleCalculator з наступним кодом:

```
import javax.swing.*;
import java.awt.FlowLayout;

public class SimpleCalculator {

  public static void main(String[] args) {

  // Створюємо панель

    JPanel windowContent= new JPanel();

  // Задаємо менеджер відображення для цієї панелі

    FlowLayout fl = new FlowLayout();
    windowContent.setLayout(fl);

  // Створюємо компоненти в пам'яті

    JLabel label1 = new JLabel("Number 1:");
    JTextField field1 = new JTextField(10);
    JLabel label2 = new JLabel("Number 2:");
    JTextField field2 = new JTextField(10);
    JLabel label3 = new JLabel("Sum:");
    JTextField result = new JTextField(10);
    JButton go = new JButton("Add");
```

```
    // Додаємо компоненти на панель

      windowContent.add(label1);
      windowContent.add(field1);
      windowContent.add(label2);
      windowContent.add(field2);
      windowContent.add(label3);
      windowContent.add(result);
      windowContent.add(go);

    // Створюємо фрейм і задаємо для нього панель

      JFrame frame = new JFrame("My First Calculator");
      frame.setContentPane(windowContent);

    // задаємо розмір і робимо фрейм видимим

      frame.setSize(400,100);
      frame.setVisible(true);

    }
}
```

Скомпілюйте і запустіть цю програму. Повинно з'явитися вікно такого вигляду:

Це, можливо, не найкрасивіший у світі калькулятор, але в даному прикладі видно, як додавати компоненти й відображати вікно. У наступній секції ми спробуємо зробити привабливіший інтерфейс за допомогою схем розміщення.

Схеми Розміщення

У деяких старомодних мовах програмування необхідно було вказувати координати і розміри кожного компонента вікна. Це працювало добре, якщо було відомо роздільна здатність екрану кожного користувача. До речі, людей, які користуються програмами, називають користувачами (users). У Java є схеми розміщення (Layout Managers), які дозволяють розмістити компоненти на екрані, не знаючи точних позицій компонентів. Схеми гарантують, що та частина інтерфейсу, за яку вони відповідають, буде виглядати правильно незалежно від розмірів вікна і роздільної здатності екрану.

Swing надає такі схеми:

- `FlowLayout`
- `GridLayout`
- `BoxLayout`
- `BorderLayout`
- `CardLayout`
- `GridBagLayout`

Щоб використовувати будь-який з цих менеджерів, необхідно створити його екземпляр і потім призначити цей об'єкт якомусь *контейнеру (container)*, наприклад панелі, як це було в прикладі з SimpleCalculator.

FlowLayout - порядкове розташування

За цією схемою компоненти розміщуються у вікні (або іншому контейнері) рядок за рядком. Наприклад, текстові мітки, іконки, текстові поля і кнопки будуть додаватися в перший умовний рядок, поки в ньому є місце. Коли перший рядок заповниться, компоненти, що залишилися будуть додаватися до наступного рядка і так далі. Якщо користувач змінить розмір вікна, картина може змінитися. Просто потягніть за кут калькулятора, щоб поміняти його розмір. Подивіться як java. awt. FlowLayout перевпорядковує елементи

вікна під час зміни його розмірів.

У наступному прикладі коду, ключове слово this представляє екземпляр класу SimpleCalculator.

```
FlowLayout fl = new FlowLayout ();
this.setLayoutManager (fl);
```

Згоден, FlowLayout не кращим чином підходить для нашого калькулятора. Давайте тепер спробуємо щось інше.

GridLayout - табличне розташування

Клас `java.awt.GridLayout` дозволяє організувати компоненти, як рядки і стовпці в таблиці. Компоненти будуть додаватися в комірки умовної таблиці. Якщо розмір вікна буде збільшений, комірки стануть більше, але положення компонентів відносно один одного залишиться колишнім. У нашому калькуляторі сім компонентів - три текстові мітки, три текстових поля і кнопка. Ми можемо розмістити їх в таблиці з чотирма рядками і двома колонками (одна комірка залишиться порожньою):

```
GridLayout gr = new GridLayout(4,2);
```

Також можна задати відстань між комірками по вертикалі і горизонталі, наприклад в п'ять пікселів:

```
GridLayout gr = new GridLayout(4,2,5,5);
```

Після невеликих змін у нашому калькуляторі (вони підсвічені нижче),

він стане виглядати набагато симпатичніше.

А тепер створіть і скомпілюйте новий клас SimpleCalculatorGrid в проекті *My Calculator*.

```
import javax.swing.*;
import java.awt.GridLayout;

public class SimpleCalculatorGrid {

  public static void main (String[] args) {

     // Створюємо панель

       JPanel windowContent = new JPanel();

     // Задаємо менеджер розташування для цієї панелі

       GridLayout gl = new GridLayout(4,2);
       windowContent.setLayout(gl);

     // Створюємо компоненти в пам'яті

       JLabel label1 = new JLabel("Number 1:");
       JTextField field1 = new JTextField(10);
       JLabel label2 = new JLabel("Number 2:");
       JTextField field2 = new JTextField(10);
       JLabel label3 = new JLabel("Sum:");
       JTextField result = new JTextField(10);
       JButton go = new JButton("Add");

     // Додаємо компоненти в панель

       windowContent.add(label1);
       windowContent.add(field1);
       windowContent.add(label2);
       windowContent.add(field2);
```

```
        windowContent.add(label3);
        windowContent.add(result);
        windowContent.add(go);

      // Створюємо фрейм і задаємо панель для нього

        JFrame frame = new JFrame("My First Calculator");
        frame.setContentPane(windowContent);

      // Задаємо розмір і відображаємо вікно

        frame.setSize(400,100);
        frame.setVisible(true);
    }
}
```

Після запуску програми SimpleCalculatorGrid, ви побачите таке вікно:

Спробуйте поміняти розміри цього вікна - розміри елементів управління будуть змінюватися разом з ним, але їх положення відносно один одного не зміниться:

Ще одна важлива річ, яку варто запам'ятати про табличний компонувальник - на ньому всі комірки мають однакову довжину і ширину.

BorderLayout - розміщення по областях

Клас `java.awt.BorderLayout` поділяє вікно на Південну, Західну, Північну, Східну і Центральну області. Північна область знаходиться нагорі вікна, Південна - знизу, Західна - ліворуч, а Східна - праворуч. Наприклад, в калькуляторі, який буде продемонстрований на наступній сторінці, текстові поля, які відображають числа, знаходяться в Північній області.

Створити `BorderLayout` і помістити в нього текстові поля можна таким чином:

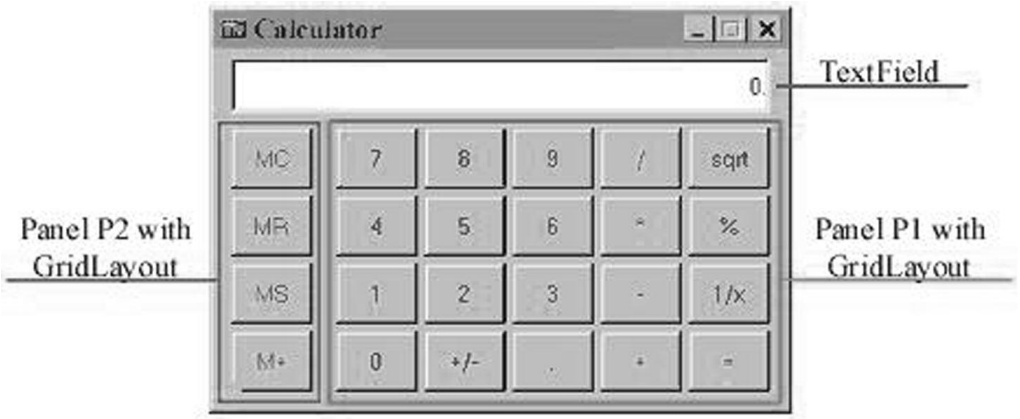
```
BorderLayout bl = new BorderLayout();
this.setLayoutManager(bl);

JTextField txtDisplay = new JTextField(20);
this.add("North", txtDisplay);
```

Зовсім не обов'язково поміщати елементи управління в усі п'ять областей. Якщо необхідні тільки Північна, Центральна і Південна області, то Центральна стане ширшою, тому що Західна і Східна пустують.

Я буду використовувати `BorderLayout` трохи пізніше, в наступній версії нашого калькулятора, який називатиметься Calculator.java.

Комбінування схем розміщення

Як ви думаєте, чи можна за допомогою `GridLayout` створити калькулятор, який буде виглядати так само, як стандартний калькулятор в Microsoft Windows?

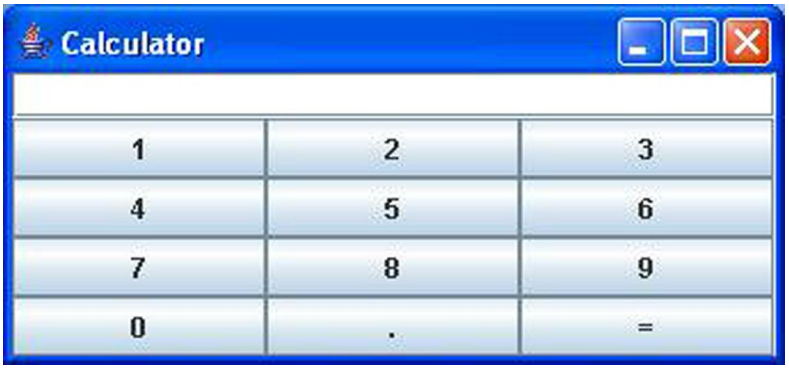


На жаль, ні, так як комірки цього калькулятора мають різні розміри - текстове поле більше кнопок. Але вміст вікна можна представити за допомогою декількох панелей, у яких схеми різні.

Спробуємо використати комбінацію декількох схем у новому калькуляторі. Для цього необхідно виконати наступні кроки:

- Призначити BorderLayout панелі фрейму, яка буде основною, і в якій міститимуться інші панелі.
- Додати JTextField в північну частину, для того щоб відображати введені числа.
- Створити панель p1 з GridLayout, додати на неї 20 кнопок і потім помістити цю панель в центральну область основної панелі.
- Створити панель p2 з GridLayout, додати на неї чотири кнопки і потім помістити панель p2 в західну область основної панелі.

Давайте почнемо з більш простої версії калькулятора, яка буде виглядати ось так:



Створіть новий клас Calculator і запустіть програму. Щоб зрозуміти, як вона працює, прочитайте коментарі у прикладі коду, продемонстрованому нижче.

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.BorderLayout;

public class Calculator {

      // Оголошення всіх компонентів калькулятора.
        JPanel windowContent;
        JTextField displayField;
        JButton button0;
        JButton button1;
        JButton button2;
        JButton button3;
        JButton button4;
        JButton button5;
        JButton button6;
        JButton button7;
        JButton button8;
        JButton button9;
        JButton buttonPoint;
        JButton buttonEqual;
        JPanel p1;
```

```
// У конструкторі створюються всі компоненти
// І додаються на фрейм за допомогою комбінації
// Borderlayout і Gridlayout
Calculator() {

  windowContent = new JPanel();

// Задаємо схему для цієї панелі
  BorderLayout bl = new BorderLayout();
  windowContent.setLayout(bl);

// Створюємо і відображаємо поле
// Додаємо його в Північну область вікна
  displayField = new JTextField (30);
  windowContent.add("North", displayField);

// Створюємо кнопки, використовуючи конструктор
// Класу JButton, який приймає текст
// Кнопки як параметр
  button0 = new JButton("0");
  button1 = new JButton("1");
  button2 = new JButton("2");
  button3 = new JButton("3");
  button4 = new JButton("4");
  button5 = new JButton("5");
  button6 = new JButton("6");
  button7 = new JButton("7");
  button8 = new JButton("8");
  button9 = new JButton("9");
  buttonPoint = new JButton(".");
  buttonEqual = new JButton("=");

// Створюємо панель з GridLayout
// Яка містить 12 кнопок - 10 кнопок з числами
// І кнопки з точкою і знаком рівності
```

```
        p1 = new JPanel();
        GridLayout gl = new GridLayout(4,3);
        p1.setLayout(gl);

      // Додаємо кнопки на панель p1
        p1.add (button1);
        p1.add (button2);
        p1.add (button3);
        p1.add (button4);
        p1.add (button5);
        p1.add (button6);
        p1.add (button7);
        p1.add (button8);
        p1.add (button9);
        p1.add (button0);
        p1.add (buttonPoint);
        p1.add (buttonEqual);

      // Розміщуємо панель p1 в центральну область вікна
        windowContent.add ("Center", p1);

      // Створюємо фрейм і задаємо його основну панель
        JFrame frame = new JFrame("Calculator");
        frame.setContentPane(windowContent);

      // Робимо розмір вікна достатнім
      // Для того, щоб вмістити всі компоненти
        frame.pack();

      // Нарешті, відображаємо вікно
        frame.setVisible(true);
      }

      public static void main (String[] args) {

        Calculator calc = new Calculator ();
      }
}
```

BoxLayout - розташування по горизонталі або вертикалі

Клас `java.swing.BoxLayout` розміщує кілька компонентів вікна горизонтально (по осі абсцис) або вертикально (по осі ординат). На відміну від менеджера `FlowLayout`, коли вікно з `BoxLayout` змінює свій розмір, його елементи керування не зміщуються зі своїх позицій. `BoxLayout` дозволяє елементам вікна мати різні розміри (чого не дозволяє `GridLayout`).

Наступні два рядки коду задають box layout з вертикальним вирівнюванням на `JPanel`.

```
JPanel p1 = new JPanel();
setLayout(new BoxLayout(p1, BoxLayout.Y_AXIS));
```

Щоб зробити цей код коротшим, я не створював змінну для зберігання посилання на об'єкт `BoxLayout`, замість цього я створив екземпляр цього класу і відразу ж передав його, як аргумент на метод `setLayout()`.

GridBag Layout - більш гнучке табличне розташування

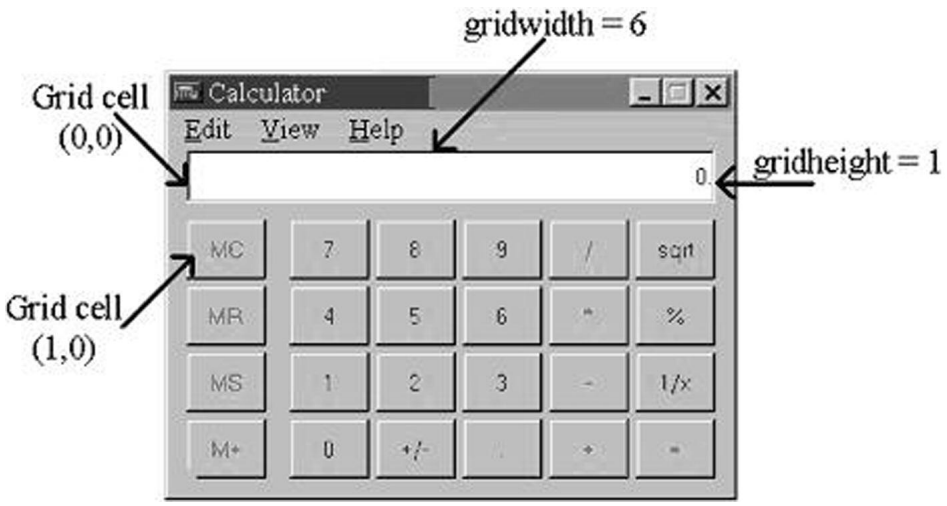
А зараз я покажу ще один спосіб створення вікна калькулятора. Тут буде використовуватися `java.awt.GridBagLayout` замість комбінації схем і панелей.

У нашому калькуляторі є рядки і стовпці, але в `GridLayout` вони повинні мати однакові розміри. Це не підходить, тому що у нас є поле для введення, ширина якого дорівнює ширині трьох кнопок з числами.

`GridBagLayout` - більш розширена схема розміщення. Вона дозволяє задавати розмір комірки, рівним декільком клітинам таблиці. `GridBagLayout` має допоміжний клас, який називається `GridBagConstraints` (обмеження на клітини таблиці). Ці обмеження не що інше, як атрибути комірок,

які необхідно задавати для кожної комірки таблиці окремо. Всі обмеження мають бути задані до того, як у комірку поміщаються компоненти. Наприклад, один з атрибутів `GridBagConstraints` називається `gridWidth`. Він дозволяє задати ширину якоїсь однієї клітинки, рівній ширині кількох інших.

Під час роботи з `GridBagLayout` необхідно спочатку створити екземпляр класу `GridBagConstraints`, і потім задати значення для його властивостей. Після того як це зроблено, можна додавати об'єкт в клітинку контейнера.



Наступний приклад коду, усипаний коментарями, які допоможуть зрозуміти, як використовувати `GridBagLayout`.

```
//Задаємо GridBagLayout для панелі вікна
  GridBagLayout gb = new GridBagLayout();
  this.setLayout(gb);

//Створюємо екземпляр класу GridBagConstraints
//Ці рядки коду потрібно повторити для кожної компоненти
//Яка додається в клітинку
  GridBagConstraints constr = new GridBagConstraints();
```

```
//задаємо обмеження для рядка введення калькулятора
//Координата x в таблиці
  constr.x = 0;

//Координата y в таблиці
  constr.y = 0;

//Ця комірка має таку ж висоту, як стандартні комірки
  constr.gridheight = 1;

//Ця комірка має ширину рівну ширині 6 стандартних комірок
  constr.gridwidth = 6;

//Заповнюємо весь простір комірки
  constr.fill = constr.BOTH;

//Пропорція по горизонталі, яку займатиме компонент
  constr.weightx = 1.0;

//Пропорція по вертикалі, яку займатиме компонент
  constr.weighty = 1.0;

//Позиція компонента всередині комірки
  constr.anchor = constr.CENTER;
  displayField = new JTextField();

//Встановлюємо обмеження для поля введення
  gb.setConstraints(displayField, constr);

//Додаємо поле введення у вікно
  windowContent.add(displayField);
```

CardLayout - колода карт

Уявіть колоду карт, в якій карти лежать сорочкою вниз так, що ви можете бачити тільки верхню карту. Схема `java.awt.CardLayout` може

бути використана, якщо необхідно створити компонент, який виглядає, як папка з вкладками.

При натисканні на вкладку вміст екрану змінюється. Насправді, всі панелі, необхідні для цього вікна, вже попередньо завантажені і лежать одна на одній. Коли користувач клацає по вкладці, програма просто "переносить цю вкладку" наверх і робить вміст інших вкладок невидимим.

Швидше за все, ви не будете використовувати цю схему, тому що бібліотека Swing включає готовий компонент для вікон з вкладками. Він називається `JTabbedPane`.

Чи можна створювати вікна, не використовуючи схеми?

Звичайно, можна! Ніщо не заважає явно задавати координати кожного компонента при додаванні на вікно. Для цього клас повинен чітко вказати, що він не використовує схеми розміщення. У Java є спеціальне слово null, яке означає "значення не задане". Ми будемо використовувати це ключове слово досить часто в майбутньому, і в наступному прикладі воно означає, що ніяка схема не використовується:

```
windowContent.setLayout(null);
```

Але, якщо відмовитися від схем, то необхідно призначити координати лівого верхнього кута, ширину і висоту кожного віконного компонента. У наступному прикладі показано, як можна встановити ширину кнопки в 40 пікселів, висоту в 20, і розмістити її на 100 пікселів вправо і 200 пікселів вниз від верхнього лівого кута вікна:

```
JButton myButton = new Button("New Game");
myButton.setBounds(100,200,40,20);
```

Компоненти вікна

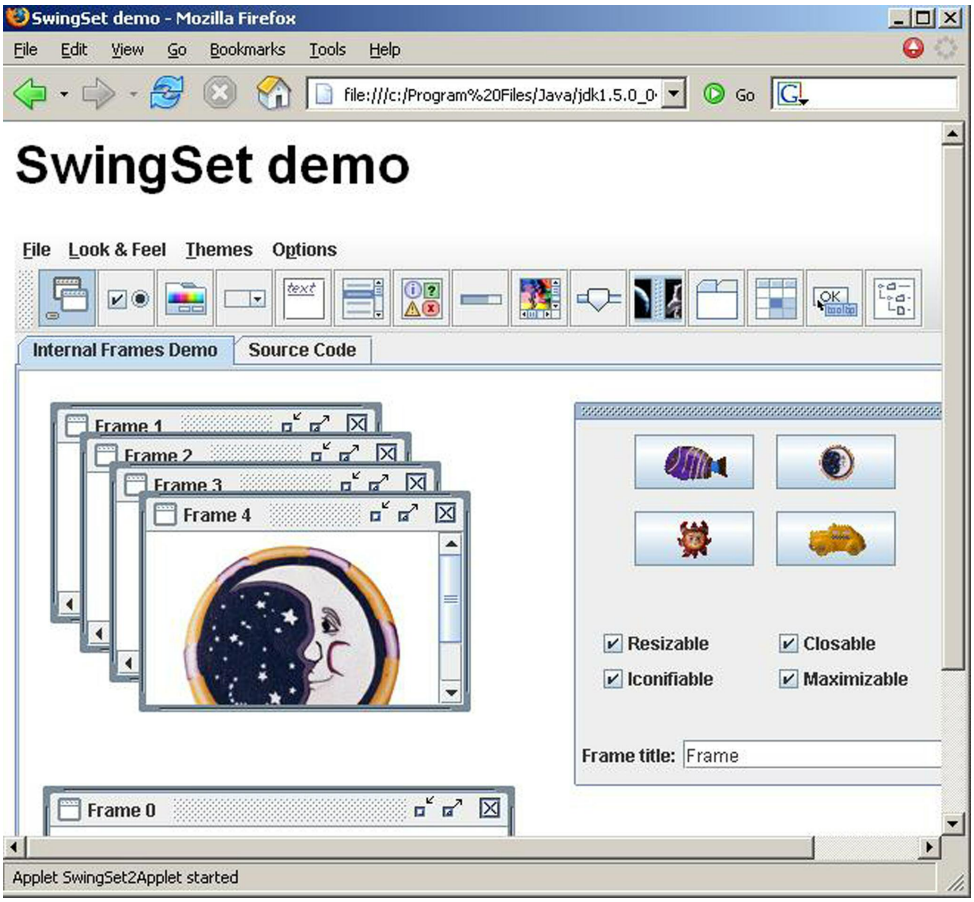
Я не буду описувати всі компоненти Swing в цій книзі, але ви можете знайти посилання на онлайн підручник з Swing, в розділі *Матеріали для додаткового читання*. У цьому керівництві є докладні описи всіх компонентів Swing. Наш калькулятор використовує тільки `JButton`, `JLabel` і `JTextField`. Ось список інших доступних компонентів:

- `JButton`
- `JLabel`
- `JCheckBox`
- `JRadioButton`
- `JToggleButton`
- `JScrollPane`
- `JSpinner`
- `JTextField`
- `JTextArea`
- `JPasswordField`
- `JFormattedTextField`
- `JEditorPane`
- `JScrollBar`
- `JSlider`
- `JProgressBar`
- `JComboBox`
- `JList`
- `JTabbedPane`
- `JTable`
- `JToolTip`
- `JTree`
- `JViewPort`
- `ImageIcon`

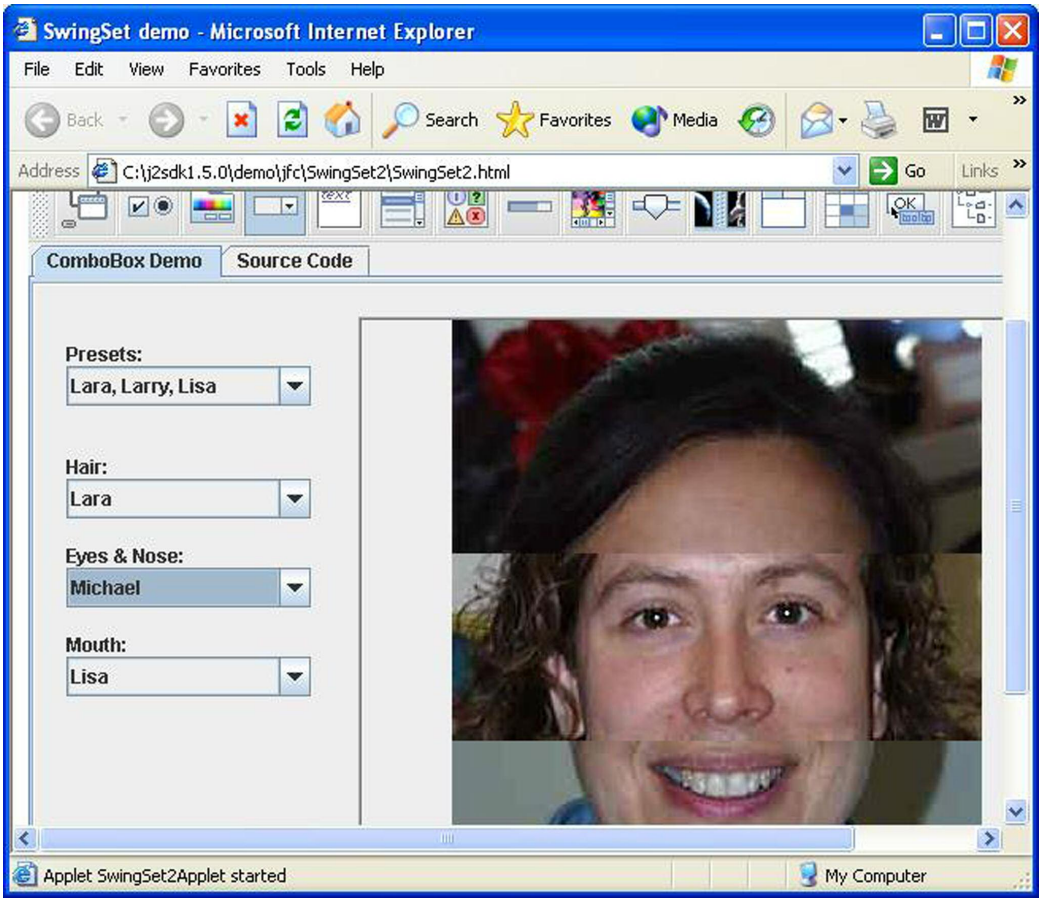
Ви також можете створювати меню (`JMenu` і `JPopupMenu`), спливаючі вікна, фрейми усередині інших фреймів (`JInternalFrame`) і використовувати стандартні вікна (`JFileChooser`,

`JColorChooser` і `JOptionPane`).

Java поставляється з відмінним демонстраційним додатком, який показує усі доступні компоненти Swing в дії. Він знаходиться в папці, де ви встановили Java SDK, наприклад *C:\Program Files\Java\jdk1.8.0_05\demo\jfc\SwingSet2*. Просто відкрийте файл SwingSet2.html, і ви побачите екран, схожий на ось цей:



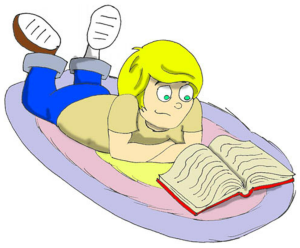
Натисніть на будь-яке зображення на панелі інструментів, щоб побачити, як та чи інша компонента Swing працює. Ви також можете знайти приклади коду, який використовувався для створення кожного вікна, вибравши вкладку Source Code. Наприклад, якщо натиснете на іконку combobox (список, що випадає), то побачите вікно, яке виглядає наступним чином:



У Swing є багато різних компонентів, щоб зробити ваші вікна симпатичними. У цьому розділі ми створювали Swing компоненти, просто вводячи код, без використання спеціальних інструментів. Але є спеціальні утиліти, які дозволяють вибрати компонент на панелі інструментів і перетягнути його в створюване вікно. Ці програми автоматично генерують відповідний Java код для компонентів Swing. Один з таких графічних дизайнерів, який дозволяє легко створювати Swing додатки, називається Matisse. Інша - Gigloo GUI Builder.

У наступному розділі буде розказано, як вікно може реагувати на дії користувача.

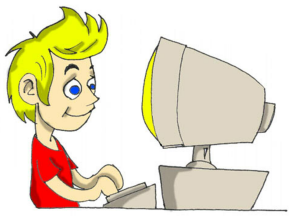
Матеріали для додаткового читання



I. Підручник з Swing:
http://download.oracle.com/javase/tutorial/uiswing/index.html

2. Клас `JFormattedTextField`:
http://download.oracle.com/javase/7/docs/api/javax/swing/JFormattedTextField.html

Практичні вправи



I. Модифікуйте клас Calculator.java додавши в нього кнопки, -, /, та *. Помістіть ці кнопки на панель p2, і покладіть цю панель на Східну область основній панелі.

2. Прочитайте про клас `JFormattedTextField` в інтернеті і змініть вихідний код калькулятора так, щоб цей клас використовувався замість `JTextField`. Метою є створення поля введення з вирівнюванням по правому краю, як у справжніх калькуляторах.

Практичні вправи для розумників і розумниць

Модифікуйте клас Calculator.java так, щоб всі кнопки з цифрами зберігалися в масиві з десятьма елементами, який повинен бути оголошений ось так:

