

COMP30820  
Java Programming (Conv)

Michael O'Mahony

# Module Organisation

- ♦ Lecturer: Michael O'Mahony ([michael.omahony@ucd.ie](mailto:michael.omahony@ucd.ie))
- ♦ TA: Sheena Davitt ([sheena.davitt@ucdconnect.ie](mailto:sheena.davitt@ucdconnect.ie))
  
- ♦ Lectures:
  - Mondays, 09:00–09:50, Room B004 CSI
  - Wednesdays, 09:00–09:50, Room 114 VET
  
- ♦ Practicals:
  - Mondays, 15:00–16:50, Room H2.38 SCH
  - This week – software installation, using Eclipse (document on Moodle)
  
- ♦ Tutorials (no tutorial this week):
  - Wednesdays, 12:00–12:50 (**50 minutes**), Room 114 VET

# Module Organisation

- ♦ All course material will be available on Moodle:
  - Check the **Announcements** on the course Moodle regularly for updates
  - Please enrol on the course Moodle today:
    - ♦ Go to: <https://csmoodle.ucd.ie/moodle/>
    - ♦ Select: COMP30820 Java Programming (Conv) - 2018-19
    - ♦ Enrolment key: COMP30820-18-19-SR2
- ♦ Please ask questions, provide feedback etc. during lecture, tutorial and practical sessions

# Assessment Components

- ◆ In-class test #1 – written test: 20% (Week 7)
- ◆ In-class test #2 – programming test: 20% (Week 9)
- ◆ End of semester examination: 60%
- ◆ This module uses the recommended School of Computer Science mark-grade mapping scale:
  - See: <https://www.cs.ucd.ie/Grading/>
- ◆ Practical each week – not graded, solutions released after practicals and covered in tutorials

# Plagiarism

- ✦ Plagiarism is a serious academic issue and the University will examine all alleged instances of plagiarism:
  - It is really obvious to us when plagiarism occurs and we actively check for it...
  - See documents on Moodle.
  - If unsure, please ask!

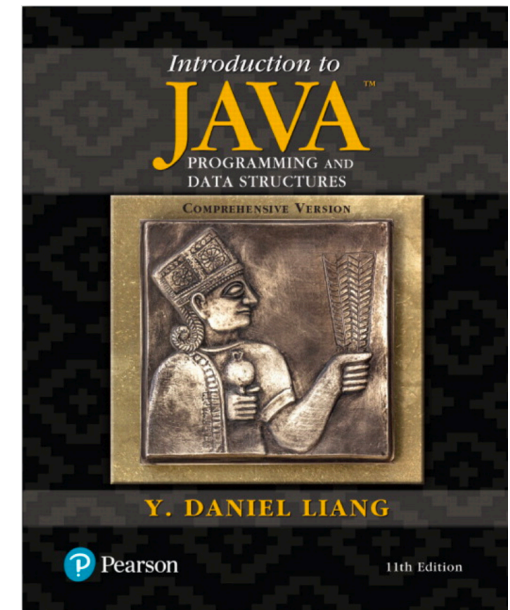
# Software

- ♦ Software required:
  - Java (version: Java SE 11.0.2)
  - Eclipse IDE (version: 2018-12 R)
- ♦ Today's practical session (15:00–16:50, Room H2.38 SCH):
  - How to install software and use the Eclipse IDE to write Java programs
  - Instructions available on Moodle – see “COMP30820 Software Requirements”

# Textbook

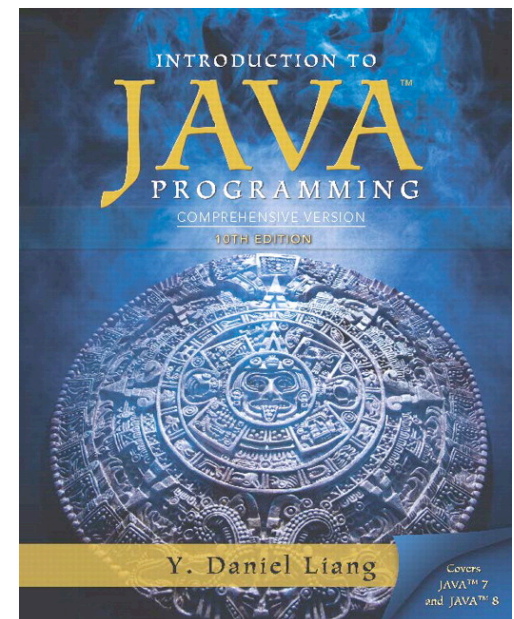
Introduction to Java Programming and Data Structures,  
Comprehensive Version, 11/E, 2018

