Software and Programming I

Introduction

Basic elements of Java

Roman Kontchakov

Birkbeck, University of London
Module Information

- Time: 6–9pm on Wednesdays in the Spring term
- Lectures: UCL Christopher Ingold, Chemistry LT
  6–7.30pm (A–K) / 7.30–9pm (L–Z)
- Labs: UCL Foster Court B29
  7.30–9pm (A–K) / 6–7.30pm (L–Z)

- web: http://www.dcs.bbk.ac.uk/~roman/sp1
  moodle (http://moodle.bbk.ac.uk)
Assessment

- In-Class Tests (weeks 5 & 9): 25% (10% & 15%)
- Two-hour examination in summer 2015: 75%
the module draws on Chapters 6–9
the lab classes are based on exercises suggested in JFE
Syllabus

- review of primitive data types and arrays
- branching and loops
- objects and classes
- methods and constructors
- instance and static variables and methods
- public and private access
- inheritance and polymorphism
- exception handling
- strings and input/output
- basic data structures and algorithms, recursion
Java Development Environments

- Java Development Kit (JDK), Java SE 8
  \[8\]
  Standard Edition

- BlueJ
  (a public project to make programming in Java easier)

- Eclipse
  (multi-language and extensible,
  free and open source software)
Java Compilation and JRE

source
HelloWorld.java

compiler
javac

bytecode
HelloWorld.class

Java Runtime Environment

JRE = JVM + (standard) classes

JDK = JRE + tools (compiler, etc.)

do not confuse the two!

running program

Virtual Machine (VM)
Compilation v Interpretation

C

**source**
HelloWorld.c

**compiler**

**machine code**
HelloWorld.exe

**executed on**
hardware

JavaScript

**source**
HelloWorld.js

**interpreted by**
**interpreter**

**executed on**
hardware
JDK: Editing

- source code can be edited in any text editor (e.g., Notepad, emacs, ...)

- *MS Word* caveat:
  - by default, Word does not save in ASCII text format

- make sure to save the code before compiling!

- the file name must be the same as the name of the class (with the .java extension) *(case sensitive!)*
Compiling with JDK

- invoke the command-line compiler:
  
  javac <source>.java

- compiles <source> and all classes it depends on into Java bytecode files (<source>.java, etc.)

- for example:
  
  javac HelloWorld.java
  produces the file HelloWorld.class

  (provided there are no errors)

- make sure the compiler and JVM are in the command path (PATH)
Execution in JDK

- starting the Java Virtual Machine (JVM):
  
  ```java
  java <source>
  ```

- the named class is loaded and execution is started (other classes are loaded as needed)

- only possible if the class has been compiled into Java bytecode

- How does the JVM know which of the methods in the class to execute?
/* HelloWorld.java
   Purpose: printing a hello message on the screen
*/

public class HelloWorld {
    // each program is a class
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}

NB. different styles of curly bracket layout
NB. watch out for semicolons
NB. names and reserved words are case-sensitive
Coding in BlueJ

- BlueJ organises files into **projects**, stored in a project-specific directories on disk
  
  _do not forget to backup!_

- types of BlueJ files:
  - bluej.pkg: contains information about classes in the package (one per package)
  - bluej.pkh: backup of the package file
  - *.java: Java source code (text files, one per class)
  - *.class: Java bytecode (binary, one per class)
  - *.ctxt: BlueJ context file with extra information about the class (one per class)
Software is Free

- available on BBK’s network
  - JDK (allows one to compile and execute programs)
  - BlueJ (preferred Java IDE)

- installing BlueJ for home use
  - download JDK from http://www.oracle.com/technetwork/java/javase/downloads/
Objects and Classes

- In object-oriented programming, tasks are solved by collaborating objects:
  
  e.g., System.out is an **object**

- In Java, a programmer does not implement a single object

- A **class** describes a set of objects with **the same behaviour**:
  
  e.g., the String class describes the behaviour of all strings—**methods** length(), substring(i, j), ...

