Cranfield University

IBIX-JAV:
Programming using Java

Sun

ORACLE

Dr Tomasz Kurowski

Lecturer in Bioinformatics

t.j.kurowski@cranfield.ac.uk

www.cranfield.ac.uk

NetBeans

Module engagement QR code

If you are unable to scan this code, please contact SAS Admin – **seeaadmin@cranfield.ac.uk**

Module Objectives

By the end of this week, you should be able to:

1. Develop programs using standard Java tools
2. Write simple programs using primitive and reference data types, control statements, methods, and arrays.
3. Design create and use methods
4. Understand Object-oriented programming
5. Develop programs with a Graphical User Interface
6. Establish a firm foundation on Java concepts

Module structure

	Mon 25/11	Tue 26/11	Wed 27/11	Thu 28/11	Fri 29/11
all-day					
08					
09	09:00 - 10:30 Introduction to Java Programming B111 LR 9	09:00 - 10:30 Control Statements B111 LR 9	09:00 - 10:30 Classes and Objects B111 LR 9	09:00 - 10:30 File I/O and Exception Handling B52A PC Lab (Vincent Building)	09:00 - 10:30 Packaging and Distribution B111 LR 9
10					
11	11:00 - 12:00 Variables and Operators B111 LR 9	11:00 - 12:00 Methods B111 LR 9	11:00 - 13:00 GUI Programming B111 LR 9	11:00 - 13:00 Practical Session 04 B52A PC Lab (Vincent Building)	11:00 - 13:00 Practical Session 06 B111 LR 9
12	12:00 - 13:00 Simple I/O and Dialog Boxes B111 LR 9	12:00 - 13:00 Data Storage B111 LR 9			
13					
14	14:00 - 15:30 Practical Session 01 B52A PC Lab (Vincent Building)	14:00 - 15:30 Practical Session 02 B52A PC Lab (Vincent Building)	14:00 - 15:30 Practical Session 03 B52A PC Lab (Vincent Building)	14:00 - 15:30 Practical Session 05 B52A PC Lab (Vincent Building)	14:00 - 15:30 Assignment Handling B52A PC Lab (Vincent Building)
15					
16	16:00 - 17:30 Practical Session 01 B52A PC Lab (Vincent Building)	16:00 - 17:30 Practical Session 02 B52A PC Lab (Vincent Building)	16:00 - 17:30 Practical Session 03 B52A PC Lab (Vincent Building)	16:00 - 17:30 Practical Session 05 B52A PC Lab (Vincent Building)	16:00 - 17:30 Practical Session 07 B52A PC Lab (Vincent Building)
17					

Introduction to Java

Java Module Format

Some teaching each day, then practical sessions

Plenty of Programming exercises – Can be emailed, reviewed but not formally assessed

Practical solutions (for **some** exercises) posted on Canvas

What Is Java?

