Seminar 1
Introduction to Java
PROGRAMMING 3

David Rizo, Pedro J. Ponce de León

Translated into English by Juan Antonio Pérez

Departament of Computer Languages and Systems

University of Alicante

Contents

Contents

1 Features

Main Java Features

- Object oriented language: everything is an object
- Source files: *.java*
- Compilation into *bytecode*: *.class*
- Libraries in *.jar* files

- Code is executed by a *virtual machine* (JVM): cross-platform
- Main integrated development environments (IDE): *Netbeans*, *Eclipse*

2 Basic syntax

Basic syntax

Naming rules for identifiers are the same than those used in C++:

```
// This file should be stored in Clase.java
// Usually, each class goes in its own file
public class Clase {
  /* Visibility should be declared for a field */
  private int campo1;
  /**
   * Documentation, comments
   */
  private float campo2; // fields are initialized to 0
  /* Constructors return nothing */
  public Clase() {
     campo1 = 0;
  }
  /* All the methods are defined inline */
  public int getCampo1() {
    return campo1;
  }
}
```

Constants, static

Constants

Constants are defined with the reserved word `final`

```
public final int KN=10;
```

Static methods and fields

They are defined with the reserved word `static`

```
private static int contador=1;
public static final int KNN=10;
public static void incrementaContador () {
  contador++;
}
```

3 Main program

Main program

main

The entrypoint *main* is a constant static method

```
// this is a normal class
public class ClaseConMain {
  // which also includes the main method
  public static final void main(String[] args) {
    // array args contains the arguments
    // the name of the executable (unlike C++)
    // is not included in args
 }
}
```

4 Compilation and execution

Command-line compilation and execution

Compilation
Compilation is performed at the prompt

```
> javac ClaseConMain.java
```

This generates the *bytecode* file ClaseConMain.class

Execution
which will be executed at the prompt with the command

```
> java ClaseConMain
```

Seminar 1.7

5 Scalar data types

Basic data types

Scalar types
Java is a strongly-typed language. It includes the following types:

```
byte, short, int, long, float, double, char, boolean
```

Notation for specifying literals:

```
float a = 10.3f;
double b = 10.3;
char c = 'a';
boolean d = true; // or false
```

Operators
The same than in C++

```
a++; if (a==1) b=2; a = (float)b;
```

Seminar 1.8

Scalar types

Wrappers
Each scalar type has a corresponding class:

```
Byte, Integer, Float, Double, Char, Boolean
```

which can be initialized as follows:

```
Integer a = null; // null by default
a = new Integer(29);
int x = a.intValue(); // x value will be 29
```

Objects

- These *wrappers* are objects
- Variables referring to these objects are pointers and they point to `null` by default
- Memory must be allocated with `new`
- Explicit memory release is not necessary; *garbage collector* takes care of this

Seminar 1.9

6 Objects

Objects

Assignment
As both variables behave like pointers

```
Integer a = new Integer(10);
Integer b = a;
```

b points to the same instance (memory zone) as variable a. In order to duplicate objects, a new one should be instantiated with `new`

Object
The class Object represents any Java object. Therefore, an object of any class is an object of class Object at the same time.

```
Object obj = new Integer(10); // Ok
obj = new Persona(); // Ok
```

Objects

instanceof operator
The expression

```
objeto  instanceof Clase
```

returns true if 'objeto' is an object of class 'Clase', and false otherwise.

Casting (type conversion)
Similar to C++:

```
int x = 10;
float f = (float) x;
```

Casting can also be used to assign any object to a reference of an already known type:

```
Object cualquiera;
...
MiClase obj = (MiClase) cualquiera;
```

Note: to ensure safe casting, the programmer needs to be sure that 'cualquiera' is an object of type 'MiClase'.

