

Introduction to Software Development (ISD)

David Weston and Igor Razgon

Autumn term 2013

Course book

- The primary book supporting the ISD module is:
Java for Everyone, by Cay Horstmann, 2nd Edition, Wiley, 2012

- There is a Companion Student Website accessible from

www.wiley.com/college/horstmann

Scroll down to find this book, click on "Visit the Companion Sites", and select "Student Companion Site"

- You can download the source code for some of the program examples, additional worked examples, additional chapters 11-15, and other resources

Lectures, Labs, Assessment

- Each Thursday we will generally start with a lecture from 6pm to 7.40pm and then a lab session from 8 to 9
- There will also be a weekly lab session on Fridays 6-9pm where you can continue with the week's assignments and seek help if needed
- The ISD module will be assessed 50% by written exam in May/June and 50% from three practical pieces of coursework that will be set during this term (the first one will be set in Week 4).

Accessing ISD module resources

- You can access copies of the lecture notes, lab sheets, coursework, coursework solutions etc. from the Bloomsbury Learning Environment (BLE), at

moodle.bbk.ac.uk

Aims of Week 1

- To learn about computers and programming
- To compile and run your first Java programs
- To recognize compile-time and run-time errors
- To describe an algorithm with pseudocode
- To understand the use of strings, integers and floating-point numbers in Java programs
- To be able to declare and initialize variables and constants
- To write arithmetic expressions and assignment statements
- To create programs that read and process inputs, and display the results

Computers and Computer Programs

- A computer program is a sequence of instructions that tell a computer what to do
- Computers execute very basic instructions in rapid succession
- Computers are very flexible in the tasks they can carry out because they can execute different programs – it is the programs (the software) that give computers their flexibility
- Programming is the design and implementation of computer programs, and this is what this module is about
- We will start by developing some simple programs in the Java programming language in the early weeks, and then some more complicated ones as the module progresses
- This module aims to give you the foundations that you need to progress onto learning about and using more advanced features of Java, and learning and using other programming languages

Computers and Computer Programs

- The main components of a computer (the hardware) include:
 - the Central Processing Unit (CPU), which is responsible for executing programs: it executes basic instructions such as arithmetic operations, fetching data from memory or secondary storage and storing processed data back
 - the main memory, or Random Access Memory (RAM), where program instructions and data are stored during program execution
 - secondary storage, such as hard disks, CD/DVD drives, flash drives etc. where programs and data are stored over the longer term (secondary storage devices are cheaper and slower to access than main memory, but their contents persist when the computer is powered down, unlike RAM)
 - input/output (IO) devices that allow users to interact with the computer: keyboard, mouse, screen, printer

Computers and Computer Programs

- IO devices allow human users to interact with computer programs:
 - users can enter input required by computer programs using a keyboard, mouse, other input devices
 - the computer transmits the output produced by computer programs to the screen, printer, speakers etc.

Running a computer program

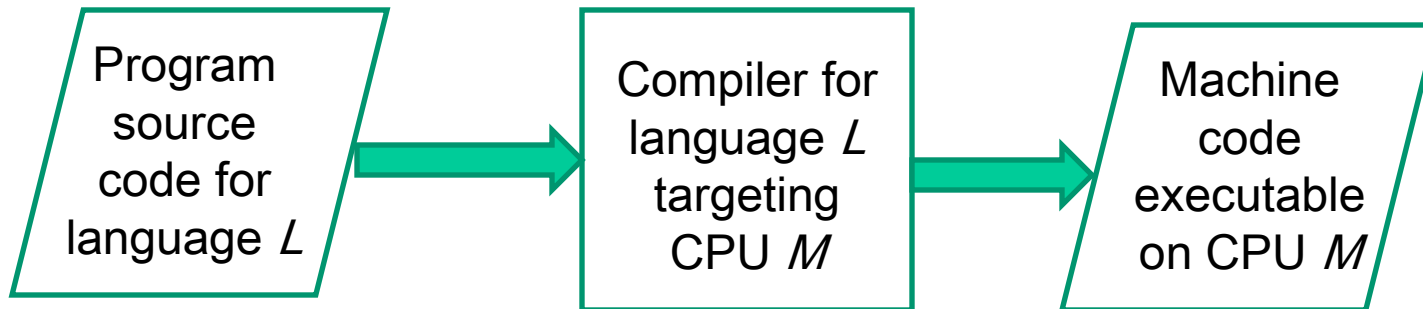
- Program instructions and the data that programs need are stored on secondary storage
- When a program is started, it is brought into main memory
- The CPU runs the program one instruction at a time. The program may request and react to user input as it runs
- As directed by these instructions and the user, the CPU reads data, modifies it, and writes it back to memory, to secondary storage, or transmits it to output devices