Characteristics of the language & alternatives

Getting Started With Java Programming

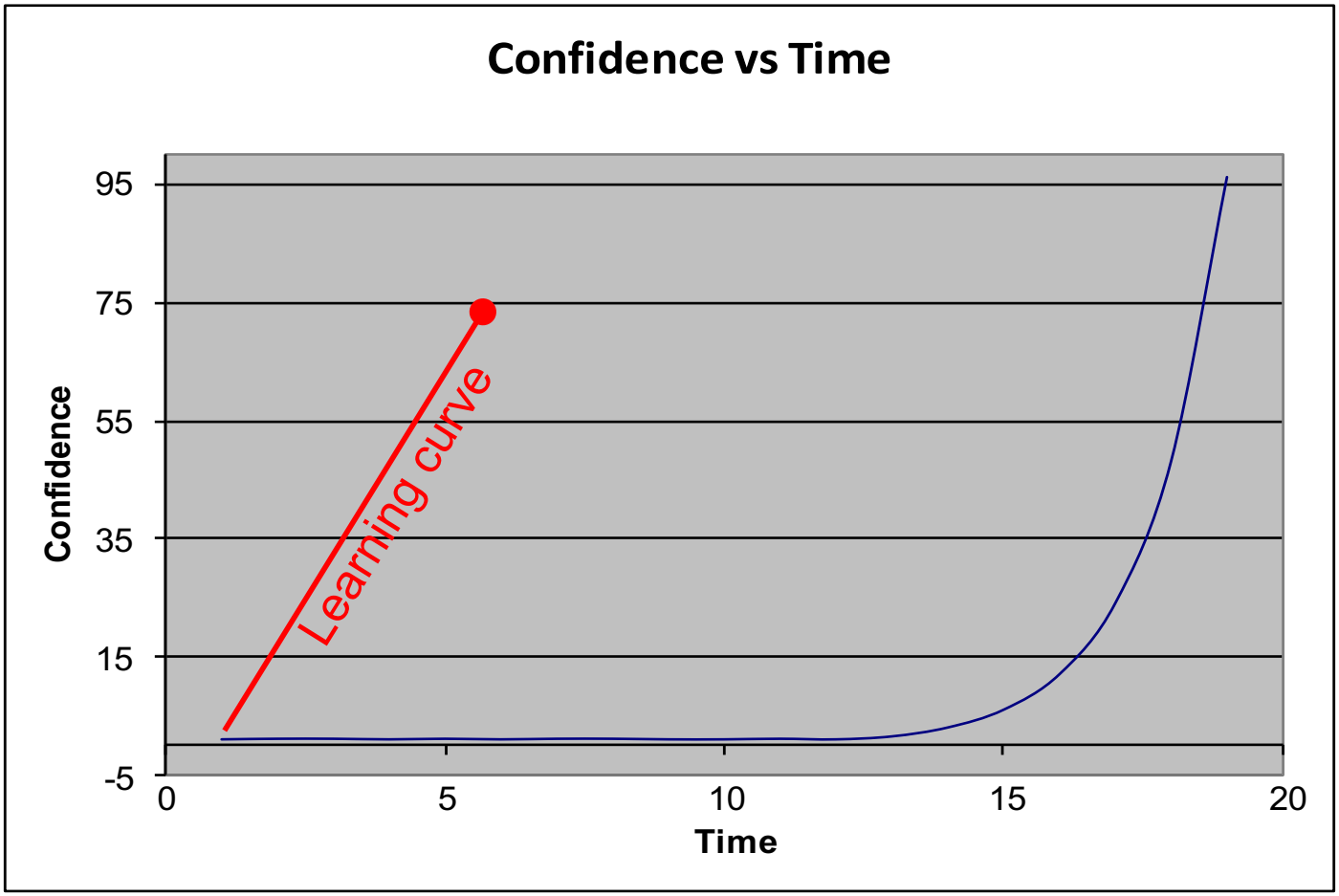
Create, Compile and Run a Java Application

A practical subject – you need to practice!

We will gradually move from the left side to the right side of the brain!

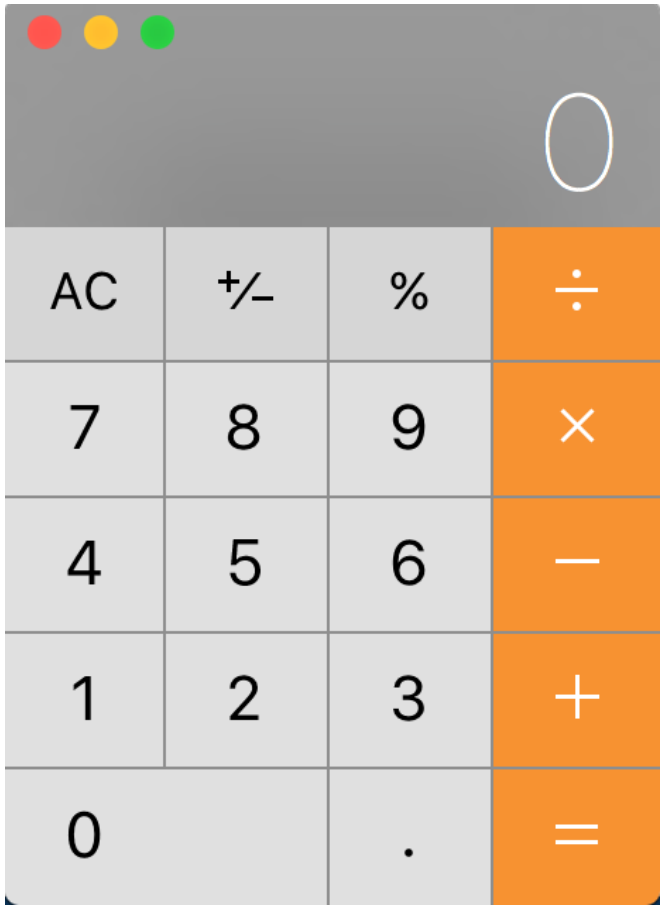
Mon Tue Wed Thu Fri Assignment

Learning to Program



What is a Program ?