```
Buttons[] numButtons = new Buttons[10];
```

Замініть 10 рядків коду, які починаються з `button0 = new JButton("0");` циклом, який створює кнопки і додає їх в масив. Підказка: зазирніть у вихідний код гри “Хрестики-Нулики”, розділ 7.

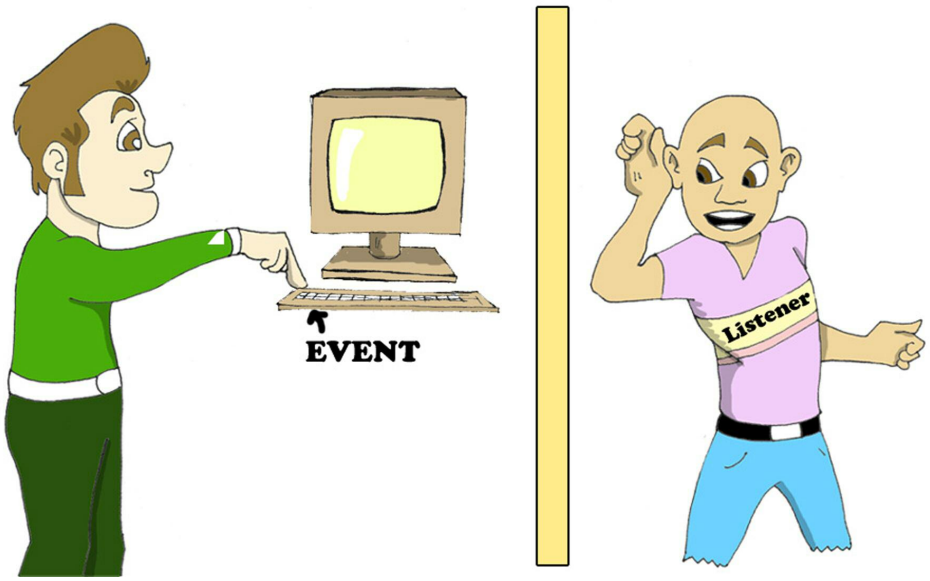
Розділ 6. Події вікна

Під час роботи програми можуть відбуватися різні події: користувач натисне на кнопку, веб-браузер вирішить перемалювати вікно, і так далі. Я впевнений, що ви намагалися натискати на кнопки вашого калькулятора з розділу 5, але ці кнопки ще не були готові реагувати на ваші дії.

Кожен компонент вікна може обробляти різні події, або, як ми говоримо, слухати ці події. Ви можете зареєструвати класи Java, які називають *слухачами (listeners)*, прив'язавши їх до компонентів вікна. Ви повинні зробити так, щоб компоненти слухали тільки ті події, які їм потрібні.

Наприклад, коли людина переміщає покажчик мишки над кнопкою калькулятора, неважливо де саме був цей покажчик, коли користувач натиснув на кнопку, поки курсор знаходився над поверхнею кнопки. Тому вам не потрібно реєструвати слухач MouseMotionListener для кнопки. З іншого боку, цей слухач корисний для всіляких програм для малювання.

Для кнопок калькулятора потрібно зареєструвати клас ActionListener, який вміє обробляти натискання на кнопки. Всі ці слухачі - це спеціальні конструкції Java, які називаються *інтерфейсами*.



Інтерфейси

Більшість класів визначають методи, які реагують на різні дії, наприклад, відповім на натискання кнопки, відповім на рух миші, і так далі. Набір таких дій називається поведінкою класу.

Інтерфейси - це спеціальні конструкції, які тільки оголошують набір певних дій без коду, який описує, що саме треба робити в оголошених методах, наприклад:

```
interface MouseMotionListener {

     void mouseDragged(MouseEvent e);
     void mouseMoved(MouseEvent e);

}
```

Як бачите, методи `mouseDragged()` і `mouseMoved()` не містять ніякого тексту програм - ці методи просто оголошені в інтерфейсі, званому MouseMotionListener. А от якщо ваш клас повинен реагувати на рух покажчика миші або на перетягування мишею, то тоді він повинен *реалізувати* цей інтерфейс.

Слово `implements` означає, що цей клас абсолютно точно буде містити методи, які могли бути оголошені в інтерфейсі, наприклад:

```
import java.awt.event.MouseMotionListener;

class MyDrawingPad implements MouseMotionListener {

  //Тут може йти текст програми, яка
  //виконує функції графічного редактора
     mouseDragged (MouseEvent e) {

       //Тут буде текст програми, коли
```

```
        //миша щось перетягує

      }

      mouseMoved (MouseEvent e) {

        //Сюди йде текст програми, коли
        //Миша просто буде тут рухатися

      }
}
```

Мабуть, вам цікаво, навіщо турбуватися про створення інтерфейсів без тексту програми? Причина в тому, що інтерфейс, зроблений одного разу, може використовуватися в багатьох класах. Наприклад, коли інші класи (або сама віртуальна машина JVM) бачать, що клас MyDrawingPad реалізує інтерфейс `MouseMotionListener`, вони знають, що в цьому класі точно є методи `mouseDragged()` і `mouseMoved()`.

Кожен раз, коли користувач рухає мишкою, JVM викликає метод `mouseMoved()` і виконує текст програми, який ви там написали. Уявіть, що якщо Іван вирішить назвати цей метод `mouseMoved()`, Маша назве його movedMouse(), а Петро віддасть перевагу mouseCrawling()? Тоді JVM заплутається і не буде знати, який же метод вашого класу викликати, щоб повідомити про рух миші.

Клас Java може реалізовувати багато інтерфейсів, наприклад, він може реагувати на рухи миші і на натискання кнопки:

```
class myDrawingProgram implements MouseMotionListener,
                                  ActionListener {

      //Тут ви повинні написати текст програми для
      //кожного методу оголошеного в обох інтерфейсах

}
```

Після того, як ви звикнете до інтерфейсів, які надає вам Java, ви зможете створювати свої власні інтерфейси, але це теми складніші і, поки що, ми не будемо цього робити.

Слухач на ім'я ActionListener

Давайте повернемося до нашого калькулятору. Якщо ви зробили завдання до попереднього розділу, візуальна частина програми готова. Тепер ми створимо ще один клас-слухач, який буде щось робити, коли користувач буде натискати на одну з кнопок. Взагалі-то, ми могли б додати текст програми, що обробляє події натискання на кнопку, відразу в клас Calculator.java, але краще не змішувати в одному класі візуальну та обробну частини.

Назвемо другий клас CalculatorEngine, і скажемо, що він повинен реалізувати інтерфейс `java.awt.ActionListener` в якому оголошено тільки один метод - `actionPerformed (ActionEvent)`. JVM викликає цей метод у класі, який реалізує цей інтерфейс кожен раз, коли хтось натискає на кнопку.

Подивимося на цей простий клас:

```
import java.awt.event.ActionListener;

public class CalculatorEngine implements ActionListener{

}
```

Якщо ви спробуєте його скомпілювати (або просто зберегти його в Eclipse), то виникне повідомлення про помилку, що, мовляв, клас повинен реалізувати метод actionPerformed (ActionEvent e). Давайте виправимо цю помилку:

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class CalculatorEngine implements ActionListener{

  public void actionPerformed (ActionEvent e) {
     //Якщо цей метод можна залишити порожнім, нічого не
     //станеться, коли JVM викличе його
  }
}
```

Наступна версія цього класу буде відкривати вікно повідомлення (*a message box*) з методу `actionPerformed()`. За допомогою класу `JOptionPane` і його методу `showConfirmDialog()` можна показувати користувачеві будь-які повідомлення. Наприклад, клас CalculatorEngine може видати таке:

Є різні версії методу showConfirmDialog (), ми будемо використовувати версію з чотирма параметрами. У тексті програми нижче null означає, що вікно повідомлення не має батьківського вікна, другий аргумент - це заголовок вікна повідомлення, потім йде саме повідомлення, а четвертий аргумент дозволяє вибрати, які кнопки будуть відображатися у вікні повідомлення (PLAIN_MESSAGE в наступному прикладі означає, що буде відображатися тільки одна кнопка - "OK").

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;

public class CalculatorEngine implements ActionListener{

  public void actionPerformed(ActionEvent e){

     JOptionPane.showConfirmDialog(null,
               "Something happened...", "Just a test",
                         JOptionPane.PLAIN_MESSAGE);

  }
}
```

Тепер я покажу, як скомпілювати і запустити наступну версію нашого калькулятора, яка показує вікно повідомлення "Something happened".

Реєстрація компонентів з ActionListener

Хто і коли буде викликати код, написаний в методі `actionPerformed()`? Сама JVM викличе цей метод, *якщо ви зареєструєте* клас CalculatorEngine в кнопках калькулятора (або зв'яжете їх з класом)! Просто додайте ці два рядки в кінець конструктора класу Calculator, щоб зареєструвати наш слухач для кнопки "Нуль":

```
CalculatorEngine calcEngine = new CalculatorEngine();
button0.addActionListener(calcEngine);
```

Тепер кожен раз, коли користувач натисне кнопку button0, JVM викличе метод `actionPerformed()` у об'єкта CalculatorEngine. Скомпілюйте і запустіть клас Calculator і натисніть на кнопку "Нуль" - і на екрані з'явиться вікно повідомлення "*Something happened*"! Інші кнопки поки не реагують, тому що в них не зареєстрований наш слухач. Додайте такі ж рядки, щоб

оживити й інші кнопки:

```
button1.addActionListener(calcEngine);
button2.addActionListener(calcEngine);
button3.addActionListener(calcEngine);
button4.addActionListener(calcEngine);
```

і т д.

Через кого подія?

Наступний крок - зробити нашого слухача трохи розумнішим - він буде показувати різні повідомлення, залежно від того, яка кнопка була натиснута. Коли відбудеться подія, JVM викличе метод вашого класу-слухача actionPerformed (ActionEvent), і передасть йому необхідну інформацію про подію в аргументі ActionEvent. Ви можете отримати цю інформацію, викликаючи відповідні методи цього об'єкту-аргументу.

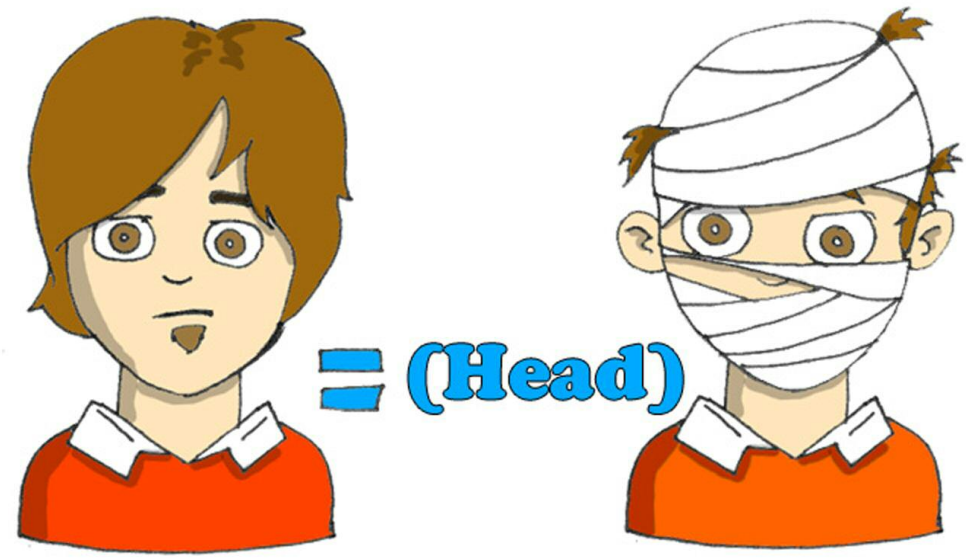
Приведення типів - casting

У наступному прикладі я покажу, як визначити, яка кнопка була натиснута, викликаючи метод `getSource()` класу `ActionEvent`. Змінна evt - це посилання на об'єкт-подію, який живе десь в пам'яті комп'ютера. Але, як написано в документації Java, метод `getSource()` повертає джерело події як екземпляр типу `Object`, який є предком всіх класів Java, включаючи компоненти вікна.

Так зроблено для того, щоб цей метод був універсальним, і працював для будь-яких компонентів. Але ми знаємо напевно, що в нашому вікні єдиною причиною такої події можуть бути тільки кнопки Тому *ми приводимо тип (we cast the type)* Object, що повертається, до типу `JButton`, розміщуючи тип в дужках перед ім'ям методу:

```
JButton clickedButton = (JButton) evt.getSource();
```

Зліва від знаку рівності оголошена змінна типу `JButton` і, хоча метод `getSource()` повертає дані типу `Object`, ми ніби говоримо JVM: *Не хвилюйся, я знаю напевно, що це - екземпляр JButton*.

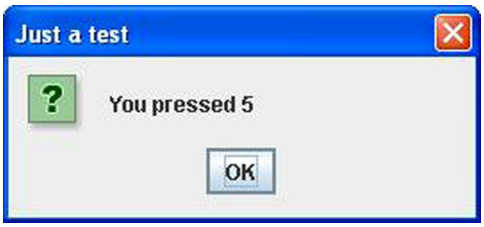


Тільки після приведення типу Object до JButton нам дозволяється викликати метод getText(), який належить класу JButton.

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;
import javax.swing.JButton;

public class CalculatorEngine implements ActionListener{

  public void actionPerformed (ActionEvent e) {

      //Отримуємо джерело події
      JButton clickedButton = (JButton) e.getSource();

      //Отримуємо напис на кнопці
      String clickedButtonLabel = clickedButton.getText();
```

```
        //Додаємо напис на кнопці до тексту вікна повідомлення
        JOptionPane.showConfirmDialog (null, "You pressed" +
              clickedButtonLabel,
              "Just a test", JOptionPane.PLAIN_MESSAGE);
    }
}
```

Наприклад, якщо ви натиснете кнопку "П'ять", то побачите це вікно повідомлення:



Але що, як події вікна будуть викликатися не тільки кнопками, а й якимись іншими компонентами? Не завжди потрібно приводити кожен об'єкт до типу `JButton`! Для цих випадків ви повинні використовувати спеціальний оператор Java, званий `instanceof`, щоб правильно зробити приведення типу. Наступний приклад спочатку перевіряє, об'єкт якого типу викликав подія, а потім робить приведення типу до `JButton` або до `JTextField`:

```
public void actionPerformed(ActionEvent evt){
      JTextField myDisplayField=null;
      JButton clickedButton=null;
      Object eventSource = evt.hetSource();
      if (eventSource instanceof JButton){
            clickedButton = (JButton) eventSource;
      }else if (eventSource instanceof JTextField){
            myDisplayField = (JTextField)eventSource;
      }
}
```

Нашому калькулятору потрібно виконувати різні частини програми для різних кнопок, і наступний приклад показує, як це зробити.

```
public void actionPerformed (ActionEvent e) {

      Object src = e.getSource();

      if (src == buttonPlus) {
        //Тут має бути текст програми, що додає числа

      }else if (src == buttonMinus) {
        //Тут має бути текст програми, що віднімає числа

      }else if (src == buttonDivide) {
        //Тут - ділення чисел

      }else if (src == buttonMultiply) {
        //Тут - текст програми, що множить числа
      }
}
```

Як передавати дані між класами

Взагалі-то, коли ви натискаєте кнопку з цифрою на справжньому калькуляторі, він не показує вікно повідомлення, а показує цю цифру на своєму дисплеї. Виникає завдання - нам потрібно отримати доступ до поля displayField класу Calculator з методу `actionPerformed()` класу CalculatorEngine. Це можна зробити, створивши в класі CalculatorEngine змінну, яка буде зберігати *посилання на екземпляр* об'єкта Calculator.

У наступній версії класу CalculatorEngine ми додамо конструктор. У цього конструктора буде один аргумент типу Calculator. Не дивуйтеся, аргументи у методів можуть мати тип класів, створених вами.

JVM виконує конструктор класу CalculatorEngine під час створення цього примірника в пам'яті. Клас Calculator створює CalculatorEngine, і пе-

редає його конструктору *посилання на себе*:

```
CalculatorEngine calcEngine = new CalculatorEngine(this);
```

Це посилання вказує на те місце в пам'яті, де знаходиться екземпляр класу Calculator. Конструктор CalculatorEngine зберігає це значення в змінній parent, щоб потім використовувати його в методі `actionPerformed()` для доступу до дисплея калькулятора.

```
parent.displayField.getText();
…
parent.displayField.setText(dispFieldText+clickedButtonLabel);
```

Ці два рядки були взяті з наступного прикладу:

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JButton;

public class CalculatorEngine implements ActionListener{

  Calculator parent; //посилання на Calculator

  //Конструктор зберігає посилання на вікно калькулятора
  //у змінній класу "parent"

  CalculatorEngine (Calculator parent) {
     this.parent = parent;
  }

public void actionPerformed (ActionEvent e) {

  //Отримати джерело поточної дії
     JButton clickedButton = (JButton) e.getSource();
```

```
    //Отримати поточний текст з поля виводу ("дисплея")
    //калькулятора
      String dispFieldText = parent.displayField.getText();

    //Отримати напис на кнопці
      String clickedButtonLabel = clickedButton.getText();
      parent.displayField.setText (dispFieldText +
                                   clickedButtonLabel);
    }
}
```

Коли ви оголошуєте змінну для зберігання посилання на екземпляр якогось класу, ця змінна повинна мати, або тип цього класу, або тип одного з його суперкласів.

Будь-який клас в Java походить від класу Object, і якщо, наприклад, клас Fish - це спадкоємець класу Pet, то кожен з цих рядків правильний:

```
Fish myFish = new Fish();
Pet myFish = new Fish();
Object myFish = new Fish();
```

Доробляємо калькулятор

Давайте визначимося з декількома правилами (з алгоритмом), за якими має працювати наш калькулятор:

1. Спочатку користувач вводить всі цифри першого числа.
2. Якщо користувач натисне якусь із кнопок арифметичної дії +, -, / чи *, то треба зберегти перше число і обрану дію в полях класу, і стерти число з дисплея калькулятора.
3. Потім користувач вводить друге число і натискає кнопку "=".
4. Сконвертувати строкове значення з дисплея в числовий тип double, щоб мати можливість зберігати великі дробові числа.

Провести арифметичну дію за допомогою обраної дії і першого числа, збережених в кроці 2.

5 Показати результат кроку 4 на дисплеї калькулятора і зберегти це значення в змінній, яка використовувалася в кроці 2.

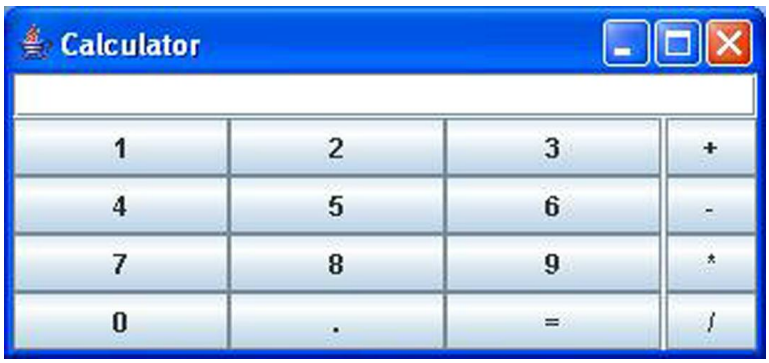
Всі ці кроки ми запрограмуємо в класі CalculatorEngine. Поки ви будете читати наступний текст програми, пам'ятайте, що метод `actionPerformed()` буде викликатися після кожного натиснення на кнопку і дані між викликами цього методу будуть зберігатися в змінних selectedAction і currentResult.

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JButton;

public class CalculatorEngine implements ActionListener{

  Calculator parent; //посилання на вікно калькулятора

  char selectedAction = ''; // +, -, /, або *
  double currentResult = 0;

//Конструктор зберігає посилання на вікно калькулятора
//у змінній екземпляра класу
  CalculatorEngine (Calculator parent) {
    this.parent = parent;
  }

  public void actionPerformed (ActionEvent e){

    //Отримати джерело дії
    JButton clickedButton = (JButton) e.getSource();
    String dispFieldText = parent.displayField.getText();
    double displayValue = 0;
```

```
//Отримати число з дисплея калькулятора,
//якщо він не порожній.
//Знак оклику - це оператор заперечення
  if (!"".equals(dispFieldText)) {
    displayValue = Double.parseDouble(dispFieldText);
  }
  Object src = e.getSource();

//Для кожної кнопки арифметичної дії
//запам'ятати її:+,-,/, або *, зберегти поточне число
//у змінній currentResult, і очистити дисплей
//для введення нового числа
  if (src == parent.buttonPlus) {
        selectedAction = '+';
        currentResult = displayValue;
        parent.displayField.setText ("");
  } else if (src == parent.buttonMinus) {
        selectedAction = '-';
        currentResult = displayValue;
        parent.displayField.setText ("");
  } else if (src == parent.buttonDivide) {
        selectedAction = '/';
        currentResult = displayValue;
        parent.displayField.setText ("");
  } else if (src == parent.buttonMultiply) {
        selectedAction = '*';
        currentResult = displayValue;
        parent.displayField.setText ("");
  } else if (src == parent.buttonEqual) {

   //Здійснити арифметичну дію, залежно
   //від selectedAction, оновити змінну currentResult
   //і показати результат на дисплеї
    if (selectedAction == '+') {
        currentResult += displayValue;
```

```
            //Конвертувати результат в рядок, додаючи його
            //до порожнього рядку і показати його
               parent.displayField.setText(""+currentResult);
            } else if (selectedAction == '-') {
               currentResult -= displayValue;
               parent.displayField.setText(""+currentResult);
            } else if (selectedAction == '/') {
               currentResult /= displayValue;
               parent.displayField.setText(""+currentResult);
             } else if (selectedAction == '*') {
               currentResult *= displayValue;
               parent.displayField.setText(""+currentResult);
             }
         } else {

          //Для всіх цифрових кнопок приєднати напис на
          //кнопці до напису в дисплеї
           String clickedButtonLabel=clickedButton.getText();
           parent.displayField.setText(dispFieldText +
                                       clickedButtonLabel);
         }
      }
   }
```

Наша заключна версія вікна калькулятора буде виглядати якось так:



Клас Calculator виконує такі дії:

- Створює і показує всі компоненти вікна.
- Створює екземпляр слухача CalculatorEngine.
- Передає CalculatorEngine посилання на себе.
- Реєструє цього слухача у всіх компонентах, які створюють події.

Ось остання версія класу Calculator:

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.BorderLayout;

public class Calculator {

  //Оголошуємо та ініціалізуємо компоненти вікна
     JButton button0=new JButton("0");
     JButton button1=new JButton("1");
     JButton button2=new JButton("2");
     JButton button3=new JButton("3");
     JButton button4=new JButton("4");
     JButton button5=new JButton("5");
     JButton button6=new JButton("6");
     JButton button7=new JButton("7");
     JButton button8=new JButton("8");
     JButton button9=new JButton("9");

     JButton buttonPoint=new JButton(".");
     JButton buttonEqual=new JButton("=");
     JButton buttonPlus=new JButton("+");
     JButton buttonMinus=new JButton("-");
     JButton buttonDivide=new JButton("/");
     JButton buttonMultiply=new JButton("*");

     JPanel windowContent=new JPanel();
     JTextField displayField=new JTextField(30);
```

```
// Конструктор
   Calculator(){

   //Встановити менеджер розташування для панелі
     BorderLayout bl = new BorderLayout();
     windowContent.setLayout (bl);

   //Додаємо дисплей у верхній частині вікна
     windowContent.add("North", displayField);

   //Створюємо панель з менеджером розташування
   //GridLayout в якій буде 12 кнопок - 10 цифр, та
   //кнопки "крапка" і "дорівнює"
     JPanel p1 = new JPanel();
     GridLayout gl = new GridLayout(4,3);
     p1.setLayout(gl);
     p1.add(button1);
     p1.add(button2);
     p1.add(button3);
     p1.add(button4);
     p1.add(button5);
     p1.add(button6);
     p1.add(button7);
     p1.add(button8);
     p1.add(button9);
     p1.add(button0);
     p1.add(buttonPoint);
     p1.add(buttonEqual);

   //Додаємо панель p1 в центр вікна
     windowContent.add("Center",p1);

   //Створюємо панель з менеджером розташування
   //GridLayout на якій буде 4 кнопки -
   //Плюс, Мінус, Розділити та Помножити
     JPanel p2 = new JPanel();
     GridLayout gl2 = new GridLayout(4,1);
```

```
        p2.setLayout(gl2);
        p2.add(buttonPlus);
        p2.add(buttonMinus);
        p2.add(buttonMultiply);
        p2.add(buttonDivide);

      //Додаємо панель p2 в праву частину вікна
        windowContent.add("East",p2);

      //Створюємо frame і додаємо в нього вміст
        JFrame frame = new JFrame ("Calculator");
        frame.setContentPane(windowContent);

      //Встановлюємо розмір вікна, так щоб вмістилися
      //Всі компоненти
        frame.pack();

      //Показуємо вікно
        frame.setVisible (true);

      //Створюємо екземпляр слухача подій і
      //Реєструємо його в кожній кнопці
        CalculatorEngine calcEngine = new
                                CalculatorEngine(this);
        button0.addActionListener(calcEngine);
        button1.addActionListener(calcEngine);
        button2.addActionListener(calcEngine);
        button3.addActionListener(calcEngine);
        button4.addActionListener(calcEngine);
        button5.addActionListener(calcEngine);
        button6.addActionListener(calcEngine);
        button7.addActionListener(calcEngine);
        button8.addActionListener(calcEngine);
        button9.addActionListener(calcEngine);
        buttonPoint.addActionListener(calcEngine);
        buttonPlus.addActionListener(calcEngine);
        buttonMinus.addActionListener(calcEngine);
        buttonDivide.addActionListener(calcEngine);
```

```
        buttonMultiply.addActionListener(calcEngine);
        buttonEqual.addActionListener(calcEngine);
    }
    public static void main(String[] args) {

        //Створюємо екземпляр класу "Калькулятор"
           Calculator calc = new Calculator();
    }
}
```

Тепер скомпілюємо проект і запустимо клас Calculator. Він працює майже так само, як і справжні калькулятори. Вітаю! Це ваша перша програма, яка може стати в нагоді багатьом людям-подаруйте її своїм друзям.

Для того, щоб краще розуміти, як працює ця програма, я рекомендую вам освоїти налагодження програм. Будь ласка, прочитайте Додаток А про налагоджувач Eclipse, а потім продовжуйте читання далі.

Деякі інші слухачі подій

Існують і інші слухачі в Java в пакеті java.awt, які непогано було б знати:

- Слухач фокусу (`FocusListener`) посилає сигнал вашому класу, коли компонент вікна отримує або втрачає фокус. Наприклад, кажуть, що текстове поле має фокус, якщо в ньому блимає курсор.
- Слухач елемента (`ItemListener`) реагує на вибір елементів у звичайному або випадаючому списку.
- Слухач клавіш (`KeyListener`) реагує на натискання клавіш.
- Слухач миші (`MouseListener`) реагує, коли натискають на кнопку миші, або вона входить в область компонента вікна або виходить з неї.
- Слухач рухів миші (`MouseMotionListener`) повідомляє вам, коли миша рухається або щось тягне. Тягти (drag) означає

рухати мишу з натиснутою клавішею.

- Слухач вікна (`WindowListener`) дає вам шанс вловити моменти, коли користувач відкриває, закриває, залишає вікно або заходить в нього.

У наступній таблиці ви побачите імена інтерфейсів слухачів, і методи, які ці інтерфейси оголошують.

Interface	Methods to implement
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent
ItemListener	itemStateChanged(ItemEvent)
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyType d (KeyEve nt)
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener	windowActivated (WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated (WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

Наприклад, інтерфейс `FocusListener` оголошує два методи: `focusGained()` і `focusLost()`. Це означає, що навіть якщо ваш клас зацікавлений тільки в обробці події отримання фокусу якимось елементом вікна, ви все одно повинні включити порожній метод `focusLost()`. Це може дратувати, тому Java надає спеціальні *класи-адаптери* для кожного слухача, щоб спростити обробку подій.

Як використовувати адаптери

Скажімо, вам потрібно зберегти якусь інформацію на диск, коли користувач закриває вікно. Відповідно до попередньої таблиці, клас, який реалізує інтерфейс `WindowsListener` повинен включати сім методів. Це означає, що вам доведеться писати текст програми в методі `windowClosing()` і ще включити шість порожніх методів.

У пакеті java.awt є адаптери, які є класами, що вже реалізували всі необхідні методи (правда, ці методи порожні всередині). Один з таких класів - так званий `WindowAdapter`. Ви можете успадкувати клас, який обробляє події, від класу `WindowAdapter` і просто перевизначити методи, які вам потрібні, наприклад метод `windowClosing()`.

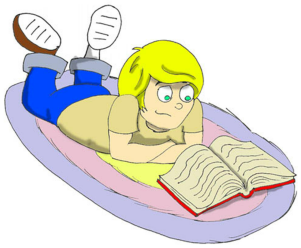
```
class MyEventProcessor extends java.awt.WindowsAdapter{

     public void windowClosing(WindowEvent e) {

          // тут знаходиться ваш текст програми,
          // що зберігає дані на диск

     }
}
```

Решта просто - просто зареєструйте цей клас, як слухач подій вікна у вашому класі:

```
MyEventProcessor myListener = new MyEventProcessor();
addWindowListener(myListener);
```

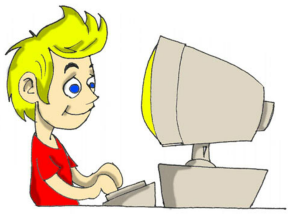
Такого ж результату можна досягти, використовуючи так звані анонімні внутрішні класи, але ця тема трохи важкувата для цієї книги.

Матеріали для додаткового читання



Пишемо свої слухачі подій:
http://download.oracle.com/javase/tutorial/uiswing/events/

Практичні вправи



Спробуйте розділити число на нуль за допомогою нашого калькулятора - дисплей покаже слово Infinity. Змініть клас CalculatorEngine, щоб відображалося повідомлення *"На нуль ділити не можна"*, якщо користувач натисне на кнопку "Розділити", коли дисплей калькулятора буде порожній.

Практичні вправи для розумників і розумниць

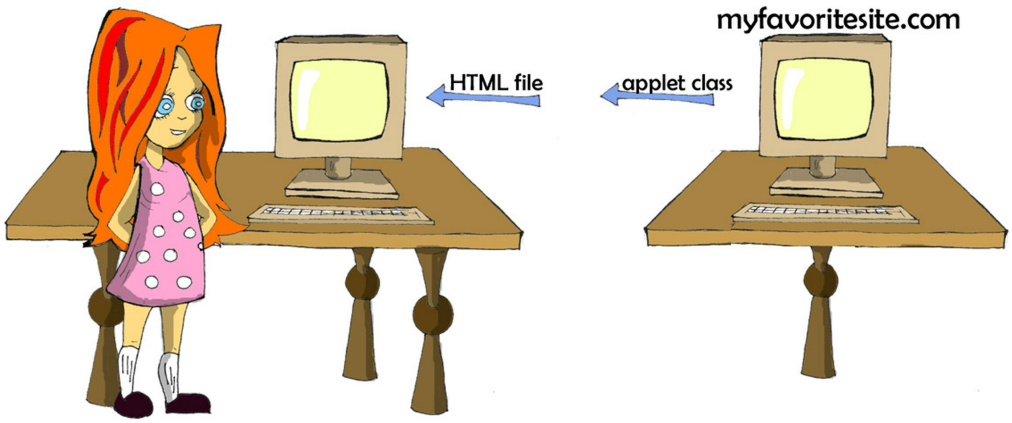
Змініть клас CalculatorEngine, щоб заборонити вводити більше однієї точки в числі.

Підказка: прочитайте про метод `indexOf()` класу `String`, щоб дізнатися, чи є вже в числі крапка.

Розділ 7. Аплет Хрестики-Нулики

Коли ви заходите на свій улюблений сайт, є невелика вірогідність, що деякі з ігор або інших програм на сайті були написані на Java за допомогою технології так званих *аплетів*. Ці спеціальні додатки живуть і працюють усередині вікна веб-браузера. Веб-браузери розуміють просту мову розмітки, що називається HTML, яка дозволяє вам вставляти спеціальні мітки ("*теги*") в текстові файли, щоб вони гарно відображалися в браузерах. Крім тексту, ви можете вставляти у файли HTML спеціальний тег <applet>, який підкаже браузеру, де знайти і як правильно показати аплет Java.

Java-аплети завантажуються на ваш комп'ютер з інтернету, як частина веб-сторінки, а браузер досить розумний, щоб запустити свою JVM для того, щоб запустити ці аплети.



У цьому розділі ви навчитеся, як створювати аплети на своєму комп'ютері, а в Додатку В дізнаєтеся, як опублікувати свої веб-сторінки в інтернеті, щоб інші люди теж могли ними користуватися.

Люди бродять в інтернеті, не знаючи, чи містять веб-сторінки аплет Java чи ні, але вони хочуть бути впевнені, що їх комп'ютерам не загрожують

які-небудь погані хлопці, які додали якийсь шкідливий аплет на сторінку. Тому аплети були розроблені з наступними обмеженнями:

- Аплети не можуть отримати доступ до файлів на вашому диску, поки ви не завантажили собі на диск спеціальний сертифікат, що дає їм таке право.
- Аплети можуть під'єднуватися по мережі тільки до того комп'ютера, з якого вони були викачані.
- Аплети не можуть запускати ніякі інші програми на вашому комп'ютері.

Щоб запустити аплет, вам знадобиться особливим чином написаний клас Java, текстовий файл HTML, що містить тег <applet>, який вказує на цей клас, і веб-браузер, що підтримує Java. Також можна тестувати аплети в Eclipse або використовуючи спеціальну програму, яка називається appletviewer. Але перш ніж вчитися робити аплети, давайте витратимо 15 хвилин на те, щоб познайомитися з деякими тегами HTML.

Вивчаємо HTML за 15 хвилин

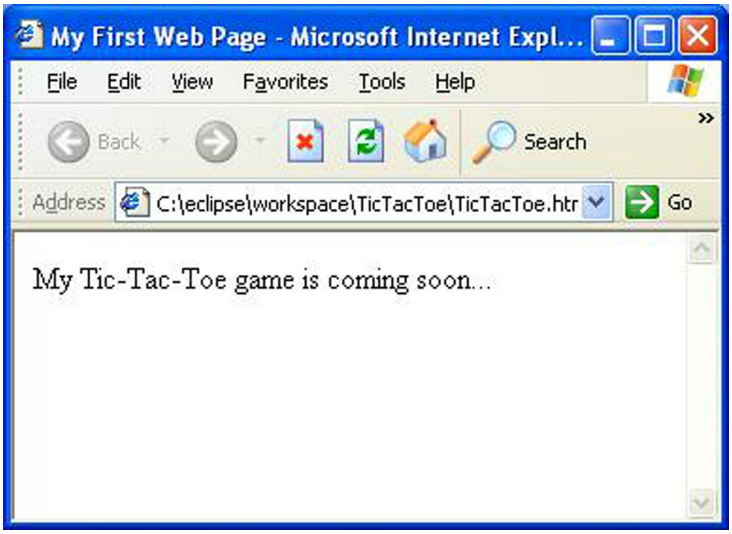
Уявіть на секунду, що ви написали і скомпілювали гру-аплет під назвою “Хрестики-Нулики” (TicTacToe). Тепер вам потрібно створити файл HTML з інформацією про нього. Спочатку створіть текстовий файл і назвіть його TicTacToe.html (до речі, Eclipse може створювати і текстові файли теж).

Імена файлів HTML закінчуються на *.html* або *.htm*. Всередині у них зазвичай є секції "заголовок" (*header*) і "тіло" (*body*). Більшість тегів HTML мають відповідні закриваючі теги, які починаються з прямого слеша ("/"), наприклад <Head> і </ Head>. Файл TicTacToe.html може виглядати ось так:

```
<HTML>

     <HEAD>
          <TITLE>My First Web Page</TITLE>
     </HEAD>
```

```
      <BODY>
           My Tic-Tac-Toe game is coming soon
      </BODY>

</HTML>
```

Ви можете розташовувати теги на одному рядку, як ми зробили з тегами <Title> і </ Title>, або ж на різних рядках. HTML - не Java, тут не важливо, як писати - великими або маленькими літерами. Відкрийте цей файл у своєму веб-браузері за допомогою меню Файл та Відкрити. У заголовку вікна буде написано *My First Web Page*, а на самій сторінці ви побачите слова *My Tic-Tac-Toe game is coming soon ...*:



Тепер змінимо цей файл, додавши туди тег для аплету "Хрестики-нулики":

```
<HTML>
  <BODY>
     Here is my Tic-Tac-Toe game:
     <APPLET code="TicTacToe.class" width=300 height=250>
     </APPLET>
  </BODY>
</HTML>
```

Тепер екран виглядає інакше:

Це не дивно: оскільки веб-браузер не зміг знайти клас TicTacToe, він просто показав сірий прямокутник. Ми створимо цей клас трохи пізніше.

Теги HTML поміщають в кутові дужки, і деякі з тегів можуть мати додаткові атрибути. Тег <APPLET> у нашому прикладі використовує такі атрибути:

- code - це ім'я класу Java-аплета.
- width - це ширина в пікселях прямокутної області екрану, де буде розташовуватися аплет. Зображення на екрані комп'ютера складаються з маленьких точок, які називаються пікселями.
- height - це висота області екрану, де буде розташовуватися аплет.

Якщо Java-аплет складається з декількох класів, упакуйте їх усі в один архівний файл за допомогою програми *jar*, яка поставляється разом з JDK (Java Developer Kit - Комплект Розробника Java). Якщо ви так зробите, атрибут "архів" (*"archive"*) повинен мати значення, рівне імені цього архіву. Ви можете прочитати про архіви jar в Додатку А.

Аплети і AWT

Навіщо використовувати бібліотеку AWT для того, щоб писати аплети, якщо Swing краще? Чи можна писати аплети, використовуючи класи Swing? Так, але ви повинні знати про деякі нюанси.

Веб-браузери поставляються зі своїми власними версіями JVM, які підтримують AWT, і можуть не підтримувати класи Swing, які включені в ваш аплет. Звичайно ж, користувачі можуть завантажити і встановити останню версію JVM, і є навіть спеціальні конвертери HTML, які змінять файл HTML так, щоб їх браузери могли завантажити цю нову версію JVM, але чи дійсно ви хочете попросити користувачів зробити це? Після того, як ваша сторінка буде опублікована в Інтернеті, ви не будете знати, хто стане нею користуватися.

Уявіть собі дідуся десь у пустелі з комп'ютером десятирічної давності - він просто піде з вашої сторінки, замість того, щоб проходити через всі ці неприємності з установкою. Уявіть, що наш аплет допомагає продавати онлайн-ігри, і ми не хочемо втратити цієї людини - він може бути нашим потенційним покупцем (у людей в пустелі теж бувають кредитні картки).

Використовуйте AWT, якщо вашими аплетами будуть користуватися неадеквати, що працюють на комп'ютерах минулого століття.

З іншого боку, все не так вже погано. Останні версії Java включають так званий плагін наступного покоління. Тепер аплети не зобов'язані виконуватися в JVM, яка йде з вашим Веб браузером - вони виконуються в окремій JVM, яка запускається цим плагіном. Аплет так само живе всередині віконця браузера, але вже не залежить від бажання (або небажання) виробників браузерів включати найсвіжіші JVM у свої поставки. Детальніше про все це можна почитати тут: https://jdk6.java.net/plugin2.

Щоб перевірити або поміняти установки цього плагіна в Microsoft Windows зайдіть в Java Control Panel - натисніть на іконку Java в системному меню Start | Control Panel. Під закладкою Advanced знайдіть Java Plug-in і переконайтеся, що обрана установка Enable the next-generation Java Plug-in. Процес підключення плагіна на макбуках описаний тут: http://blogs.sun.com/thejavatutorials/entry/enabling_the_next_generation_java.

Між нами, вже і Swing застарів для написання графічних програм для інтернет додатків. Розробники Java придумали нову мову спеціально для цих цілей. Вона називається JavaFX і, якщо хто цікавиться, то вам сюди: http://javafx.com/.

Як писати аплети

Java-аплети AWT повинні бути успадковані від класу `java.applet.Applet`, наприклад:

```
class TicTacToe extends java.applet.Applet {

}
```

Якщо використовувати Swing, то успадковуватися потрібно від класу JApplet:

```
class TicTacToe extends javax.swing.JApplet {

}
```

На відміну від звичайних програм Java, аплетам не потрібен метод main (), тому що веб-браузер сам *завантажить* і запустить їх, як тільки зустріне на сторінці тег <applet>. Також браузер буде посилати сигнали аплетам, коли будуть відбуватися важливі події, наприклад запуск аплету, перемальовування аплету, і так далі. Щоб переконатися, що аплет реагує на ці події, ви повинні написати спеціальні *методи зворотного виклику ("callback methods")*: `init()`, `start()`, `paint()`, `stop()` і `destroy()`. JVM веб-браузера буде викликати ці методи у таких випадках:

- `init()` викличеться, коли аплет завантажується браузером. Він викликається тільки один раз, таким чином, цей метод грає роль конструктора в звичайних класах Java.
- `start()` викличеться відразу після `init()`. Він також викликається, коли користувач повертається на цю сторіночку після відвідування інший сторінки.
- `paint()` викличеться, коли буде потрібно показати або оновити вікно аплета після якихось дій на екрані. Наприклад, аплет перекривається якимось іншим вікном і браузеру потрібно його перемалювати.
- `stop()` викличеться, коли користувач залишає веб-сторінку, яка містить аплет.
- `destroy()` - викличеться, коли браузер знищує аплет. Вам доведеться писати текст програми в цьому методі тільки якщо аплет використовує деякі зовнішні ресурси, наприклад, він підтримує з'єднання з комп'ютером, з якого він був завантажений.

І хоча ви не зобов'язані писати всі ці методи, в кожному аплеті повинен бути хоча б один з цих методів: `init()` або `paint()`. Ось текст програми аплету, який показує слова *"Привіт, Світ!"*. У цьому аплеті є тільки один метод `paint()`, який отримує екземпляр об'єкта `Graphics` від JVM веб-браузера. У цього об'єкта є цілий набір методів для малювання. У наступному прикладі використовується метод `drawString()`, щоб намалювати текст *"Привіт, Світ!"*.

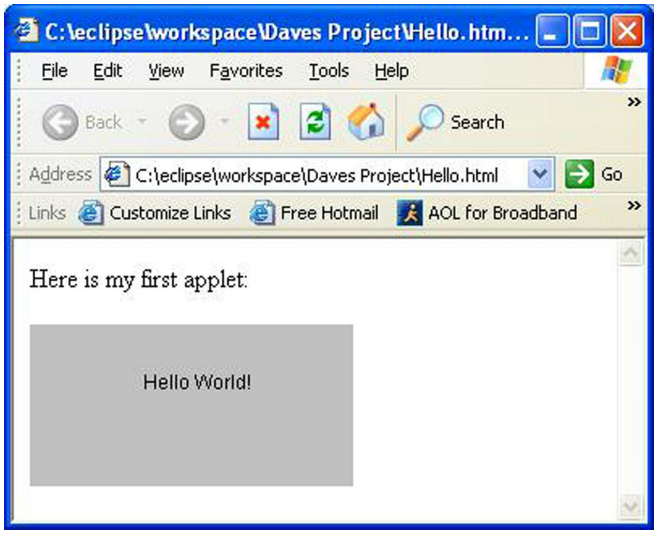
```
public class HelloApplet extends java.applet.Applet {

     public void paint(java.awt.Graphics graphics) {
          graphics.drawString("Привіт, Світ!", 70, 40);
     }
}
```

Створіть цей клас в Eclipse. Потім у вікні *"Run"* виберіть *"Java Applet"* в лівому верхньому кутку, натисніть кнопку *"New"*, і введіть HelloApplet в полі *"Applet Class"*.

Щоб протестувати цей аплет у веб-браузері, створіть файл *Hello.html* в тій самій папці, де знаходиться ваш аплет:

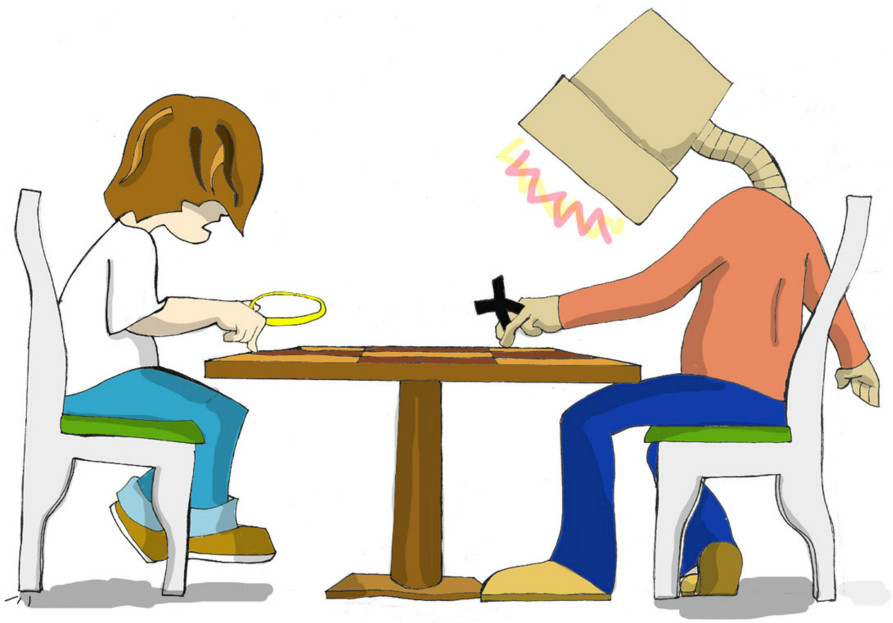
```
<HTML>
  <BODY>
   Here is my first applet:<P>
   <APPLET code="HelloApplet.class" width=200 height=100>
   </APPLET>
  </BODY>
</HTML>
```

Тепер запустіть ваш веб-браузер і відкрийте файл *Hello.html* за допомогою пунктів меню *File* і *Open*. Вікно має виглядати приблизно так:



Як ви думаєте, після цього простого прикладу ми зможемо написати гру? Ще б пак! Пристебніть ваші ремені ...

Пишемо гру “Хрестики-нулики”



Стратегія

Кожна гра використовує певний алгоритм - набір правил чи стратегію, яка застосовується в залежності від дій гравця. Алгоритми для однієї і тієї ж гри можуть бути простими або дуже складними. Коли ви чуєте, що чемпіон світу з шахів Гаррі Каспаров грає проти комп'ютера, насправді він грає проти програми. Цілі команди експертів намагаються винайти витончені алгоритми, щоб обіграти його. Гра Хрестики-нулики також може бути запрограмована за допомогою різних стратегій, ну а ми використаємо найпростішу:

- У нас буде дошка розміром 3x3. Людина буде грати хрестиками, а комп'ютер буде грати нуликами.

- Щоб перемогти, треба повністю заповнити ряд, колонку або діагональ поля однаковими символами.
- Після кожного ходу програма повинна перевіряти, чи є переможець. Якщо є переможець, то виграшна комбінація повинна виділятися іншим кольором і гра повинна закінчуватися.
- Гра також повинна закінчуватися, якщо більше не залишилося вільних клітин. Щоб почати нову гру, людина повинна натиснути кнопку "New Game".
- Коли комп'ютер приймає рішення, куди поставити наступний нулик, він повинен спробувати знайти ряд, колонку або діагональ, в якій вже є два нулика, і поставити там третій.
- Якщо таких рядів, колонок або діагоналей немає, то комп'ютер повинен спробувати знайти такі ж ряди з двома хрестиками, і поставити там нулик, щоб заблокувати виграшний хід гравця.
- Якщо не було знайдено ні виграшного, ні блокуючого ходу, то комп'ютер повинен спробувати зайняти центральну клітку, або ж зайняти будь-яку випадкову вільну клітинку.

Текст програми

Тут я коротко опишу програму, тому що в тексті програми аплету є багато коментарів, які допоможуть вам зрозуміти, як вона працює.

Аплет буде використовувати BorderLayout, у верхній частині вікна буде розташована кнопка "New Game". У центрі вікна будуть розташовуватися дев'ять кнопок, що представляють клітини поля, а в нижній частині будуть відображатися повідомлення:

Всі компоненти вікна будуть створюватися в методі аплету `init()`. Всі події будуть оброблятися слухачем `ActionListener` в методі `actionPerformed()`. Метод lookForWinner () буде викликатися після кожного ходу, щоб перевірити, чи закінчилася гра.

Останні три правила нашої стратегії запрограмовані в методі computerMove (), якому може знадобитися генерувати *випадкові числа*. Це робиться за допомогою класу `Java Math` і його методу `random()`.

Вам також може здатися незвичайним синтаксис, коли кілька методів викликаються в одному `виразі`, наприклад:

```
if(squares[0].getLabel().equals(squares[1].getLabel())){…}
```

Цей рядок робить програму коротшою, тому що в ній здійснюються такі ж дії, які могли бути зроблені в наступних рядках:

```
String label0 = squares[0].getLabel();
String label1 = squares[1].getLabel();

if(label0.equals(label1)){…}
```

Java обчислює вираз в дужках, перш ніж виконувати будь-які інші обчислення. Коротка версія цього тексту програми спочатку отримує результат виразу в дужках, та одразу ж використовує його як аргумент для методу `equals()`, який застосовується для результату першого виклику `getLabel()`.

Хоча текст програми гри займає кілька сторінок, він не повинен бути складним для розуміння. Просто читайте код і всі коментарі в програмі.

```
/**
*  Гра хрестики-нулики на дошці 3x3
*/

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class TicTacToe extends Applet implements ActionListener{

Button squares[];
Button newGameButton;
Label score;
int emptySquaresLeft=9;

/**
 * Метод init – це конструктор аплету
 */

  public void init(){

  //Встановлюємо менеджер розташування аплету, шрифт та колір
    this.setLayout(new BorderLayout());
    this.setBackground(Color.CYAN);

  //Змінюємо шрифт аплету так, щоб він був жирним
  //і мав розмір 20
    Font appletFont=new Font("Monospased",Font.BOLD, 20);
    this.setFont(appletFont);
```

```
//Створюємо кнопку "Нова гра" і реєструємо в ній
//слухач дії
  newGameButton=new Button("New Game");
  newGameButton.addActionListener(this);
  Panel topPanel=new Panel();
  topPanel.add(newGameButton);
  this.add(topPanel,"North");
  Panel centerPanel=new Panel();
  centerPanel.setLayout(new GridLayout(3,3));
  this.add(centerPanel,"Center");
  score=new Label("Your turn!");
  this.add(score,"South");

//створюємо масив, щоб зберігати посилання на 9 кнопок
  squares=new Button[9];

//Створюємо кнопки, зберігаємо посилання на них в масиві
//реєструємо в них слухача, фарбуємо їх
//в помаранчевий колір і додаємо на панель
  for(int i=0;i<9;i++){
    squares[i]=new Button();
    squares[i].addActionListener(this);
    squares[i].setBackground(Color.ORANGE);
    centerPanel.add(squares[i]);
  }
}

/**
* Цей метод буде обробляти всі події
* @param ActionEvent об'єкт
*/

public void actionPerformed(ActionEvent e) {

  Button theButton = (Button) e.getSource();
```

```
//Це кнопка New Game?
  if (theButton == newGameButton){

    for (int i=0;i<9;i++){
      squares[i].setEnabled(true);
      squares[i].setLabel("");
      squares[i].setBackground(Color.ORANGE);
    }

    emptySquaresLeft=9;
    score.setText("Your turn!");
    newGameButton.setEnabled(false);
    return; //виходимо з методу
  }

  String winner = "";

//Це одна з клітинок?
  for (int i=0; i<9; i++) {

    if (theButton == squares[i]) {
      squares[i].setLabel("X");
      winner = lookForWinner();
      if(!"".equals(winner)){
        endTheGame();
      } else {
        computerMove();
        winner = lookForWinner();
        if(!"".equals(winner)){
          endTheGame();
        }
      }

      break;
    }
  } //кінець циклу for
```

```
    if (winner.equals("X")) {
      score.setText("You won!");
    } else if (winner.equals("O")){
      score.setText("You lost!");
    } else if (winner.equals("T")){
      score.setText("It's a tie!");
    }
  } //кінець методу actionPerformed

/**
* Цей метод викликається після кожного ходу, щоб дізнатись,
* чи є переможець. Він перевіряє кожен ряд, колонку та
* діагональ, щоб знайти три клітинки з однаковими написами
* (не пустими)
* @return "X", "O", "T" - нічия, "" - ще немає переможця
*/

  String lookForWinner() {

    String theWinner = "";
    emptySquaresLeft--;

    if (emptySquaresLeft==0){
      return "T";  //це нічия. T від англійского слова tie
    }

  //Перевіряємо ряд 1 - елементи масиву 0,1,2
    if (!squares[0].getLabel().equals("") &&
     squares[0].getLabel().equals(squares[1].getLabel()) &&
     squares[0].getLabel().equals(squares[2].getLabel())) {
       theWinner = squares[0].getLabel();
       highlightWinner(0,1,2);

  //Перевіряємо ряд 2 - елементи масиву 3,4,5
    } else if (!squares[3].getLabel().equals("")  &&
     squares[3].getLabel().equals(squares[4].getLabel()) &&
     squares[3].getLabel().equals(squares[5].getLabel())) {
```

```
        theWinner = squares[3].getLabel();
        highlightWinner(3,4,5);

    //Перевіряємо ряд 3 - елементи масиву 6,7,8
      } else if (! squares[6].getLabel().equals("") &&
       squares[6].getLabel().equals(squares[7].getLabel()) &&
       squares[6].getLabel().equals(squares[8].getLabel())) {
        theWinner = squares[6].getLabel();
        highlightWinner(6,7,8);

    //Перевіряємо колонку 1 - елементи масиву 0,3,6
      } else if (! squares[0].getLabel().equals("") &&
       squares[0].getLabel().equals(squares[3].getLabel()) &&
       squares[0].getLabel().equals(squares[6].getLabel())) {
        theWinner = squares[0].getLabel();
        highlightWinner(0,3,6);

    //Перевіряємо колонку 2 - елементи масиву 1,4,7
      } else if (! squares[1].getLabel().equals("") &&
       squares[1].getLabel().equals(squares[4].getLabel()) &&
       squares[1].getLabel().equals(squares[7].getLabel())) {
        theWinner = squares[1].getLabel();
        highlightWinner(1,4,7);

    //Перевіряємо колонку 3 - елементи масиву 2,5,8
      } else if (! squares[2].getLabel().equals("") &&
       squares[2].getLabel().equals(squares[5].getLabel()) &&
       squares[2].getLabel().equals(squares[8].getLabel())) {
        theWinner = squares[2].getLabel();
        highlightWinner(2,5,8);

    //Перевіряємо першу діагональ - елементи масиву 0,4,8
      } else if ( ! squares[0].getLabel().equals("") &&
       squares[0].getLabel().equals(squares[4].getLabel()) &&
       squares[0].getLabel().equals(squares[8].getLabel())) {
        theWinner = squares[0].getLabel();
        highlightWinner(0,4,8);
```

```
  //Перевіряємо другу діагональ - елементи масиву 2,4,6
    } else if ( ! squares[2].getLabel().equals("") &&
     squares[2].getLabel().equals(squares[4].getLabel()) &&
     squares[2].getLabel().equals(squares[6].getLabel())) {
      theWinner = squares[2].getLabel();
      highlightWinner(2,4,6);
    }

    return theWinner;
  }

/**
 * Цей метод застосовує набір правил, щоб знайти
 * кращий комп'ютерний хід. Якщо гарний хід
 * не знайдено, вибирається випадкова клітинка.
*/

  void computerMove() {

    int selectedSquare;

  //Спочатку комп'ютер намагається знайти порожню клітинку
  //поряд з двома клітинками з нуликами, щоб виграти
    selectedSquare = findEmptySquare("O");

  //Якщо він не може знайти два нулика, то хоча б
  //спробує не дати опонентові зробити ряд з 3-х
  //хрестиків, помістивши нулик поряд з двома хрестиками
    if (selectedSquare == -1){
      selectedSquare = findEmptySquare("X");
    }

  //якщо selectedSquare все ще дорівнює -1, то
  //спробує зайняти центральну клітинку
    if ((selectedSquare == -1) &&
                    (squares[4].getLabel().equals(""))){
      selectedSquare=4;
    }
```

```
  //не пощастило з центральною клітинкою ...
  //просто займаємо випадкову клітинку
    if (selectedSquare == -1){
       selectedSquare = getRandomSquare();
    }

    squares[selectedSquare].setLabel("O");
  }

/**
 * Цей метод перевіряє кожен ряд, колонку і діагональ
 * щоб дізнатися, чи є в ній дві клітинки
 * З однаковими написами і порожньою клітинкою.
 * @param   передається X - для користувача і O - для компа
 * @return  кількість вільних клітинок,
 *          або -1, якщо не знайдено дві клітинки
 *          з однаковими написами
*/

  int findEmptySquare(String player) {

    int weight[] = new int[9];

    for (int i = 0; i < 9; i++ ) {
      if ( squares[i].getLabel().equals("O") )
           weight[i] = -1;
        else if (squares[i].getLabel().equals("X") )
           weight[i] = 1;
        else
           weight[i] = 0;
    }

    int twoWeights = player.equals("O") ? -2 : 2;

  //Перевіримо, чи є в ряду 1 дві однакові клітинки і
  //одна порожня.
    if (weight[0] + weight[1] + weight[2] == twoWeights) {
```

```
      if (weight[0] == 0)
        return 0;
      else if (weight[1] == 0)
        return 1;
      else
        return 2;
    }

  //Перевіримо, чи є в ряду 2 дві однакові клітинки і
  //одна порожня.
    if (weight[3] + weight[4] + weight[5] == twoWeights) {
      if (weight[3] == 0)
        return 3;
      else if (weight[4] == 0)
        return 4;
      else
        return 5;
    }

  //Перевіримо, чи є в ряду 3 дві однакові клітинки і
  //одна порожня.
    if (weight[6] + weight[7] + weight[8] == twoWeights) {
      if (weight[6] == 0)
        return 6;
      else if (weight[7] == 0)
        return 7;
      else
        return 8;
    }

  //Перевіримо, чи є в колонці 1 дві однакові клітинки і
  //одна порожня.
    if (weight[0] + weight[3] + weight[6] == twoWeights) {
      if (weight[0] == 0)
        return 0;
      else if (weight[3] == 0)
        return 3;
      else
```

```
        return 6;
    }

  //Перевіримо, чи є в колонці 2 дві однакові клітинки і
  //одна порожня.
    if (weight[1] + weight[4] + weight[7] == twoWeights) {
      if (weight[1] == 0)
        return 1;
      else if (weight[4] == 0)
        return 4;
      else
        return 7;
    }

  //Перевіримо, чи є в колонці 3 дві однакові клітинки і
  //одна порожня.
    if (weight[2] + weight[5] + weight[8] == twoWeights ){
      if ( weight[2] == 0 )
        return 2;
      else if ( weight[5] == 0 )
        return 5;
      else
        return 8;
    }

  //Перевіримо, чи є в діагоналі 1 дві однакові клітинки і
  //одна порожня.
    if (weight[0] + weight[4] + weight[8] == twoWeights ){
      if ( weight[0] == 0 )
        return 0;
      else if ( weight[4] == 0 )
        return 4;
      else
        return 8;
    }

  //Перевіримо, чи є в діагоналі 2 дві однакові клітинки і
```

```
  //одна порожня.
    if (weight[2] + weight[4] + weight[6] == twoWeights ){
      if ( weight[2] == 0 )
        return 2;
      else if ( weight[4] == 0 )
        return 4;
      else
        return 6;
    }

  //Не знайдено двох однакових сусідніх клітинок
    return -1;

  } //кінець методу findEmptySquare()

/**
 * Цей метод вибирає будь-яку пусту клітинку
 * @return випадково вибраний номер клітинки
 */

  int getRandomSquare() {

    boolean gotEmptySquare = false;
    int selectedSquare = -1;

    do {
      selectedSquare = (int) (Math.random() * 9 );
      if (squares[selectedSquare].getLabel().equals("")){
        gotEmptySquare = true; //щоб закінчити цикл
      }
    } while (!gotEmptySquare );

    return selectedSquare;

  } //кінець методу getRandomSquare()
```

```
/**
 * Цей метод виділяє виграшну лінію.
 * @param преша, друга і третя клітинки для виділення
 */

  void highlightWinner(int win1, int win2, int win3) {
    squares[win1].setBackground(Color.CYAN);
    squares[win2].setBackground(Color.CYAN);
    squares[win3].setBackground(Color.CYAN);
  }