- Y. Daniel Liang
- ISBN-13: 9780134670942



Introduction to Java Programming, Comprehensive  
Version, 10/E, 2015

- Y Daniel Liang
- ISBN-13: 9780133761313



Lecture material based on the above

# Module Overview

## Part I: Fundamentals of Programming

**Chapter 1 Introduction to Computers, Programs, and Java**



**Chapter 2 Elementary Programming**



**Chapter 3 Selections**



**Chapter 4 Mathematical Functions, Characters, and Strings**



**Chapter 5 Loops**



**Chapter 6 Methods**



**Chapter 7 Single-Dimensional Arrays**



**Chapter 8 Multidimensional Arrays**

## Part II: Object-Oriented Programming

**Chapter 9 Objects and Classes**



**Chapter 10 Thinking in Objects**



**Chapter 11 Inheritance and Polymorphism**



**Chapter 12 Exception Handling and Text I/O**



**Chapter 13 Abstract Classes and Interfaces**





# Chapter 1 Introduction to Computers, Programs, and Java

# Objectives

- ♦ To understand the meaning of Java language specification, API, JDK, JVM (§1.6).
- ♦ To write a simple Java program (§1.7).
- ♦ To explain the basic syntax of a Java program (§1.7).
- ♦ To explain the differences between syntax errors, runtime errors, and logic errors (§1.10).
- ♦ To use sound Java programming style and document programs properly (§1.9).
- ♦ To understand computer basics and programs (§§1.2–1.3).
- ♦ To create, compile, and run Java programs (§1.8).
- ♦ To understand the characteristics of Java (§1.5).

# The Java Language Specification, API, JDK, JVM and IDE

- ♦ The *Java language specification* is a technical definition of the Java programming language's syntax and semantics.
- ♦ The *Application Program Interface* (API), also known as library, contains predefined classes and interfaces for developing Java programs:
  - Go to: <https://docs.oracle.com/en/java/javase/11/docs/api/index.html>
- ♦ The *Java Development Toolkit* (JDK) is the software for developing and running Java programs
- ♦ The *Java Virtual Machine* (JVM) is a program that interprets Java bytecode
- ♦ An IDE is an *integrated development environment* for rapidly developing programs (we will use the Eclipse IDE in this module)

# Java Programming Language Platforms

- ◆ Java Platform, Standard Edition (Java SE)
  - Java SE provides the core functionality of the Java programming language and is used to develop Java applications.
  - Used in this module.
- ◆ Java Platform, Enterprise Edition (Java EE)
  - Java EE can be used to develop server-side applications using e.g. Java servlets, Java ServerPages etc.
- ◆ Java Platform, Micro Edition (Java ME)
  - Java ME can be used to develop applications for mobile devices.

# Programs

Computer *programs*, known as *software*, are instructions to the computer.

You tell a computer what to do through programs. Without programs, a computer is an empty machine...

Programs are written using programming languages.

# A Simple Java Program

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

# A Simple Java Program

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

## Anatomy of a Java program:

- Class name
- Main method
- Statements
- Statement terminator
- Reserved words
- Comments
- Blocks

# Class Name

Every Java program must have at least one class. Each class has a name.

By convention, class names start with an uppercase letter – in this example, the class name is `Welcome`

Note – the filename of this program should be `Welcome.java`

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



# Main Method

Line 2 defines the main method. In order to run a class, the class must contain a method named `main`. The program is executed from the main method.

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

# Statement

A statement represents an action or a sequence of actions.

The following statement ...

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

... displays the string "Welcome to Java!"

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

# Statement Terminator

Every statement in Java ends with a semicolon (;).

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

# Reserved words

Reserved words or keywords are words that have a specific meaning to the compiler and 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.

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

# Special Symbols

Character Name	Description	
{ }	Opening and closing braces	Denotes a block to enclose statements.
( )	Opening and closing parentheses	Used with methods.
[ ]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
;	Semicolon	Marks the end of a statement.

# Blocks

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

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

Class block

Method block

# Parentheses

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

# Brackets

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



# Comments

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

# Quotation Marks

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

# Statement Terminator

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

# Trace a Program Execution

Enter main method

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

# Trace a Program Execution

Execute statement

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

# Trace a Program Execution

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



Welcome to Java!

print a message to the  
console

# Another Simple Example

```
// This program prints three messages...
public class WelcomeThreeMessages {
    public static void main(String[] args) {
        System.out.println("Programming is fun!");
        System.out.println("Fundamentals First");
        System.out.println("Problem Driven");
    }
}
```

# “Template” for Java Programs

Initially we will use the following “template” for Java programs.