- Example Instructions:
  - Input a number
  - Input some characters
  - Do a calculation
  - Output a number
  - Draw a graphical image to the screen
  - Respond to the user clicking the mouse



Accuracy of Instructions

- Programming a computer is a precise activity – no room for ambiguity. The following instructions appear sensible at first glance, but have errors :

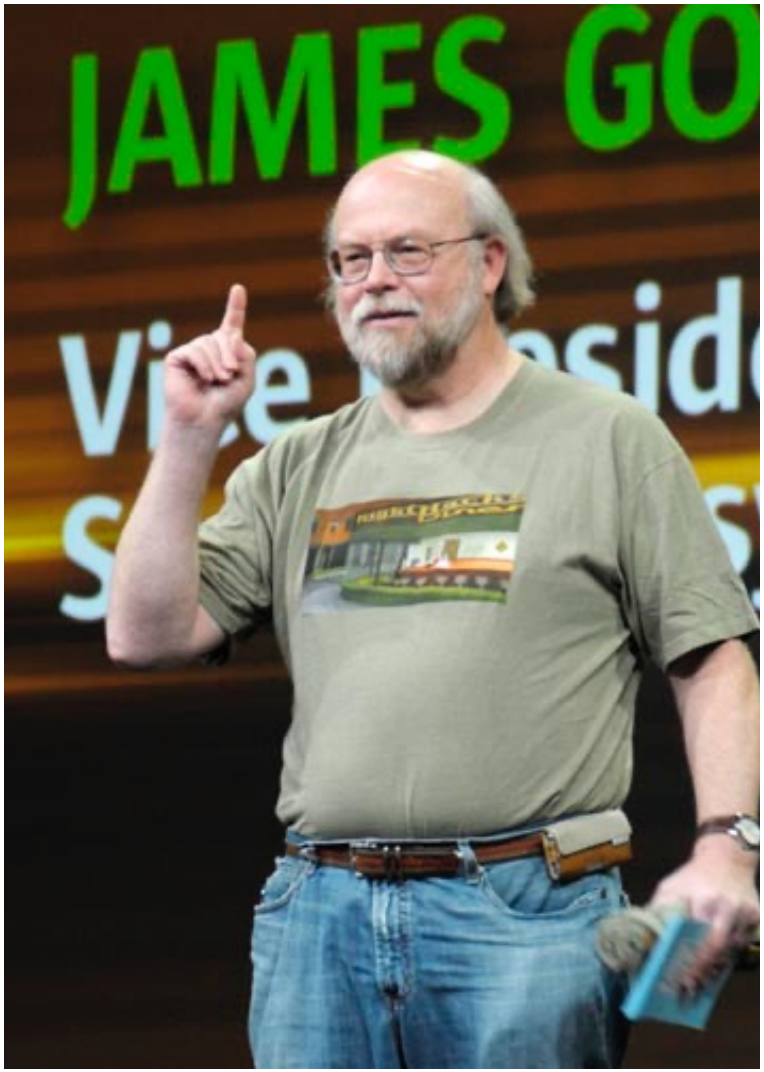
- **Logic is important**
  - "Only take this ride if you are over 7 or under 70"
  - (*A 6 year old is under 70 – replace or with and*)
  - Computers are machines – they will execute accurately and reliably the exact instructions given to them.
  - Garbage In – Garbage Out

History of Java

- Initiated by James Gosling in 1991
- The language, initially called *Oak*

- He aimed to build virtual machine and a language that had a familiar C/C++ style of notation

- Java 1.0 was released in 1996 by Sun Microsystems as "*Write once, run everywhere*"



History of Java

- Nov, 2006: Sun released much of Java as open source software under the terms of the GNU General Public License (GPL).
- 2009-2010: Oracle bought Sun Microsystems.
- 2010 – Present: Despite scepticism by developers (including James Gosling) - Java continue to thrive as an open-source PL

Primary goals

1. It should be "simple, object oriented, and familiar".
2. It should be "robust and secure".
3. It should be "architecture neutral and portable".
4. It should execute with "high performance".
5. It should be "interpreted, threaded, and dynamic".

Language history

HotJava - The first Java-enabled Web browser

Bytecode designed to run as ‘Applet’ within browser

JDK Evolutions - J2ME, J2SE, and J2EE

Emphasis has evolved from Web Browser add-in to general purpose platform independent programming language.

Microsoft – Visual Java, J++, now C#

Java Versions

Oracle Java SE Support Roadmap*†				
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**	**Sustaining Support**
8 (LTS)**	March 2014	March 2022	December 2030*****	Indefinite
9 - 10 (non-LTS)	September 2017 - March 2018	March 2018 - September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	January 2032	Indefinite
12 - 16 (non-LTS)	March 2019 - March 2021	September 2019 - September 2021	Not Available	Indefinite
17 (LTS)	September 2021	September 2026****	September 2029****	Indefinite
18 (non-LTS)	March 2022	September 2022	Not Available	Indefinite
19 (non-LTS)	September 2022	March 2023	Not Available	Indefinite
20 (non-LTS)	March 2023	September 2023	Not Available	Indefinite
21 (LTS)***	September 2023	September 2028	September 2031	Indefinite
22 (non-LTS)***	March 2024	September 2024	Not Available	Indefinite
23 (non-LTS)***	September 2024	March 2025	Not Available	Indefinite
24 (non-LTS)***	March 2025	September 2025	Not Available	Indefinite
25 (LTS)***	September 2025	September 2030	September 2033****	Indefinite

The latest Long Term Support version is 21, but anything 8 or newer will work for us.