  //Робимо недоступними клітинки та доступною
  //кнопку "New Game"
  void endTheGame(){
    newGameButton.setEnabled(true);
    for (int i=0; i<9; i++){
      squares[i].setEnabled(false);
    }
  }
} //кінець класу
```

Вітаю! Ви створили вашу першу гру на Java!

При використанні бібліотеки Swing, потрібно поміняти `Button` на `JButton`, `Panel` на `JPanel`, і т.д. Більш детальна інформація, правда англійською, тут:

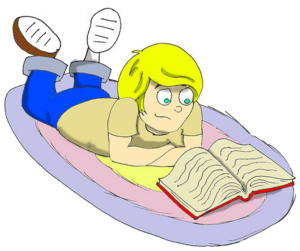
http://download.oracle.com/javase/tutorial/deployment/applet.

Цей аплет можна запустити або прямо з Eclipse, або відкривши файл TicTacToe.html, який ми створили на початку розділу, тільки скопіюйте файли *TicTacToe.html* і *TicTacToe.class* в одну папку. У нашому класі TicTacToe є маленька помилка - можливо, ви її навіть не помітили, але я впевнений, що вона зникне після того, як ви зробите друге завдання нижче.

Наш клас TicTacToe використовує просту стратегію, тому що наша мета - просто навчитися писати програми, але якщо ви хочете покращити цю гру,

вивчіть так звану стратегію мінімаксу, яка дозволить комп'ютеру вибрати найкращий хід. У цій книзі немає опису стратегії мінімаксу, проте він доступний онлайн.

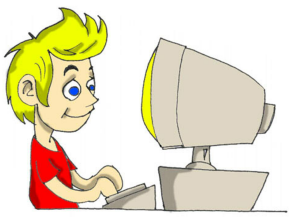
Матеріали для додаткового читання



Аплети Java:
http://download.oracle.com/javase/tutorial/deployment/applet/

Клас Math
http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html

Практичні вправи



1. Додайте на верхню панель класу TicTacToe два написи для підрахунку виграшів і програшів. Для цього оголосіть дві змінні в класі і збільшуйте відповідну змінну щоразу, коли людина виграє або програє. Рахунок повинен оновлюватися відразу після того, як програма виводить повідомлення "You won" або "You lost".

2. Наша програма дозволяє клацати на клітинці, в якій вже є хрестик або нулик. Це помилка! Програма продовжує працювати, як ніби ви зробили правильний хід. Змініть текст програми так, щоб натискання на такі клітинки ігнорувалися.

3. Додайте метод main () до класу TicTacToe, щоб мати можливість запускати гру не як аплет, а як Java-додаток.

Практичні вправи для розумників і розумниць

1. Перепишіть TicTacToe, щоб замінити одновимірний масив, який зберігає кнопки

```
JButton squares []
```

на двовимірний масив 3x3:

```
JButton squares [] []
```

2. Почитайте про багатовимірні масиви в інтернеті.

Розділ 8. Виключення - помилки в програмах

Скажімо, ви забули закрити фігурні дужки у своєму Java-коді. Це призведе до помилки компіляції, яку можна легко виправити. Але існують ще помилки часу виконання (*run-time errors*), коли, абсолютно несподівано, програма перестає працювати, як належить. Наприклад, Java-клас зчитує файл з рахунком у грі. Що станеться, якщо хтось видалить цей файл? Чи зупиниться програма з довгим і страшним повідомленням про помилку, або продовжить працювати, видавши доброзичливе повідомлення, типу:

```
Любий друже, з якоїсь причини, мені не вдалося прочитати файл scores.txt. Будь ласка, перевір, чи існує цей файл?
```

Створюйте програми, готові до незвичайних ситуацій. У багатьох мовах програмування обробка помилок залежить тільки від доброї волі програміста. Але Java змушує додавати код обробки помилок, інакше програма навіть не скомпілюється.

Помилки часу виконання в Java називаються *виключеннями (exceptions)*, а обробка таких помилок називається *обробкою виключень (exception handling)*. Код, який може викликати виключення, розміщуйте в так званий блок `try/catch`. Це наче сказати JVM наступне: *Спробуй прочитати файл з рахунком гри, але якщо щось піде не так, перехопи помилку і запусти код,*

```
try{
     fileScores.read();
} catch (IOException e){
```

```
        System.out.println("Любий друже, мені не вдалося
                            прочитати файл scores.txt");
}
```

Ви навчитеся працювати з файлами в розділі 9, а зараз запам'ятайте новий термін *I/O* або *введення/виведення (input/output)*. Операції читання і запису (на диск або інший пристрій) називаються введенням/виведенням, звідси `IOException` - це клас, який містить інформацію про помилки введення/виведення.

Метод *генерує (throws)* виключення у разі помилки. Для різних типів помилок генеруються різні виключення. Якщо в програмі є блок `catch` для даного конкретного типу помилки, то вона перехопиться і програма перейде до виконання коду в блоці `catch`. Програма продовжить роботу, і вважатиме, що про це виключення подбали.

У коді, наведеному вище, виведення тексту відбудеться тільки у випадку помилки читання файлу.

Читання трасування стеку

Якщо трапляється непередбачений виняток, який не обробляється програмою, на екран виводиться багаторядкове повідомлення про помилку. Таке повідомлення називається *трасуванням стеку (stack trace)*. Якщо ваша програма викликала кілька методів перед тим, як зіткнулася з проблемою, трасування стеку допоможе відстежити проблему і знайти рядок коду, який викликав помилку.

Давайте напишемо програму `TestStackTrace`, яка спеціально ділила б на нуль (номери рядків не є частиною коду).

```
1  class TestStackTrace{
2    TestStackTrace()
3    {
4      divideByZero();
5    }
6
7    int divideByZero()
8    {
9      return 25/0;
10   }
11
12   static void main(String[] args)
13   {
14     new TestStackTrace();
15   }
16 }
```

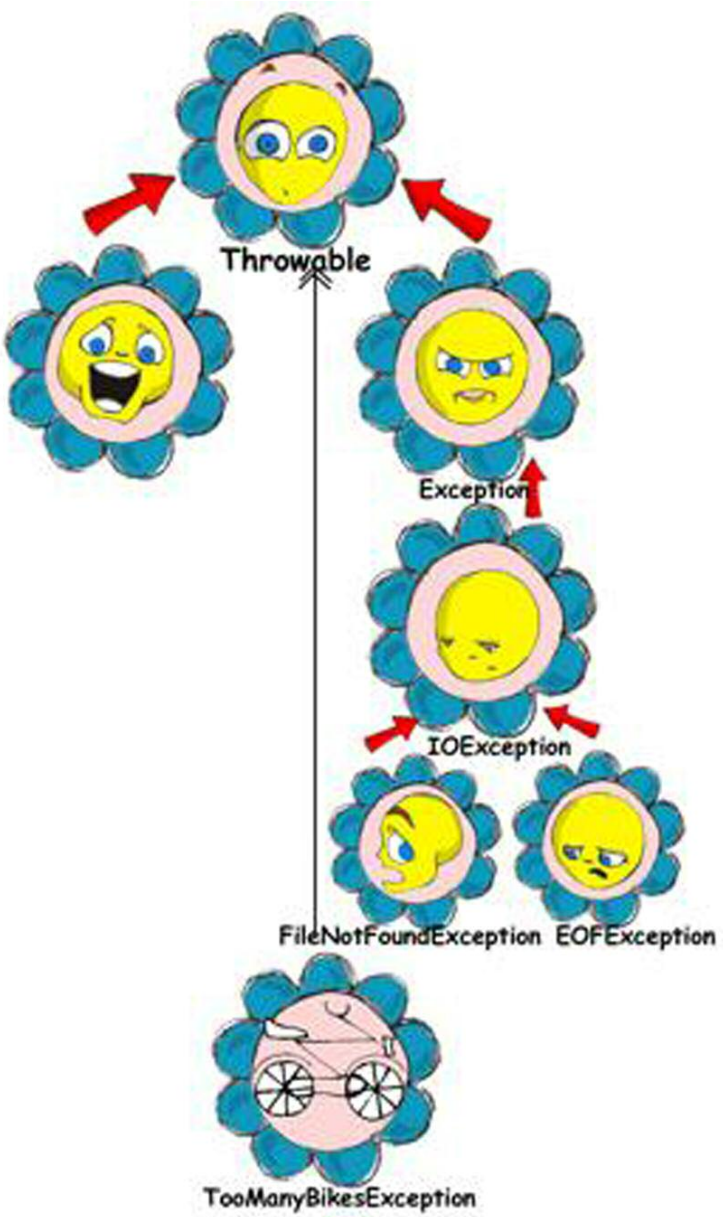
Виведення цієї програми показує послідовність методів, викликаних до моменту виникнення помилки часу виконання. Починайте читати це виведення з останнього рядка, просуваючись вгору.

```
Exception in thread "main"
java.lang.ArithmeticException: /by zero
  at TestTrace.divideByZero(TestStackTrace.java:9)
  at TestStackTrace.<init>(TestStackTrace.java:4)
  at TestStackTrace.main(TestStackTrace.java:14)
```

Це означає, що виконання програми почалося з методу `main()`, потім перейшло до `init()` - конструктору, потім був викликаний метод `divideByZero()`. Числа 14, 4 і 9 вказують, у яких рядках програми ці методи були викликані. Після цього, було викликано виняток `ArithmeticException`, тому що в рядку 9 програма намагалася розділити на нуль.

Генеалогічне дерево виключень

Виключення в Java - це теж класи, і деякі з них представлені ось у такій ієрархії (стрілка вгору вказує на батька):



Нащадки класу `Exception` називаються *некритичними виключеннями (checked exceptions)*, і ви повинні обробляти їх у своєму коді. Тобто програма якось буде продовжувати працювати після таких виключень, але отримані результати будуть не ті, що нам всім би хотілося.

Нащадки класу `Error` - це фатальні помилки JVM, з обробкою яких не може впоратися виконувана програма.

`TooManyBikesException` (занадто багато велосипедів) - це приклад виключення, який може бути створено програмістом.

Як програмісту дізнатися заздалегідь, що якийсь Java-метод якогось класу може викликати виключення, і що необхідно використовувати блок `try/catch`? Не варто турбуватися! Якщо ви викликаєте метод, який може призвести до виключення, компілятор Java видасть попередження про помилку, подібне до такого:

```
"ScoreReader.java":  unreported exception:
java.io.IOException; must be caught or declared to be
thrown at line 57
```

Додаткову інформацію про те, які виключення можуть бути викликані тими чи іншими методами, можна знайти в документації по Java. У частині розділу, що лишилася, ви навчитеся поводитися з цими виключеннями.

Блок try/catch

Для обробки помилок в Java можуть бути використані п'ять ключових слів: `try`, `catch`, `finally`, `throw` та `throws`.

Після одного блоку `try` можна поставити кілька блоків `catch`, якщо передбачається, що може відбутися більш ніж одна помилка. Наприклад, коли програма намагається прочитати файл, його може не виявитися на місці, і виникне виключення `FileNotFoundException`. Або ж, якщо файл знайдений, але програма продовжує зчитувати після кінця цього файлу, ви отримаєте виняток `EOFException`.

Наступний фрагмент коду виведе повідомлення українською мовою,

якщо програма не зможе знайти файл з рахунком гри або досягне кінця файлу. Для інших помилок читання код виведе повідомлення *Проблема при читанні файлу* і технічний опис проблеми.

```
public void getScores(){
  try{
    fileScores.read();
    System.out.println("Рахунок гри успішно завантажено");
  }catch(FileNotFoundException e){
    System.out.println("Не знайдено файл Scores");
  }catch(EOFException e1){
    System.out.println("Досягнуто кінця файлу");
  }catch(IOException e2){
    System.out.println("Проблема при читанні файлу " +
                                          e2.getMessage());
  }
}
```

Якщо виконання методу `read()` перерветься, програма перестрибне через рядок з викликом `println()` прямо до блоку `catch` для відповідної помилки. Якщо такий блок знайдений, то виконається відповідний `println()`, а якщо відповідний блок `catch` знайти не вдасться, метод `getScores()` перенаправляє це виключення методу, що його викликав.

Якщо ви пишете кілька блоків `catch`, вам слід розташовувати їх у порядку, згідно з тим, як відповідні виключення успадковані один від одного. Наприклад, оскільки `EOFException` - це підклас `IOException`, потрібно розташувати блок `catch` для підкласу спочатку. Якщо ж першим помістити `catch` для `IOException`, тоді програма ніколи не досягне `FileNotFound` або `EOFException`, тому що перший `catch` буде їх перехоплювати.

Починаючи з Java 7, можна відловлювати відразу кілька виключень в одному блоці `catch`:

```
public void getScores(){
  try{
    fileScores.read();
    System.out.println("Рахунок гри успішно завантажено");
  }catch(FileNotFoundException | EOFException |
                                          IOException e){
    System.out.println("Проблема при читанні файлу " +
                                          e.getMessage());
  }
}
```

Ледарі запрограмують метод `getScores()` ось так:

```
public void getScores(){
  try{
    fileScores.read();
  }catch(Exception e){
    System.out.println("Проблема при читанні файлу " +
                                          e.getMessage());
  }
}
```

Це приклад поганого стилю програмування. При написанні програми, завжди потрібно пам'ятати, що хтось інший може захотіти її прочитати, і вам може бути соромно за свій код.

Блок `catch` отримує об'єкт типу `Exception`, який містить короткий опис проблеми, що виникла, а його метод `getMessage()` повертає цей опис. Іноді, якщо опис помилки не до кінця зрозуміло, спробуйте використовувати метод `toString()`:

```
catch(Exception e){
  System.out.println("Проблема при читанні файлу " +
                                          e.toString());
}
```

Якщо потрібна більш детальна інформація про виключення, використовуйте метод printStackTrace (). Він виведе послідовність викликів, яка призвела до виникнення виключення, таку ж, як у прикладі з розділу Читання трасування стека.

Давайте спробуємо «вбити» програму-калькулятор з розділу 6. Запустіть клас Calculator і введіть з клавіатури abc. Натисніть будь-яку з кнопок арифметичних дій і ви отримаєте на екрані консолі щось на зразок цього:

```
java.lang.NumberFormatException: For input string: "abc" at
java.lang.NumberFormatException.forInputString(NumberFormatEx
ception.java:48)
     at java.lang.FloatingDecimal.readJavaFormatString(Float
ingDecimal.java:1213)
     at
java.lang.Double.parseDouble(Double.java:202)
     at
CalculatorEngine.actionPerformed(CalculatorEngine.java:27)
     at
javax.swing.AbstractButton.fireActionPerformed(AbstractBut
ton.java:1764)
```

Це був приклад реакції програми у разі необробленого виключення. У методі `actionPerformed()` класу `CalculatorEngine` є наступний рядок:

```
displayValue=Double.parseDouble(dispFieldText);
```

Якщо змінній `dispFieldText` присвоєно не числове значення, метод `parseDouble()` не зможе конвертувати його в значення типу double і вик-

личе виключення NumberFormatException.

Давайте обробимо це виключення й відобразимо повідомлення про помилку, яке пояснить ситуацію користувачеві. Рядок, що містить `parseDouble()` повинна бути поміщений в блок `try/catch`, і Eclipse в цьому допоможе. Виділіть цей рядок і клацніть на ньому правою кнопкою миші. В меню оберіть пункт *Source and Surround with try/catch block*. Вуаля! Код змінився:

```
try {
  displayValue=Double.parseDouble(dispFieldText);
} catch (NumberFormatException e1) {
  // TODO Auto-generated catch block
  e1.printStackTrace();
}
```

Замініть рядок з `printStackTrace()` на наступний код:

```
javax.swing.JOptionPane.showConfirmDialog(null,
     "Будь ласка, введіть число", "Неправильне введення",
                    javax.swing.JOptionPane.PLAIN_MESSAGE);
return;
```

Так ми позбулися страхітливого трасування стеку в повідомленні про помилку і тепер відображаємо просте для розуміння повідомлення *Будь ласка, введіть число:*

Тепер виключення `NumberFormatException` буде оброблятися.

Ключове слово throws

У деяких випадках, більш доцільно обробляти виключення не в тому методі, де воно виникло, а в тому, який його викликав. У таких випадках, в описі методу необхідно оголосити (попередити), що він може викликати деяке виключення. Це робиться за допомогою спеціального ключового слова `throws`. Давайте використаємо той самий приклад з читанням файлу.

Оскільки метод `read()` може викликати виключення `IOException`, ви повинні або обробляти його, або оголосити його в описі методу. У наступному прикладі ми оголосимо, що метод `getAllScores()` може викликати виключення `IOException`:

```
class MySuperGame{

  void getAllScores() throws IOException{
    //Не використовуйте try/catch, якщо ви не
    //обробляєте виключення всередині цього методу
    file.read();
  }

  public static void main(String[] args){
    MySuperGame msg = new MySuperGame();
    System.out.println("Список результатів гри");
    try{
      //Оскільки getAllScores() оголошує виключення,
      //ми обробляємо його тут
      msg.getAllScores();
    }catch(IOException e){
      System.out.println("Вибачте, список результатів гри
                                              недоступний");
    }
  }
}
```

Оскільки ми навіть не намагаємося перехоплювати виключення в методі `getAllScores()`, виключення `IOException` перенаправиться з нього в метод, що його викликав, `main()`. Тепер це виключення повинен обробити головний метод.

Ключове слово finally

Виконання будь-якого коду всередині блоку `try/catch` може закінчитися одним з таких варіантів:

- Виконання коду в блоці `try` закінчилося успішно, і програма продовжує роботу.
- Код всередині блоку `try` досяг виразу `return` і вийшов з методу.
- Код всередині блоку `try` викликав виключення, і контроль перейшов до відповідного блоку `catch`, який або обробляє помилку, або перенаправляє виключення до методу, який його викликав.

Якщо є код, який необхідно виконати у будь-якому випадку, помістіть його після ключового слова `finally`:

```
try{
     file.read();
}catch(Exception e){
     printStackTrace();
}finally{
  // код, який завжди повинен виконуватись
  // поміщається сюди, наприклад file.close();
}
```

Якщо написати код для закриття файлу в блок finally, то він виконається незалежно від того, чи пройшла операція читання успішно чи ні. Зазвичай, до блоку finally поміщається код, який звільняє будь-які ресурси комп'ютера, наприклад, здійснює відключення від мережі або закриття файлу.

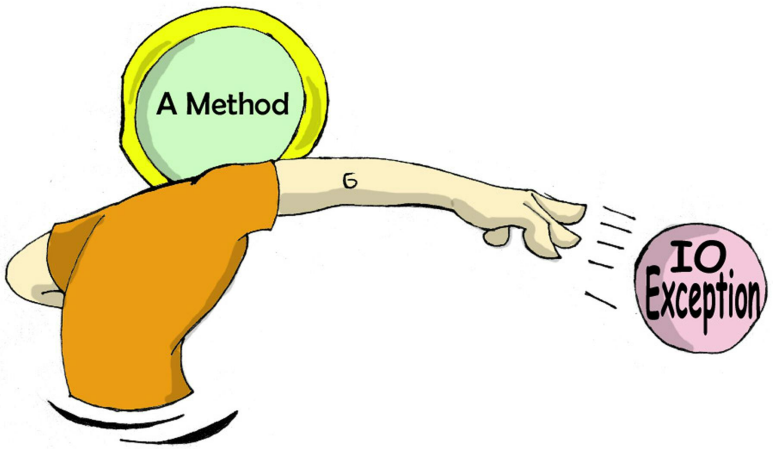
Якщо ви не плануєте обробляти виключення в поточному методі, вони будуть передані метод, що їх викликав. У такому випадку, ви можете використовувати ключове слово finally навіть без блоку catch:

```
void myMethod () throws IOException{

  try{
    //помістіть сюди код, що зчитує файл
  }

  finally{
    //помістіть сюди код, що закриває файл
  }
}
```

Ключове слово throw

Якщо в методі виникло виключення, але його має обробити викликаючий метод, 6просто перенаправте його в цей метод. Іноді вам може знадобитися перехопити одне виключення, а відправити далі інше, з іншим описом помилки, як у наведеному далі коді.

Вираз `throw` (кидати) призначено для "кидання" (відправлення) Java-об'єктів. Об'єкти, які кидаються програмою, повинні бути *киданими*. Це означає, що ви можете кидати тільки об'єкти, прямо або побічно успадковані від класу `Throwable`. Всі виключення в Java є нащадками цього класу.

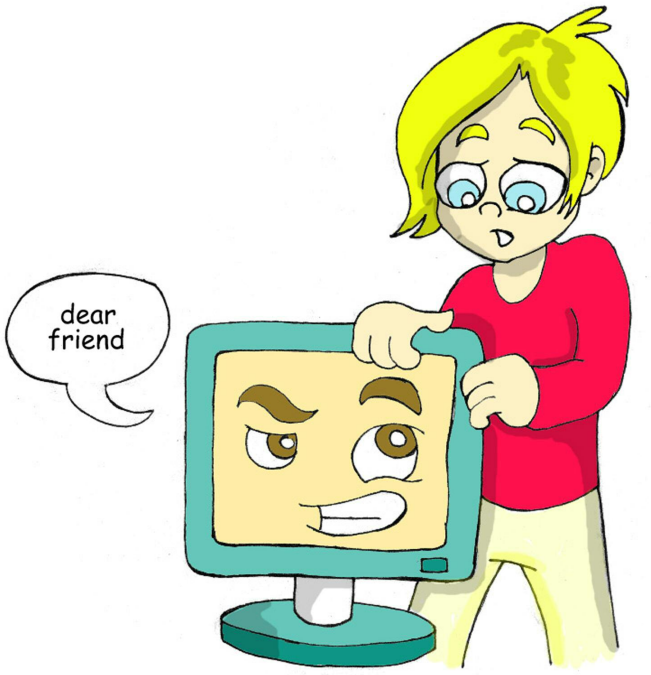


У наступному фрагменті коду, метод `getAllScores()` перехоплює виключення `IOException`, створює новий об'єкт типу `Exception` з більш зрозумілим описом помилки, і відправляє його в метод `main()`. Тепер, метод `main()` не скомпілюється, якщо ви не помістите рядки з викликом методу `getAllScores()` в блок `try/catch`, тому що цей метод може викликати виключення `Exception`, яке має бути, або оброблено, або перенаправлено далі. Метод main () не повинен створювати жодних виключень, тому повинен обробляти їх.

```
class ScoreList{

  //Для компіляції цього класу потрібен додатковий код
  static void getAllScores() throws Exception{

    try{
      file.read(); //цей рядок може викликати виключення
    } catch (IOException e) {
      throw new Exception("Любий друже, у файлі Scores
                                          є проблеми");
    }
  }
```

```
  public static void main(String[] args){

    System.out.println("Scores");
    try{
      getAllScores();
    } catch(Exception e1){
        System.out.println(e1.getMessage());
    }
  }
}
```

У випадку помилки роботи з файлом, головний метод обробить її, а метод `e1.getMessage()` виведе повідомлення *Любий друже ...*



Створення власних виключень

Програмісти можуть створювати класи виключень, яких спочатку не було в Java. Такі класи повинні бути успадковані від одного з існуючих виключень. Припустимо, ви займаєтеся продажем велосипедів і повинні перевіряти замовлення покупців. Залежно від моделей, у вашу вантажівку може поміститися різна кількість велосипедів.

Наприклад, ви можете завантажити в неї не більше трьох велосипедів FireBird. Створіть підклас класу `Exception` під назвою `TooManyBikesException`. Тепер, якщо хтось замовить більше трьох таких велосипедів, викличте виключення.

```
class TooManyBikesException extends Exception{

  //Конструктор
  TooManyBikesException (){
    //Просто викличте конструктор суперкласу
    //і передайте йому повідомлення, яке потрібно
    //відобразити
    super("Ми не зможемо доставити стільки велосипедів
                                          за один раз.");
  }
}
```

Цей клас містить тільки конструктор, який отримує повідомлення, яке описує помилку. Це повідомлення конструктор передає для зберігання суперкласу. Коли в блок `catch` потрапляє це виключення, можна дізнатися, що саме сталося, викликавши метод `getMessage()`.

Уявіть, що користувач вибирає у вікні `OrderWindow` кілька велосипедів і натискає кнопку *Розмістити замовлення*. Як ви пам'ятаєте з шостого розділу, ця дія спричинить виклик методу `actionPerformed()`, який перевірить, чи може замовлення бути виконане.

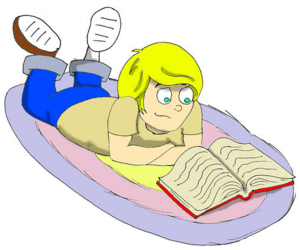
У наступному прикладі коду показано, як метод `checkOrder()` цього

вікна оголошує, що він може викликати виключення `TooManyBikesException`. Якщо замовлення не поміщається у вантажівку, цей метод викликає виключення, яке потім перехоплюється блоком catch. Повідомлення про помилку відображається в текстовому полі вікна.

```
class OrderWindow implements ActionListener{

//Тут потрібно помістити код для створення
//компонентів вікна.
//Користувач натиснув на кнопку Розмістити замовлення
  String selectedModel = txtFieldModel.getText();
  String selectedQuantity = txtFieldQuantity.getText();
  int quantity = Integer.parseInt(selectedQuantity);

  void actionPerformed(ActionEvent e){

    try{
      bikeOrder.checkOrder("FireBird", quantity);
      //Наступний рядок не виконується у разі виключення
      txtFieldOrderConfirmation.setText(
               "Розміщення вашого замовлення завершено");
    }catch(TooManyBikesException e){
      txtFieldOrderConfirmation.setText(e.getMessage());
    }
  }

  void checkOrder(String bikeModel, int quantity)
                        throws TooManyBikesException{

  //Напишіть код, який перевіряє, чи поміщається необхідна
  //кількість велосипедів заданої моделі у вантажівку.
  //Якщо не поміщається, зробити наступне:
    throw new TooManyBikesException("Неможливо доставити"
       + quantity + " велосипедів моделі " + bikeModel +
                                    " за одну доставку");
   }
}
```

В ідеальному світі будь-яка програма повинна працювати без проблем. У реальності, ми повинні бути готові до несподіваних ситуацій. Те, що Java змушує писати код, готовий до таких ситуацій, дійсно корисно.

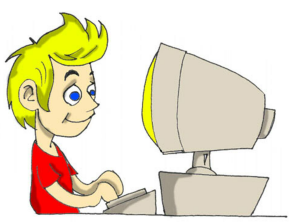
Матеріали для додаткового читання



Обробка помилок за допомогою виключень:

http://download.oracle.com/javase/tutorial/essential/exceptions/index.html

Практичні вправи



1. Створіть Swing-додаток для розміщення замовлень на покупку велосипедів. Він повинен містити два текстових поля *Модель велосипеда* і *Кількість*, кнопку *Розмістити замовлення* і повідомлення для підтвердження замовлення.

2. Використовуйте код з прикладів з `OrderWindow` і `TooManyBikesException`. Спробуйте кілька комбінацій моделей велосипедів та кількості, які викликали б виключення.

Практичні вправи для розумників і розумниць

1. Змініть додаток з попереднього завдання, замінивши текстове поле *Модель велосипеда* випадаючим меню з декількома моделями, щоб користувач міг вибрати зі списку, замість того, щоб вводити з клавіатури.

2. Прочитайте в Інтернеті про Swing-компоненти `JComboBox` і `ItemListener` для обробки подій, коли користувач вибирає модель велосипеда.

Розділ 9. Збереження рахунку гри

Після того, як програма закінчує роботу, вона вивантажується з його оперативної пам'яті. Це означає, що всі класи, методи та змінні перестають існувати до тих пір, поки програма не буде запущена знову. Щоб зберегти результати роботи програми, їх потрібно записати на жорсткий диск, флешку, або інший пристрій, здатний зберігати дані довгий час. У цьому розділі ви дізнаєтеся, як працювати з даними на диску, використовуючи потоки (*streams*).

Просто кажучи, відкривається потік між програмою і диском (або іншим пристроєм, навіть віддаленим). Якщо потрібно зчитати з диска, це повинен бути *потік введення*, якщо ж необхідно записати дані на диск, то *потік виведення*. Наприклад, якщо гравець завершує гру і необхідно запам'ятати його результат, можна записати його у файл під назвою *scores.txt*, використовуючи потік виведення.

Програма зчитує і записує дані в потік *послідовно*, байт за байтом, символ за символом і т.д. Програма може використовувати різні типи даних, наприклад, `String`, `int`, `double` і інші, тому необхідно вибирати відповідний тип потоку Java, наприклад, потік байтів, потік символів, потік даних.

Класи для роботи з потоками знаходяться в пакетах `java.io.` і `java.nio`.

Незалежно від того, який тип файлової системи буде використаний, необхідно провести три дії:

- Відкрити потік, який вказує на деякий файл.
- Зчитати або записати дані в цей потік.
- Закрити потік.

Байтові потоки

При написанні програми, яка зчитує дані з файлу, а потім відображає їх на екрані, необхідно знати, дані якого типу в цьому файлі містяться. З іншого боку, програма, яка просто копіює файли з одного місця в інше, може навіть не знати, що в них - зображення, текст або ж музика. Така програма зчитує первісний файл в оперативну пам'ять, а потім записує його в папку призначення байт за байтом, використовуючи класи Java `FileInputStream` або `FileOutputStream`.

Наступний приклад показує, як використовувати клас `FileInputStream` для читання графічного файлу *abc.gif* з директорії *c:\practice*. В операційній системі Microsoft Windows, для уникнення плутанини зі спеціальними символами Java, які починаються з зворотного слеша, для розділення назв директорій і файлів слід використовувати подвійний слеш: *c:\\practice*. Ця невелика програма виводить на екран не зображення, а всього лише цифри, що показують, в якому закодованому вигляді воно зберігається на диску. Кожному байту відповідає позитивне значення від 0 до 255. Клас `ByteReader` виводить ці значення, розділені пропуском.

Зверніть увагу, що клас `ByteReader` закриває потік в блоці `finally`. Ніколи не викликайте метод `close()` всередині блоку `try/catch` відразу після завершення читання з файлу, робіть це в блоці `finally`. У разі виникнення виключення при читанні файлу, виконання переступить через викреслений виклик `close()` і потік не буде закрито! Читання припиняється, коли метод `FileInputStream.read()` повертає негативне значення.