High level programming languages

- Java is an example of a high-level programming language
- High-level programming languages are easier to write programs in than are low-level languages such as *assembly languages* (assembly languages represent symbolically the basic machine code instructions of a particular CPU architecture)
- High-level languages are more oriented towards the human programmer and they include language abstractions away from a specific CPU architecture and machine code
- Different high-level languages lend themselves naturally to different programming paradigms i.e. different styles of writing computer programs
- Java lends itself most naturally to object-oriented programming
- We will discuss programming paradigms later in this course

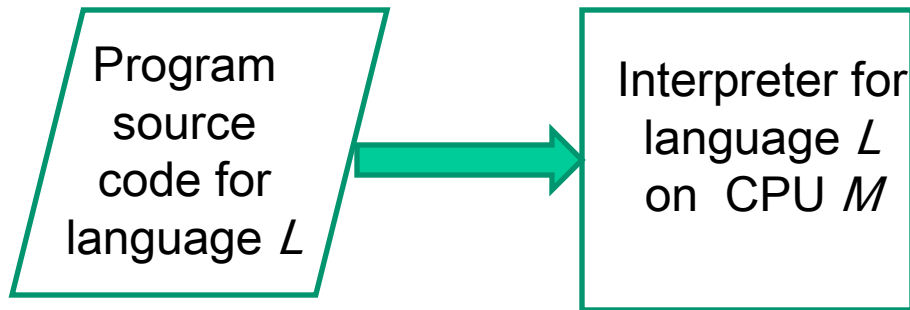
Compilers and Interpreters

- A compiler is a computer program that translates programs written in one programming language (typically a high-level language) into another (typically machine code)
- If the target language is machine code, then the translated program can be executed directly on the target machine:



Compilers and Interpreters

- An interpreter is a computer program that takes as input a programs written in a particular programming language and executes that program on a target machine



Compilers and Interpreters

- Java combines these techniques:
 - the Java compiler translates Java programs into class files containing Java bytecode
 - the Java Virtual Machine (JVM) executes Java bytecode:



The Java Language

- Java programs are 'portable', in the sense that they can be run on different computers without needing modification:
 - Java programs are compiled into Java bytecode instructions for the JVM, rather than into an actual machine language
 - The JVM is available for most computer operating systems, making Java programs platform-independent
- Java programs can also be run within Web Browsers (*applets*)
- Java has a many existing library packages e.g. for
 - graphics, user interfaces, networking, database interactionwhich helps to write useful, portable programs (see Appendix D of *Java for Everyone* for a list of commonly used packages)
- We will explore and use some of these in this module as we go along

Java Development Environments

- You need to install the Java SDK (Software Development Kit) to create Java programs
 - We have this installed in the computer science labs
- There are many development tools available for Java
 - We will be using BlueJ on this course
- Components of an IDE include:
 - Source code Editor, which helps programmers by:
 - listing line numbers of a program
 - highlighting different aspects of the program
 - auto-indenting source code
 - Window for listing program output
 - Debugger

Our First Program

- This is the traditional 'Hello World' program in Java:

```
1 public class HelloPrinter
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, World!");
6     }
7 }
```

- We will type in and run this program in the Lab session. Note that:
 - Java has a set of *reserved words* which can't be used for other purposes in your programs e.g. **public**, **class**, **static**, **void**
 - Java uses some special characters which also can't be used for other purposes e.g. { } () ;
 - Java is case-sensitive, so **Public** is not the same as **public**, **System** is not the same as **system** etc.


```
1 public class HelloPrinter
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, World!");
6     }
7 }
```

Line 1 declares a class called HelloPrinter, whose definition starts at Line 2 and ends at Line 7

- Every Java program consists of one or more classes
- The word **public** indicates that a class can be used elsewhere in a program
- In Java, every program source file can contain at most one public class; the name of this class must match the name of the file containing it

```
1 public class HelloPrinter
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, World!");
6     }
7 }
```

Line 3 declares a method called `main`, whose definition starts at Line 4 and ends at Line 6

- a method contains Java statements that carry out a particular task
- each statement must end with a semicolon (;)
- every Java program has exactly one `main` method, which is the entry point where the program starts; usually there are other methods in the program as well
- We will discuss the meaning of `static` and `void` in the weeks to come
- `String[] args` indicates that the `main` method is passed a parameter – we will discuss parameters and methods in the weeks to come

```
1 public class HelloPrinter
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, World!");
6     }
7 }
```