...What's so great about Java?!

- Java is simple – the language that is, 10,000's library components
- Java is portable - Write Once, Run anywhere
- Java is object-oriented – supports modern software design
- Java is distributed – network aware. Client / Server apps
- Java is robust – Error trapping - No low level access – use Native code
- Java is secure – No low level access!
- Java manages memory for you – Garbage collection!
- Java's performance – JIT
- Java is multithreaded – run many tasks in parallel
- Java is popular – just look at the job adverts
- Especially popular within the bioinformatics community – eg. GATK, Cytoscape, MapOptics, Omicsbox

...What's so great about Java?!

- Platform independence
  - It runs everywhere! → Java Virtual Machine (JVM)
  - Breakthrough in programming!

...What's so great about Java?!

Before java, other programming languages promised platform independence, by providing different compilers for different platforms.

The compiler does not translate Java into machine code, but to the JVM language, the **Bytecode**

The Java Runtime Environment (JRE) runs the bytecode on the JVM

Bytecode

- This is what Java Virtual Machine Executes.
- As the name suggests each bytecode instruction is one byte in length
- As a Java programmer... you don’t really need to understand Java Bytecode

Why is Java so great?!

- Java is Object-Oriented!
- Java treats program components as virtual objects

Objects and classes

- Data: information in a form that computers can use
- Information: is any knowledge that can be communicated (incl. ideas & concepts)
- Object: is the combination of data and the operations that can be applied to it

- Java treats program components as **objects** (that are **instances** of **classes)**

Object Oriented programming

1. Objects have identities (instances):

- e.g. a1 → 

a2 → 

- Although both apples are nearly identical, but they are not the same, each apple (object) has it's own location in the computer memory

a1 & a2 are both **objects** and **instances** of a **class** like "Object" or "Fruit"

Object Oriented programming

1. Objects are instances of a non-primitive type (a class)

- e.g. a1 → 

o1 → 

- a1 and o1 are two different objects of the same class "Fruit"

Object Oriented programming

1. Objects have properties (states)

- e.g. a1 → 

o1 → 

Property	a1	o1
Colour	green	orange
type	Fruit	Fruit

Object Oriented programming

1. Objects have behaviour (methods)

- e.g. Fred →

Lilly →

behaviour	Fred	Lilly
Walk	Yes	Yes
Bark	Yes	No
Climb a tree	No	Yes

Getting Started

Java Tools

Java SDK – free download from Oracle

NetBeans (what we will use this week – also free to download)

Eclipse (Open Source, IBM sponsored)