Replace the parts in red font with your code...

```
public class SomeName {  
    public static void main(String[] args) {  
        Instruction 1...  
        Instruction 2...  
        Instruction 3...  
        ...  
    }  
}
```



# Programming Errors

- ◆ Syntax Errors

- Detected by the compiler

- ◆ Runtime Errors

- Causes the program to abort

- ◆ Logic Errors

- Produces incorrect result

# Syntax Error – Example

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

# Runtime Error – Example

```
public class ShowRuntimeErrors {  
    public static void main(String[] args) {  
        System.out.println(1 / 0);  
    }  
}
```

# Logic Error – Example

```
public class ShowLogicErrors {  
    public static void main(String[] args) {  
        System.out.println("Celsius to Fahrenheit:");  
        System.out.println((9 / 5) * 35 + 32);  
    }  
}
```

# Programming Style and Documentation

- ◆ Appropriate Comments
- ◆ Naming Conventions
- ◆ Proper Indentation and Spacing Lines
- ◆ Block Styles

# Appropriate Comments

Include a summary at the beginning of the program to explain what the program does, its key features, and any unique techniques it uses.

Include comments in the code to explain what is being done.

In your work, always include your name, student number, module title, date, practical number, question number, and a brief description at the beginning of the program.

# Naming Conventions

- ♦ Choose meaningful and descriptive names.
- ♦ Class names:
  - Capitalize the first letter of each word in the name; for example: HelloWorld.

# Proper Indentation and Spacing

- ◆ Indentation:

- Indent two spaces.


- ◆ Spacing:

- Use a blank line to separate segments of the code.




# Block Styles

*Next-line  
style*



```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

*End-of-line  
style*



```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

End-of-line style is *generally* preferred for braces.

# Programs

Computer *programs*, known as *software*, are instructions to the computer.

You tell a computer what to do through programs. Without programs, a computer is an empty machine...

Programs are written using programming languages.

# Programming Languages

**Machine Language**    Assembly Language    High-Level Language

Machine language is a set of primitive instructions built into every computer.

The instructions are in the form of binary code.

Programming with native machine language is a tedious process. Programs are highly difficult to read and modify.

For example, to add two numbers, you might write an instruction in binary like this:

```
1101101010011010
```

# Programming Languages

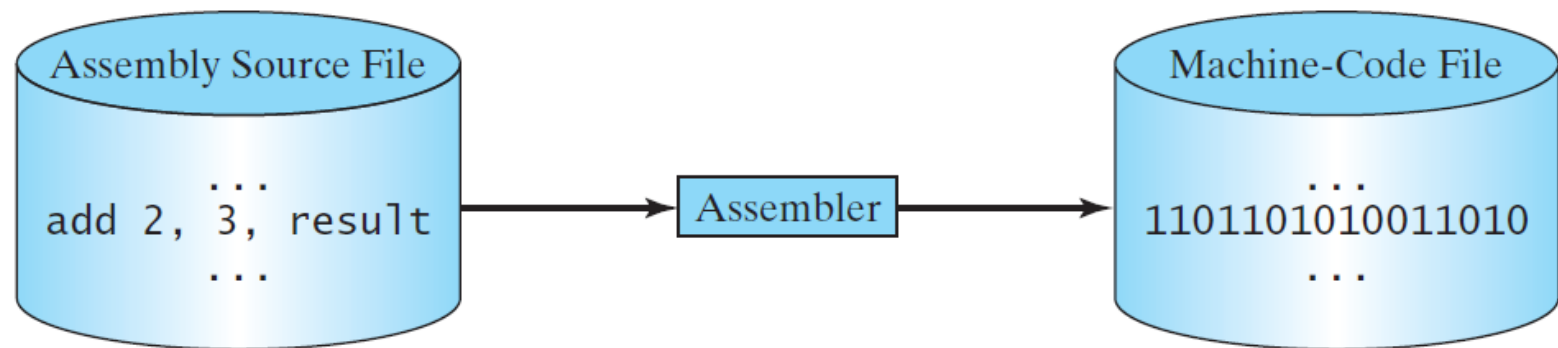
Machine Language    **Assembly Language**    High-Level Language

Assembly languages were developed to make programming easier.

