

COMP20220

Programming II (Conversion)

Michael O'Mahony



Module Organisation

- ♦ Lecturer: Michael O'Mahony (michael.omahony@ucd.ie)
- ♦ TA: Akhilesh Sharma (akhilesh.sharma@ucdconnect.ie)

- ♦ Lectures:
 - Mondays, 09:00–09:50, Room CS B0.03
 - Wednesdays, 09:00–09:50, Room CS B0.03

- ♦ Practicals:
 - Mondays, 15:00–16:50, Room SCE E2.16
 - Wednesdays, 12:00–12:50, Room CS B0.03 (optional)

- ♦ Tutorials (announced in advance):
 - Wednesdays, 13:00–13:50, Room CS B0.03

- ♦ This week – practical session on Monday (today) only
 - Software installation

Module Organisation

- ◆ All course material will be available on Moodle (updated weekly):
 - Check the **News Forum** on Moodle regularly for course updates
 - Please enrol on the course Moodle today!
 - ◆ Go to <https://csmoodle.ucd.ie/moodle/>
 - ◆ Select COMP20220 Programming II (Conversion) 2016-2017
 - ◆ Enrolment key: COMP20220-2016-2017-Sem-2
- ◆ Please ask questions,, provide feedback etc. during lecture, tutorial and practical sessions

Assessment Components

- ♦ Programming assignments (2x): 20%
- ♦ In-class tests (2x): 20%
- ♦ End of semester examination: 60%
- ♦ School of Computer Science Marks to Grades Conversion Scale
 - See: <https://www.cs.ucd.ie/Grading/>

Assessment Components

♦ Schedule (provisional):

- Assignment #1 - Week 5 (due end Week 5)
- Assignment #2 - Week 10 (due end Week 10)
- In-class Test #1 - Week 7 (week before Study Break)
- In-class Test #2 - Week 8 (week after Study Break)

Plagiarism

- ✦ Please note that students must work on all assignments **individually** and not in groups/teams – avoid 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...
 - Do not copy (any part of) the work of other students or copy material from online sources
 - See **Plagiarism Policy and Procedures** document on Moodle
 - If unsure, please ask!

Software

- ♦ Software required:
 - Java (JDK1.8)
 - Eclipse (Neon)
- ♦ Go to: <https://www.cs.ucd.ie/ConversionSoftware/>
 - Refer to the relevant sections in file “COMP30670 Software Necessities.pdf” for details
 - Assistance available during practical session this afternoon

Textbook

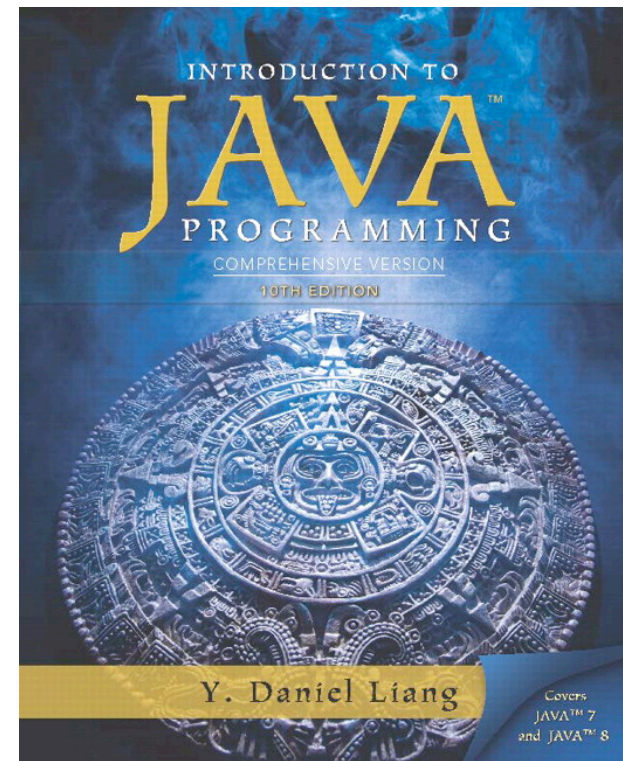
Introduction to Java Programming, Comprehensive Version, 10/E, 2015