Some editors have simple Java support

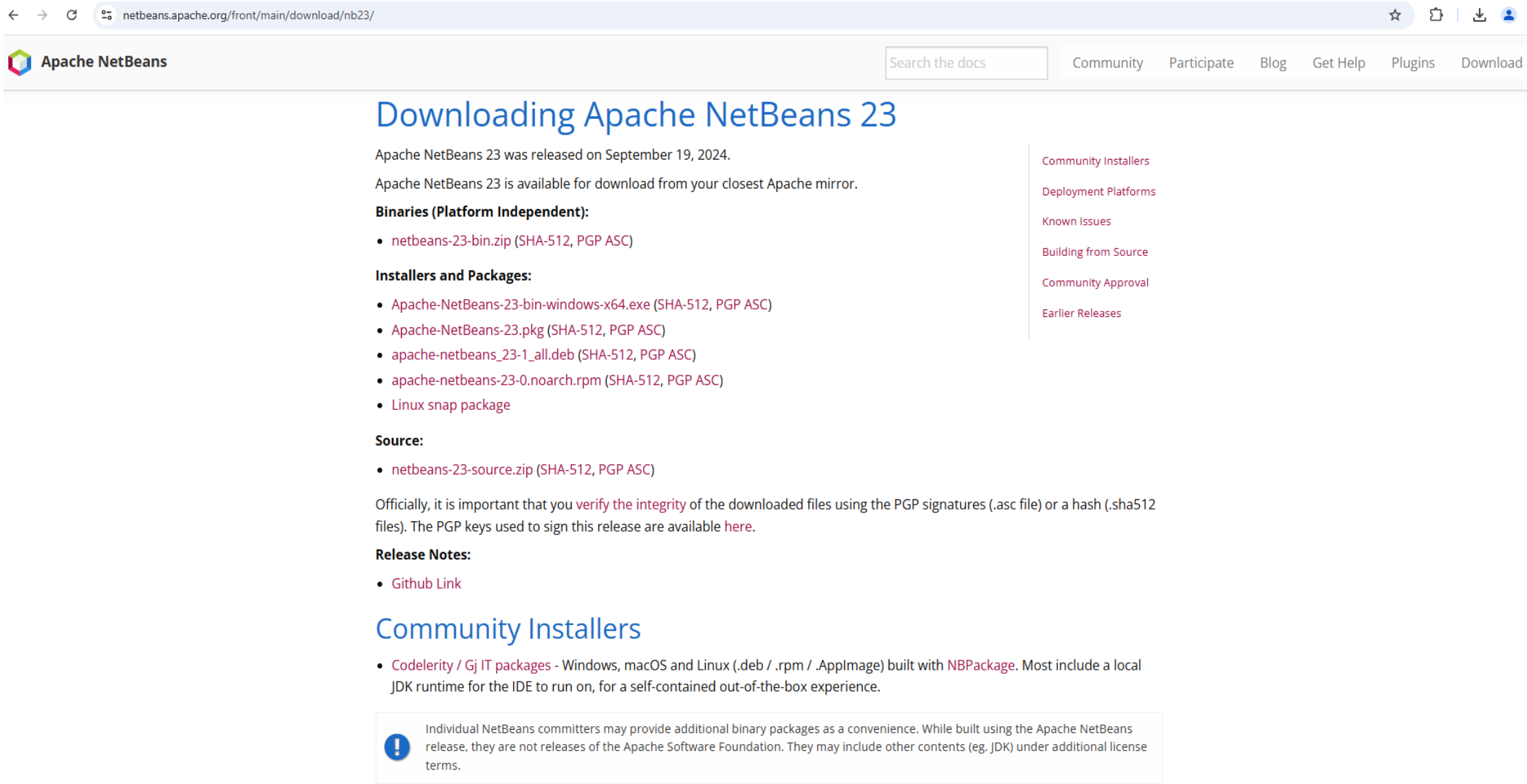
Syntax Highlighting, Error Parsing
Integrated Compile / Run commands
e.g TextPad, UltraEdit, Prog.Notepad, Emacs

- For the purpose of this module

PCLabs have Netbeans installed

For the purpose of this module mainly Netbeans will be used, but you are free to use any IDE according to your personal preference

To install Netbeans (netbeans.apache.org)



Java tools that may be useful for this module

- We will be using Netbeans, which also runs under Windows, MacOS or Linux.
- However, you can use any basic text editor with command line.
- Netbeans (**recommended**) is useful for big projects & GUI designs – So is Eclipse
- All software are ‘free’ & can be installed on your own computers.