The computer cannot understand assembly language – a program called an *assembler* is used to convert assembly language programs into machine code.

For example, to add two numbers, you might write an instruction in assembly code like this:

```
add 2, 3, result
```



# Programming Languages

Machine Language    Assembly Language    **High-Level Language**

The high-level languages are English-like and (relatively!) easy to learn and program.

For example, the following is a high-level language statement that adds two integers:

```
result = 2 + 3
```

# Interpreting/Compiling Source Code

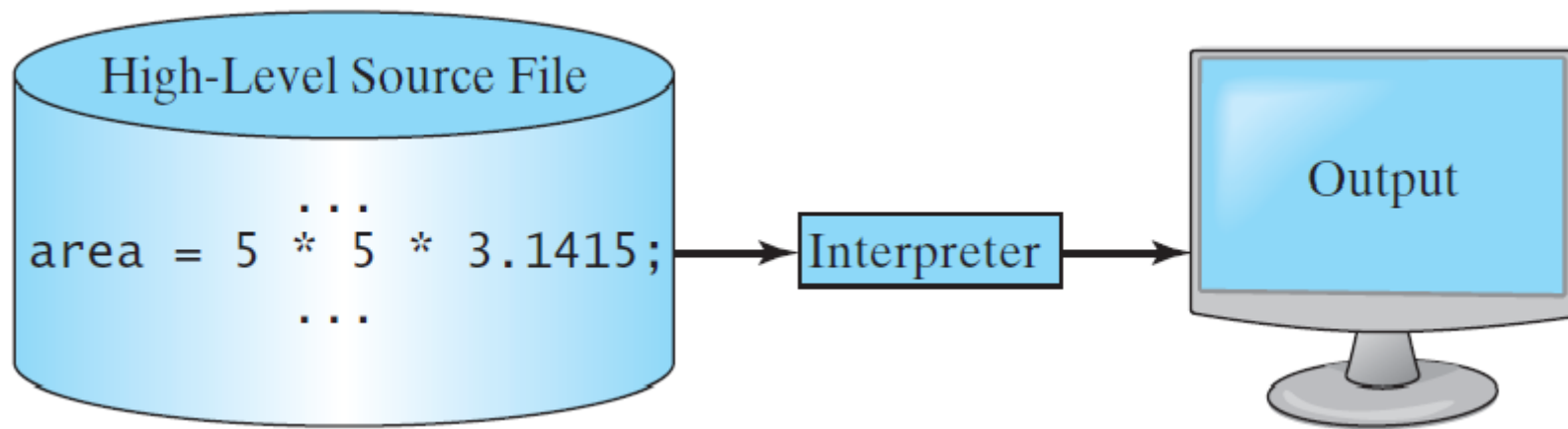
A program written in a high-level language is called a *source program* or *source code*.

Source programs must be translated into machine code for execution.

The translation can be done using another programming tool called an *interpreter* or a *compiler*.

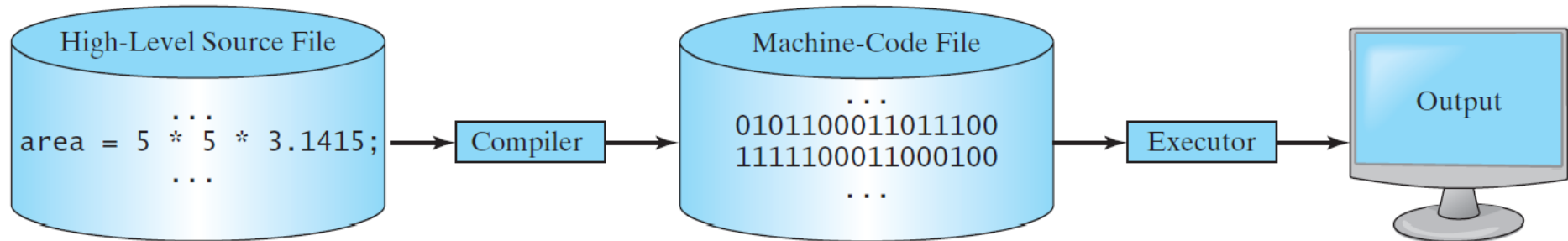
# Interpreting Source Code

An *interpreter* reads one statement from the source code, translates it to the machine code, and then executes it right away.



# Compiling Source Code

A *compiler* translates the entire source code into a machine-code file, and the machine-code file is then executed.