Y Daniel Liang, Armstrong Atlantic State University

ISBN-10: 0133761312

ISBN-13: 9780133761313

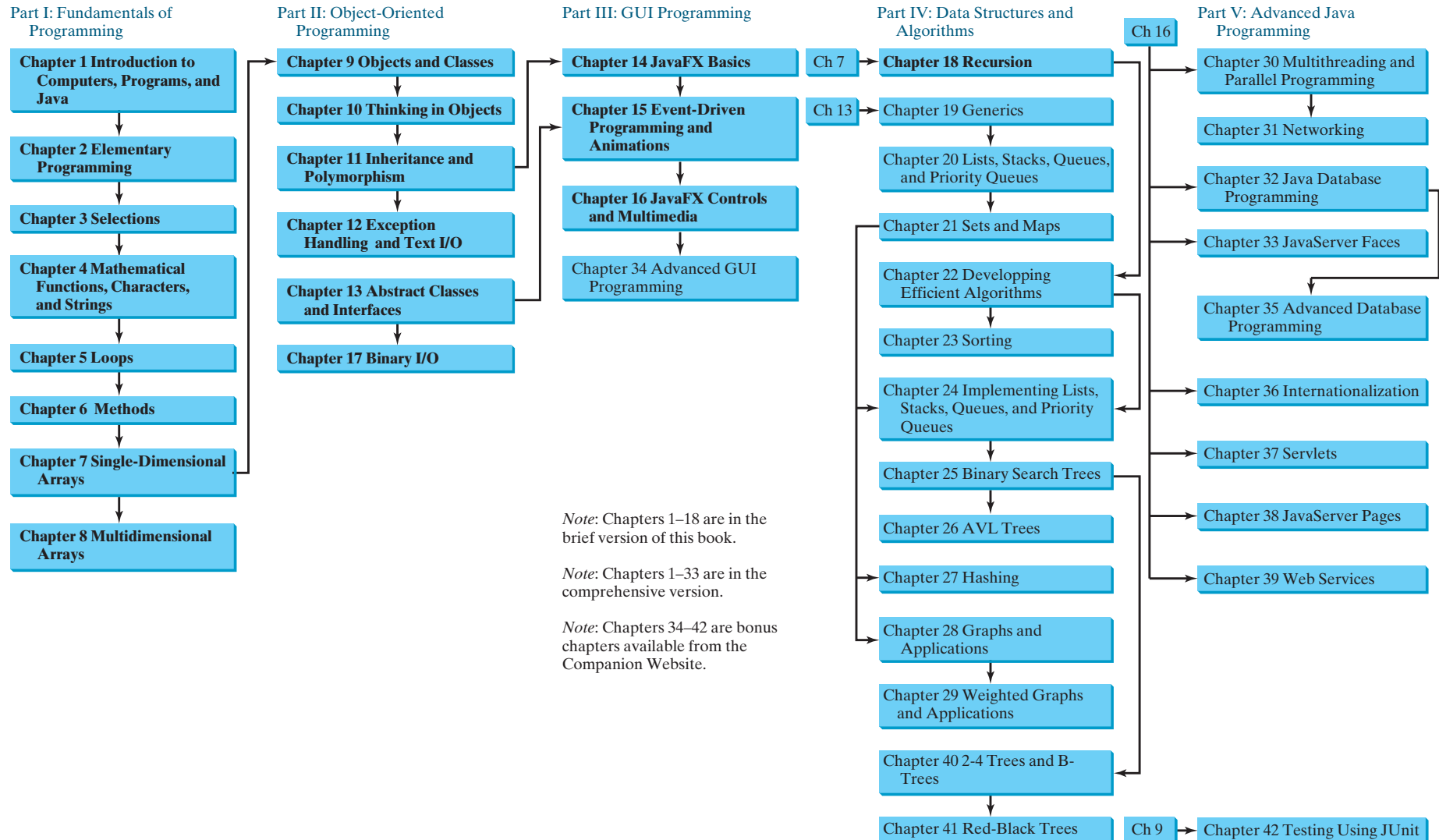
Lecture material based on this book



Module Overview

- ✦ Introduction
- ✦ Elementary Programming
- ✦ Selections
- ✦ Mathematical Functions, Characters and Strings
- ✦ Loops
- ✦ Methods
- ✦ Arrays
- ✦ Objects and Classes
- ✦ Object-Orientated Thinking
- ✦ Inheritance and Polymorphism
- ✦ Exception Handling and Text I/O
- ✦ Abstract Classes and Interfaces