- A Simple Java Application
- Compiling Programs
- Executing Applications

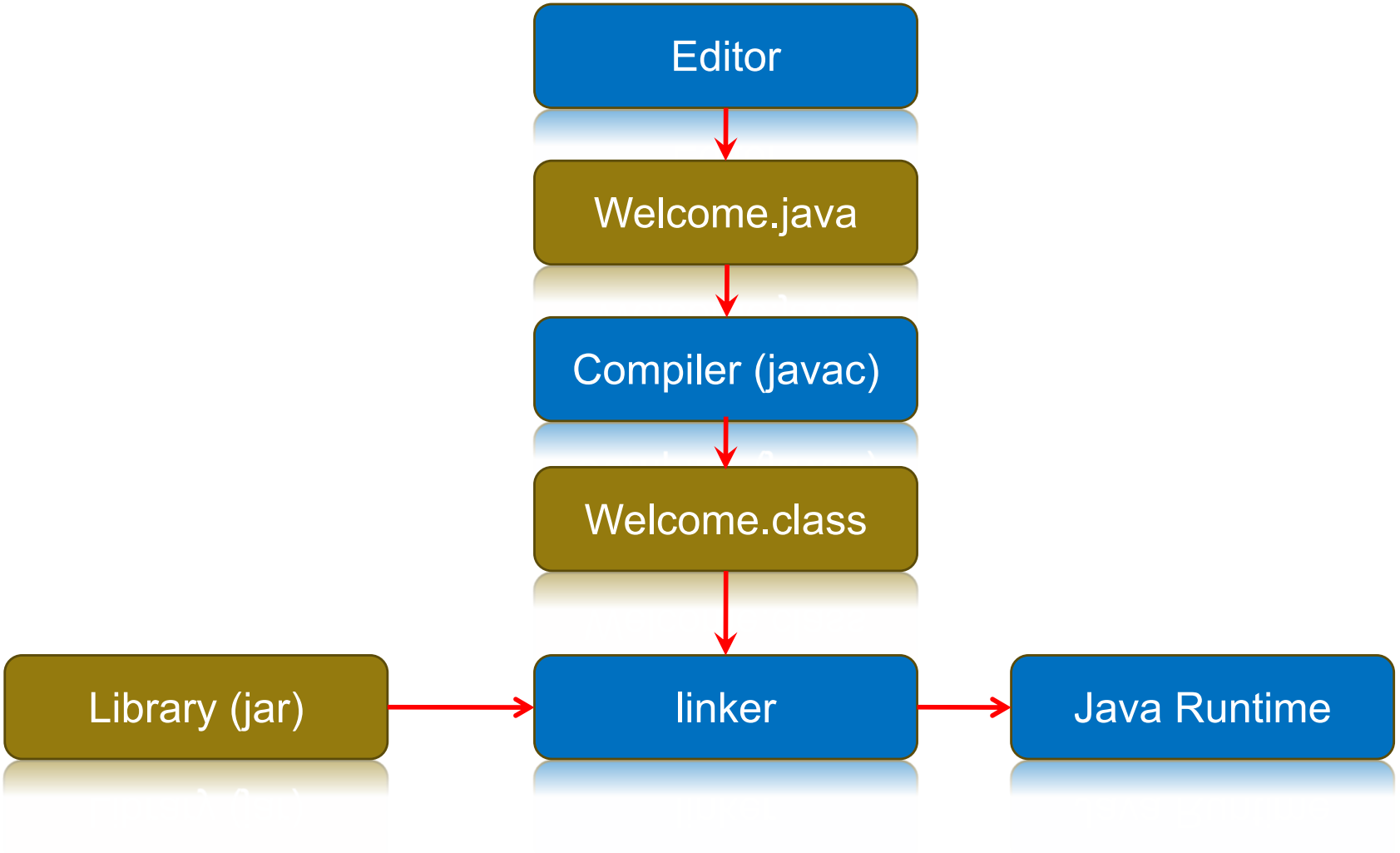
A Simple Application

Example 1.1

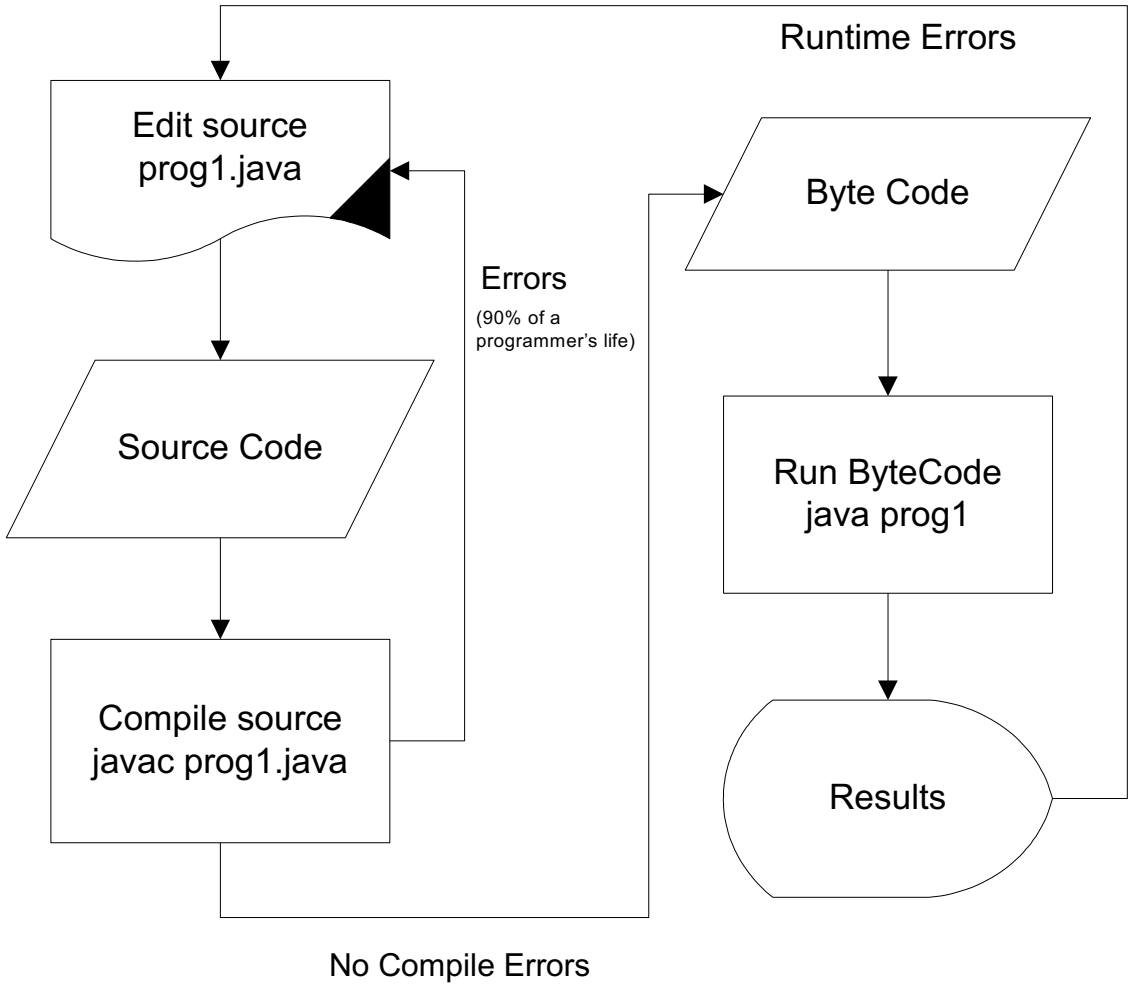
```
// This application program prints
// Welcome to Java! to the terminal
// (This is a comment by the way)

public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

• Enter this into an editor of your choice and save as Welcome.java – note classname matches filename.

Creating and Compiling Programs

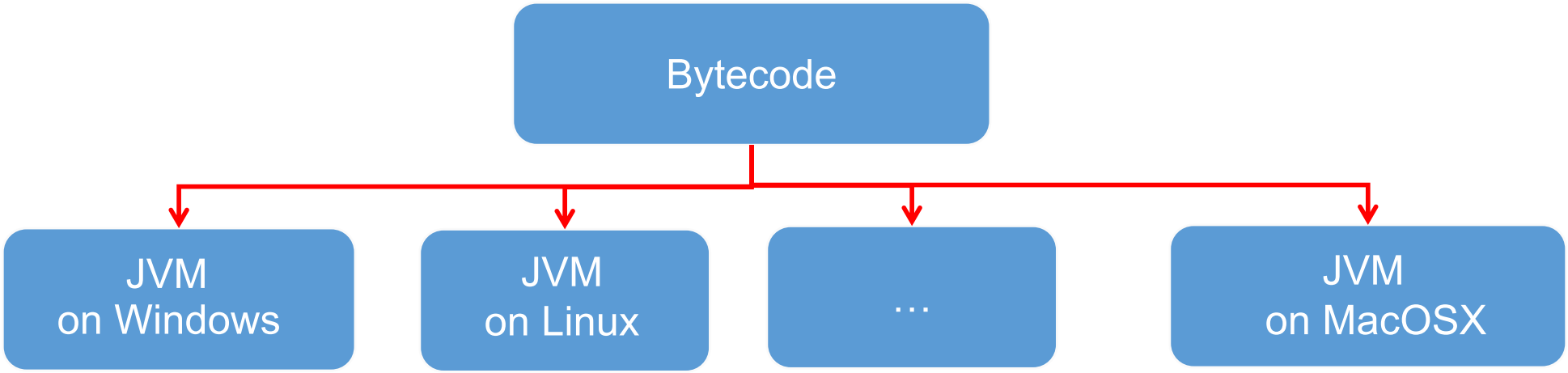


Creating and Compiling Programs



Executing Applications

- On the command line :
- javac Welcome.java
- java Welcome
- output ….



Compile (with JDK)

- Compile a java program from command line:
  - `>javac Welcome.java`

- Common syntax errors
  - Case must match (uppercase/lowercase)
  - File name must be same as class file name
  - Spelling - braces – punctuation

- If no errors:
- No message indicates successful compilation and linking
  - You can compile multiple Java files at once
  - `> javac *.java`

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Blocks
- Modifiers
- Statements
- Classes
- Methods
- The main method

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Comments

```
// This application program prints
// Welcome to Java! to the terminal
// (This is a comment by the way)

public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Comments

- There are two forms of Java Comments :
- Two slashes // (single line comment)
- When the compiler sees //, it ignores all text after // until the end of the same line.
- Block Comment /* …. */ (multiple lines)
- When the compiler sees /* it scans for the next */ (over multiple lines if necessary) ignoring any text between /* and */.
- Use plenty of comments – programs are written once but read many times