Objects

———————————–

Comparison
This expression

```
a==b
```

compares memory addresses. Comparing objects should be roughly done like this:

```
a.equals(b)
```

Method 'equals'
To compare two objects of a class defined by us, the method 'equals' must be implemented.

```
public boolean equals(Object obj)
```

The parameter of 'equals' is a reference to an object of class 'Object', which implies that objects of any class can be passed to the method 'equals' (although usually the object will be of the same type than the one it is being compared with).

Objects

Implementation of 'equals'
In order to implement 'equals', note that the equality operation must satisfy the following properties: reflexive, symmetric, and transitive. Also, it must satisfy that

```
x.equals(null) == false // for any non-null x
```

Besides that, in order to compare the attributes of the parameter with those of the object `this`, the parameter must be converted into a reference of the class. Consequently, any implementation of 'equals' must satisfy:

```
public boolean equals(Object obj) {
  if (obj == this) return true; // both references
                      // point to the same object
  if (obj == null) return false;
  if (!(obj instanceof MiClase)) return false;
  MiClase elotro = (MiClase) obj;
  // From this point on, the attributes of both objects
  // ('this' and 'elotro') have to be compared to
  // determine whether they are equal or not.
  // Watch out! If an attribute is a reference to an
  // object, use 'equals' to compare them.
}
```

Objects

Boxing
When doing

```
Integer b = 3;
```

this is what is being done internally:

```
Integer b = new Integer(3);
```

Unboxing
On the contrary,

```
int x = new Integer(100);
```

behaves in a similar way to this:

```
int x = (new Integer(100)).intValue();
```

Objects

Exceptions: concept

- An exception is a mechanism designed to manage error situations by modifying the normal control flow of a program.
- Examples of exceptions include access to invalid memory positions, division by zero or referring to a negative index of an array.
- In their simplest form, when an exception happens, the invoked method aborts its execution and returns control to the invoking method; this operation repeats until going back to the main program, which stops the execution of the application.
- Exceptions are objects instantiated from classes whose name usually has the form <Name>Exception.

Objects

The two most common exceptions are:

NullPointerException
It is thrown when accessing an uninitialized memory position (because the object has not been created with `new`). For instance:

```
Integer a, b;
if (a.equals(b)) {
// NullPointerException exception thrown
.....
```

ArrayIndexOutOfBoundsException
It is thrown after accessing to an invalid component of an array.

```
int [] v = new int[10];
v[20] = 3;
// ArrayIndexOutOfBoundsException exception thrown
```

7 Strings

Strings

String
Java comes with a class for string manipulation

```
String s = new String("Hola");
```

Recall how to compare:

```
s == "Hola" // wrong
s.equals("Hola") // right
```

toString()
Classes usually define the `toString()` method.

```
Float f = new Float(20);
String s = f.toString();
```

Seminar 1.17

String

Concatenation
Strings can be concatenated with the + operator, non-string types are converted into string first

```
int i=100;
... "El␣valor␣de␣i␣es␣=␣" + i;
```

Behind the scenes 4 objects are created:

```
String s1 = new String("El␣valor␣de␣i␣es␣=␣");
String s2 = new Integer(i).toString();
String s3 = s1.concatenate(s2); // creates another object
```

StringBuilder
To reduce the number of objects, use `StringBuilder` [1]:

```
StringBuilder sb = new StringBuilder();
sb.append("El␣valor␣de␣i␣es␣=␣");
sb.append(i);
sb.toString(); // string object
```

Seminar 1.18

8 Arrays

Arrays

Arrays are similar to dynamic arrays in C++:

```
int [] v; // v is a pointer to null
```

Allocation:

```
v = new Integer[100];
```

Now `v` components (`v[0], v[1], ...`) are `null` and should be initialized

```
// v.length is the array length
for (int i=0; i<v.length; i++) {
  v[i] = new Integer(0);
  // or v[i] = 0 (equivalent thanks to boxing)
}
```

Allocation of literal arrays:

```
int [] v = new int []{1,2,3,4,5};
```

Arrays can be copied with a loop or by using the static method `arraycopy` in class `System`

```
int [] origen = new int []{1,2,3,4,5};
int [] destino = new int[origen.length];
System.arraycopy(origen, 0, destino, 0, origen.length);
```