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

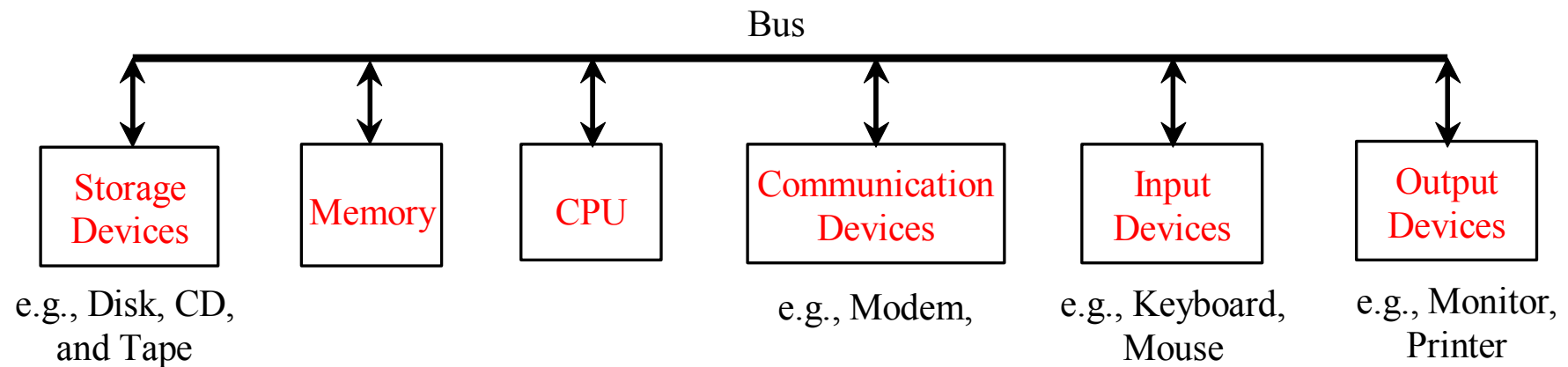


Objectives

- ♦ To understand computer basics and programs (§§1.2–1.3).
- ♦ To understand the characteristics of Java (§1.5).
- ♦ 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 create, compile, and run Java programs (§1.8).
- ♦ To use sound Java programming style and document programs properly (§1.9).
- ♦ To explain the differences between syntax errors, runtime errors, and logic errors (§1.10).

What is a Computer?

A computer consists of a CPU, memory, hard disk, I/O devices, and communication devices.

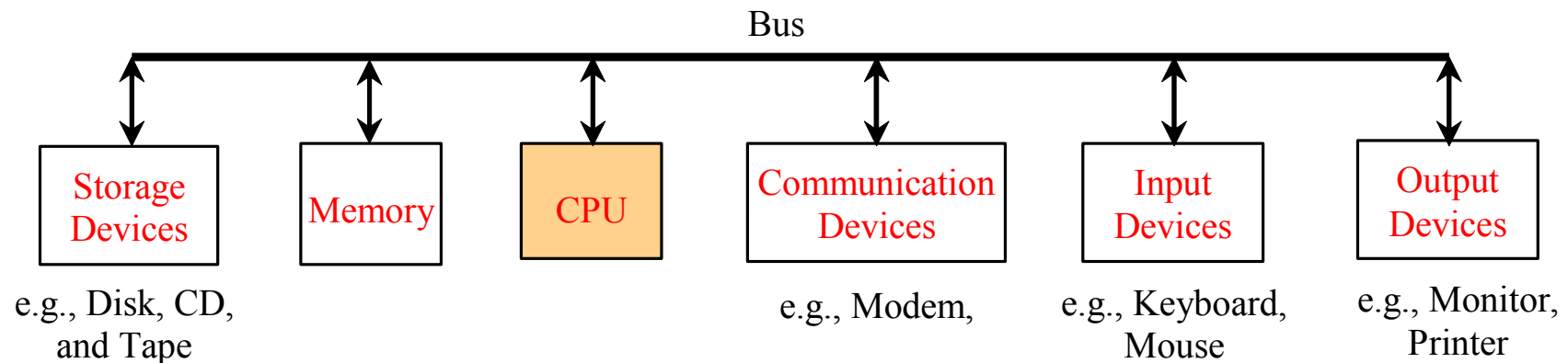


CPU

The central processing unit (CPU) – the “brain” of a computer.

It retrieves instructions from memory and executes them.

The CPU speed is measured in megahertz ($1 \text{ MHz} = 10^6 \text{ Hz}$) or gigahertz ($1 \text{ GHz} = 10^9 \text{ Hz}$) – 1 GHz equals one thousand million pulses per second.