```
import java.io.FileInputStream;
import java.io.IOException;

public class ByteReader {

  public static void main(String[] args) {

    FileInputStream myFile = null;
```

```
    try {
      //Відкриття байтового потоку, що вказує на файл
      myFile = new FileInputStream("c:\\temp\\abc.gif");

      while(true) {
        int intValueOfByte = myFile.read();
        System.out.print(" " + intValueOfByte);
          if (intValueOfByte  == -1){
            //досягнуто кінець файлу потрібно вийти з циклу
            break;
          }
      } //кінець циклу while

      // myFile.close(); не розміщуйте цей виклик тут
    }catch (IOException e) {
            System.out.println("Неможливо прочитати файл: "
                                              + e.toString());
    }finally{
        try{
            myFile.close();
        }catch (Exception e1){
                e1.printStackTrace();
        }

        System.out.println("Читання файлу завершено
                                                  успішно");
    }
  }
}
```

Наступний фрагмент коду записує кілька байт, представлених цілочисельними значеннями, у файл *xyz.dat*, використовуючи клас `FileOutputStream`:

```
int somedata[]= {56,230,123,43,11,37};

  FileOutputStream myFile = null;

  try {
    //Відкривається файл xyz.dat, в який
    //записуються дані з масиву
    myFile = new  FileOutputStream("xyz.dat");
    for (int i = 0; i <some data.length; i++){
      file.write(data[i]);
    }
  }catch (IOException e) {
    System.out.println("Неможливо записати дані у файл: "
                                          + e.toString());
  }finally{
    try{
      myFile.close();
    }catch (Exception e1){
      e1.printStackTrace();
    }
  }
```

Буферизовані потоки

Отже, зараз ми зчитуємо і записуємо файли побайтно. Це означає, що для читання файлу довжиною в 1000 байт, програма `ByteReader` повинна буде звернутися до диску 1000 разів. Але робота з даними на диску відбувається значно повільніше, ніж в оперативній пам'яті. Для мінімізації кількості звернень до диску, Java надає так звані буфери, щось на кшталт "резервуарів для даних".

Клас `BufferedInputStream` дозволяє швидко заповнити буфер в пам'яті даними з `FileInputStream`. Буферізірованний потік зчитує велику порцію байтів з файлу в пам'ять за раз, після чого метод `read()` отримує окремі байти з буфера набагато швидше.

У програмі можна з'єднувати потоки, так само, як водопровідник з'єд-

нує труби. Давайте трохи змінимо приклад, який зчитує файл. Спочатку, дані з потоку `FileInputStream` будуть надходити в `BufferedInputStream`, а потім - методу `read()`:

```
FileInputStream myFile = null;
BufferedInputStream buff = null;

try {
  myFile = new  FileInputStream("abc.dat");

  //з'єднуємо потоки FileInputStream і BufferedInputStream
  buff = new BufferedInputStream(myFile);
  while (true) {
    int byteValue = buff.read();
    System.out.print(byteValue + " ");

    if (byteValue  == -1)
      break;
  }

} catch (IOException e) {
  e.printStackTrace();

}finally{
  try{
    buff.close();
    myFile.close();
  } catch(IOException e1){
    e1.printStackTrace();
  }
}
```

А якого розміру такий буфер? Це залежить від версії JVM, наприклад 600 байт, але його можна встановити і вручну. Наприклад, для встановлення розміру буфера в 5000 байтів, використовуйте конструктор з двома аргументами:

```
BufferedInputStream buff = new BufferedInputStream(myFile, 5000);
```

Буферизовані потоки не змінюють спосіб читання, а просто прискорюють його.

`BufferedOutputStream` діє аналогічно, але тільки він не читає, а пише в буфер, використовуючи клас `FileOutputStream`.

```
int somedata[]= {56,230,123,43,11,37};

FileOutputStream myFile = null;
BufferedOutputStream buff = null;

try {
  myFile = new FileOutputStream("abc.dat");

  // з'єднуємо потоки
  buff = new BufferedOutputStream(myFile);

  for (int i = 0; i <somedata.length; i++){
    buff.write(somedata[i]);
  }

} catch (IOException e) {
  e.printStackTrace();

}finally{
  try{
    buff.flush();
    buff.close();
    myFile.close();
  } catch(IOException e1){
    e1.printStackTrace();
  }
}
```

Щоб бути впевненим, що всі байти з буфера були передані у файловий потік, після закінчення запису в BufferedOutputStream, викличте метод `flush()`.

Аргументи командного рядка

Програма `ByteReader` зберігає ім'я файлу abc.gif прямо в своєму коді, або, як кажуть програмісти, ім'я файлу жорстко зашито (на сленгу захардкодовано) у програму. Це означає, що для отримання схожої програми, яка б зчитувала файл xyz.gif, потрібно було б змінити код і перекомпілювати його, що не дуже зручно. Було б набагато краще передавати ім'я файлу в командному рядку, при запуску програми.

Будь-яку Java-програму можна запустити з аргументами командного рядка, наприклад:

```
java ByteReader xyz.gif
```

Тут методу `main()` передається всього один аргумент - xyz.gif. Якщо пам'ятаєте, у методу `main()` є аргумент:

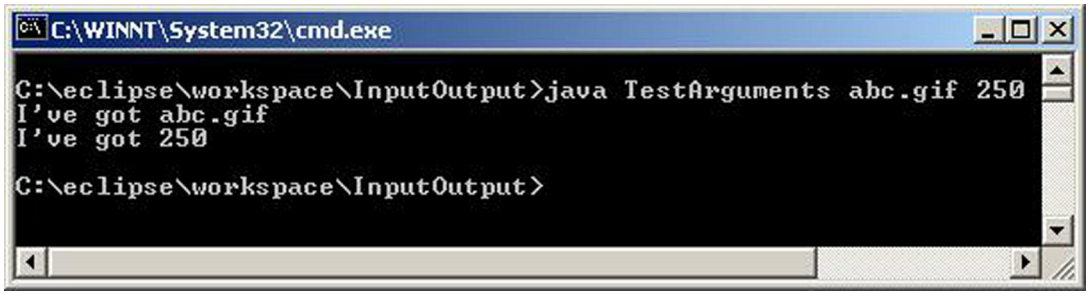
```
public static void main(String[] args) {…}
```

Так, це масив елементів типу String, який JVM передає в головний метод. Якщо запустити програму без аргументів, цей масив залишиться порожнім. Інакше, масив буде мати стільки елементів, скільки аргументів командного рядка було передано програмі.

Давайте подивимося, як ці аргументи командного рядка можна використовувати в дуже простому класі, який просто виведе їх на екран:

```
public class TestArguments {

  public static void main(String[] args) {

    //Скільки отримано аргументів?
    int numberOfArgs = args.length;

    for (int i=0; i<numberOfArgs; i++){
      System.out.println("I've got " + args[i]);
    }
  }
}
```

На наступному скріншоті показано, що станеться, якщо запустити програму з двома аргументами - xyz.gif і 250. JVM помістить значення xyz.gif в елемент `args[0]`, а другий аргумент потрапить у `args[1]`.



Аргументи командного рядка завжди передаються в програму як рядки (String). Програма самостійно повинна конвертувати дані у відповідний формат, наприклад:

```
int myScore = Integer.parseInt(args[1]);
```

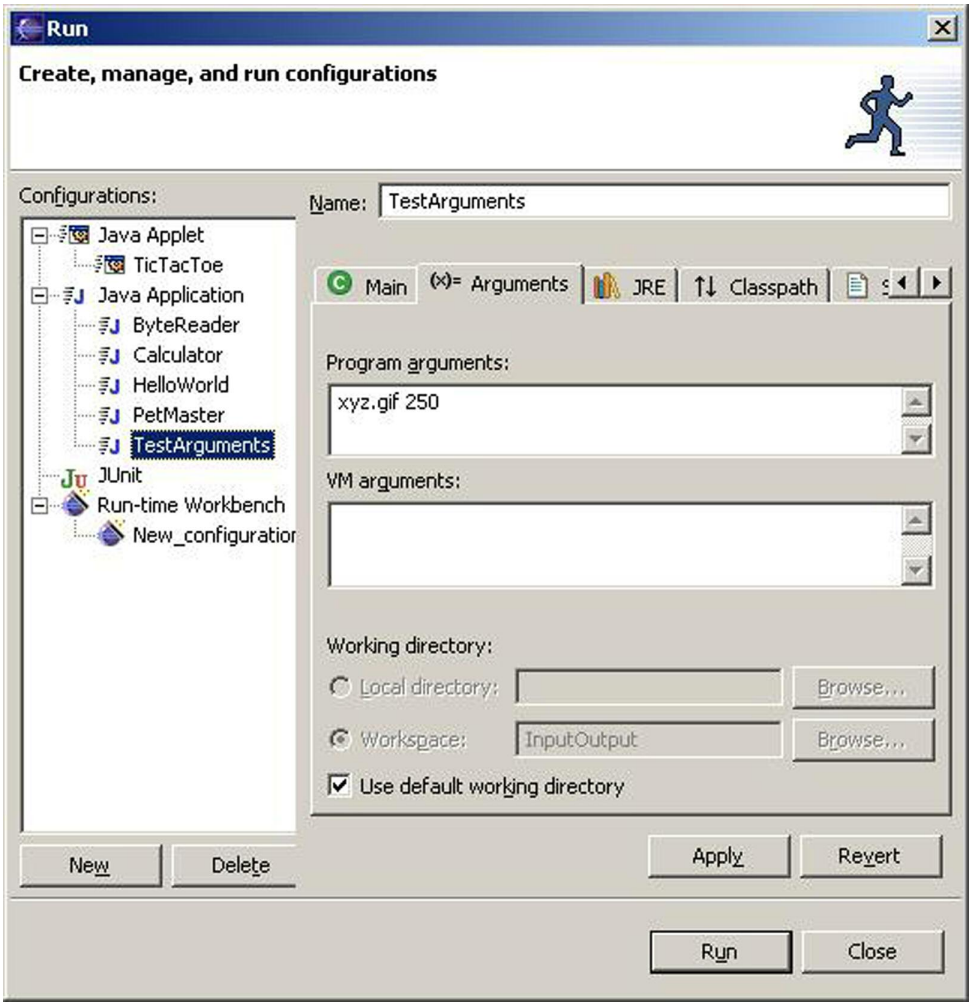
Раджу перевіряти, чи правильне число аргументів передано в команд-

ному рядку. Робіть це прямо на початку методу `main()`. Якщо кількість аргументів не відповідає очікуваному, то програма повинна видати коротке повідомлення про це і негайно закінчити роботу, використовуючи спеціальний метод `System.exit()`:

```
public static void main(String[] args) {

  if (args.length != 2){
    System.out.println("Будь ласка, передайте параметри,
                                                  наприклад:");
    System.out.println("java TestArguments xyz.gif 250");

    //Вихід з програми
    System.exit(0);
  }
}
```

Наприкінці цього розділу вам належить написати програму для копіювання файлів. Для того, щоб програма працювала з будь-якими файлами, імена вихідного і кінцевого файлів повинні передаватися в програму через аргументи командного рядка.

У Eclipse, з метою тестування, теж можна вказувати аргументи командного рядка для всіх програм, що запускаються. У вікні *Run Configurations*, виберіть закладку з написом `(x) = Arguments` і введіть необхідні аргументи у поле *Program Arguments*.



Поле VM arguments дозволяє вказати аргументи для JVM. Це дозволить, наприклад, запросити більше пам'яті для виконання програми, налаштувати саму JVM, тощо. У розділі *Матеріали для додаткового читання* є посилання на сайт з більш докладним описом цих параметрів.

Читання текстових файлів

Java використовує двобайтні символи для зберігання літер, а класи `FileReader` і `FileWriter` призначені для зручної роботи з текстовими файлами. Ці класи можуть зчитувати файли посимвольно, використовуючи

метод `read()`, або ж порядково, за допомогою методу `readLine()`. У класів `FileReader` і `FileWriter` також є аналоги, `BufferedReader` і `BufferedWriter`, що дозволяють прискорити роботу з файлами.

Наступний клас `ScoreReader` зчитує вміст файлу `scores.txt` рядок за рядком. Виконання програми завершується, коли метод `readLine()` повертає `null`, що вказує на кінець файлу.

Використовуючи будь-який текстовий редактор, створіть файл `c:\scores.txt`, що містить такі чотири рядки:

David 235
Brian 190
Anna 225
Zachary 160

Запустіть програму ScoreReader, яка виведе вміст цього файлу на екран. Додайте ще кілька рядків з результатами ігор. Тепер запустіть програму знову, щоб пересвідчитися, що додані рядки теж будуть виводитися на екран.

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class ScoreReader {

  public static void main(String[] args) {

    FileReader myFile = null;
    BufferedReader buff = null;

    try {
      myFile = new FileReader("c:\\scores.txt");
      buff = new BufferedReader(myFile);
```

```
      while (true) {

        //зчитується рядок з файлу scores.txt
        String line = buff.readLine();

        //перевірка досягнення кінця файлу
        if (line == null) break;
          System.out.println(line);кінець циклу
      } //кінець циклу while

    }catch (IOException e){
      e.printStackTrace();

    }finally {
      try{
        buff.close();
        myFile.close();
      }catch(IOException e1){
        e1.printStackTrace();
      }
    }
  } //кінець методу main
}
```

Якщо програма повинна записувати текстові дані на диск, використовуйте один з перевантажених методів `write()` класу `FileWriter`. Ці методи дозволяють записати у файл символ, рядок `String` або цілий масив символів.

У класу `FileWriter` є кілька перевантажених (overloaded) конструкторів. Якщо відкрити файл, задавши тільки його ім'я, то він буде перезаписаний наново при кожному запуску програми:

```
FileWriter fOut = new FileWriter("Scores.txt");
```

Якщо потрібно дозаписати дані у вже існуючий файл, використовуйте конструктор з двома аргументами (`true` означає режим додавання, якщо файл існує):

```
FileWriter fOut = new FileWriter("Scores.txt", true);
```

У наступному прикладі, клас `ScoreWriter` записує три рядки з масиву scores в файл `c:\scores.txt`.

```
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;

public class ScoreWriter {

  public static void main(String[] args) {

    FileWriter myFile = null;
    BufferedWriter buff = null;
    String[] scores = new String[3];

    //заповнення масиву результатами гри
    scores[0] = "Mr. Smith 240";
    scores[1] = "Ms. Lee 300";
    scores[2] = "Mr. Dolittle 190";

    try {
      myFile = new FileWriter("c:\\scores2.txt");
      buff = new BufferedWriter(myFile);
      for (int i=0; i < scores.length; i++) {
        //запис рядків з масиву у файл scores2.txt
        buff.write(scores[i]);
        System.out.println("Записується  " + scores[i] );
      }
      System.out.println("Запис файлу завершено");
```

```
    } catch (IOException e){
      e.printStackTrace();
    } finally {
      try {
        buff.flush();
        buff.close();
        myFile.close();
      } catch(IOException e1){
        e1.printStackTrace();
      }
    }

  } //кінець методу main
}
```

Виведення програми буде виглядати ось так:

```
Записується  Mr. Smith 240
Записується  Ms. Lee 300
Записується  Mr. Dolittle 190
Запис файлу завершено
```

Клас File

Клас `java.io.File` містить безліч зручних методів, які дозволяють перейменувати файл, видалити файл, перевірити існування файлу і т.д. Припустимо, програма зберігає деякі дані у файл і потрібно видати користувачеві попередження, якщо такий файл вже існує. Для цього, необхідно створити екземпляр класу `File`, вказавши ім'я файлу, а потім викликати метод `exists()`. Якщо метод поверне `true`, значить файл abc.txt знайдений на диску і необхідно вивести попередження, інакше такий файл ще не існує:

```
File aFile = new File("abc.txt");

if (aFile.exists()){
   //Сюди йде код для виведення в консоль або через
   //JOptionPane для відображення попередження у віконці
}
```

Конструктор класу `File`, *насправді, не створює файл* - він просто створює в пам'яті екземпляр цього класу, який вказує на реальний файл. Якщо дійсно потрібно створити файл, використовуйте для цього метод `createNewFile()`.

Нижче наведені деякі корисні методи класу `File`.

Им'я методу	**Прзначення методу**
`createNewFile()`	Створює новий порожній файл з ім'ям, вказаним при створенні об'єкта типу File. Новий файл створюється тільки, якщо він ще не існує.
`delete()`	Видаляє файл або директорію
`renameTo()`	Перейменовує файл
`length()`	Повертає розмір файла в байтах
`exists()`	Повертає `true`, якщо файл існує
`list()`	Повертає масив рядків з іменами файлів/директорій, що містяться у вказаній директорії
`lastModified()`	Повертає час останньої зміни файлу
`mkDir()`	Створює директорію

Наступний фрагмент коду перейменовує файл `customers.txt` в `customers.txt.bak`. Якщо файл. Bak вже існує, він буде перезаписаний.

```
File file = new File("customers.txt");
File backup = new File("customers.txt.bak");

if (backup.exists()){
  backup.delete();
}

file.renameTo(backup);
```

У версії Java 7 з'явився новий клас `Files`, який спрощує створення, видалення та копіювання файлів. На відміну від класу `File`, `Files` створює або видаляє реальні файли на диску, а не в пам'яті. А новий клас `Path` - це програмне подання повного імені файлу, незалежно від операційної системи користувача. Ось як це виглядає для файлу Customers.txt під Windows і на MacOS:

```
Path pathCustomers=
FileSystems.getDefault().getPath(“.”,”c\\Customers.txt”);

Path pathCustomers=
FileSystems.getDefault().getPath(“.”,”/Customers.txt”);
```

А так можна читати файл до колекції рядків:

```
List customers=
  Files.readAllLines(pathCustomers,Charset.defaultCharset());
```

Так само можна використовувати буфер:

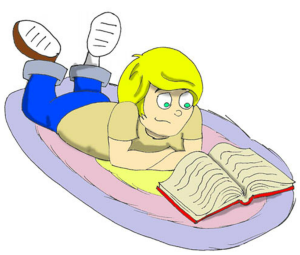
```
Reader reader=
  Files.newBufferedReader(pathCustomers, Charset.defaultCharset());
```

А створити новий файл *Customers.txt* можна так:

```
Path fileName=Paths.get("c:\\Customers.txt");
Path customers=Paths.createFile(fileName);
```

Цей розділ був присвячений тільки роботі з файлами на диску вашого комп'ютера, але Java дозволяє створювати потоки, що вказують на інші комп'ютери в мережі. Такі комп'ютери можуть перебувати досить далеко один від одного. Наприклад, NASA використовувала Java для управління марсоходами. Я впевнений, що для цього вони просто спрямували свої потоки на Марс. ☺

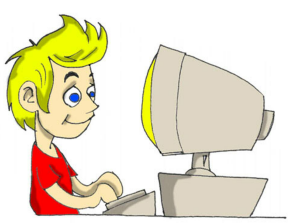
Матеріали для додаткового читання



1. Параметри командного рядка JVM
http://download.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html

2. Використання файлових потоків
http://docs.oracle.com/javase/tutorial/essential/io/streams.html

Практичні вправи



Напишіть програму копіювання файлів під назвою `FileCopy`, об'єднавши для цього фрагменти коду з розділу про байтові потоки.
Відкрийте два потоки (введення і виведення) і викличте методи `read()` і `write()` *в одному і тому ж циклі*. Використовуйте аргументи командного рядка для передачі програмі імен вихідного і кінцевого файлів, наприклад:

```
java FileCopy c:\\temp\\scores.txt
c:\\backup\\scores2.txt
```

Практичні вправи для розумників і розумниць

Створіть Swing-додаток, що дозволяє користувачам вибирати імена файлів для копіювання, використовуючи клас JFileChooser, який створює стандартне вікно вибору файлу. Це вікно має відкриватися при натисканні користувачем однієї з кнопок Browse. Потрібно буде додати кілька рядків коду для відображення обраного імені файлу в текстовому полі.

Після того, як користувач натискає кнопку Copy, код в методі actionPerformed () повинен скопіювати вибраний файл. Спробуйте заново використовувати код з попередніх завдань, але без прямого копіювання / вставки.

Розділ 10. Різні корисні штучки

У нас була можливість використовувати різні елементи мови Java в попередніх розділах. Ми навіть створили гру “Хрестики-нулики”. Однак, я упустив деякі важливі прийоми і елементи Java і саме час розповісти про них.

Робота з датами і часом

У кожному комп'ютері є внутрішній годинник. Будь-яка програма Java може дізнатися поточні дату і час, і відобразити їх у різних форматах. Наприклад, 15.06.2011 або 15 липня 2011 року. У Java є безліч класів, які працюють з датами. Але два з них - `java.util.Date` і `java.text.SimpleDateFormat` - охоплюють велику частину ваших потреб при роботі з датами і часом.

Дуже просто створити об'єкт, який зберігає поточну системну дату і час з точністю до мілісекунд:

```
Date today = new Date();
System.out.println("Дата: " + today);
```

Вихідні дані такого фрагмента коду можуть виглядати наступним чином:

```
Дата: Sat Oct 08 20:41:44 MSD 2011
```

Клас `SimpleDateFormat` дозволяє відобразити дату і час в різних форматах. Перше, вам необхідно створити екземпляр даного класу з необхідним форматом. Потім викликати у нього метод `format()`, якому як аргумент слід передати об'єкт `Date`. Наступна програма виконує форматування та друк поточної дати в декількох форматах.

```
import java.util.Date;
import java.text.SimpleDateFormat;

public class MyDateFormat {

  public static void main( String [] args ) {

    //Створюється об'єкт Date
    //і виконується друк у форматі за замовчуванням
    Date today = new Date();
    System.out.println("Дата: " + today);

    //Формат, який виводить дату у вигляді 10-08-11
    SimpleDateFormat sdf= new SimpleDateFormat("MM-dd-yy");
    String formattedDate=sdf.format(today);
    System.out.println("Дата(мм-дд-рр): " + formattedDate);

    //Формат, який виводить дату у вигляді 08-10-2011
    sdf = new SimpleDateFormat("dd-MM-yyyy");
    formattedDate=sdf.format(today);
    System.out.println("Дата(дд-мм-рррр): "+formattedDate);

    //Формат, який виводить дату у вигляді Пт, жов 27, '11
    sdf = new SimpleDateFormat("EEE, MMM d, ''yy");
    formattedDate=sdf.format(today);
    System.out.println("Дата (день тижня, міс д, 'рр) " +
                                          formattedDate);

    //Формат, який виводить дату у вигляді 07:18:51 AM
    sdf = new SimpleDateFormat("hh:mm:ss a");
    formattedDate=sdf.format(today);
    System.out.println("Час (гг:хх:сс) " + formattedDate);
  }
}
```

Відкомпілюйте і запустіть клас MyDateFormat, в результаті він виведе на екран щось схоже на:

```
Дата: Sat Oct 08 20:54:37 MSD 2011
Дата(мм-дд-рр): 10-08-11
Дата(дд-мм-рррр): 08-10-2011
Дата (день тижня, міс д, 'рр) Сб, жов 8, '11
Час (гг:хх:сс) 08:54:37 PM
```

У документації Java для класу `SimpleDateFormat` описані інші формати. Додаткові методи, що працюють з датами, можна знайти в іншому класі Java на ім'я `java.util.Calendar`.

Перевантаження методів

Клас може містити декілька методів з однаковим ім'ям, але з різними списками аргументів. Таку можливість називають *перевантаженням методів (method overloading)*. Наприклад, метод `println()` класу `System` може бути викликаний з аргументами різного типу: `String, int, char` та ін.

```
System.out.println("Привіт!");
System.out.println(250);
System.out.println('A');
```

Незважаючи на те, що це виглядає як три виклики одного і того ж методу `println()`, фактично ж викликаються три різних методи. Ви можете запитати, чому б не створити методи з різними іменами, наприклад, `printString(),printInt(),printChar()`? Одна з причин в тому, що набагато простіше запам'ятати одне ім'я методу для друку, ніж запам'ятовувати кілька імен. Є й інші причини для використання перевантаження методів, але вони трохи складніші, щоб пояснювати їх тут. Ці причини (polymorfism) потрібно розглядати в більш складних підручниках.

Як ви пам'ятаєте, наш клас `Fish` з розділу 4, містив метод `dive()`, який приймав один аргумент:

```
public int dive(int howDeep)
```

Давайте створимо ще одну версію цього методу, яка не вимагатиме аргументів. Цей метод змусить рибку пірнати на 5 метрів, поки глибина занурення не перевищить 100 метрів. Нова версія класу `Fish` містить `final` змінну `DEFAULT_DIVING` (глибина пірнання за замовчуванням), значення якої буде рівним п'яти метрам.

Тепер клас Fish містить два перевантажених методи `dive()`.

```
public class Fish extends Pet {
     int currentDepth=0;
     final int DEFAULT_DIVING = 5;

     public int dive() {
          currentDepth=currentDepth + DEFAULT_DIVING;
          if (currentDepth > 100){
            System.out.println("Я маленька рибка "+
               " і не можу плавати глибше 100 метрів");
            currentDepth=currentDepth - DEFAULT_DIVING;
          }
          else{
            System.out.println("Занурююся ще на " +
                        DEFAULT_DIVING + " метрів");
            System.out.println("Я на глибині " +
                        currentDepth + " метрів");
          }
          return currentDepth;
     }

     public int dive(int howDeep) {
          currentDepth=currentDepth + howDeep;
          if (currentDepth > 100){
            System.out.println("Я маленька рибка "+
               " і не можу плавати глибше 100 метрів");
            currentDepth=currentDepth - howDeep;
          }
          else{
            System.out.println("Занурююся ще на " +
```

```
                        howDeep + " метрів");
            System.out.println("Я на глибині " +
                        currentDepth + " метрів");
        }
        return currentDepth;
    }
    public String say(String something){
        return "То хіба ви не знаєте, що риби
                                не говорять?";
    }

    // constructor
    Fish(int startingPosition){
      currentDepth=startingPosition;
    }
}
```

Тепер клас `FishMaster` може викликати будь-який з перевантажених методів `dive()`:

```
public class FishMaster {
    public static void main(String[] args) {
        Fish myFish = new Fish(20);
        myFish.dive(2);
        myFish.dive(); //новий перевантажений метод
        myFish.dive(97);
        myFish.dive(3);
        myFish.sleep ();
    }
}
```

Конструктори також можуть бути перевантажені, але при створенні об'єкта буде використовуватися тільки один з них. JVM буде виконувати виклик конструктора з відповідним списком аргументів. Наприклад, якщо в

клас Fish додати конструктор за замовчуванням (без аргументів), клас FishMaster зможе створити його примірник за допомогою одного з таких способів:

```
Fish myFish = new Fish(20);
```
або
```
Fish myFish = new Fish();
```

Читання даних з клавіатури

У цьому розділі ви дізнаєтеся, як програма може друкувати питання в командному вікні і розуміти відповіді, які користувач вводить з клавіатури. Цього разу ми видалимо з класу `FishMaster` всі жорстко задані значення, які він передає класу `Fish`. Тепер програма буде задавати питання: «*На яку глибину?*», а риба буде занурюватися відповідно до відповіді користувача.

Ви вже багато разів користувалися *стандартним виведенням даних* `System.out`. Між іншим, змінна `out` має тип `java.io.OutputStream`. Зараз я вам поясню, як працювати зі *стандартним виведенням даних* `System.in`. Як ви, напевно, здогадалися, змінна `in` має тип `java.io.InputStream`.

Наступна версія класу `FishMaster` виводить в системну консоль *рядок введення* і чекає відповіді від користувача. Після того, як користувач введе один або кілька символів і натисне клавішу Enter, JVM розміщує ці символи в об'єкт класу `InputStream` і передає їх програмі.