- The statement in Line 5 outputs the message **Hello, World!** to the screen
- It calls the method `System.out.println` (which is pre-defined in the Java library)
- The **"Hello, World!"** between round brackets is a parameter that is passed to this method
- **"Hello, World!"** is an example of a string. Strings in Java are enclosed in quotation marks
- You can also print out numbers using the `System.out.println` method e.g. `System.out.println(7); System.out.println(3+4);`

println and print

- The `System.out.println` method prints a string or a number and then starts a new line e.g.

```
System.out.println("3 + 4 = ");
```

```
System.out.println(3 + 4);
```

results in the output:

```
3 + 4 =
```

```
7
```

- There is also a `System.out.print` method that prints a string or a number and then does not start a new line e.g.

```
System.out.print("3 x 4 x 5 = ");
```

```
System.out.println(3 * 4 * 5); // * is multiplication
```

results in the output:

```
3 x 4 x 5 = 60
```

Writing, compiling and running a Java program

- In the lab session, you will be writing some simple Java programs
- The general program development process is:
 1. design your program (we will discuss this first step shortly);
 2. type in your program using the IDE editor;
 3. compile your program;
 - if it fails to compile because it violates the Java language rules, you need to find the errors, correct them and re-compile the program; these are called *compile-time*, or *syntax*, errors;
 4. once the program has successfully compiled, you can run it;
 - if it does not behave as you intended (e.g. its output is wrong or it stops unexpectedly), then you need to find the errors, correct them and go back to step 2; these are called *run-time*, or *logic*, errors.

Designing programs: Algorithms

- An algorithm is a “recipe” or “set of instructions”, which can be translated into a particular programming language to be run on a computer
- We can define algorithms using a structured form of English, called pseudocode, which is independent of any particular programming language
- Writing pseudocode allows us to focus on designing how an algorithm works without having to worry about the syntactic details of implementing it in a particular programming language
- Pseudocode supports:
 - sequencing of instructions
 - repetition of a group of instructions
 - calling of sub-algorithms
 - following different branches of the “recipe”

An example algorithm

- Suppose you put £1,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?
- Here is some pseudocode that defines an algorithm to solve this problem (as we will see in the upcoming weeks, this pseudocode can be translated straightforwardly into Java code):

```
year = 0;                // set year to 0
balance = 1000;          // set balance to 1000
while (balance < 2000) {
    year = year + 1;
    balance = balance x 1.05; // a 5% increase
}
output year value
```

Variables

- Computer programs hold temporary values in named memory locations called *variables*
- We declare a variable in a program by stating what type (and hence what size) of variable we need and what name we will use to refer to it e.g.

```
int cansPerPack;
```

cansPerPack



- We can then set the contents of the variable i.e. assign a value to it:

```
cansPerPack = 6;
```

cansPerPack



- We can combine declaration and assignment and use just one statement:

```
int cansPerPack = 6;
```


Types of Variables

- There are three common types of variables that we will discuss this week:

A whole number type: `int`

A number with a fraction part type: `double`

A sequence of characters type: `String`

There are other numeric types too in Java (see Special Topic 2.1 in Java for Everyone) which we will introduce as we need to during this course



- After you have declared and initialised a variable you can use it e.g.

```
int cansPerPack = 6;
double canVolume = 12.0;
System.out.print("Volume of a pack is ");
System.out.println(cansPerPack * canVolume);
```

Number Literals

When a number such as 2 or 12.0 occurs in a Java program it is called a *number literal*.

Table 2 Number Literals in Java

Number	Type	Comment
6	int	An integer has no fractional part.
-6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	double	A number with a fractional part has type double.
1.0	double	An integer with a fractional part .0 has type double.
1E6	double	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type double.
2.96E-2	double	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
 100,000		Error: Do not use a comma as a decimal separator.
 3 1/2		Error: Do not use fractions; use decimal notation: 3.5

Common Errors

- You must declare a variable before you use it. So the compiler will complain about this:

```
double canVolumeInLitres = 12.0 * litrePerOunce;  
double litrePerOunce = 0.0296;
```

- You must initialize a variable's contents before you use it. So the compiler will complain about this:

```
int bottles;  
int bottleVolumeInLitres = bottles * 2;
```