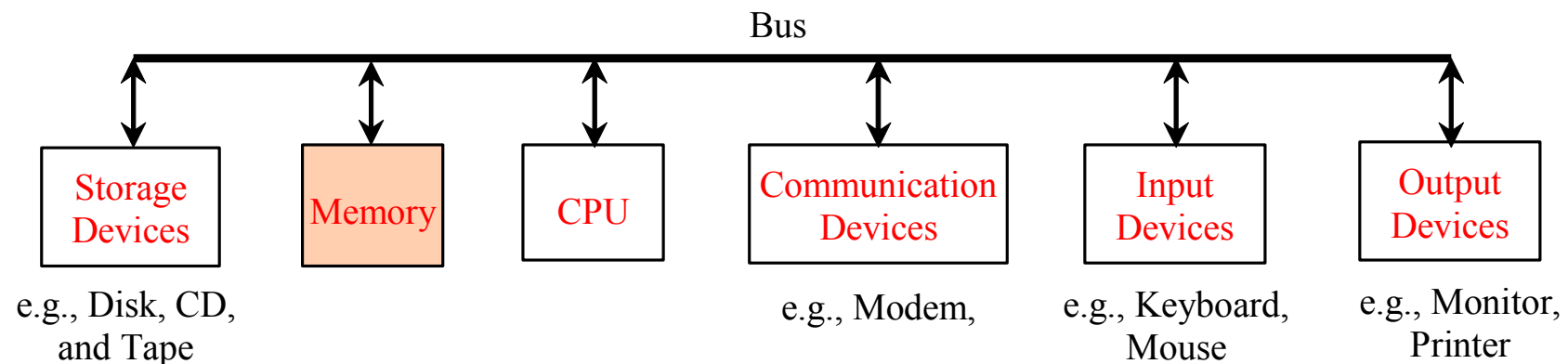


Memory

Memory is used to store data and program instructions for the CPU to execute.

A memory unit is an ordered sequence of *bytes*, each holds eight *bits*.

A program and its data must be brought to memory before they can be executed.



Bytes, Kilobytes etc.

- A byte is 8 bits (binary digits)
- A kilobyte (KB) is about 1,000 bytes ($1024 = 2^{10}$ bytes)
- A megabyte (MB) is about 1 million bytes ($1,048,576 = 2^{20}$ bytes)
- A gigabyte (GB) is about 1 billion bytes ($1,073,741,824 = 2^{30}$ bytes)
- A terabyte (TB) is about 1 trillion bytes ($1,099,511,627,776 = 2^{40}$ bytes)

How Data is Stored?

Data such as numbers, characters, and strings, are encoded as a series of bits (zeros and ones).

Computers use zeros and ones because digital devices have two stable states, referred to as *zero* and *one*.

The encoding and decoding of data is performed automatically by the system based on the encoding scheme.

The encoding scheme varies. For example, character 'J' is represented by 01001010 in one byte. A small number such as 3 can be stored in a single byte.

If computer needs to store a large number that cannot fit into a single byte, it uses a number of adjacent bytes.

No two data can share or split the same byte. A byte is the minimum storage unit.

Memory address	Memory content	
.	.	
.	.	
.	.	
2000	01001010	Encoding for character 'J'
2001	01100001	Encoding for character 'a'
2002	01110110	Encoding for character 'v'
2003	01100001	Encoding for character 'a'
2004	00000011	Encoding for number 3

How Data is Stored?

I need to represent 4 characters (e.g. A, B, C and D). How many bits are required?

How Data is Stored?

I need to represent 4 characters (e.g. A, B, C and D). How many bits are required?

Answer: $\log_2(4) = 2$

How Data is Stored?

I need to represent 4 characters (e.g. A, B, C and D). How many bits are required?

Answer: $\log_2(4) = 2$, e.g. A = 00, B = 01, C = 10, D = 11

How Data is Stored?

I need to represent 4 characters (e.g. A, B, C and D). How many bits are required?

Answer: $\log_2(4) = 2$, e.g. A = 00, B = 01, C = 10, D = 11

I need to represent 256 characters – how many bits are required?

How Data is Stored?

I need to represent 4 characters (e.g. A, B, C and D). How many bits are required?

Answer: $\log_2(4) = 2$, e.g. A = 00, B = 01, C = 10, D = 11

I need to represent 256 characters – how many bits are required?

Answer: $\log_2(256) = 8$

How Data is Stored?

I need to represent 4 characters (e.g. A, B, C and D). How many bits are required?

Answer: $\log_2(4) = 2$, e.g. A = 00, B = 01, C = 10, D = 11

I need to represent 256 characters – how many bits are required?

Answer: $\log_2(256) = 8$

How many characters can be represented by 8 bits (1 byte)?

How Data is Stored?

I need to represent 4 characters (e.g. A, B, C and D). How many bits are required?