# Compiled vs Interpreted Languages

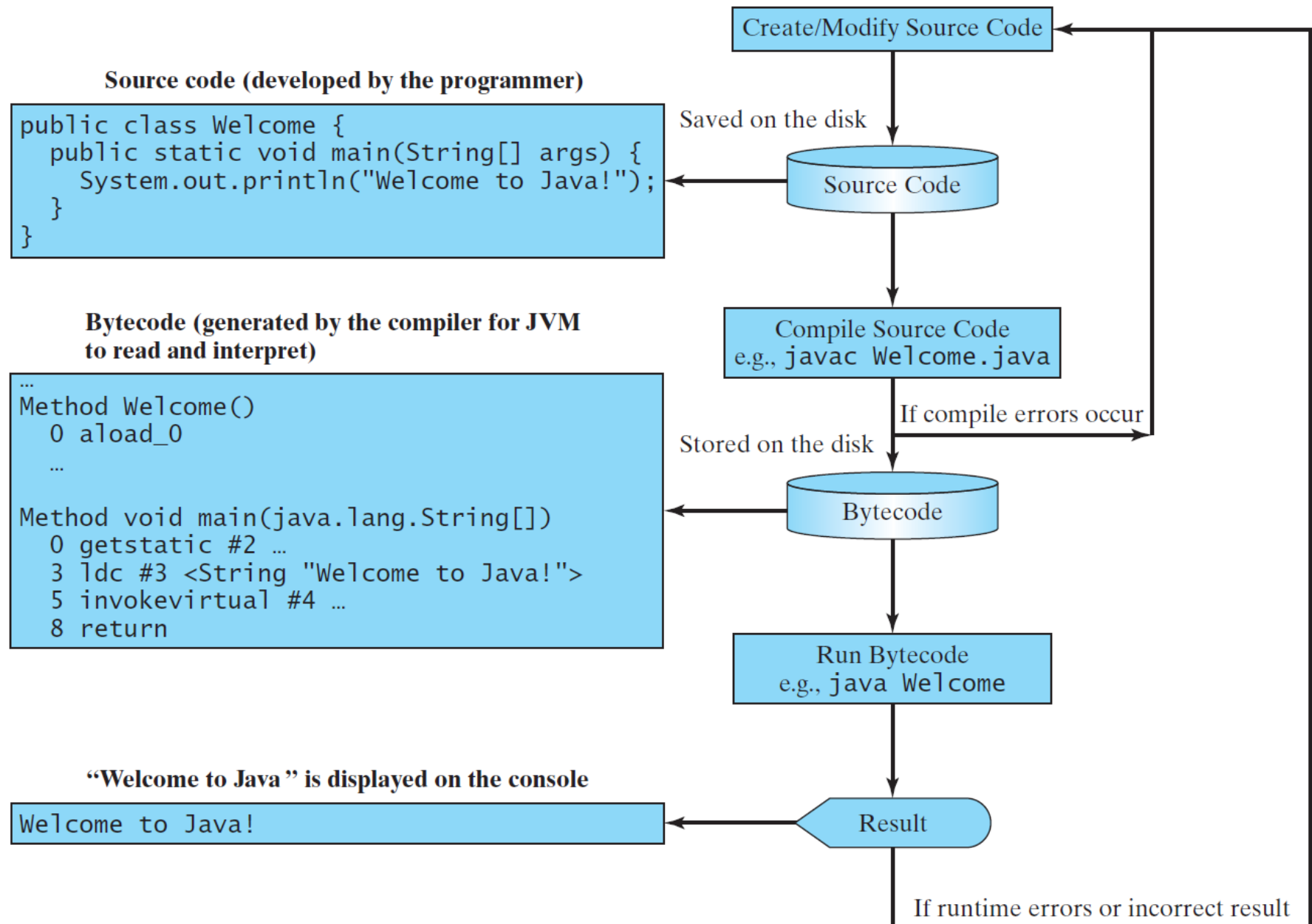
## ♦ Compiled languages:

- Can result in faster performance by directly using the native code of the machine
- Can apply optimisations during the compile stage

## ♦ Interpreted languages:

- Easier to implement – developing good compilers is non-trivial
- No need to run a compilation stage – can execute code directly "on the fly"