Naming Variables

- Pick a name for the variable that describes its purpose e.g. `canVolume` is better than `cv`
- Java has some rules about permissible variable names:
 - Use only letters, digits or the underscore (`_`)
 - also, a variable name can't start with a digit
 - No spaces are permitted in a variable name; you can show separate words using 'camelHump' notation e.g. `canVolume`
 - Variable names are *case-sensitive* e.g. `canVolume` and `canvolume` are different names
 - Java reserved words can't be used as variable names (see Appendix C of *Java for Everyone* for the full list of reserved words in Java)
 - By convention (but not mandatory), variable names start with lower case and class names with upper case letters in Java programs (so it is easier to distinguish them)

Comments in Java programs

- There are three forms of comments:

`//` single-line comment, or rest of line to the right

`/*`

multi-line comment

`*/`

`/**`

multi-line comment that explains the purpose of a class

`*/`

- Use comments to add explanations for other people who read your code so that they can understand what it does more easily. The Java compiler ignores comments.

Example program

```
1  /**
2   * This program computes the volume (in liters) of a six-pack of soda cans.
3   */
4  public class Volume1
5  {
6      public static void main(String[] args)
7      {
8          int cansPerPack = 6;
9          double canVolume = 0.355; // Liters in a 12-ounce can
10
11          System.out.print("A six-pack of 12-ounce cans contains ");
12          System.out.print(cansPerPack * canVolume);
13          System.out.println(" liters.");
14      }
15 }
```

- Lines 1-3 are Javadoc comments for the class Volume1
- Line 9 uses a single-line comment to clarify the unit of measurement

Assigning new values to variables

Example:

counter

5

```
counter = counter + 1;
```

1. This statement does the right hand side of the assignment first i.e. finds the value stored in the variable counter and adds 1 to it:

$$5 + 1 = 6$$

2. It then stores the result in the variable named on the left side of the assignment operator (which is counter again in this case):

counter

6

Reading Input

You might need to ask the user to enter input that is needed by a program:

1. Import the `Scanner` class from its package `java.util` by writing this at the start of the program file:

```
import java.util.Scanner;
```

- Java classes are grouped into *packages*
- The `import` statement is used to allow a program to access a class (e.g. `Scanner`) from a package (e.g. `java.util`).
- The classes in the `java.lang` package are automatically available in every program and do not have to be explicitly imported.

Reading Input

2. Create an object `in` of type `Scanner` where you need it in the program:

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

- We will be discussing classes and objects in detail later in the course; for now, use this statement as shown.

(On the right-hand-side, `System` is a class in the `java.lang` package; `in` is an object in that class, and so is `out`.)

Reading Input

We can now use existing Java methods available in the Scanner class to get the input the program needs:

```
int cansPerPack = in.nextInt();  
    /* waits for the user to input an integer  
       and then sets the contents of the  
       variable cansPerPack to this value */
```

```
double canVolume = in.nextDouble();  
    /* waits for the user to input a double and  
       then sets the contents of the variable  
       canVolume to this value */
```

See example program on next slide:

```
/** This program asks the user to specify the number of cans
    in a pack and the volume (in litres) of a can, and outputs
    the total volume of the pack
*/
import java.util.Scanner;

public class Volume2 {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter number of cans per pack : ");
        int cansPerPack = in.nextInt();
        System.out.print("Enter volume of a can : ");
        double canVolume = in.nextDouble();
        System.out.print("The volume of a pack is ");
        System.out.println(cansPerPack * canVolume);
    }
}
```

Constants

- It is good practice to declare values that will not change as 'constants' in the program
 - To declare a constant use the reserved word `final` before the type e.g.

```
final double VAT_RATE = 0.175;
```

- Then this constant name can be used instead of the value e.g.

```
double cost = in.nextDouble();  
double totalCost = cost * VAT_RATE;
```

- Constants are usually declared near the beginning of a class
- A new value cannot be assigned to a constant (unlike a variable)

Arithmetic and Expressions

- Java supports the basic arithmetic operators: +, -, *, /
- The order of precedence is (highest to lowest): parentheses; exponent; * and /; + and -

- Maths:

$$\frac{a + b}{2}$$

$$b \times \left(1 + \frac{r}{100}\right)^n$$

- Java:

$$(a + b) / 2$$

$$b * \text{Math.pow}(1 + r / 100, n)$$

- Such combinations of variables, literals, operators, method calls and parentheses are called expressions