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class FishMaster {

  public static void main(String[] args) {
```

```
    Fish myFish = new Fish(20);
    String feetString="";
    int feet;

    //Створюємо обробник читання вхідного потоку
    //InputStreamReader, який підключений до System.in
    //і передаємо його буферизованому обробнику читання
    //BufferedReader
    BufferedReader stdin = new BufferedReader
                        (new InputStreamReader(System.in ));

    //Занурюємося кілька разів поки користувач не натисне
    //клавішу "Q"
    while(true) {
      System.out.println("Готова до занурення. На яку глиину?");
      try {
        feetString = stdin.readLine();
        if (feetString.equals("Q")){
          //Вихід з програми
          System.out.println("До зустрічі!");
          System.exit(0);
        }else {
          //Перетворимо feetString в ціле число
          //і занурюємося на глибину,
          //яка визначається змінною feet
          feet = Integer.parseInt(feetString);
          myFish.dive(feet);
        }
      } catch (IOException e) {
        e.printStackTrace();
      }
    } //Кінець while
  } //Кінець main
}
```

Діалог між користувачем і програмою FishMaster може виглядати як:

```
Готова до занурення. На яку глибину?
14
Занурююся на 14 м.
Я на 34 м. нижче рівня моря
Готова до занурення. На яку глибину?
30
Занурююся на 30 м.
Я на 64 м. нижче рівня моря
Готова до занурення. На яку глибину?
Q
До зустрічі!
```

Спочатку клас `FishMaster` створює потік `BufferedReader`, підключений до стандартного вхідного потоку `System.in` . Після цього він відображає повідомлення *«Готова до занурення. На яку глибину?»* І метод `readLine()` призупиняє виконання програми, поки користувач не натисне клавішу Enter. Введене значення матиме тип `String`, тому клас `FishMaster` перетворює це значення в тип `int` і викликає метод `dive()` класу `Fish`. Дана дія виконується в циклі, поки користувач не введе символ *Q*, щоб вийти з програми. Рядок `feetString.equals("Q")` порівнює значення змінної `feetString` типу `String` і символ *Q*.

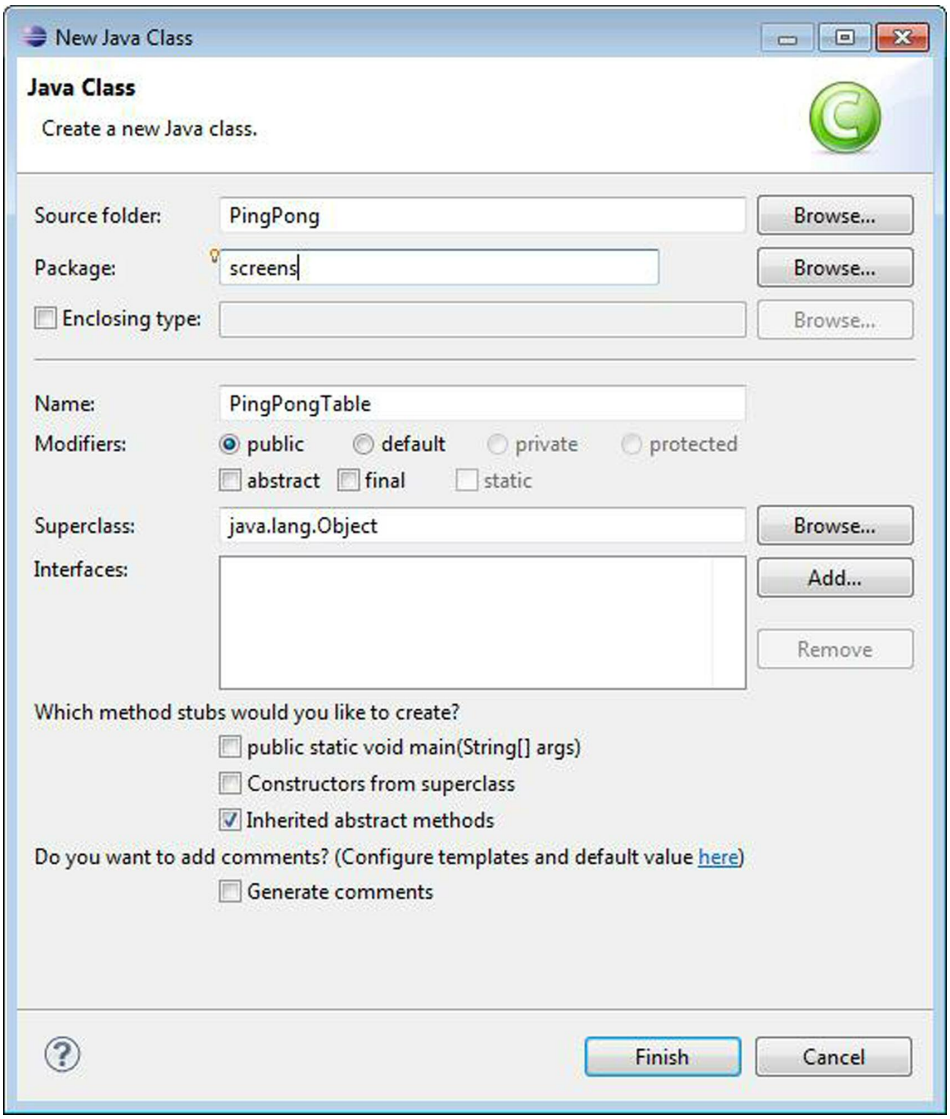
Щоб повністю отримати рядок, введений користувачем, за одну операцію ми використовували метод `readLine()`. Однак є й метод `System.in.read()`, який дозволяє обробляти дані, що вводяться користувачем, посимвольно.

Тобі пакет

Коли програмісти працюють над великим проектом, який містить величезну кількість класів, вони, як правило, групують їх у різних пакетах. Наприклад, один пакет може містити всі класи, які відображають вікна (форми),

а інший пакет може містити обробники подій. Мова Java також зберігає свої класи в пакетах. Наприклад, у пакеті `java.io` містяться класи, які відповідають за обробку операцій введення/виведення. У пакеті `javax.swing` містяться класи графічних компонентів Swing.

Давайте створимо в Eclipse новий проект під назвою *PingPong*. Цей проект буде містити класи у двох пакетах: `screens` і `engine`. Тепер створіть новий клас `PingPongTable` і в полі *Package* введіть назву пакета `screens`:

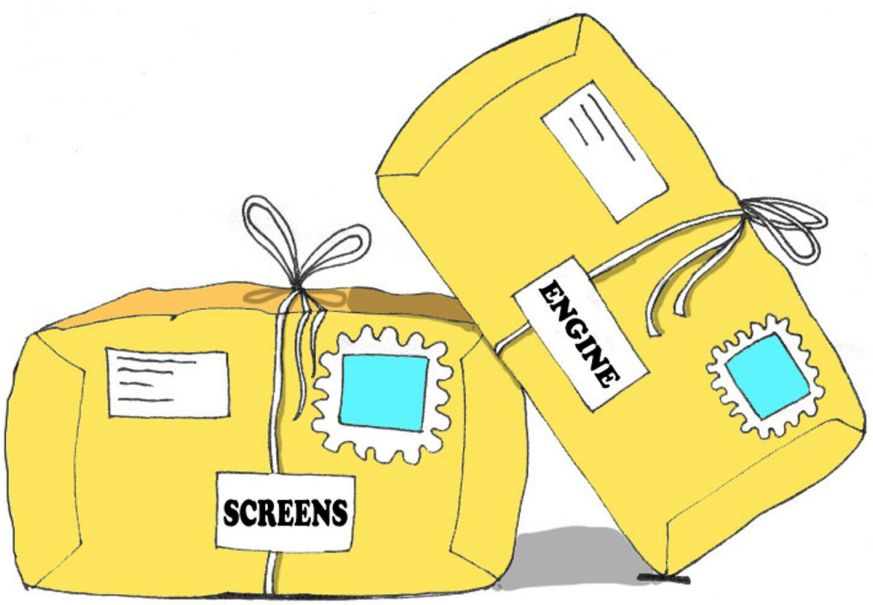


Натисніть кнопку *Finish* ("Готово") і Eclipse згенерує код, який буде містити рядок з ім'ям пакета.

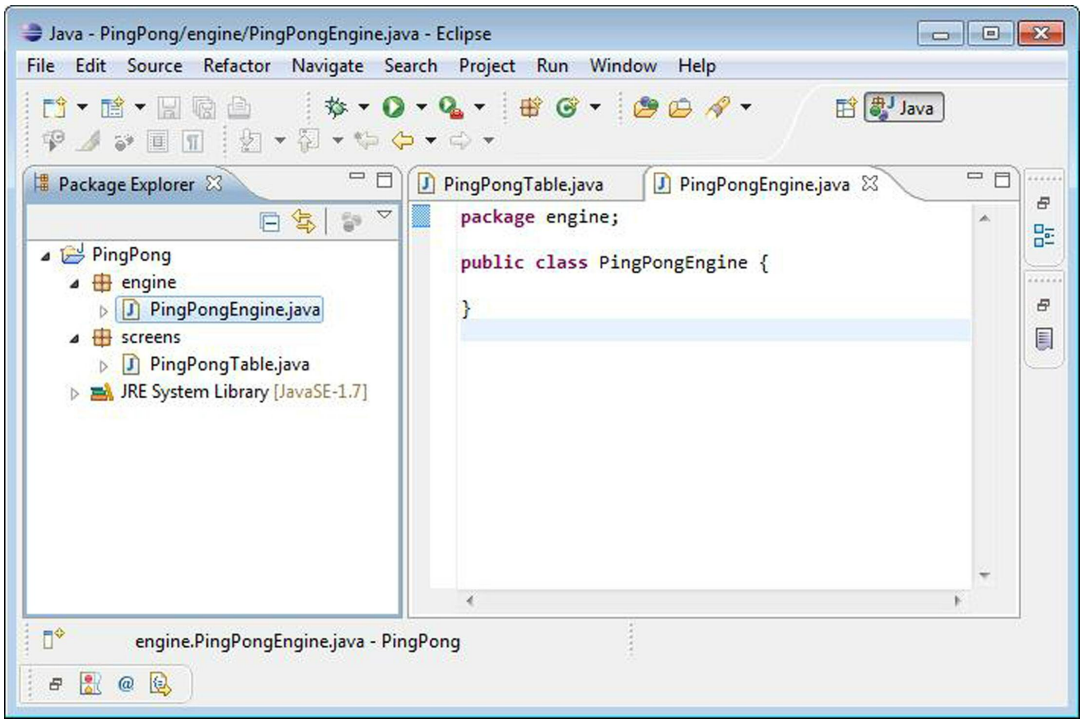
```
package screens;

public class PingPongTable {
     public static void main(String[] args) {

     }
}
```

До речі, якщо у вашому класі є рядок з ключовим словом `package`, вище цього рядка дозволяється розміщувати тільки коментарі і нічого більше.

Оскільки кожен пакет зберігається в окремому каталозі на диску, Eclipse створить каталог screens і розмістить в ньому файл *PingPongTable.java*. Перевірте - на диску повинен бути каталог *c:\eclipse\workspace\PingPong\screens* з файлами *PingPongTable.java* і *PingPongTable.class*.



Тепер створіть інший клас з назвою `PingPongEngine`, а в якості імені пакета введіть `engine`. Проект `PingPong` тепер містить два пакети:



Оскільки наші два класи розташовані в двох різних пакетах (і, відповідно, директоріях), клас `PingPongTable` не бачитиме `PingPongEngine`, поки не додати в нього вираз `import`.

```
package screens;

import engine.PingPongEngine;

public class PingPongTable {

  public static void main(String[] args) {
     PingPongEngine gameEngine = new PingPongEngine();
  }
}
```

Пакети Java допомагають не тільки структурувати ваші класи. Їх можна використовувати для обмеження доступу до класів в пакеті для зовнішніх класів, які розташовуються в інших пакетах.

Рівні доступу

Класи Java, методи й змінні класу можуть мати такі рівні доступу: `public`, `private`, `protected` і *package*. Наш клас `PingPongEngine` має рівень доступу `public`. Це означає, що у будь-якого класу є доступ до нього. Давайте проведемо простий експеримент - видалимо ключове слово `public` з оголошення класу `PingPongEngine`. Тепер клас `PingPongTable` не компілюватиметься, вказуючи на помилки *PingPongEngine cannot be resolved to a type* (Неможливо визначити тип PingPongEngine) і *The type engine.PingPongEngine is not visible* (Тип даних `engine.PingPongEngine` невидимий). Це означає, що клас `PingPongTable` не бачить більше клас `PingPongEngine`.

Якщо рівень доступу не вказується явно, то мається на увазі рівень доступу *package*. Це означає, що клас буде доступний тільки для класів, які знаходяться в одному з ним пакеті (директорії).

Так само, якщо ви забудете дати доступ до методів класу PingPongEngine, клас PingPongTable також вкаже вам на це, повідомивши, що ці методи для нього невидимі. У наступному розділі в процесі створення гри в пінг-понг ви побачите, як використовуються рівні доступу.

Рівень доступу `private` використовується для приховування методів або змінних класу від зовнішнього світу. Уявіть собі автомобіль. Велика частина людей поняття не мають, яка кількість деталей знаходиться у нього під капотом, а також про те, що реально відбувається, коли водій натискає на педаль гальма.

Погляньте на наступний фрагмент коду. У мові Java ми можемо сказати, що об'єкт `Car` показує тільки один `public` метод - `brake()`, усередині якого можуть бути викликані кілька інших методів, знати про які водієві немає ніякої необхідності. Наприклад, якщо водій надто сильно натискає на педаль гальма, комп'ютер автомобіля може включити спеціальні антиблокувальні гальма. Я вже згадував раніше, що програми на Java керують такими складними роботами, як марсіанські всюдиходи, не кажучи вже про прості автомобілі.

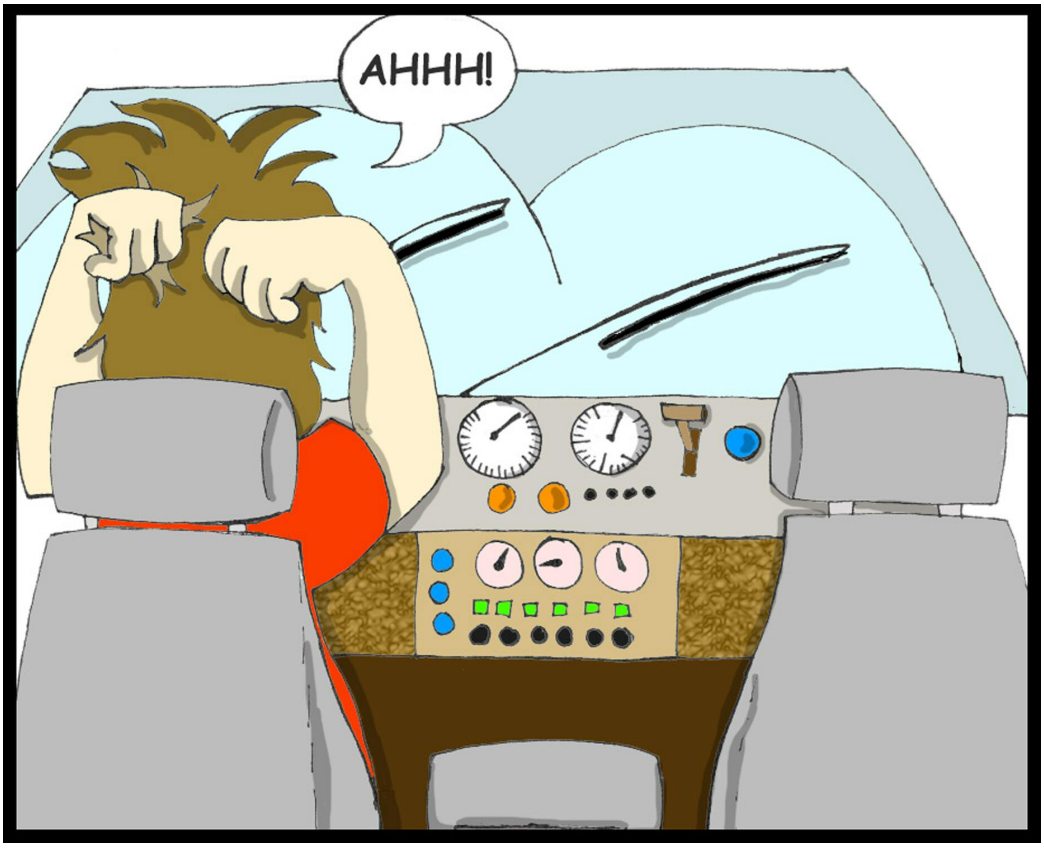
```
public class Car {

  //Ця private змінна може використовуватись
  //тільки всередині класу
  private String brakesCondition;

  //public метод brake() викликає private методи,
  //щоб вирішити, які гальма використовувати
  public void brake(int pedalPressure){
    boolean useRegularBrakes;
    useRegularBrakes=checkForAntiLockBrakes(pedalPressure);

    if (useRegularBrakes==true){
      useRegularBrakes();
    }else{
      useAntiLockBrakes();
    }
  }

  //Цей private метод, що перевіряє гальма з авто-блокуванням
  //може бути викликаний тільки всередині цього класу
```

```
  private boolean checkForAntiLockBrakes(int pressure){
    if (pressure > 100){
      return true;
    }else {
       return false;
    }
  }

  //Цей private метод може бути викликаний
  //тільки всередині цього класу
  private void useRegularBrakes(){
    //тут буде код, що посилає сигнал звичайним гальмам
  }

  //Цей private метод може бути викликаний
  //тільки всередині цього класу
  private void useAntiLockBrakes(){
    //код, що посилає сигнал антиблокувальним гальмам
  }
}
```

Є ще одне ключове слово в Java - `protected` - яке також встановлює рівень доступу. Якщо ви використовуєте це ключове слово в сигнатурі методу, то цей метод буде доступний в класах-спадкоємців, а також в інших класах, які розташовані в цьому ж пакеті. Однак він не буде доступний для класів, які знаходяться в інших пакетах.

Одне з важливих властивостей об'єктно-орієнтованих мов називається ***інкапсуляцією***. Це означає здатність ховати і захищати елементи класу від доступу з інших класів.

У процесі розробки класу приховуйте методи й змінні класу, які не повинні бути видимі в зовнішньому світі. Якщо конструктор автомобіля не приховає управління частиною внутрішніх операцій, то водій зіткнеться з необхідністю взаємодії з сотнями кнопок, перемикачів та приладів.



У наступному розділі ви зможете знайти клас `Score`, який приховує свої властивості в змінних з доступом `private`.

Повертаємося до масивів

У розділі 9 програма `ScoreWriter` створювала масив об'єктів `String` і зберігала імена і бали гравців у файл. Настав час дізнатися, як використовувати масиви для зберігання будь-яких об'єктів.

Цього разу для представлення рахунку у грі ми створимо об'єкт. Цей об'єкт буде містити такі атрибути, як ім'я та прізвище гравця, рахунок і останню дату гри.

Нижче представлений клас `Score`. Він містить спеціальні методи для читання (*getter*) і запису (*setter*) кожного з свого атрибутів, який оголошений з модифікатором доступу *private*. Напевно, може здатися неочевидним, чому

викликаючий клас просто не встановлює значення атрибуту безпосередньо, наприклад, так:

```
Score.score = 250;
```

Замість цього він це робить так:

```
Score.setScore(250);
```

А що, як пізніше ми вирішимо, що наша програма повинна програвати мелодію, коли гравець досягає рахунку в 500 балів? Якщо в класі Score є метод `setScore()`, все що потрібно зробити, це змінити тільки цей метод. Потрібно додати в нього код, який перевіряє кількість балів і при необхідності програє мелодію.

Викликаючий клас продовжить викликати музичну версію методу `setScore()` точно таким же чином. Якщо викликаючий клас буде встановлювати рахунок безпосередньо, то всі «музичні» зміни потрібно буде реалізувати у викликаючому класі. А якщо буде потрібно використовувати клас у двох різних ігрових програмах? У разі прямої зміни значень атрибутів буде потрібно реалізувати ці зміни у двох викликаючих класах. Але, якби у вас був setter-метод, то зміни були б інкапсульовані в ньому і викликаючі класи міняти не довелося б.

```
import java.util.Date;

public class Score {

  private String firstName;
  private String lastName;
  private int score;
  private Date playDate;
```

```
  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName(){
    return lastName;
  }

  public void setLastName(String lastName){
    this.lastName = lastName;
  }

  public int getScore(){
    return score;
  }

  public void setScore(int score){
    this.score=score;
  }

  public Date getPlayDate(){
    return playDate;
  }

  public void setPlayDate(Date playDate){
    this.playDate=playDate;
  }

//Об'єднуємо всі атрибути в рядок (String)
//І в кінці додаємо символ переведення на новий рядок.
//Цей метод зручно використовувати, якщо викликаючий клас
//Хоче за один раз роздрукувати всі значення, наприклад
//System.out.println(myScore.toString());
```

```
  public String toString(){
     String scoreString = firstName + " " +
         lastName + " " +  score + " " + playDate +
        System.getProperty("line.separator");
     return scoreString;
  }
}
```

Програма `ScoreWriter2` створить екземпляри об'єкта `Score` і призначить значення атрибутам цих екземплярів.

```
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Date;

public class ScoreWriter2 {

  /** Метод main виконує такі дії:
   1. Створює екземпляр масиву
   2. Створює об'єкти Score і заповнює ними масив
   3. Записує рахунок гри у файл
  */
  public static void main(String[] args) {
    FileWriter myFile = null;
    BufferedWriter buff = null;
    Date today = new Date();
    Score scores[] = new Score[3];

    // The player #1
    scores[0]=new Score();
    scores[0].setFirstName("Микола");
    scores[0].setLastName("Смирнов");
    scores[0].setScore(250);
    scores[0].setPlayDate(today);
```

```
    // The player #2
    scores[1]=new Score();
    scores[1].setFirstName("Анна");
    scores[1].setLastName("Єгорова");
    scores[1].setScore(300);
    scores[1].setPlayDate(today);

    // The player #3
    scores[2]=new Score();
    scores[2].setFirstName("Сергій");
    scores[2].setLastName("Данилов");
    scores[2].setScore(190);
    scores[2].setPlayDate(today);

    try {
      myFile = new FileWriter("c:\\scores2.txt");
      buff = new BufferedWriter(myFile);
      for (int i=0; i < scores.length; i++) {
        //Перетворює кожен рахунок у об'єкт String
        //і записує його в scores2.txt
        buff.write(scores[i].toString());
        System.out.println("Запис  " +
                        scores[i].getLastName());
      }
      System.out.println("Запис файлу завершено");
    }catch (IOException e){
      e.printStackTrace();
    }finally {
      try{
        buff.flush();
        buff.close();
        myFile.close();
      }catch(IOException e1){
        e1.printStackTrace();
      }
    }
  }
}
```

Якщо програма намагається звернутися до елементу масиву, який знаходиться за його межами, наприклад, `scores[5].GetLastName()`, то Java викликає виключення `ArrayIndexOutOfBoundsException`.

Клас ArrayList

Пакет `java.util` містить класи, які вельми зручно використовувати у випадку, якщо програмі потрібно зберігати кілька екземплярів (колекцію) деяких об'єктів в пам'яті. Ось деякі з популярних класів колекцій: `ArrayList`, `HashTable`, `HashMap` і `List`. Я покажу вам, як використовувати клас `java.util.ArrayList`.

Недоліком звичайного масиву є те, що вам заздалегідь потрібно знати число елементів у масиві. Пам'ятайте, щоб створити екземпляр масиву, потрібно вказати число між квадратними дужками:

```
String[] myFriends = new String[5];
```

Клас `ArrayList` не має такого обмеження - можна створити екземпляр цієї колекції, не знаючи про те, скільки об'єктів буде в ньому. Просто додавайте стільки елементів, скільки потрібно.

Навіщо тоді використовувати масиви? Давайте завжди використовувати клас `ArrayList`! На жаль, безкоштовний сир буває тільки в мишоловці. За зручність потрібно заплатити - `ArrayList` працює дещо повільніше, ніж звичайний масив. Крім того, в ньому можна зберігати тільки об'єкти, але не можна просто так зберегти набір чисел з типом `int` (їх потрібно загортати в класи-обгортки або користуватися так званим автобоксінгом - autoboxing).

Щоб створити і заповнити об'єкт `ArrayList`, спочатку потрібно створити його екземпляр. Далі створити екземпляри об'єктів, які плануєте в ньому зберігати і додати їх у `ArrayList`, за допомогою його методу `add()`. Нижче представлена невелика програма, яка заповнить об'єкт `ArrayList` об'єктами з типом `String` і виведе вміст отриманої колекції.

```
import java.util.ArrayList;

public class ArrayListDemo {

  public static void main(String[] args) {

    //Створюємо і заповнюємо ArrayList
    ArrayList friends = new ArrayList();
    friends.add("Олена");
    friends.add("Анна");
    friends.add("Микола");
    friends.add("Сергій");

    //Скільки в нього друзів?
    int friendsCount = friends.size();

    //Друкуємо вміст ArrayList
    for (int i=0; i<friendsCount; i++){
      System.out.println("Друг №" + i + " це " +
                                          friends.get(i));
    }
  }
}
```

Ця програма надрукує такі рядки:

```
Друг №0 це Олена
Друг №1 це Анна
Друг №2 це Микола
Друг №3 це Сергій
```

Метод `get()` *витягує* з об'єкта `ArrayList` елемент, який розташовується за вказаним індексом. Оскільки ви можете зберігати в колекції будь-які об'єкти, метод `get()` повертає кожен елемент з типом `Object`. Відповідальність за приведення цього об'єкта до правильного типу даних

лягає на програму. Ми не зобов'язані були робити це в попередньому прикладі тільки тому, що зберігали в колекції friends об'єкти з типом String.

Java автоматично виконує перетворення об'єкта з типом `Object` в об'єкт з типом `String`. Але, якщо ви вирішите зберігати в `ArrayList` інші об'єкти, наприклад, екземпляри класу `Fish`, правильний код для додавання і вилучення конкретних об'єктів `Fish` може виглядати так, як у програмі `FishTank`, яка наводиться далі.

Спочатку ця програма створює кілька екземплярів класу Fish, призначає якесь значення кольору, ваги й поточної глибини занурення і зберігає ці значення в об'єкт ArrayList з ім'ям fishTank. Потім програма отримує об'єкти з колекції, приводить їх тип до типу Fish і друкує значення цих об'єктів.

```
import java.util.ArrayList;

public class FishTank {

  public static void main(String[] args) {
    ArrayList fishTank = new ArrayList();
    Fish theFish;
    Fish aFish = new Fish(20);
    aFish.color = "червону";
    aFish.weight = 2;
    fishTank.add(aFish);
    aFish = new Fish(10);
    aFish.color = "зелену";
    aFish.weight = 5;
    fishTank.add(aFish);
    int fishCount = fishTank.size();
    for (int i=0;i<fishCount; i++){
      theFish = (Fish) fishTank.get(i);
      System.out.println("Впіймав "+ theFish.color +
          " рибу вагою " + theFish.weight + " кг. Глибина:"
          + theFish.currentDepth);
    }
  }
}
```

Нижче наводяться вихідні дані програми FishTank:

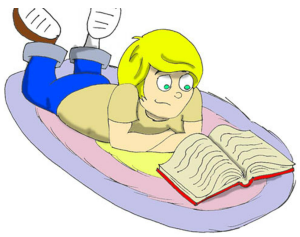
```
Упіймав червону рибу вагою 2.0 кг. Глибина: 20
Упіймав зелену рибу вагою 5.0 кг. Глибина: 10
```

Тепер, коли ви прочитали про рівні доступу в мові Java, можна трохи змінити класи `Pet` і `Fish`. Такі змінні як `age`, `color`, `weight` і `height` можна оголошувати з модифікатором доступу protected, якщо ви думаєте, що хтось захоче наслідувати свої класи з них. А змінна `currentDepth` повинна бути private. Також потрібно додати нові `public` методи, наприклад, `getAge()`, щоб повертати значення змінної `age`, і метод `setAge()`, щоб встановлювати значення цієї змінної.

Програмісти з гарними манерами не дозволяють одному класу безпосередньо змінювати властивості іншого класу. Клас повинен надавати методи, які б змінювали його внутрішні елементи. Ось чому клас `Score` з попереднього розділу був спроектований з private змінними, які могли бути отримані або змінені тільки за допомогою спеціальних getter- і setter-методів для читання і запису відповідно.

У цьому розділі я показав вам різні елементи і прийоми в мові Java, які, на перший погляд, здаються непов'язаними один з одним. Проте всі ці елементи дуже часто використовуються професійними програмістами на Java. Після виконання практичних вправ до цього розділу, ви будете краще розуміти як ці елементи працюють разом.

Матеріали для додаткового читання



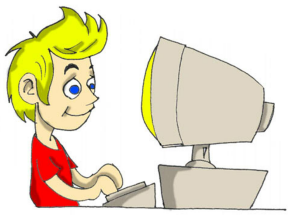
1. Колекції Java:
http://download.oracle.com/javase/tutorial/collections/intro/index.html

2. Клас `ArrayList`:
http://download.oracle.com/javase/7/docs/api/java/util/ArrayList.html

3. Робота с датами:
http://www.vogella.de/articles/JavaDateTimeAPI/article.html

4. Клас `Calendar`:
http://download.oracle.com/javase/7/docs/api/java/util/Calendar.html

Практичні вправи



1. Додайте перевантажений конструктор за замовчуванням (без аргументів) в клас `Fish`. Цей конструктор повинен встановлювати значення стартової позиції рівне 10 метрам. Клас `FishMaster` створюватиме екземпляр об'єкта `Fish` дуже просто, ось так:

```
Fish myFish = new Fish();
```

2. Додайте в клас `Score` конструктор з чотирма аргументами. Створіть програму `ScoreWriter3`, яка буде заповнювати екземпляри класу `Score` не за допомогою setter-методів, а в момент створення об'єкту, на-

приклад так:

```
Score aScore = new Score("Микола",
                         "Смирнов", 250, today);
```

Практичні вправи для розумників і розумниць

Погугліть як використовувати клас `HashMap`. Спробуйте створити програму `HashMapDemo`, яка схожа на програму `ArrayListDemo`, але в колекції зберігаються імена і телефони.

Розділ 11. Повертаємося до графіки. Пінг-Понг

У розділах 5, 6 і 7 ми використовували деякі компоненти з AWT і Swing бібліотек. Тепер я покажу вам, як можна малювати і рухати такі фігури, як овали, прямокутники та лінії всередині вікна. Також, ви дізнаєтеся, як обробляти події миші і клавіатури. Щоб додати трохи веселощів в ці нудні теми, в цьому розділі ми будемо вивчати їх при створенні гри пінг-понг. У грі будуть два учасники, я називаю їх дитина і комп'ютер.

Стратегія

Давайте встановимо правила гри:

- Гра продовжується, поки один з гравців (дитина або комп'ютер) не набере 21 бал.
- Ракетка дитини буде управлятися за допомогою миші.
- Рахунок гри буде відображатися в нижній частині вікна.
- Нова гра починається при натисканні кнопки N на клавіатурі, Q - завершує гру і S - подає м'яч.
- Подати м'яч може тільки дитина.
- Для того, щоб виграти бал, м'яч повинен потрапити в область за вертикальною лінією ракетки, коли ракетка не блокує м'яч.
- Коли комп'ютер відбиває м'яч, м'яч може рухатися тільки горизонтально вправо.
- Якщо м'яч торкається ракетки у верхній половині столу, він може рухатися тільки ліворуч і вгору. Якщо м'яч знаходився в нижній частині столу, він може рухатися тільки вниз і ліворуч.

Можливо, ви думаєте, що написати таку програму дуже складно. Хитрість у тому, щоб розбити складне завдання на кілька маленьких і простих завдань і зробити кожну з них окремо.

Ця хитрість називається *аналітичним мисленням* і вона допомагає не тільки в програмуванні, але і в повсякденному житті. Не турбуйтеся, якщо ви не можете досягти великої мети, розбийте її на маленькі і виконуйте окремо.

Тому перша версія програми буде виконувати тільки деякі з правил - програма буде малювати стіл, переміщати ракетку і показувати координати кліків миші.

Замість того, щоб говорити «Мій комп'ютер не працює» (велика проблема), спробуйте подивитися, що саме не працює (знайдіть проблему трохи менше).

1. Чи підключений комп'ютер до розетки? (так/ні)? *Так.*
2. Коли я запускаю комп'ютер, чи відображаються іконки на екрані (так/ні)? *Так.*
3. Чи можна переміщати мишу по екрану (так/ні)? *Ні.*
4. Чи правильно підключений кабель миші (так/ні)? *Ні.*

Просто підключіть кабель миші і комп'ютер знову буде працювати! Велика проблема вирішується всього лише підключенням кабеля миші.

Код

Гра буде складатися з трьох класів:

- Клас `PingPongGreenTable` візьме на себе візуальну частину. Протягом гри він буде відображати стіл, ракетки і м'яч.
- Клас `PingPongGameEngine`, який вважатиме координати м'яча і ракеток, починати і завершувати гру, подавати м'яч. Клас передаватиме поточні координати компонентів класу `PingPongGreenTable`, який буде перемальовуватись відповідно з даними.
- Інтерфейс `GameConstants` оголошуватиме всі константи, які знадобляться в грі, наприклад довжину і ширину столу, початкові позиції ракеток та ін.

Ось як буде виглядати стіл для пінг-понгу:

Перша версія цієї гри буде робити тільки три речі:

- Відображати зелений стіл для пінг-понгу.
- Відображати координати покажчика миші, коли ви клікаєте.
- Рухати ракетку дитини вгору і вниз.

Через дві сторінки ви зможете побачити наш клас `PingPongGreenTable`, який успадковується від класу `JPanel` з `Swing`. Подивіться на цей код, поки читайте текст нижче.

Оскільки наша програма повинна знати точні координати покажчика миші, конструктор `PingPongGreenTable` створюватиме об'єкт класу-обробника подій `PingPongGameEngine`.

Цей клас буде виконувати деякі дії, коли дитина клацає кнопкою миші або просто рухає її. Метод `addPaneltoFrame()` створює текстовий елемент, який буде відображати координати миші.

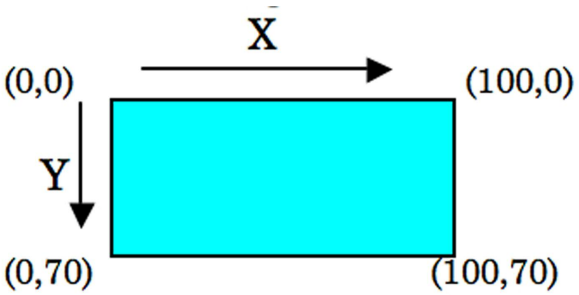
Цей клас не аплет, тому замість методу `paint()`, використовується `paintComponent()`. Цей метод може бути викликаний JVM, коли потрібно перемалювати вікно, так і нашою програмою, за допомогою методу `repaint()`. Так, ви прочитали правильно, метод `repaint()` усередині викликає `paintComponent()` і надає вашому класу доступ до об'єкта

`Graphics`, щоб ви змогли малювати всередині вікна. Ми будемо викликати цей метод кожен раз після перерахунку координат ракеток або м'яча для їх правильного відображення.

Щоб намалювати ракетку, спочатку задайте колір, а потім заповніть їм прямокутник за допомогою методу `fillRect()`. Цей метод повинен знати `X` і `Y` координати верхнього лівого кута прямокутника, а також ширину і висоту в пікселях.

Коло малюється за допомогою методу `fillOval()`, для цього треба знати координати центру овалу, його висоту і ширину. Коли висота і ширина однакові, овал виглядає як коло.

Координата X у вікні збільшується зліва направо, координата Y зростає зверху вниз. Припустимо, ширина ось цього прямокутника 100 пікселів, а висота - 70:



Координати X і Y кутів прямокутника показані в дужках.

Ще один цікавий метод - `getPreferredSize()`. Для того, щоб встановити розміри столу, ми створимо екземпляр класу `Dimension` з Swing. Віртуальній машині Java потрібно знати про розміри вікна, тому вона викликає метод `getPreferredSize()` об'єкта `PingPongGreenTable`. Цей метод повертає об'єкт `Dimension`, який ми створили відповідно до розмірів нашого столу.

Обидва класи `PingPongGreenTable` і `PingPongGameEngine` використовують деякі константи, які не змінюються. Наприклад, у класі `PingPongGreenTable` використовується ширина і висота столу, а `PingPongGameEngine` повинен знати, на скільки пікселів рухати м'яч - чим менше це значення, тим більш плавним буде рух.

Зручно зберігати всі константи (змінні `final`) у інтерфейсі. У нашій грі інтерфейс називається `GameConstants`. Якщо в класі знадобляться ці значення, просто додайте `implements GameConstants` до оголошення класу і використовуйте будь-які змінні `final` з цього інтерфейсу так, як ніби вони задані у цьому ж класі! Тому обидва класи `PingPongGreenTable` і `PingPongGameEngine` реалізують інтерфейс `GameConstants`.

Якщо ви вирішите поміняти розмір столу, м'яча або ракетки, єдине місце, де це потрібно буде зробити - інтерфейс `GameConstants`. Давайте подивимося на код класу `PingPongGreenTable` і інтерфейс `GameConstants`.

```
package screens;

import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.WindowConstants;
import java.awt.Point;
import java.awt.Dimension;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Color;
import engine.PingPongGameEngine;

/**
Цей клас малює стіл для пінг-понгу і відображає координати
точки, де користувач клікнув мишею
*/
public class PingPongGreenTable extends JPanel

implements GameConstants{
  JLabel label;
  public Point point = new Point(0,0);
  public int ComputerRacket_X = 15;
  private int kidRacket_Y =KID_RACKET_Y_START;
```

```
Dimension preferredSize=new Dimension(TABLE_WIDTH,TABLE_HEIGHT);

// Цей метод встанавлює розмір
// Викликається віртуальною Java машиною
public Dimension getPreferredSize() {
  return preferredSize;
}

//Конструктор. Створює обробник подій миші.
PingPongGreenTable(){
  PingPongGameEngine gameEngine=new PingPongGameEngine(this);

  //Обробляє кліки миші для відображення її координат
  addMouseListener(gameEngine);

  //Обробляє рухи миші для пересування ракеток
  addMouseMotionListener(gameEngine);
}

//Додати панель з JLabel у вікно
void addPaneltoFrame(Container container) {
  container.setLayout(new BoxLayout(container,
                                    BoxLayout.Y_AXIS));
  container.add(this);

  label = new JLabel("Click to see coordinates");
  container.add(label);
}

//Перемалювати вікно. Цей метод викликається віртуальною
//машиною, коли потрібно оновити екран або
//викликається метод repaint() з PingPointGameEngine
public void paintComponent(Graphics g) {
  super.paintComponent(g);
  g.setColor(Color.GREEN);
```

```
    //Намалювати стіл
    g.fillRect(0,0,TABLE_WIDTH,TABLE_HEIGHT);
    g.setColor(Color.yellow);

    //Намалювати праву ракетку
    g.fillRect(KID_RACKET_X_START,kidRacket_Y,5,30);
    g.setColor(Color.blue);

    //Намалювати ліву ракетку
    g.fillRect(ComputerRacket_X,100,5,30);
    g.setColor(Color.red);
    g.fillOval(25,110,10,10); //Намалювати м'яч
    g.setColor(Color.white);
    g.drawRect(10,10,300,200);
    g.drawLine(160,10,160,210);

    //Відобразити точку як маленький квадрат 2x2 пікселів
    if (point != null) {
      label.setText("Coordinates (x,y): " + point.x +
                                          ", " + point.y);
      g.fillRect(point.x, point.y, 2, 2);
    }
  }

  //Встановити поточне положення ракетки дитини
  public void setKidRacket_Y(int xCoordinate){
    this.kidRacket_Y = xCoordinate;
  }

  //Повернути поточне положення ракетки дитини
  public int getKidRacket_Y(int xCoordinate){
    return kidRacket_Y;
  }

  public static void main(String[] args) {
    //Створити екземпляр вікна
    JFrame f = new JFrame("Ping Pong Green Table");
```

```
        //Переконатися, що вікно може бути закрите після
        //натискання на хрестик в кутку
        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        PingPongGreenTable table = new PingPongGreenTable();
        table.addPaneltoFrame(f.getContentPane());

        //Встановити розмір вікна і зробити його видимим
        f.pack();
        f.setVisible(true);
    }
}
```

Тепер подивимося на інтерфейс `GameConstants`. Всі значення змінних в пікселях. У назвах `final` змінних використовуйте великі літери:

```
package screens;

public interface GameConstants {
    public final int TABLE_WIDTH =  320;
    public final int TABLE_HEIGHT = 220;
    public final int KID_RACKET_Y_START = 100;
    public final int KID_RACKET_X_START = 300;
    public final int TABLE_TOP = 12;
    public final int TABLE_BOTTOM = 180;
    public final int RACKET_INCREMENT = 4;
}
```

Запущена програма не зможе поміняти значення цих змінних, тому що вони оголошені як `final`. Але, якщо ви вирішите поміняти, наприклад, розмір столу, досить змінити значення `TABLE_WIDTH` і `TABLE_HEIGHT` і перекомпілювати інтерфейс `GameConstants`.

Рішення в цій грі приймає клас `PingPongGameEngine`, який реалізує 2 інтерфейси, пов'язані з подіями миші.

`MouseListener` буде використовуватися в методі

`mousePressed()`. На кожне натискання миші цей метод буде малювати маленьку білу точку на столі і відображати її координати. Чесно кажучи, в нашій грі цей код марний, але він покаже простий спосіб дістати з об'єкта `MouseEvent` координати миші, які були передані JVM.

Метод `mousePressed()` передає координати натиснутої кнопки миші змінній `point`. Після того, як координати передані, цей метод просить віртуальну машину перемалювати стіл.

`MouseMotionListener` відстежує рух миші над столом, а метод `mouseMoved()` використовуватиметься для переміщення ракетки дитини вниз або вгору.

Метод `mouseMoved()` рахує наступну позицію ракетки гравця.

Якщо покажчик миші знаходиться над ракеткою (координата Y миші менше, ніж координата Y ракетки), то цей метод гарантує, що ракетка не вийде за межі столу.

Коли конструктор `PingPongGreenTable` створює об'єкт класу `PingPongGameEngine`, він передає йому посилання об'єкта столу (ключове слово `this` означає посилання на область пам'яті з об'єктом `PingPongGreenTable`). Тепер, `PingPongGameEngine` може «розмовляти» зі столом, наприклад, встановлювати нові координати м'яча або перемалювати стіл, коли потрібно. Якщо ця частина не зовсім зрозуміла, перечитайте частину розділу 6 про передачу даних між класами.

У нашій грі ракетки переміщаються вертикально по 4 пікселя, як ми задали в інтерфейсі `GameConstants` (клас `PingPongGameEngine` реалізує цей інтерфейс). Наприклад, наступний рядок віднімає 4 від значення змінної `kidRacket_Y`:

```
kidRacket_Y -= RACKET_INCREMENT;
```

Наприклад, якщо координата Y ракетки була 100, після цього рядка коду вона стає рівною 96 і ракетка повинна піднятися вгору. Той самий результат можна отримати таким виразом:

```
kidRacket_Y = kidRacket_Y - RACKET_INCREMENT;
```

Якщо ви пам'ятаєте, ми говорили про різні способи змінити значення змінної в Розділі 3.

Далі - клас `PingPongGameEngine`.

```
package engine;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import screens.*;

public class PingPongGameEngine implements
       MouseListener, MouseMotionListener, GameConstants{
  PingPongGreenTable table;
  public int kidRacket_Y = KID_RACKET_Y_START;

  //Конструктор. Містить посилання на об'єкт столу
  public PingPongGameEngine(PingPongGreenTable greenTable){
     table = greenTable;
  }

  //Обов'язкові методи з інтерфейсу MouseListener
  public void mousePressed (MouseEvent e) {

    //Взяти X і Y координати покажчика миші
    //і встановити їх "Білій точці" на столі
    table.point.x = e.getX();
    table.point.y = e.getY();

    //Усередині викликає метод paintComponent () і оновлює вікно
    table.repaint();
  }

  public void mouseReleased(MouseEvent e) {};
  public void mouseEntered(MouseEvent e) {};
  public void mouseExited(MouseEvent e) {};
  public void mouseClicked(MouseEvent e) {};
```

```
  //Обов'язкові методи з інтерфейсу MouseMotionListener
  public void mouseDragged(MouseEvent e) {}

  public void mouseMoved(MouseEvent e) {
    int mouse_Y = e.getY();

    //Якщо миша знаходиться вище ракетки дитини
    //і не виходить за межі столу -
    //пересунути її вгору, в іншому випадку - опустити вниз
    if (mouse_Y < kidRacket_Y && kidRacket_Y > TABLE_TOP){
      kidRacket_Y -= RACKET_INCREMENT;
    }else if (kidRacket_Y < TABLE_BOTTOM) {
      kidRacket_Y += RACKET_INCREMENT;
    }

    //Встановити нове положення ракетки
    table.setKidRacket_Y(kidRacket_Y);
    table.repaint();
  }
}
```

Основи багатопотоковості

До цього всі дії в наших програмах виконувалися послідовно - одна за іншою. Якщо програма викликає два методи, другий метод чекає, поки не виконається перший. Іншими словами, кожна з наших програм має тільки *один потік виконання (a thread)*.

Проте, в реальному житті ми можемо робити кілька речей одночасно, наприклад, їсти, розмовляти по телефону, дивитися телевізор і робити домашнє завдання. Щоб виконувати всі ці дії *паралельно*, ми використовуємо кілька *процесорів*: руки, очі і рот.

На сьогодні, у багатьох комп'ютерах теж два або більше процесорів (CPU). Хоча, можливо, у вашому комп'ютері тільки один процесор, який рахує, посилає команди монітору, диску, віддаленим комп'ютерам, тощо.

Але навіть один процесор може виконувати декілька дій відразу, якщо програма використовує *декілька потоків (multiple threads)*. Один Java-клас може запустити декілька потоків виконання, які будуть змінюватися і отримувати свою частку процесорного часу.

Хороший приклад програми, яка створює кілька потоків - це web-браузер. Ви можете дивитися Інтернет-сайти, поки качаються кілька файлів - одна програма запускає два потоки виконання.

У наступній версії нашої пінг-понг гри буде один потік для промальовки столу. Другий потік буде рахувати координати м'яча і ракеток, і посилати команди по відображенні вікна першому потоку. Але спочатку, я покажу вам дві дуже прості програми, щоб ви краще зрозуміли, навіщо потрібні потоки.

Кожна з цих програм буде відображати кнопку і текстове поле.

Коли ви натиснете кнопку *Kill Time*, програма почне цикл, який збільшує значення змінної триста тисяч разів. поточне значення цієї змінної буде відображатися в заголовку вікна. У класі `NoThreadsSample` тільки один потік виконання і ви не зможете нічого надрукувати в текстовому полі, поки цикл не закінчиться. Цей цикл забирає увесь процесорний час, тому вікно заблоковано.

Скомпілюйте і запустіть цей клас, щоб переконатися, що вікно блокується протягом деякого часу. Зауважте, що цей клас створює екземпляр класу `JTextField` і передає його в панель контенту без створення змінної. Якщо ви не плануєте отримувати або встановлювати атрибути цього об'єкта,

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class NoThreadsSample extends JFrame
                             implements ActionListener{
  //Конструктор
  NoThreadsSample(){
    //Створити вікно з кнопкою і текстовим полем
    GridLayout gl = new GridLayout(2,1);
    this.getContentPane().setLayout(gl);
    JButton myButton = new JButton("Kill Time");
    myButton.addActionListener(this);
    this.getContentPane().add(myButton);
    this.getContentPane().add(new JTextField());
  }

  //Оброблювач натискання кнопки
  public void actionPerformed(ActionEvent e){
    //Просто заморозити на деякий час,
    //щоб показати, що вікно заблоковано
```

```
    for (int i=0; i<300000;i++){
      this.setTitle ("i="+i);
    }
  }

  public static void main(String[] args) {
    //Створити вікно
    NoThreadsSample myWindow = new NoThreadsSample();

    //Переконатися, що вікно закривається при натисканні
    //на хрестик в кутку
    myWindow.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

    //Встановити розміри вікна - координати лівого
    //верхнього кута і висоту з шириною
    myWindow.setBounds(0,0,150, 100);

    //Зробити вікно видимим
    myWindow.setVisible(true);
  }
}
```

Наступна версія цього маленького віконця буде створювати і запускати окремий потік для циклу, і головний потік вікна дозволить друкувати в текстовому полі, поки цикл виконується.

Ви можете створити потік одним із таких способів:

- Створити екземпляр Java-класу Thread і передати йому об'єкт, який реалізує інтерфейс Runnable. Якщо ваш клас реалізує інтерфейс Runnable, код буде виглядати так:

```
Thread worker = new Thread (this);
```

Цей інтерфейс вимагає написати в методі run() код, який буде виконуватися в окремому потоці.

```
worker.start();
```

- Створити підклас класу Thread і реалізувати метод run(). Для того, щоб запустити потік, треба викликати метод start().

```
public class MyThread extends Thread {

  public static void main(String[] args) {
    MyThread worker = new MyThread();
    worker.start();
  }

  public void run(){
    //Тут буде ваш код
  }
}
```

У класі `ThreadsSample` я буду використовувати перший спосіб, тому що цей клас вже успадковується від `JFrame`, а успадковувати більше одного класу в Java не можна.

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class ThreadsSample extends JFrame
                  implements ActionListener, Runnable {

  //Конструктор
  ThreadsSample(){

    //Створити вікно з кнопкою і текстовим полем
    GridLayout gl = new GridLayout(2,1);
```

```
    this.getContentPane().setLayout(gl);
    JButton myButton = new JButton("Kill Time");
    myButton.addActionListener(this);
    this.getContentPane().add(myButton);
    this.getContentPane().add(new JTextField());
  }

  public void actionPerformed(ActionEvent e){

    //Створити потік і виконати "заморожуючий" код
    //без блокування
    Thread worker = new Thread(this);
    worker.start();  //викликає метод run()
  }

  public void run(){

    //Заморозити на деякий час, щоб показати, що
    //елементи вікна НЕ блокуються
    for (int i=0; i<300000;i++){
      this.setTitle("i="+i);
    }
  }

  public static void main(String[] args) {

    ThreadsSample myWindow = new ThreadsSample();
    //Переконайся, що вікно закривається після натискання
    //на хрестик в кутку
    myWindow.setDefaultCloseOperation(
                        WindowConstants.EXIT_ON_CLOSE);
    //Встанови розміри вікна і зроби його видимим
    myWindow.setBounds(0,0,150, 100);
    myWindow.setVisible(true);
  }
}
```

Після натискання на кнопку *Kill Time*, клас `ThreadsSample` запускає новий потік. Після цього, потік з циклом і головний потік отримують по своїй частці процесорного часу. І тепер ви можете друкувати в текстовому полі (головний потік), поки інший потік виконує цикл!

Вивчення потоків заслуговують набагато більшої уваги, ніж ці кілька сторінок, і я рекомендую вам почитати додатковий матеріали з цієї теми.

Закінчуємо гру Пінг-Понг

Тепер, після короткого вступу до потоків ми готові поміняти класи нашої гри в пінг-понг. Давайте почнемо з класу `PingPongGreenTable`. Нам не треба відображати білу точку по кліку миші - це була просто навчальна вправа для відображення координат покажчика миші. Тому ми видалимо оголошення змінної `point` і рядки, які малюють білу точку з методу `paintComponent()`. Також, в конструкторі більше не потрібен `MouseListener`, так як він тільки показує координати точки.

З іншого боку, цей клас повинен реагувати на деякі кнопки клавіатури (N - для початку нової гри, S - для подачі м'яча і Q - для виходу з гри). У цьому нам допоможе метод `addKeyListener()`.

Для того, щоб зробити наш код більш інкапсульованим, я перемістив виклики `repaint()` з класу `PingPongGameEngine` в клас `PingPongGreenTable`. Тепер, коли знадобиться, `PingPongGreenTable` буде перемальовувати себе сам.

Також, я додав методи для зміни положення м'яча, ракетки комп'ютера і для відображення повідомлень.

```
package screens;

import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.WindowConstants;
import java.awt.Dimension;
```

```
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Color;
import engine.PingPongGameEngine;

/**
*Цей клас малює зелений стіл для пінг-понгу, кулю, ракетки,
відображає рахунок
*/

public class PingPongGreenTable extends JPanel
                                implements GameConstants{
  private JLabel label;
  private int computerRacket_Y = COMPUTER_RACKET_Y_START;
  private int kidRacket_Y = KID_RACKET_Y_START;
  private int ballX = BALL_START_X;
  private int ballY = BALL_START_Y;
  Dimension preferredSize = new
                    Dimension(TABLE_WIDTH,TABLE_HEIGHT);

  //Встановлюємо розміри вікна. Викликається віртуальною машиною
  public Dimension getPreferredSize() {
    return preferredSize;
  }

  //Конструктор. Створює обробник подій миші.
  PingPongGreenTable(){
    PingPongGameEngine gameEngine = new PingPongGameEngine(this);

    //Обробляємо рухи миші для пересування ракеток
    addMouseMotionListener(gameEngine);

    //Обробляємо події клавіатури
    addKeyListener(gameEngine);
  }
```

```
//Додамо у вікно панель с JLabel
void addPaneltoFrame(Container container) {
  container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
  container.add(this);
  label = new JLabel(
        "Press N for a new game, S to serve or Q to quit");
  container.add(label);
}

//Перемалювати вікно. Цей метод викликається віртуальною
//машиною, коли потрібно оновити екран або
//викликається метод repaint() із PingPointGameEngine
public void paintComponent(Graphics g) {
  super.paintComponent(g);

  //Намалювати зелений стіл
  g.setColor(Color.GREEN);
  g.fillRect(0,0,TABLE_WIDTH,TABLE_HEIGHT);

  //Намалювати праву ракетку
  g.setColor(Color.yellow);
  g.fillRect(KID_RACKET_X, kidRacket_Y,
                              RACKET_WIDTH, RACKET_LENGTH);

  // Намалювати ліву ракетку
  g.setColor(Color.blue);
  g.fillRect(COMPUTER_RACKET_X, computerRacket_Y,
                               RACKET_WIDTH,RACKET_LENGTH);

  //Намалювати м'яч
  g.setColor(Color.red);
  g.fillOval(ballX,ballY,10,10);

  //Намалювати білі лінії
  g.setColor(Color.white);
  g.drawRect(10,10,300,200);
  g.drawLine(160,10,160,210);
```

```
    //Встановити фокус на стіл, щоб
    //оброблювач клавіатури міг посилати команди столу
    requestFocus();
  }

  //Встановити поточне положення ракетки дитини
  public void setKidRacket_Y(int yCoordinate){
    this.kidRacket_Y = yCoordinate;
    repaint();
  }

  //Повернути поточне положення ракетки дитини
  public int getKidRacket_Y(){
    return kidRacket_Y;
  }

  //Повернути поточне положення ракетки комп'ютера
  public void setComputerRacket_Y(int yCoordinate){
    this.computerRacket_Y = yCoordinate;
    repaint();
  }

  //Встановити ігрове повідомлення
  public void setMessageText(String text){
    label.setText(text);
    repaint();
  }

  //Встановити позицію м'яча
  public void setBallPosition(int xPos, int yPos){
    ballX=xPos;
    ballY=yPos;
    repaint();
 }
```

```
  public static void main(String[] args) {

    //Створити екземпляр вікна
    JFrame f = new JFrame("Ping Pong Green Table");

    //Переконатися, що вікно може бути закрите після
    //натискання на хрестик в кутку

    f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    PingPongGreenTable table = new PingPongGreenTable();
    table.addPaneltoFrame(f.getContentPane());

    //Встановити розмір вікна і зробити його видимим
    f.setBounds(0,0,TABLE_WIDTH+5, TABLE_HEIGHT+40);
    f.setVisible(true);
  }
}
```

Я додав кілька `final` змінних в інтерфейс `GameConstants`, і ви повинні здогадатися за їхніми іменами, для чого вони потрібні.

```
package screens;

/**
*Цей інтерфейс містить всі константи, які використовуються у грі
*/
public interface GameConstants {

  //Розміри столу
  public final int TABLE_WIDTH =  320;
  public final int TABLE_HEIGHT = 220;
  public final int TABLE_TOP = 12;
  public final int TABLE_BOTTOM = 180;
```

```
    //Крок переміщення м'яча в пікселях
    public final int BALL_INCREMENT = 4;

    //Максимальні і мінімальні координати м'яча
    public final int BALL_MIN_X = 1+ BALL_INCREMENT;
    public final int BALL_MIN_Y = 1 + BALL_INCREMENT;
    public final int BALL_MAX_X = TABLE_WIDTH - BALL_INCREMENT;
    public final int BALL_MAX_Y = TABLE_HEIGHT - BALL_INCREMENT;

    //Початкові координати м'яча
    public final int BALL_START_X = TABLE_WIDTH/2;
    public final int BALL_START_Y = TABLE_HEIGHT/2;

    //Розміри, розташування і крок переміщення ракеток
    public final int KID_RACKET_X = 300;
    public final int KID_RACKET_Y_START = 100;
    public final int COMPUTER_RACKET_X = 15;
    public final int COMPUTER_RACKET_Y_START = 100;
    public final int RACKET_INCREMENT = 2;
    public final int RACKET_LENGTH = 30;
    public final int RACKET_WIDTH = 5;
    public final int WINNING_SCORE = 21;

    //Уповільнити швидкі комп'ютери - змініть це значення,
    //Якщо знадобиться
    public final int SLEEP_TIME = 10; //час в мілісекундах
}
```

Нижче я перерахував основні зміни, які зробив у класі `PingPongGameEngine`:

- Видалив інтерфейс `MouseListener` і всі його методи, тому що ми більше не обробляємо кліки миші. Всі рухи миші будуть оброблятися `MouseMotionListener`.
- Тепер цей клас реалізує інтерфейс `Runnable`, а вся логіка знаходиться в методі `run()`. Подивіться на конструктор - там я створюю і запускаю новий потік. Метод `run()` обробляє пра-

вила гри в кілька кроків, всі ці кроки запрограмовані всередині умови `if(ballServed)`. Це скорочений варіант виразу `if(ballServed==rue)`.

- Будь ласка, зверніть увагу на умову, яка встановлює значення змінної `canBounce` в першому кроці. Залежно від цього виразу, значення змінної буде або `true`, або `false`.
- Клас реалізує інтерфейс `KeyListener`, і метод `keyPressed()` перевіряє, яка кнопка була натиснута для початку/завершення гри, або подачі м'яча. Цей метод дозволяє обробляти як великі, так і маленькі літери, наприклад N і n.
- Я додав кілька `private` методів: `displayScore()`, `kidServe()` і `isBallOnTheTable()`. Вони оголошені приватними, тому що використовуються тільки всередині цього класу і інші класи навіть не підозрюють про їх існування. Це приклад *інкапсуляції* в дії.
- Деякі комп'ютери настільки швидкі, що контролювати рух м'яча стає важко. Тому я уповільнив гру за допомогою методу `Thread.sleep()`. Статичний метод `sleep()` призупинить поточний потік на задану в конструкторі кількість мілісекунд.
- Щоб гра стала трохи веселішою, м'яч тепер рухається по діагоналі після удару ракеткою дитини. Тому змінюється не тільки X координата м'яча, але і Y.