# Creating, Compiling, and Running Java Programs



# Creating, Compiling, and Running Java Programs

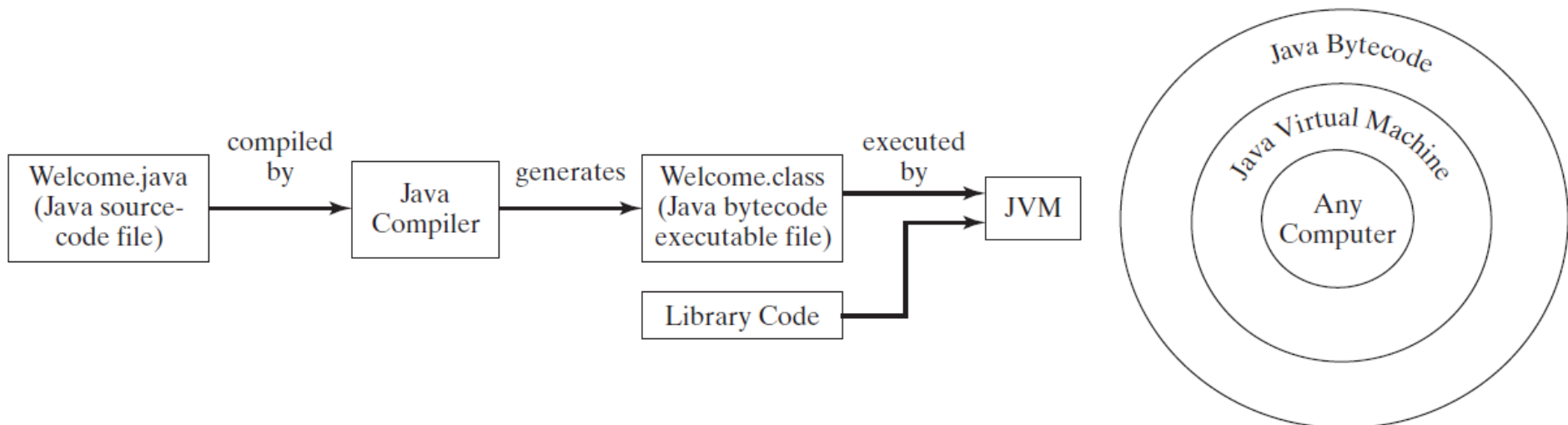
You can port a source program to any machine with appropriate compilers.

However, the source program must be recompiled because the object program can only run on a specific machine.

Java is designed to run object programs on any platform.

With Java, you write the program once, and compile the source program into a special type of object code, known as *bytecode*.

The bytecode can then run on any computer with a Java Virtual Machine (JVM) - software that interprets Java bytecode.



# Why Java?

- ♦ Java is a general purpose programming language.
- ♦ Java is a widely used Internet programming language.
- ♦ Java can be used to:
  - Develop standalone applications.
  - Develop applications running from a browser (removed from Java SE 11).
  - Develop applications for hand-held devices. The software for Android phones is based on Java.
  - Develop applications for Web servers.

# Characteristics of Java

- ♦ Java Is (Relatively!) Simple
- ♦ Java Is Object-Oriented
- ♦ Java Is Robust
- ♦ Java Is Architecture-Neutral  
and Portable

# Characteristics of Java

- ✦ Java Is (Relatively!) Simple
- ✦ Java Is Object-Oriented
- ✦ Java Is Robust
- ✦ Java Is Architecture-Neutral and Portable

Java is partially modeled on C++, but simplified and improved.

Some people refer to Java as "C++--" because it is like C++ but with more functionality and fewer negative aspects.

# Characteristics of Java

- ✦ Java Is (Relatively!) Simple
- ✦ Java Is Object-Oriented
- ✦ Java Is Robust
- ✦ Java Is Architecture-Neutral and Portable

Java is inherently object-oriented.

One of the central issues in software development is code reuse. Object-oriented programming provides flexibility, modularity, clarity, and reusability.

# Characteristics of Java

- ✦ Java Is (Relatively!) Simple
- ✦ Java Is Object-Oriented
- ✦ **Java Is Robust**
- ✦ Java Is Architecture-Neutral and Portable

Java compilers can detect many problems that would first show up at execution time in other languages.

Java has eliminated certain types of error-prone programming constructs found in other languages.

Java has a runtime exception-handling feature to provide programming support for robustness.



# Characteristics of Java

- ◆ Java Is (Relatively!) Simple

- ◆ Java Is Object-Oriented

- ◆ Java Is Robust

- ◆ Java Is Architecture-Neutral  
and Portable

Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Write once, run anywhere.

# Next Lecture

- ♦ Writing Java programs to perform simple computations
- ♦ Look at Java primitive data types, variables, constants, data types, operators, expressions, input and output...