Projet Java

Responsables: **Ocan Sankur**, Guillaume Scerri
(LSV, ENS Cachan)

Objectives

- Apprendre à programmer en Java
- Travailler à plusieurs sur un gros projet qui a plusieurs aspects: graphisme, interface utilisateur, réseau, concurrence, optimisation.
- Initiation à quelques notions de génie logiciel: classes/modules réutilisable, documentation, design patterns, tests unitaires, développement de test par scénarios.

Projet

- Un projet à plusieurs aspects qui nécessitent un partage des tâches et le respect des interfaces fixées.
- Le contenu du projet sera expliqué à une séance prochaine.

Page web du cours:

http://www.lsv.ens-cachan.fr/~sankur/java

Evaluation

Il y aura

- un mini devoir: **1/6** (dans un mois)
- la première étape du projet: **2/6** -(fin Mars)
- la deuxième étape du projet: **3/6** (fin Mai)

Calendrier

- 25/01 – 08/02: Trois TP Java.
- 15/02: Data limite du mini devoir.
- 29/03: Première partie
- 31/05: Deuxième partie (et soutenance)

Il y aura des TP complémentaires sur le graphisme et réseaux, gestionnaires de version ou autre sur demande.

Introduction

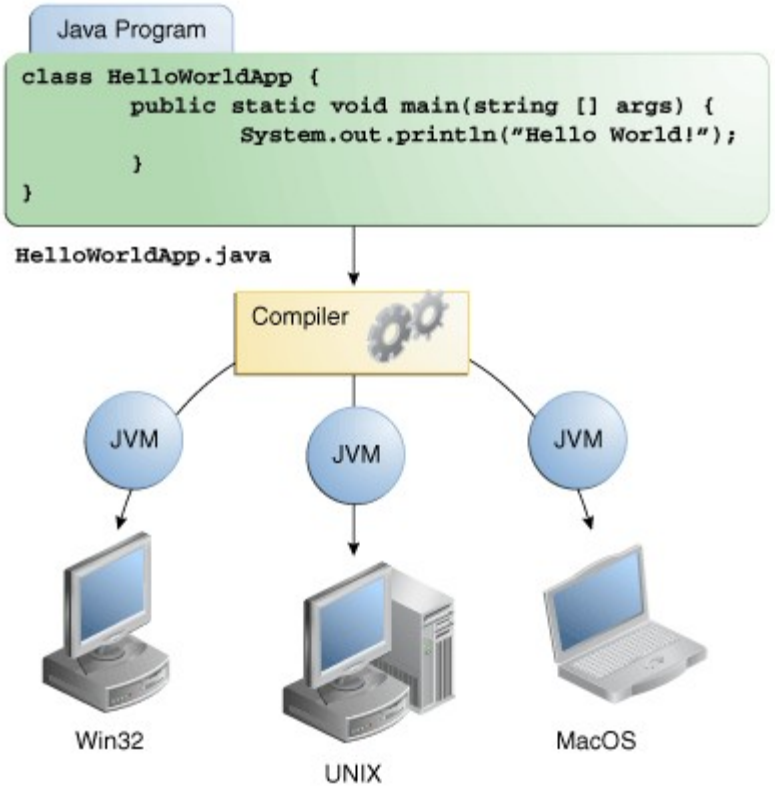
- Present the syntax of Java
- Introduce the Java API
- Demonstrate how to build
  - stand-alone Java programs
  - Java applets, which run within browsers e.g. Netscape
- Example programs

Why Java?

- Very popular general-purpose language
- It’s almost entirely object-oriented
- It has a vast library of predefined objects and operations
- It’s platform independent
  - this makes it great for Web programming
- It’s more secure (than C, C++)
- It isn’t C++

Java Virtual Machine

- The *.class* files generated by the compiler are not executable binaries
- Instead, they contain “byte-codes” to be executed by the **Java Virtual Machine**



Building JAVA Programs (on UNIX)

- Prepare the file `Foo.java` using an editor
- Invoke the compiler: `javac Foo.java`
- This creates `Foo.class`
- Run the java interpreter: `java Foo`

HelloWorld

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- This code must be put in a file named **HelloWorld.java**, that is, the name of the public class.

- *public static void main(String [] args)* is the **main** function to be executed first (at most one such function must be given).