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

```
// This application program prints
// Welcome to Java! to the terminal
// (This is a comment by the way)

import com.sun.data.provider.RowKey;
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Blocks
- Modifiers
- Statements
- Classes
- Methods
- The main method

Reserved words

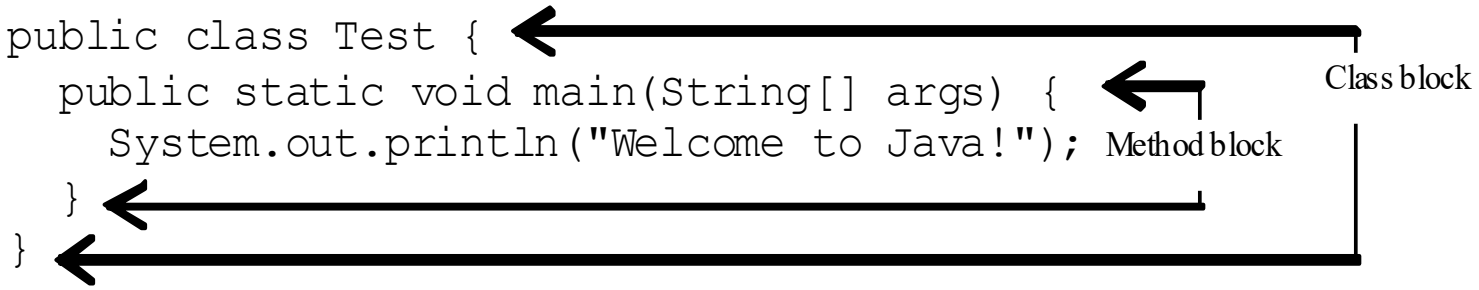
```
public class Welcome {
  public static void main(String[]
  args) {
    System.out.println("Welcome to
  Java!");
  }
}
```

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Blocks
- Modifiers
- Statements
- Classes
- Methods
- The main method

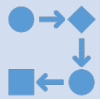
Blocks

- A pair of braces { } in a program forms a block that groups components of a program.



- Java programs are made up from one or more classes.
- A class contains code (statements) and data (variables)
- Even the simplest Java program is wrapped in a class.

Statements



A *statement* represents an action or a sequence of actions. The statement

System.out.println("Welcome to Java!");

in the previous example is a statement to display the greeting "Welcome to Java!"

Every statement in Java ends with a semicolon (;)

Java syntax is case sensitive.

Proper Indentation and Spacing

- Java source code can be free format – as much white space, as many newlines as desired.
- Java is case sensitive : ix, IX not the same.
- Identifiers / keywords separated by white space
- Be careful with String literals e.g “my string “.
- Program layout helps readability & maintenance
- Recommendation – Indentation
  - Indent a block of code by two or more spaces.
- Recommendation - Spacing
- Use blank lines to separate segments of the code.
- Netbeans – right click (or ALT+SHIFT+F) on text to reformat code !

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Methods
What is System.out.println?

- It can be used even without fully understanding the details of how it works. But check spelling !

- It is a *method*: a collection of statements that performs a sequence of operations to display a message on the console.

- It is used by invoking the statement with a string argument. The string argument `"Welcome to Java!"` is enclosed within parentheses.

Methods
What is System.out.println?

- You can call the same println method with a different argument to print a different message.

- You can build up a string within the argument e.g.

```
double r = 3.5;
  System.out.println ("Circle Radius = " + r);
```

main Method

- The main method provides the start point to the flow of program
- The main method looks like this:

```
public static void main (String[] args) {
    // Statements;
    // More Statements
}
```

- The Java interpreter executes an application by invoking the main method of a class – each application must have a class with main()

main Method

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

}
```

- main must be '*public*' - visible to other classes outside
- main must be '*void*' – returns no results

- You can pass arguments to the program at run time – these appear as an array of Strings in args[]

Classes

- The *class* is the essential Java construct.
- Even the simplest Java program requires at least one class.
- A class is a template or blueprint for an object.
- To program in Java, you must understand classes and be able to write and use them.
- The mystery of the class will continue to be unveiled throughout this module.
- For now, though, understand that a program is defined by using one or more classes.

Modifiers

- Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used.

- Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected.

Modifiers

- A public datum, method, or class can be accessed by other programs.
- A private datum or method cannot be accessed by other programs. Modifiers are discussed later.

Reserved Words

- *Reserved words* or *keywords* are words that have a specific meaning to the compiler
- They cannot be used for other purposes in the program.
- For example, when the compiler sees the word class, it understands that the word after class is the name for the class.
- Other reserved words in the previous example are public, static, and void.
- Java syntax is case sensitive
- Their use will be introduced later.

Java Keywords

- abstract
- assert
- boolean
- break
- Byte
- case
- Catch
- char
- class
- const
- continue
- default
- do
- double
- else
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while