Answer: $\log_2(4) = 2$, e.g. A = 00, B = 01, C = 10, D = 11

I need to represent 256 characters – how many bits are required?

Answer: $\log_2(256) = 8$

How many characters can be represented by 8 bits (1 byte)?

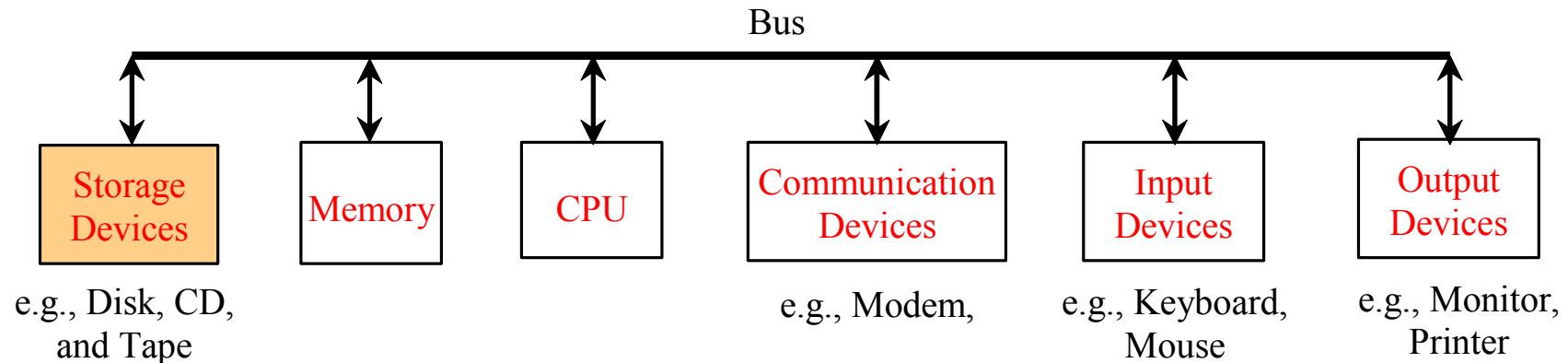
Answer: $2^8 = 256$

Storage Devices

Memory is volatile – information is lost when the power is off.

Programs and data are permanently stored on storage devices and are moved to memory when the computer actually uses them.

There are three main types of storage devices: disk drives, CD drives (CD-R and CD-RW), and tape drives.



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

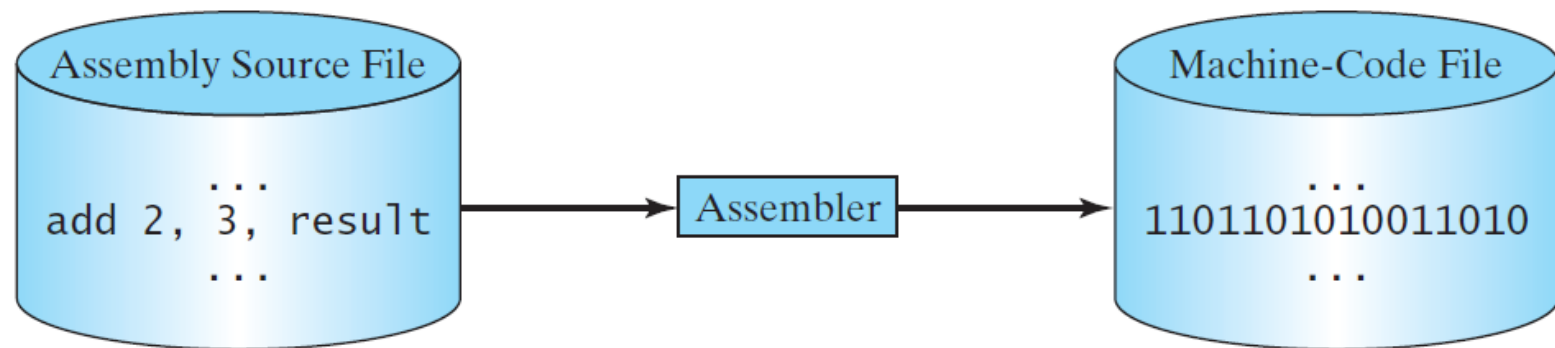
Machine Language **Assembly Language** High-Level Language

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 easy (!) to learn and program.