Seminar 1.19

[1] `StringBuffer` is synchronized but less efficient

9 Methods

Methods, functions

Methods
Everything is an object in Java: member functions are called methods

Parameters
Parameters are passed by value

```
void F(int a, String x, int [] v) {
  a=10; // this will not affect the original value
  x += "Hola"; // this creates a new object
               // (new takes place behind the scenes)
               // and the original remains unaffected
  v[2] = 7; // arrays are references,
            // v[2] is changed in the original array
}
```

Seminar 1.20

10 Writing

Console output

Output
Printing through standard output:

```
System.out.print("Cadena"); // no newline printed at the end
System.out.println(10+3); // newline printed
```

Printing through standard error:

```
System.err.println("Ha␣ocurrido␣un␣error...");
```

Seminar 1.21

11 Flow control

Flow control

Similar to C++. Java 1.7 will allow to use strings in the `switch` sentence.

Loops
To iterate over a vector:

```
List<String> v = Arrays.asList("Azul", "Verde", "Rojo");

for (int i=0; i<v.size(); i++) {
  System.out.println(v.get(i));
}

for (String color: v) {
  System.out.println(color); // one color per line
}

// using iterators
Iterator<String> iterador = v.iterator();
while (iterador.hasNext()) {
  String color = iterador.next();
  System.out.println(color); // one color per line
}
```

Seminar 1.22

12 Packages

Packages

`Package`
Class files are physically organized into directories, which make up `packages`

A class will belong to a package if:

- The class file is located in the directory corresponding to the package
- The name of the `package` is declared at the beginning of the class file (directory names are separated with dots)

```
package prog3.ejemplos;
class Ejemplo {
}
```

In this case, file `Ejemplo.java` should be stored in the directory `prog3/ejemplos`.

Packages

Modularization
Packages are recommended but not mandatory. Before using a class belonging to a different package, it must be imported.

```
package prog3.ejemplos;
// class belonging to Java API (libraries)
import java.util.ArrayList;

// class created by us but listed in another package
import prog3.otrosejemplos.Clase;

// This "includes" all the classes in package
// prog3.practicas;
// It is recommended not to use the *:
import prog3.practicas.*;
```

Classes in `java.lang` are included by default:

```
// not necessary:
import java.lang.String;
```

13 Java libraries

Java API

API
The Java platform includes an extensive library of classes (`http://download.oracle.com/javase/6/docs/api/overview-summary.html`)

Vectors
`ArrayList` can be used for lineal dynamic storage.

```
import java.util.ArrayList;
.....
ArrayList v = new ArrayList();
v.add(87); // internally v.add(new Integer(87));
v.add(22); // increases the size of the vector

// get will return an Object
// It should be casted to an Integer (87 in this case)
Integer a = (Integer)v.get(0);
v.get(100); // an exception is thrown (execution error)
```

Java API

Generic classes
To avoid casting of variables, the type stored in the vector must be indicated:

```
ArrayList<Integer> v = new ArrayList<Integer>();
v.add(87); // internally v.add(new Integer(87));
Integer a = v.get(0); // no casting needed
System.out.println(v.size());
```

Initialization
Vector initialization:

```
List<String> v = Arrays.asList("Azul", "Verde", "Rojo");
// v is initialized as an ArrayList object
```

Seminar 1.26

14 CLASSPATH

CLASSPATH

ClassNotFoundException
This exception is usually thrown before starting the execution of the main program, when the JVM tries to load all the class files required by the application and one of them is not found.

Ejemplo

```
mihome> java Main
Exception in thread "main" java.lang.NoClassDefFoundError: Main
Caused by: java.lang.ClassNotFoundException: Main
  at java.net.URLClassLoader\$1.run(URLClassLoader.java:202)
  at java.security.AccessController.doPrivileged(Native Method)
  at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
  at sun.misc.Launcher\$AppClassLoader.loadClass(Launcher.java:301)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
```

Required files must be *in the classpath*.