```
package engine;

import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import screens.*;
```

```
/**
*Цей клас - обробник подій миші і клавіатури.
*Розраховує рух м'яча і ракеток, зміну їх координат.
*/

public class PingPongGameEngine implements Runnable,
          MouseMotionListener, KeyListener, GameConstants{
  private PingPongGreenTable table; //посилання на стіл
  private int kidRacket_Y = KID_RACKET_Y_START;
  private int computerRacket_Y=COMPUTER_RACKET_Y_START;
  private int kidScore;
  private int computerScore;
  private int ballX;   //координата X м'яча
  private int ballY;   //координата Y м'яча
  private boolean movingLeft = true;
  private boolean ballServed = false;

  //Значення вертикального пересування м'яча в пікселях
  private int verticalSlide;

  //Конструктор. Містить посилання на об'єкт столу
  public PingPongGameEngine(PingPongGreenTable greenTable){

    table = greenTable;
    Thread worker = new Thread(this);
    worker.start();
  }

  //Обов'язкові методи з інтерфейсу MouseMotionListener
  //(деякі з них порожні, але повинні бути включені)
  public void mouseDragged(MouseEvent e) {
  }

  public void mouseMoved(MouseEvent e) {

    int mouse_Y = e.getY();
```

```
    //Якщо миша знаходиться вище ракетки дитини
    //і не виходить за межі столу - пересунути її вгору,
    //в іншому випадку - опустити вниз
    if (mouse_Y<kidRacket_Y && kidRacket_Y>TABLE_TOP){
      kidRacket_Y -= RACKET_INCREMENT;
    }else if (kidRacket_Y < TABLE_BOTTOM) {
      kidRacket_Y += RACKET_INCREMENT;
    }

    //Встановити нове положення ракетки
    table.setKidRacket_Y(kidRacket_Y);
  }

  //Обов'язкові методи з інтерфейсу KeyListener
  public void keyPressed(KeyEvent e){

    char key = e.getKeyChar();
    if ('n' == key || 'N' == key){
      startNewGame();
    } else if ('q' == key || 'Q' == key){
      endGame();
    } else if ('s' == key || 'S' == key){
      kidServe();
    }
  }

  public void keyReleased(KeyEvent e){}

    public void keyTyped(KeyEvent e){}

    //Почати нову гру

    public void startNewGame(){

      computerScore=0;
      kidScore=0;
      table.setMessageText("Score Computer: 0  Kid: 0");
```

```
    kidServe();
  }

  //Завершити гру
  public void endGame(){
    System.exit(0);
  }

  //Обов'язковий метод run() з інтерфейсу Runnable
  public void run(){

    boolean canBounce=false;

    while (true) {
      if (ballServed){ //якщо м'яч рухається
        //Крок 1. М'яч рухається ліворуч?
        if (movingLeft && ballX > BALL_MIN_X){
          canBounce = (ballY >= computerRacket_Y &&
          ballY<(computerRacket_Y+RACKET_LENGTH)?true:false);
          ballX-=BALL_INCREMENT;

          //Додати зміщення вгору або вниз до будь-яких
          //рухів м'яча ліворуч або праворуч
          ballY-=verticalSlide;
          table.setBallPosition(ballX,ballY);

          //Може відскочити?
          if (ballX <= COMPUTER_RACKET_X  && canBounce){
            movingLeft=false;
          }
        }

        //Крок 2. М'яч рухається ліворуч?
        if (!movingLeft && ballX <= BALL_MAX_X){
          canBounce = (ballY >= kidRacket_Y && ballY <
             (kidRacket_Y + RACKET_LENGTH)?true:false);
          ballX+=BALL_INCREMENT;
```

```
        table.setBallPosition(ballX,ballY);

        //Може відскочити?
        if (ballX >= KID_RACKET_X && canBounce){
          movingLeft=true;
        }
      }

      //Крок 3. Переміщати ракетку комп'ютера вгору або
      //вниз, щоб блокувати м'яч
      if (computerRacket_Y < ballY
                && computerRacket_Y < TABLE_BOTTOM){
        computerRacket_Y +=RACKET_INCREMENT;
      }else if (computerRacket_Y > TABLE_TOP){
        computerRacket_Y -=RACKET_INCREMENT;
      }
      table.setComputerRacket_Y(computerRacket_Y);

      //Крок 4. Призупинити
      try {
        Thread.sleep(SLEEP_TIME);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      //Крок 5. Оновити рахунок, якщо м'яч
      //в зеленій області, але не рухається
      if (isBallOnTheTable()){
        if (ballX > BALL_MAX_X ){
          computerScore++;
          displayScore();
        }else if (ballX < BALL_MIN_X){
          kidScore++;
          displayScore();
        }
      }
```

```
      } // Кінець if ballServed
    } // Кінець while
  } // Кінець run()

  //Подати з поточної позиції ракетки дитини
  private void kidServe(){
    ballServed = true;
    ballX = KID_RACKET_X-1;
    ballY=kidRacket_Y;

    if (ballY > TABLE_HEIGHT/2){
      verticalSlide=-1;
    }else{
      verticalSlide=1;
    }

    table.setBallPosition(ballX,ballY);
    table.setKidRacket_Y(kidRacket_Y);
  }

  private void displayScore(){
    ballServed = false;

    if (computerScore ==WINNING_SCORE){
      table.setMessageText("Computer won! " +
                                      computerScore +
                                      ":" + kidScore);
    }else if (kidScore ==WINNING_SCORE){
       table.setMessageText("You won! " + kidScore +
                                    ":" + computerScore);
    }else{
      table.setMessageText("Computer: "+ computerScore +
                                    " Kid: " + kidScore);
    }
  }
```

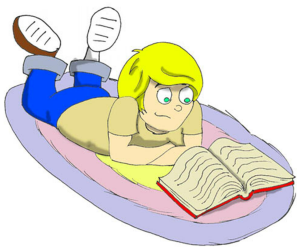
```
    //Перевірити, чи не перетнув м'яч верхню
    //або нижню межу столу
    private boolean isBallOnTheTable(){
      if (ballY >= BALL_MIN_Y && ballY <= BALL_MAX_Y){
        return true;
      } else {
        return false;
      }
    }
  }
}
```

Вітаю! Ви закінчили розробку вашої другої гри. Скомпілюйте класи і грайте в неї. Після того, як ви розберетеся в коді, спробуйте змінити його - я впевнений, у вас є ідеї, як зробити цю гру краще.

Якщо хочете продовжити програмувати ігри, погляньте на Robocode - ця програмована гра, дозволяє вивчати Java, створюючи роботів:

http://robocode.sourceforge.net/?Open&ca=daw-prod-robocode

Матеріали для додаткового читання

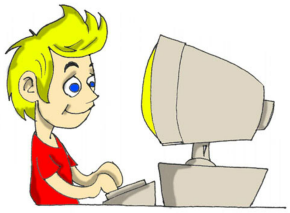


Посібник по сокетах в Java:
http://www.oracle.com/technetwork/java/socket-140484.html

Вступ до потоків Java:
http://www-106.ibm.com/developerworks/edu/j-dw-javathread-i.html

Клас java.awt.Graphics:
http://download.oracle.com/javase/6/docs/api/java/awt/Graphics.html

Практичні завдання



1. Клас `PingPongGameEngine` встановлює координати білої точки за допомогою такого коду:

```
table.point.x = e.getX();
```

Зробіть змінну `point` приватною в класі `PingPongGreenTable` і додайте `public` метод `setPointCoordinates(int x, int y)`.

Використовуйте цей метод в класі `PingPongGameEngine`.

2. У нашій грі пінг-понг є помилка: після того, як один з гравців виграв, все ще можна натиснути кнопку S і гра продовжиться. Виправте цю помилку.

Практичні вправи для розумників і розумниць

1. Спробуйте поміняти значення `RACKET_INCREMENT` і `BALL_INCREMENT`. Чим більше ці значення, тим швидше рухається м'яч і ракетки. Поміняйте код так, щоб користувач міг вибирати рівень від 1 до 10. Використовуйте обрані значення як коефіцієнт руху для м'яча і ракеток.

2. Коли ракетка дитини відбиває м'яч у верхній частині столу, м'яч відскакує по діагоналі і швидко падає зі столу. Змініть програму так, щоб м'яч відскакував від верхньої частини столу по діагоналі вниз, а від нижньої частини столу діагонально вгору.

Додаток A. Java архіви - JARs

Дуже часто користувачам комп'ютерів потрібно обмінюватися файлами. Вони можуть скопіювати файли на USB flash drive, CD, скористатися електронною поштою або просто відправити дані через мережу. Існує спеціальна програма, яка стискає декілька файлів в один файл-архів.

Розмір такого архіву зазвичай менше, ніж загальний розмір усіх файлів окремо, його швидше копіювати і він займає менше місця на ваших дисках.

Java поставляється з програмою під назвою jar, яка архівує кілька Java класів в один файл з розширенням `.jar`.

Внутрішній формат `jar` файлів такий же, як і в популярній програмі WinZip (ми її використовували в 2-му розділі).

Наступні три команди ілюструють, як можна використовувати програму `jar`.

Для того, щоб створити архів jar з файлами з розширенням `.Class`, відкрийте чорне вікно терміналу, увійдіть в папку, де знаходяться ваші класи і наберіть наступну команду:

```
jar cvf myClasses.jar *.class
```

Після слова `jar` ви повинні вказати опції для цієї команди. В останньому прикладі *c* - для створення нового архіву, *v* - для відображення того, що відбувається і *f* означає те, що ми вказали ім'я файлу для нового архіву.

Тепер ви можете скопіювати цей файл на інший диск або відправити електронною поштою вашому другу. Для того, щоб розпакувати файли з архіву `myClasses.jar`, наберіть таку команду:

```
jar xvf myClasses.jar
```

Всі файли будуть розпаковані в поточну папку. У цьому прикладі x - для вилучення файлів з архіву.

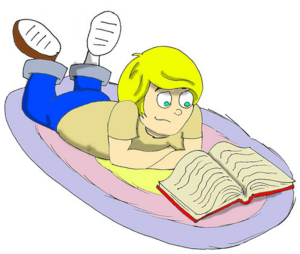
Якщо ви просто хочете подивитися вміст jar-архіву без розпакування, скористайтеся командою, де *t* - зміст.

```
jar tvf myClasses.jar
```

Насправді, я віддаю перевагу програмі WinZip для того, щоб подивитися, що знаходиться в jar-архіві.

У більшості випадків Java програми з реального світу складаються з безлічі класів, які знаходяться в jar-архівах. Хоча й існує багато інших опцій, які можна використовувати з командою jar, три приклади з цього розділу - це головне, що потрібно знати для більшості ваших майбутніх проектів.

Матеріали для додаткового читання



Jar – Інструмент для java-архівів:
http://download.oracle.com/javase/7/docs/technotes/tools/windows/jar.html

Додаток Б. Поради для роботи в Eclipse

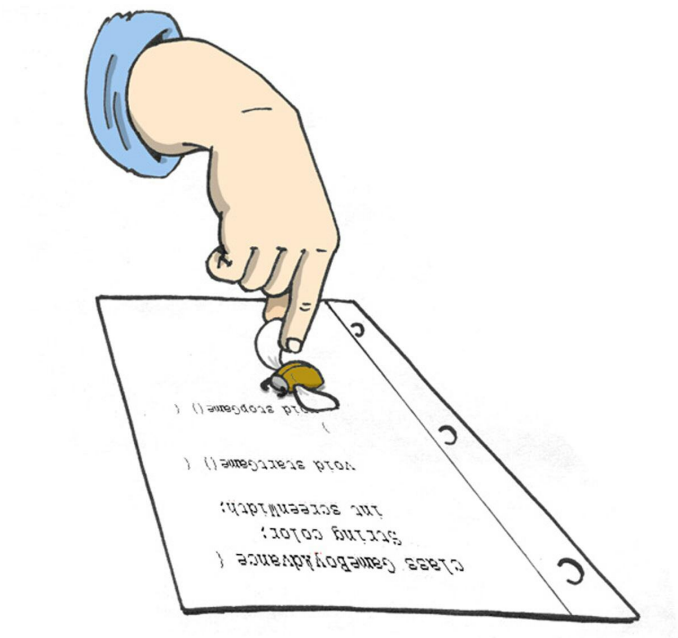
В Eclipse існує безліч маленьких зручних команд, які роблять програмування на Java трохи швидшим. Я перерахував тут деякі корисні поради для Eclipse, і я впевнений, що ви знайдете їх ще більше, коли почнете використовувати цей інструмент.

- Якщо ви бачите маленьку зірочку на вкладці з класом - це означає, що клас містить не збережені зміни в коді.
- Виділіть ім'я класу або методу у вашому коді і натисніть клавішу F3. Ця команда перемістить вас на рядок, де оголошено цей клас або метод.
- Якщо деякі рядки позначені червоними колами з помилками, то навівши курсор миші на коло, ви побачите текст помилки.
- Натисніть *Ctrl-F11*, щоб запустити останню виконану програму.
- Помістіть курсор після фігурної дужки, і Eclipse виділить іншу відповідну їй закриваючу або відкриваючу дужку.
- Для того, щоб скопіювати клас з одного пакету в інший, виберіть клас і натисніть Ctrl-C. Оберіть пакет, в який хочете його скопіювати і натисніть Ctrl-V.
- Для того, щоб перейменувати клас, змінну і метод, клікніть правою кнопкою миші на ньому і виберіть Refactor і Rename із спливаючого меню. Eclipse перейменує це ім'я скрізь, де воно згадується.
- Якщо у вашому проекті потрібні зовнішні jar-архіви (наприклад, зроблені кимось іншим), клікніть правою кнопкою на імені проекту, виберіть Properties, Java Build Path та натисніть кнопку Add External Jars.

Налагоджувач Eclipse

Подейкують, близько 50 років тому, коли комп'ютери були великими і навіть не помістилися б у вашій кімнаті, раптом, одна з програм почала видавати невірні результати. Ці проблеми були викликані маленьким жучком (англ. bug), який сидів всередині комп'ютера десь в проводах. Коли люди дістали цього жука, програма знову стала працювати правильно. Починаючи з цього моменту, налагоджувати програму (англ. debug) стало означати знаходження причини некоректних результатів програми.

Не плутайте логічні помилки з помилками компіляції. Наприклад, замість того, щоб помножити змінну на 2, ви помножите її на 22. Ця помилка не викличе ніяких помилок компіляції, але результат буде невірний. Налагоджувачі дозволяють крокувати в програмі рядок за рядком із зупинками, і ви можете бачити або міняти значення всіх змінних в будь-який момент виконання програми.

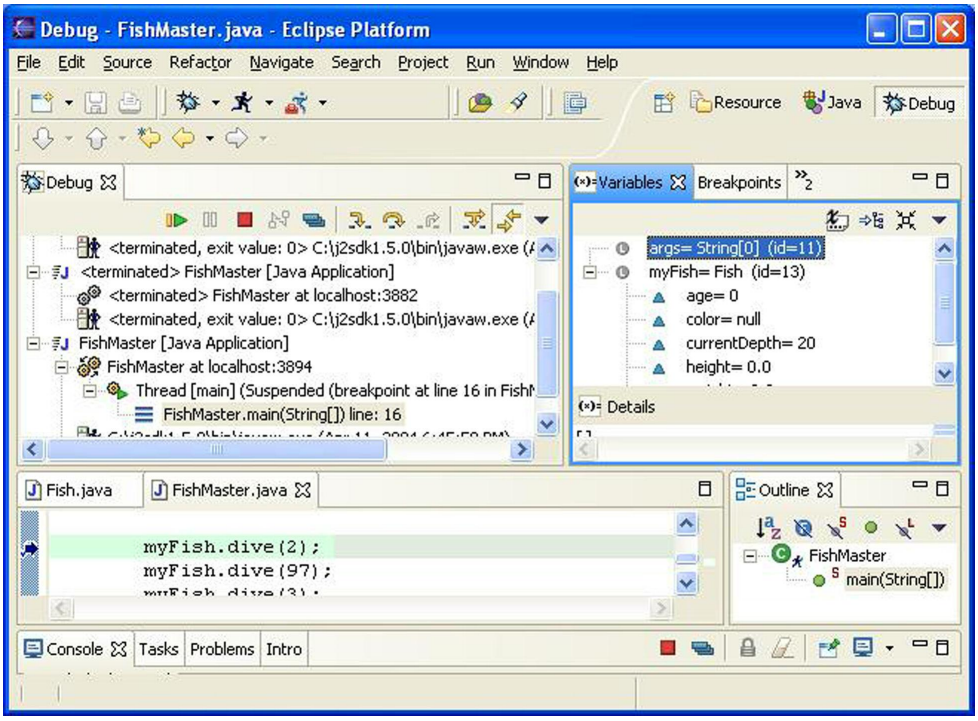


Я покажу, як використовувати налагоджувач Eclipse на прикладі програми FishMaster з четвертого розділу.

Точка зупину (breakpoint) - це рядок коду, де ви хочете, щоб програма зупинилася для того, щоб спостерігати / міняти поточні значення змінних, та іншу інформацію часу виконання. Для того, щоб встановити точку зупину, просто зробіть подвійне клацання на сірій вертикальній смузі зліва від лінії, де ви хочете зупинити програму. Давайте зробимо це в класі FishMaster, на рядку `myFish.dive(2)`. Ви побачите круглий маркер на рядку з точкою зупину. Тепер, виберіть у меню *Run, Debug ....* Виберіть програму FishMaster і натисніть кнопку *Debug*.

FishMaster запуститься в режимі налагодження і як тільки програма досягне рядка `myFish.dive(2)`, зупиниться і буде чекати ваших подальших дій.

Ви побачите вікно налагоджувача, схоже на це:

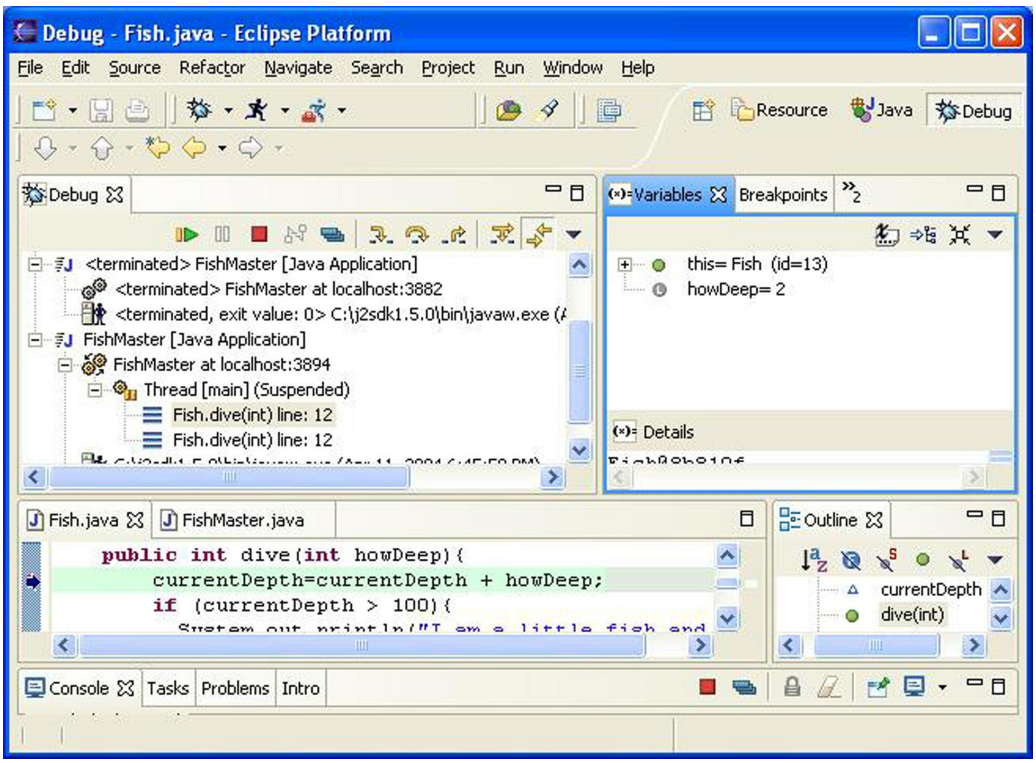


У лівій нижній частині перспективи налагодження ви побачите, що рядок з точкою зупину підсвічується. Синя стрілка вказує на рядок, який виконується. З правого боку знаходиться вікно *Variables*, клікніть на маленькому значку плюса у змінної `myFish`. Т.к. змінна вказує на об'єкт `Fish`, ви

побачите всі елементи цього класу та їх поточний стан, наприклад, `currentDepth = 20`.

Стрілки у верхній лівій частині дозволяють продовжити виконання програми в різних режимах. Перша жовта стрілка означає увійти всередину методу. Якщо ви натиснете цю стрілку (або F5), то ви потрапите всередину методу `dive()`. Вікно зміниться, і ви побачите значення аргументу `howDeep = 2`, як на наступному малюнку. Натисніть на маленькому плюсі біля слова `this` для того, щоб побачити поточні значення атрибутів цього класу.

Для того, щоб поміняти значення змінної, клікніть правою кнопкою на ній і введіть нове значення. Це допоможе вам, якщо ви не розумієте, чому програма працює неправильно і якщо ви хочете пограти у відгадайку - як би працювала програма, якщо значення змінної було б іншим.



Для того, щоб продовжити виконання програми по одному рядку, на-

тискайте наступну стрілку - переступити (або клавішу F6).

Якщо хочете продовжити виконання програми в швидкому режимі, натисніть маленький зелений трикутник або клавішу F8.

Для того, щоб видалити точку зупину, просто клікніть двічі на маеленькому круглому маркері, і вона зникне. Я люблю використовувати налагоджувач, навіть якщо в моїй програмі немає помилок - це допомагає мені краще зрозуміти, що саме відбувається всередині програми.

Де ставити точку зупину? Якщо ви здогадуєтеся, який метод може створювати проблеми, поставте точку зупину перед підозрілим рядком. Якщо ви не знаєте, в чому проблема, просто поставте її в першому рядку методу `main()` і повільно покроково йдіть за програмою.

Додаток В. Як опублікувати Веб-сторінку

Інтернет-сторінки складаються з HTML файлів (з можливими вставками коду на мові JavaScript), зображень, звукових і відео файлів, тощо HTML був коротко згаданий у Розділі 7, але якщо ви плануєте стати Веб-дизайнером, вам слід приділити більше часу вивченню HTML. Почати можна тут: www.w3schools.com. Насправді, існує безліч веб-сайтів і програм, які дозволяють створювати веб-сторінку за кілька хвилин, навіть якщо ви не знаєте, як це робиться. Ці програми генерують HTML, але сам процес створення буде приховано від вас. Але, якщо ви освоїли цю книгу, я оголошую вас джуніором - **Молодшим Java Програмістом** (я не жартую!) І вивчення HTML для вас - дріб'язкова справа.

Для того, щоб розробити Веб-сторінку, зазвичай, вам досить створити один або кілька HTML файлів на диску вашого комп'ютера, але проблема в тому, що ваш комп'ютер *невидимий* для інших користувачів Інтернету. Тому, коли сторінка готова, вам потрібно скопіювати (*завантажити*) ці файли в таке місце, де їх всі можуть побачити. Одне з таких місць - диск на комп'ютері вашого *Інтернет-провайдера*.

По-перше, у вас повинна бути своя папка на комп'ютері Інтернет-провайдера. Зв'яжіться з провайдером по телефону або електронною поштою, скажіть, що ви створили HTML сторінку і хочете її опублікувати. Зазвичай вони надають таку інформацію:

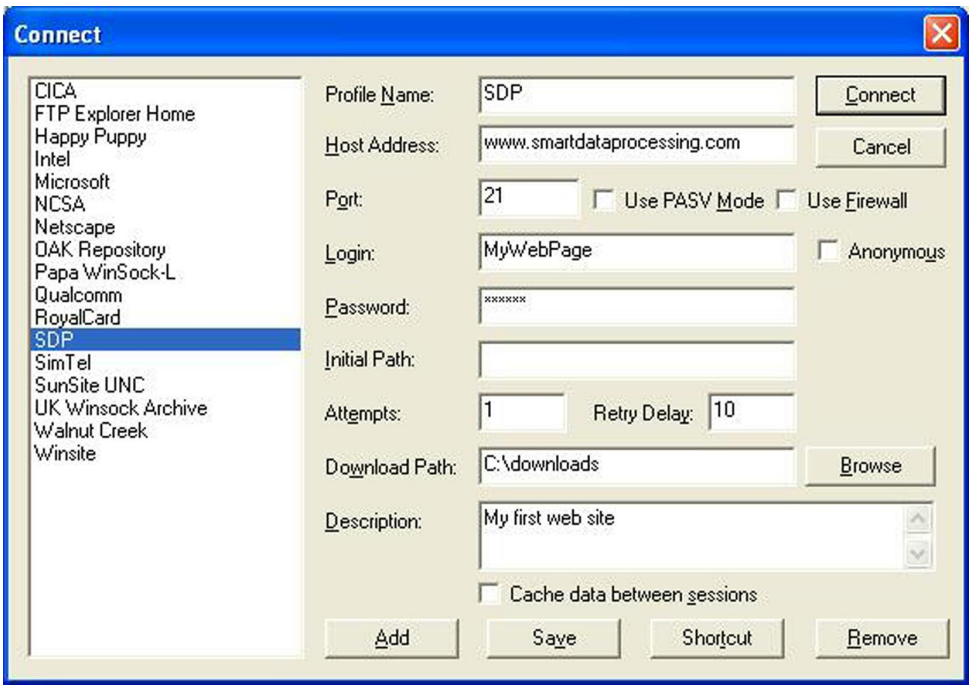
- Мережеве ім'я їх комп'ютера (головний комп'ютер).
- Ім'я папки на їх комп'ютері, де ви можете зберігати свої файли.
- Інтернет-адресу (URL) вашої нової сторінки - ви зможете дати її всім, хто захоче подивитися вашу сторінку.
- Ім'я користувача і пароль, які вам знадобляться для того, щоб завантажити нові або змінити старі файли.

Більшість провайдерів зараз можуть надати вам як мінімум 100Мб місця на їх жорстких дисках безкоштовно, що більш, ніж достатньо для біль-

шості людей.

Ще вам знадобиться програма, що дозволяє скопіювати файли з вашого комп'ютера на комп'ютер провайдера. Копіювання файлів з вашого комп'ютера на комп'ютер в інтернеті називається викладанням, а копіювання файлів з інтернету на ваш комп'ютер називається скачуванням. Ви можете викладати або завантажувати файли за допомогою програми *FTP-клієнта*.

Один з простих у використанні FTP-клієнтів - це програма *FTP Explorer*, ви можете завантажити його на сайті www.ftpx.com. Встановіть цю програму і додайте комп'ютер вашого Інтернет-провайдера в список з'єднань FTP-клієнта - запустіть FTP Explorer і перше вікно, яке ви побачите, буде вікном з'єднань. Також, ви можете клікнути на пункт Connection в меню Tools (в останніх версіях цієї програми вікна і меню виглядають інакше).

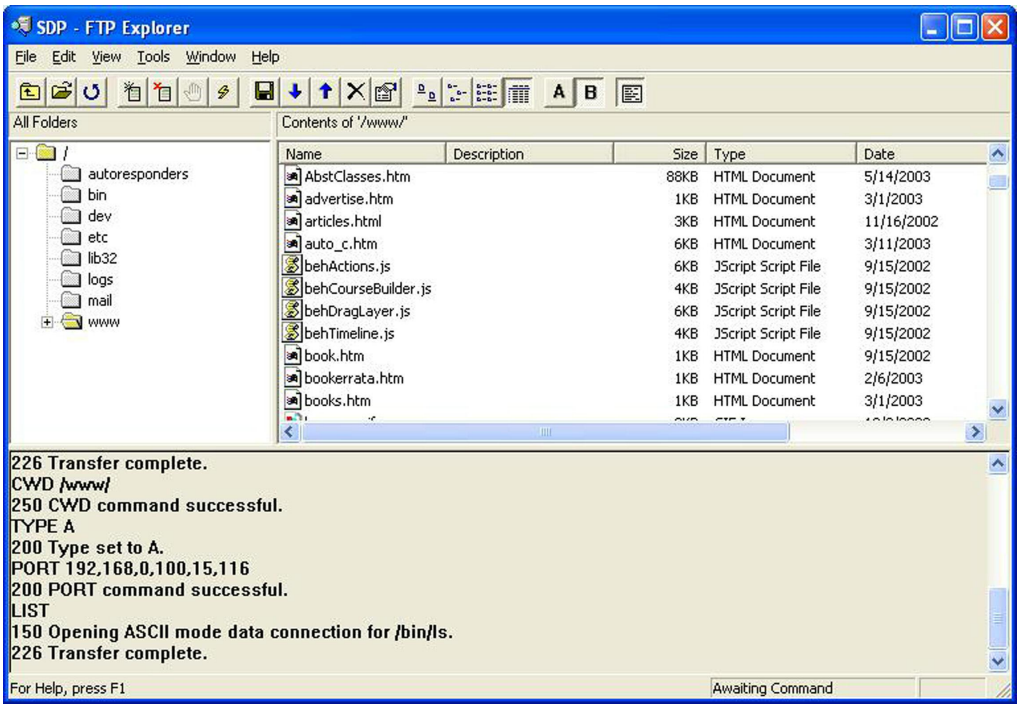


Натисніть кнопку *Add* і введіть адресу комп'ютера, ім'я користувача та пароль, які ви отримали від вашого Інтернет-провайдера. У полі Profile Name просто введіть назву вашого провайдера. Якщо ви все зробили правильно, то буде створено новий профіль з'єднання в списку доступних FTP-серверів.

Натисніть кнопку *Connect* і ви побачите папки на комп'ютері вашого

провайдера. Знайдіть вашу папку і починайте викладати файли, як описано нижче.

У панелі інструментів знаходяться дві сині стрілки. Стрілка, яка вказує вгору - для викладання файлів. Натисніть цю стрілку і ви побачите стандартне вікно, де можна увійти в папку з вашими HTML файлами. Виберіть файли, які плануєте викласти і натисніть кнопку *Open*. Через декілька секунд ви побачите ці файли на комп'ютері вашого провайдера.



Зверніть увагу на нижню частину вікна і переконайтеся, що при викладанні не було ніяких повідомлень про помилки.

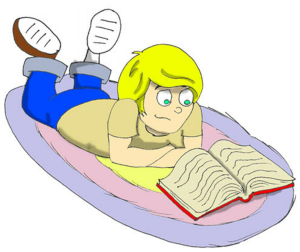
Назвіть головний файл `index.html`. Тоді URL вашої сторінки буде коротшим, і людям не потрібно буде друкувати ім'я цього файлу. Наприклад, якщо ім'я папки на комп'ютері провайдера www.xyz.com/~David та ім'я головного файлу вашої сторінки `myMainPage.html`, то адреса вашої сторінки www.xyz.com/~David/myMainPage.html. Але, якщо ім'я головного файлу - `index.html`, то адреса вашої сторінки буде коротшою - www.xyz.com/~David. Тепер, будь-хто, хто знає цю адресу, зможе побачити

її в Інтернеті. Якщо пізніше ви вирішите змінити цю сторінку, просто повторіть весь процес - зробіть зміни на вашому диску. Після цього завантажте файли, і старі файли заміняться новими.

Якщо ви вирішите стати Веб-дизайнером, наступна мова, яку потрібно вивчити - це JavaScript. Ця мова набагато простіше, ніж Java і дозволить вам зробити ваші Веб-сторінки яскравішими і цікавішими.

Також, ви можете скористатися одним з безкоштовних сервісів для зберігання HTML файлів в інтернеті, наприклад - Сайти Google (https://sites.google.com/) - прим. перекладача.

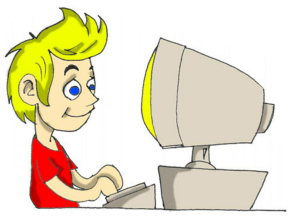
Матеріали для додаткового читання



1. How to make/create you own Web site:
http://www.thesitewizard.com/gettingstarted/startwebsite.shtml

2.The World Wide Web
http://www.w3schools.com/web/web_www.asp

Практичні вправи



Створіть веб-сторінку і опублікуйте гру “Хрестики-Нулики” з Розділу 7. Для того, щоб почати, просто завантажте на вашу сторінку файли `TicTacToe.html` і `TicTacToe.class`.

Кінець цієї книжки

... Але у мене є й інша, правда, на інгліш і не для дітей. Все-ж погляньте, а раптом сподобається? А називається вона «Java Tutorial. 24-Hour Trainer ». З книжкою йде і DVD, де я записав скрінкасти майже до всіх уроків. Її можна купити на Амазон - найбільшому американському інтернет-магазині з відмінною репутацією. Вони доставляють товари в усі країни світу. Ось URL цієї книженції:

http://www.amazon.com/Java-Programming-24-Hour-Trainer-Yakov/dp/0470889640

До зустрічі!

Індекс