For example, the following is a high-level language statement that computes the area of a circle with radius 5:

```
area = 5 * 5 * 3.1415;
```

Interpreting/Compiling Source Code

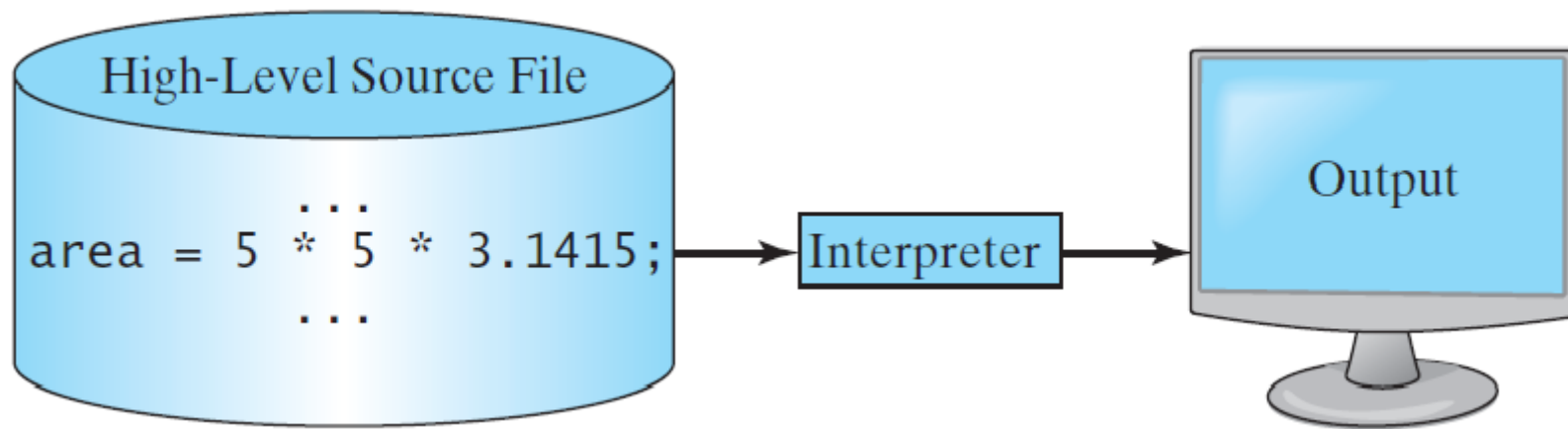
A program written in a high-level language is called a *source program* or *source code*.

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*.

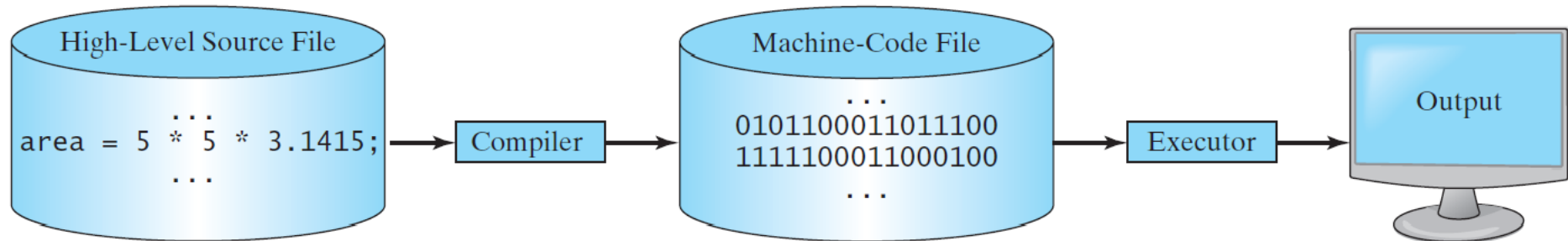
Interpreting Source Code

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"

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: <http://docs.oracle.com/javase/8/docs/api/>
- ♦ The *Java Development Toolkit* (JDK) is the software for developing and running Java programs (**you need to install JDK 1.8, a.k.a. JDK 8**)
- ♦ The *Java Virtual Machine* (JVM) is a program that interprets Java bytecode.
- ♦ An IDE is an *integrated development environment* for rapidly developing programs (**you need to install Eclipse Neon**).

JDK Editions

♦ Java Standard Edition (J2SE)

- J2SE can be used to develop client-side standalone applications.
- Used in this module.

♦ Java Enterprise Edition (J2EE)

- J2EE can be used to develop server-side applications such as Java servlets, Java ServerPages, and Java ServerFaces.

♦ Java Micro Edition (J2ME)

- J2ME can be used to develop applications for mobile devices.

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.
 - Develop applications for hand-held devices. (The software for Android phones is developed using Java.)
 - Develop applications for Web servers.