Seminar 1.27

CLASSPATH

classpath
The *classpath* is the list of the directories where the JVM looks for the .class files required by an application.
By default, it includes:

- current directory
- JRE (Java Runtime Environment) libraries (.class files for the Java API).

Seminar 1.28

CLASSPATH

Let's assume that your main program is compiled in a file called `Main.class` under `/home/mihome/miapp`.

First scenario
All classes in the same directory. `package` not used. From that directory,

```
/home/mihome/miapp> java Main
```

Seminar 1.29

CLASSPATH

Scenario 2
`java` is run from a different directory than the one containing our `.class` files. The *classpath* must be defined:

Option 1
Define the environment variable **CLASSPATH**, which should include the paths to all the directories containing .class files (it is recommended to use absolute paths).

```
.../otrodirectorio> export CLASSPATH=/home/mihome/miapp

.../otrodirectorio> java Main
```

Option 2
Use the option **-cp** or **-classpath** when invoking `java`:

```
.../otrodirectorio> java -cp /home/mihome/miapp Main
```

Seminar 1.30

CLASSPATH

Scenario 3
The .class files are distributed through different directories.

```
> export CLASSPATH=/home/mihome/milibjava:/home/mihome/miapp
> java Main
```

or you can also used **-cp**. Warning: '-cp' cancels any previous value of CLASSPATH; you must use one or the other, but not both.

Seminar 1.31

Packages and *classpath*

Let's assume that our classes are organized into packages.

Project's structure
modelo/MiClase.java:

```
package modelo;
public class MiClase {...}
```

mains/Main.java:

```
package mains;
public class Main {...}
```

modelo/m2/OtraClase.java:

```
package modelo.m2;
public class OtraClase {...}
```

The *classpath* has to contain the **root directory** of package structure.

Seminar 1.32

Packages and *classpath*

Project directory is `/home/mihome/miapp`. In order to use **OtraClase** in Main.java:

```
import modelo.m2.OtraClase;
```

When executing

```
.../otrodir>java -cp /home/mihome/miapp Main
```

in order to execute the class Main, 'java' will search in the *classpath* directories for a directory `modelo/m2` which contains `OtraClase.class`.

Seminar 1.33

15 Archivos JAR

JAR files

jar is a Java utilty (similar to **tar**) which aggregates many Java classes and information on their packages in a single archive with extension **.jar**.

JAR
From the working directory:

```
> jar cvf MisClases.jar *.class
```

Now, MisClases.jar can be freely moved to a different directory (for example, /home/mihome/libs) and used from any other directory:

```
> java -cp /home/mihome/libs/MisClases.jar Main
```

To list the contents of a .jar archive:

```
> jar tvf MisClases.jar
```

Seminar 1.34

16 Documentation

Documentation

Javadoc
Java uses a format for comments based on annotations (`@`) embedded in comments starting with `/**`.

```
package paquete;
/**
 * Example class: brief description of the class...
 * @author drizo
 * @version 1.8.2011
 */
 public class Ejemplo {

 /**
  * This field is used for...
  */
  private int x;

  private int y; // This comment will be ignored by javadoc


   /**
    * Constructor: objects....
    * @param ax Radius of...
    * @param ab If true, then...
    */
   public Ejemplo(int ax, boolean ab) {
    ....
   }

    /**
     * Getter.
     * @return x: number of times...
     */
    public double getX() {
        return x;
    }
}
```

Generation of documentation
HTML documentation is generated with

```
javadoc -d doc package1 package2
```

which creates a directory `doc` with the documentation contained in the classes in both packages.

17 ANT

ANT

ant
Apache Ant is a software tool for automating software build processes. It is similar to Make. It will be used in this subject as part of the script used for assignment evaluation.

'Ant' tutorial
Follow this link for a brief introduction to ant:
`http://www.vogella.com/articles/ApacheAnt/article.html`