- The **new** operator constructs objects (=**instances of classes**):

  Scanner in = new Scanner(System.in);
Methods

- A **method** is a named sequence of instructions
- **Parameter values** are supplied when a method is called
- The **return value** is the result that the method computes
- Method $\approx$ algorithm $\approx$ mathematical function
Example: Function as a Method

Consider $y = x^2$

```java
public class PrintSquares {
    // compute $x^2$
    public static int sq(int x) {
        return x * x; // x is a parameter variable
    }

    public static void main(String[] args) {
        System.out.println(7 + "^2=" + sq(7));
        System.out.println(9 + "^2=" + sq(9));
    }
}
```
Variables

- A **variable** is a storage location with a name
- When declaring a variable, you specify
  - the **type** of its values
  - and usually its **initial value**

```java
int cansPerPack = 6;
```

- Variable names must start with a letter (or _) and the remaining characters must be letters, _, or numbers (cannot be a reserved word)
- Names are **case-sensitive**
  by convention, variable names start with a lower-case letter
**Primitive Data Types**

- **int** 32-bit two’s complement integer
  
  \((-2,147,483,648 \text{ to } 2,147,483,647)\)

- **long** 64-bit two’s complement integer

- **short** 16-bit two’s complement integer

- **byte** 8-bit two’s complement integer

- **double** double-precision 64-bit IEEE 754 floating point

- **float** single-precision 32-bit IEEE 754 floating point

- **boolean** Boolean value (true or false)

- **char** 16-bit Unicode character
Variable Assignment

An assignment statement stores a new value in a variable, replacing the previously stored value (so, the previous value is lost)

cansPerPack = 8;

The assignment operator = does not denote mathematical equality

\[ x = x + 1; \]

- Pascal uses := for assignment

“Software is getting slower more rapidly than hardware becomes faster”
(Niklaus Wirth, 1995)
The for Loop

The **for** loop is used when instructions are executed repeatedly and a value runs from a starting point to an ending point with a constant increment (or decrement).

```java
for(int i = 5; i <= 10; i++) {
    sum = sum + i;
}
```
1 public class PrintHelloWorld {
2   public static void main(String arg[]) {
3       for (int i = 1; i <= 10; i++)
4           System.out.println("Hello, World!");
5   }
6 }

How many times is the phrase printed?
- for(int i = 0; i < 10; i++) {
  ...
}
- for(int i = 0; i <= 10; i++) {
  ...
}
Arithmetic Expressions

- Java uses the natural precedence of arithmetic operations: *, %, / before +, -

if in doubt, use brackets

- $2 * 6 / 4 + 5 - 2 * 3 = ?$
- $2 * 6.0 / (5 + 3) - 2 * 3 = ?$
- $2 * 6 / (5 + 3) - 2 * 3 = ?$

NB: beware of the unintended integer division
Strings

- strings are sequences of characters:
  ```java
  String name = "Harry";
  ```

- the `length` method yields the number of characters in the string:
  ```java
  int n = name.length();
  ```
  the empty string "" is of `length` 0

- use the `+` operator to concatenate strings
  ```java
  String lastname = "Morgan";
  String fullname = name + " " + lastname;
  ```

**NB:** whenever one of the arguments of `+` is a string, the other argument is converted to a string
Substrings

- string that is made up of the characters starting at position \(i\) and containing all the characters up to, *but not including*, the position \(j\):

  ```java
  String greeting = "Hello!";
  String sub = greeting.substring(0, 2);
  ```

- all characters from the position \(i\) to the end of the string:

  ```java
  String tail = greeting.substring(2);
  ```

- these are all examples of instance methods of the class String
Strings and Characters

- 
  string **positions** are counted starting with 0
  
  ```java
  char start = name.charAt(0);  // 'H'
  char last = name.charAt(name.length() - 1);  // 'y'
  ```

- do not confuse characters ('H')
  and strings containing a single character ("H")
Scope of a Variable

- The scope of a variable is the part of the program in which it is visible
  - the entire method of a method’s parameter variable
  - from its declaration until the end of the block, for a local variable
  - the for statement, for a local variable declared in a for statement

- Two variables can have the same name provided their scopes do not overlap
Scope: an Example

What is wrong here?

```java
public static int sumOfSquares(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        int n = i * i;
        sum = sum + n;
    }
    return sum;
}
```
Overview

- compiler, bytecode and JVM
- interpretation v compilation
- JDK and BlueJ
- variables: declaration and initialisation
- primitive data types
- arithmetic operations
- strings
- class and instance
- methods