Characteristics of Java

- ♦ Java Is Simple
- ♦ Java Is Object-Oriented
- ♦ Java Is Interpreted
- ♦ Java Is Robust
- ♦ Java Is Architecture-Neutral
- ♦ Java Is Portable

Characteristics of Java

- ◆ Java Is Simple
- ◆ Java Is Object-Oriented
- ◆ Java Is Interpreted
- ◆ Java Is Robust
- ◆ Java Is Architecture-Neutral
- ◆ Java Is 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 Simple
- ✦ Java Is Object-Oriented
- ✦ Java Is Interpreted
- ✦ Java Is Robust
- ✦ Java Is Architecture-Neutral
- ✦ Java Is 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 Simple
- ✦ Java Is Object-Oriented
- ✦ **Java Is Interpreted**
- ✦ Java Is Robust
- ✦ Java Is Architecture-Neutral
- ✦ Java Is Portable

Programs are first **compiled** into bytecode.

An **interpreter** is used to run Java bytecode.

Bytecode is machine-independent and can run on any machine that has a Java interpreter.

Characteristics of Java

- ✦ Java Is Simple
- ✦ Java Is Object-Oriented
- ✦ Java Is Interpreted
- ✦ **Java Is Robust**
- ✦ Java Is Architecture-Neutral
- ✦ Java Is 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 Simple
- ♦ Java Is Object-Oriented
- ♦ Java Is Interpreted
- ♦ Java Is Robust
- ♦ **Java Is Architecture-Neutral**
- ♦ Java Is Portable

Write once, run anywhere

With a Java Virtual Machine (JVM),
you can write one program that will
run on any platform.

Characteristics of Java

- ♦ Java Is Simple
- ♦ Java Is Object-Oriented
- ♦ Java Is Interpreted
- ♦ Java Is Robust
- ♦ Java Is Architecture-Neutral
- ♦ Java Is Portable

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

A Simple Java Program

Welcome.java

```
// 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 must 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!");  
    }  
}
```

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

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.

{ ... }

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

(...)

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

•
;

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


// ...

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

""
...

```
// 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!");  
    }  
}
```

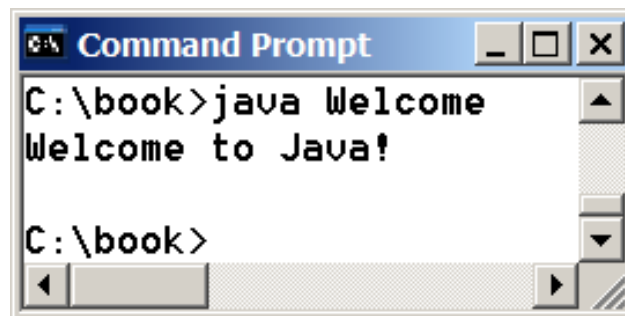
Trace a Program Execution

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!");  
    }  
}
```



```
Command Prompt  
C:\book>java Welcome  
Welcome to Java!  
  