Integer Division

- If both numbers are of type integer, you need to be careful e.g.
 $11 / 4$
 returns the value 2 i.e. the fractional part of the answer is dropped
 (it is *truncated*)
- To find the remainder in integer division, use the 'modulus' operator %

```
int first = 11, second = 4, answer, remainder;  
answer = first / second;      // set to 2  
remainder = first % second;  // set to 3
```
- If one of the numbers is of type double, then the overall result is automatically a double e.g.
 $11 / 4.0$
 $11.0 / 4$
 $11.0 / 4.0$
 all return the value 2.75

Strings

- We can declare variables of type `String` e.g.

```
String name = "Mary";
```

where "Mary" is a string literal

- We can use the method `length` to find the number of characters in a string:

```
int n = name.length();
```

(n will be assigned the value 4)

- An empty string (of length 0) is shown as ""

String concatenation

- We can append (*concatenate*) one string onto the end of another:

```
String firstName = "Mary";  
String lastName = "Smith";  
String name = firstName + lastName;  
           // name now has value "MarySmith"
```

- Or:

```
String name = firstName + " " + lastName;  
           // name now has value "Mary Smith"
```


String Input

- We can read a string from the input using the `next` method, which reads the next word:

```
System.out.print("Please enter your name: ");  
String name = in.next();
```

- We can read an entire line from the input using the `nextLine` method:

```
System.out.print("Please enter your address: ");  
String address = in.nextLine();
```

Extracting part of a String

- The `substring` method returns a portion of a String, starting at a given position, for a number of characters:

0	1	2	3	4	5
H	e	l	l	o	!

```
String greeting = "Hello!";
```

```
String sub = greeting.substring(0, 3);
```

```
// sub has value "Hel" and length 3-0 i.e. 3
```

```
String sub2 = greeting.substring(3, 5);
```

```
// sub2 has value "lo" and length 5-3 i.e. 2
```

```
len = greeting.length(); // len has value 6
```

```
String sub3 = greeting.substring(0, len);
```

```
// sub3 has value "Hello!" and length len-0 i.e. len
```

Aims of Week 1



- To learn about computers and programming
- To compile and run your first Java programs
- To recognize compile-time and run-time errors
- To describe an algorithm with pseudocode
- To understand the use of strings, integers and floating-point numbers in Java programs
- To be able to declare and initialize variables and constants
- To write arithmetic expressions and assignment statements
- To create programs that read and process inputs, and display the results

Week 1 Homework

- Complete Lab Sheet 1 – not assessed. Solutions will be posted next week on the BLE.
- Read Chapters 1 and 2 of *Java for Everyone* and do the self-check questions as you go along – the answers are at the end of each chapter!
 - You can skip Sections 2.2.5 on “Converting Floating-Point Numbers to Integers”, 2.5.5 on “Strings and Characters”, and 2.3.2 on “Formatted Output” for now – we will look at these topics next week.
- Make sure you read How To 1.1 (Describing an algorithm with pseudocode)
- If you have time, do some of the review and programming exercises from Chapters 1 and 2 of *Java for Everyone*

Example Variable Declarations

Table 1 Variable Declarations in Java

Variable Name	Comment
<code>int cans = 6;</code>	Declares an integer variable and initializes it with 6.
<code>int total = cans + bottles;</code>	The initial value need not be a constant. (Of course, <code>cans</code> and <code>bottles</code> must have been previously declared.)
 <code>bottles = 1;</code>	Error: The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.2.
 <code>int bottles = "10";</code>	Error: You cannot initialize a number with a string.
<code>int bottles;</code>	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.
<code>int cans, bottles;</code>	Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

From: Java for Everyone by Cay Horstmann

Arithmetic Expressions

Mathematical Expression	Java Expression	Comments
$\frac{x + y}{2}$	<code>(x + y) / 2</code>	The parentheses are required; <code>x + y / 2</code> computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	<code>x * y / 2</code>	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	<code>Math.pow(1 + r / 100, n)</code>	Use <code>Math.pow(x, n)</code> to compute x^n .
$\sqrt{a^2 + b^2}$	<code>Math.sqrt(a * a + b * b)</code>	<code>a * a</code> is simpler than <code>Math.pow(a, 2)</code> .
$\frac{i + j + k}{3}$	<code>(i + j + k) / 3.0</code>	If i , j , and k are integers, using a denominator of 3.0 forces floating-point division.
π	<code>Math.PI</code>	<code>Math.PI</code> is a constant declared in the <code>Math</code> class.

Shorthand for Incrementing and Decrementing

- Incrementing (+1) and decrementing (-1) integer types is so common that there are shorthand version for each

Long Way	Shortcut
<code>counter = counter + 1;</code>	<code>counter++ ;</code>
<code>counter = counter - 1;</code>	<code>counter-- ;</code>