Comments are almost like C++

- ```
  /* This kind of comment can span multiple lines
  */
  ```
- ```
  // This kind is to the end of the line
  ```
- ```
  /**
    * This kind of comment is a special
    * ‘javadoc’ style comment
    */
  ```
  ```

Primitive data types are like C

- Main data types are `int, double, boolean, char`
- Also have `byte, short, long, float`
- `boolean` has values `true` and `false`
- Declarations look like C, for example,
  - `double x, y;`
  - `int count = 0;`

Expressions are like C

- Assignment statements mostly look like those in C; you can use `=`, `+=`, `*=` etc.
- Arithmetic uses the familiar `+ - * / %`
- Java also has `++` and --
- Java has boolean operators `&& || !`
- Java has comparisons `< <= == != >= >`
- Java does *not* have pointers or pointer arithmetic

Control statements are like C

- `if (x < y) smaller = x;`
- `if (x < y){ smaller=x;sum += x;} else { smaller = y; sum += y; }`
- `while (x < y) { y = y - x; }`
- `do { y = y - x; } while (x < y)`
- `for (int i = 0; i < max; i++)`
  `sum += i;`
- BUT: conditions must be boolean !

Control statements II

```
switch (n + 1) {
  case 0: m = n - 1; break;
  case 1: m = n + 1;
  case 3: m = m * n; break;
  default: m = -n; break;
}
```

- Java also has exceptions (more on that later)

So, what is a class?

- A **class** is a blueprint for constructing objects, consisting of
  - a collection of *fields*, or *variables*
  - all the operations (called *methods*) that can be performed on those fields
- A class describes objects and operations defined on those objects

An example of a class

```
class Person {
   String name;
   int age;

   void birthday ( ) {
      age++;
      System.out.println (name+' is now '+age);
   }
   int getAge(){

      return age;

   }

}
```

An example of a class (cont'd)

```
Person p = new Person();
p.name = “John”;
p.age = 100;
p.birthday();
p.birthday();
```

→ ?

Here, **p** is an **object.**

While **Person** is its description.

A better example of a class

```
class Person {
   private String name;
   private int age;

   public Person(String name, int age){
      this.age = age;
      this.name = name;
   }

   void birthday ( ) {
      age++;
      System.out.println (name+' is now '+age);
   }
    int getAge(){ return age; }

}
```

The method *Person(String name, int age)* is a **constructor**.

An example of a class (cont'd)

```
Person p = new Person();
p.name = “John”;
p.age = “100”;
p.birthday();
p.birthday();
```

Compiler error!
Attributes are private

```
Correct (and nicer) version:
Person p = new Person(“John”, 100);
p.birthday();
p.birthday();
```

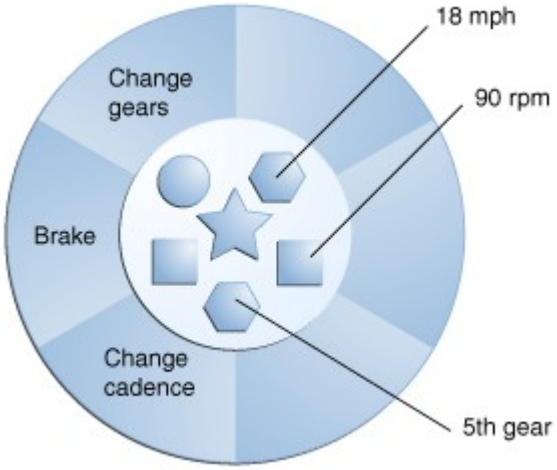
In the second version, we avoided modifying the attributes by hand. This is a good practice (see next).

Object Oriented Programming

- (Almost) any entity manipulated in your programs are objects, defined by classes. Classes:

  - **encapsulate** all data related to the object.
  - **hide** data related to the *implementation*.
  - Provide **methods** by which one modifies the state of the object.

General principle: You don't need to understand **how** a class works in order to use it. Just use the methods.



The class hierarchy

- Classes are arranged in a hierarchy
- The root, or topmost, class is `Object`
- Every class but `Object` has at least one superclass
- A class may have subclasses
- Each class *inherits* all the fields and methods of its (possibly numerous) superclasses

Example of hierarchy

```
Class Person{
   int age;
   String name;
}

class Driver extends Person {
   long driversLicenseNumber;
   Date expirationDate;
}
```

Driver is a subclass of Person.

```
Driver d = new Driver();
d.name = “Nash”; d.age = 100;
d.birthday();
```

Name conventions

- **Class names** begin with a capital letter
- All other names begin with a lowercase letter
- Subsequent words are capitalized:
  printBirthday()
  getDataFromFile()
- Underscores are not used in names:
  ~~print_birthday()~~
- These are *very strong* conventions!

Compilation

- For now, edit with your favorite text editor compile with:

```
javac *.java
```

- Or, you can try the popular IDE eclipse:

http://www.eclipse.org/

Java API and Resources

- Java has a huge standard library (The Java API).
- From now on, your primary reference shall be:

http://docs.oracle.com/javase/7/docs/api/

Very useful tutorials:

Java Language:

http://docs.oracle.com/javase/tutorial/java/index.html

Other concepts (Network, concurrency, graphics …)

http://docs.oracle.com/javase/tutorial/