C:\book>
```

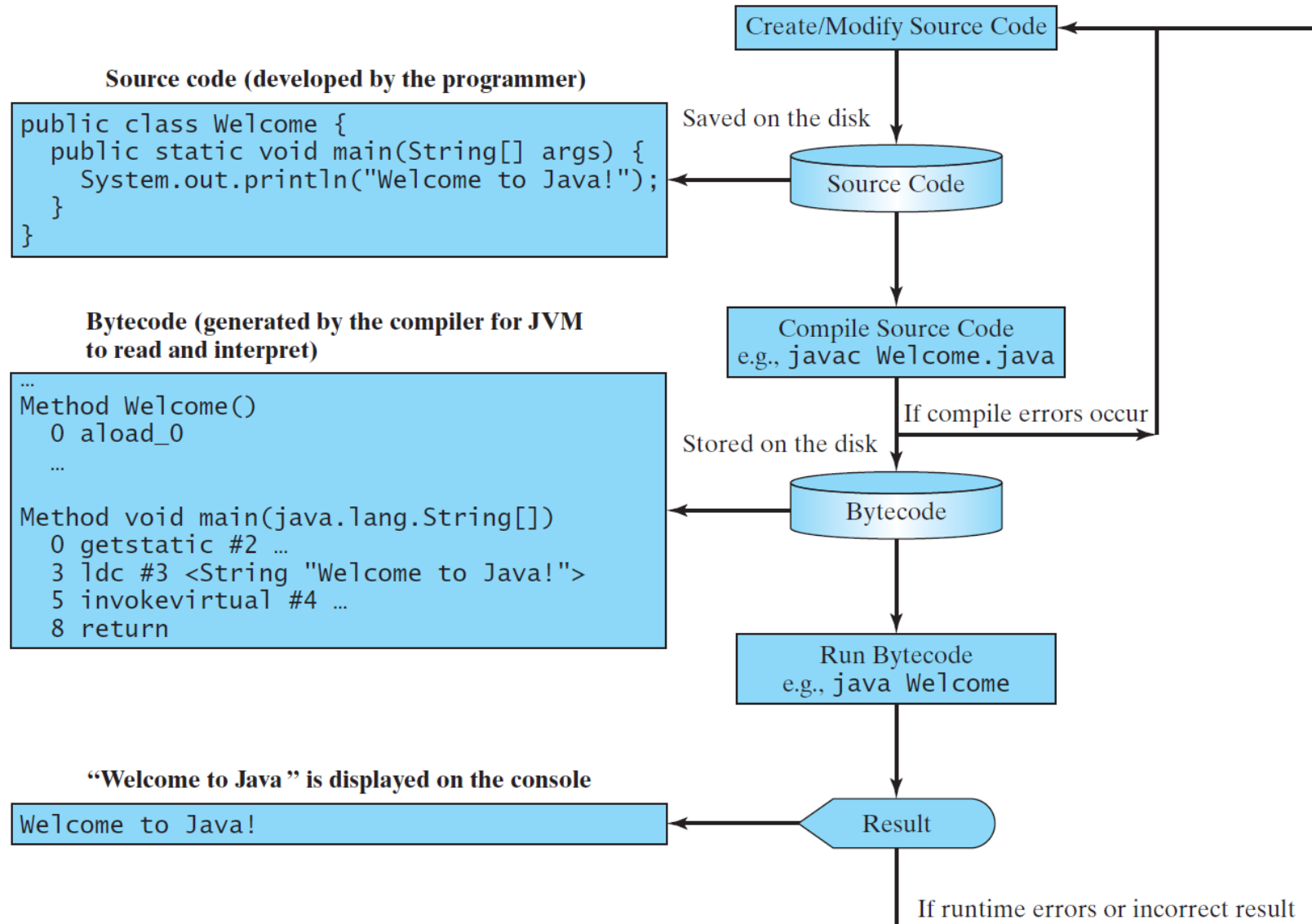
print a message to the console

Another Simple Example



WelcomeWithThreeMessages

Creating, Compiling, and Running Programs



Compiling Java Source Code

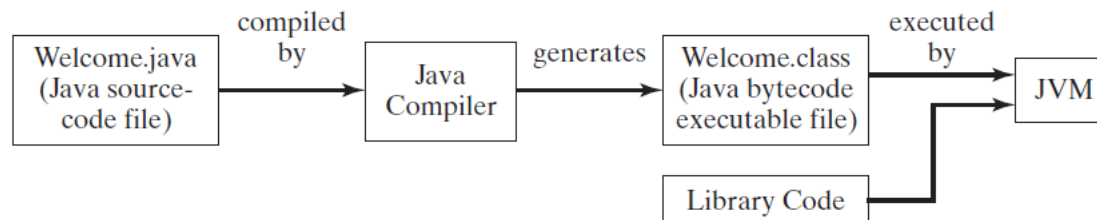
You can port a source program to any machine with appropriate compilers.

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

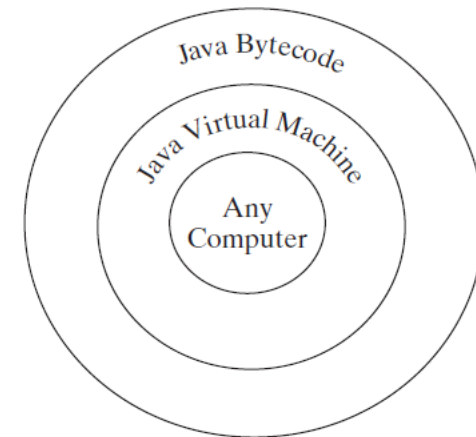
Java is designed to run object programs on any platform.

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.



(a)



(b)

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, its supporting data structures, and any unique techniques it uses.

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

In your work, always include your name, the module title, date, 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 *generally* preferred for braces.

Programming Errors

- ◆ Syntax Errors

- Detected by the compiler

- ◆ Runtime Errors

- Causes the program to abort

- ◆ Logic Errors

- Produces incorrect result

Syntax Error – Examples

```
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 35 is Fahrenheit degree ");  
        System.out.println((9 / 5) * 35 + 32);  
    }  
}
```



Next Lecture

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