Chapter 1

Introduction to Computers, Programs, and Java

Objectives

- To Know the basics
- To write a simple Java program
- To know error types
- To Know basic syntax of a Java program

Programs

Computer *programs*, known as *software*, are instructions to the computer. They tell the computer what to do through programs.

Computers do not understand human languages, so you need to use computer languages to communicate with them.

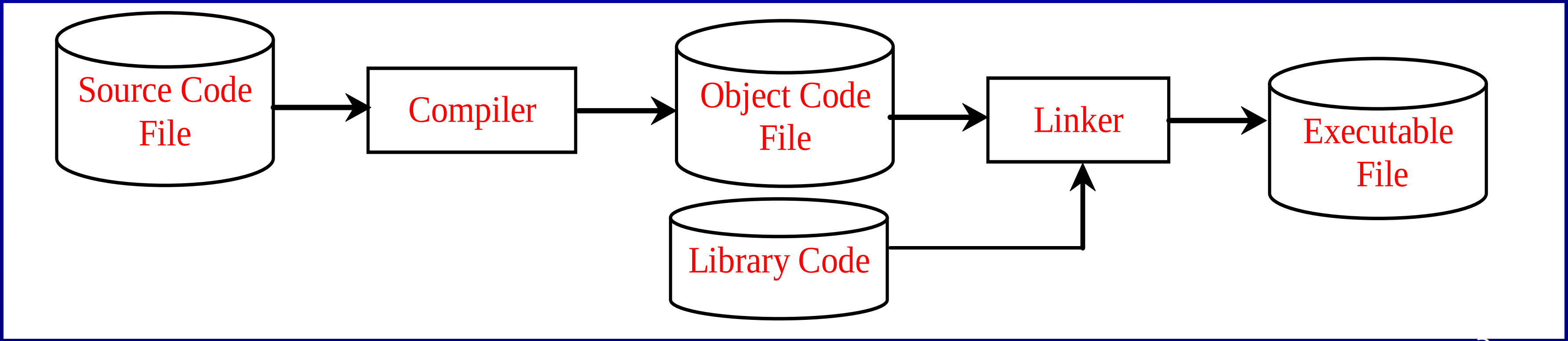
Programs are written using programming languages.

Popular High-Level Languages

- COBOL (COmmon Business Oriented Language)
- FORTRAN (FORmula TRANslation)
- BASIC (Beginner All-purpose Symbolic Instructional Code)
- Pascal (named for Blaise Pascal)
- Ada (named for Ada Lovelace)
- C (whose developer designed B first)
- Visual Basic (Basic-like visual language developed by Microsoft)
- Delphi (Pascal-like visual language developed by Borland)
- C++ (an object-oriented language, based on C)
- C# (a Java-like language developed by Microsoft)
- **Java (We use in this course and textbook)**

Compiling (Java) Source Code

A program written in a high-level language is called a *source program*. Since a computer cannot understand a source program. Program called a *compiler* is used to translate the source program into a machine language program called an *object program (byte code)*. The object program is often then linked with other supporting library code before the object code can be executed on the machine. JVM then converts **byte** code to **machine/executable** code.

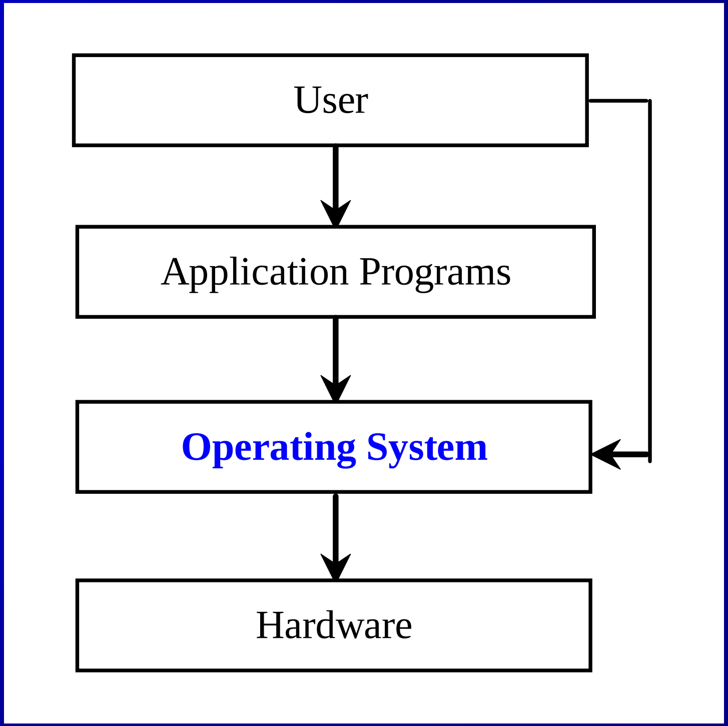


Operating Systems

The ***operating system*** **(OS)** is a program that manages and controls a computer's hardware activities.

For example, Windows 98, NT, 2000, XP, or ME.

Application programs such as an Internet browser and a word processor cannot run without an operating system.



Why Java?

The answer is that Java enables users to develop and deploy applications on the Internet for servers, desktop computers, and small hand-held devices. The future of computing is being profoundly influenced by the Internet, and Java promises to remain a big part of that future. Java is the Internet programming language.

- Java is a general purpose programming language.
- Java is the Internet programming language.
- Java is portable (machine-independent)

Java, Web, and Beyond

- Developed by James Gosling at Sun Microsystems (May 20, 1995)
- Java can be used to develop Web applications
- Java Supports Applets
- Java can also be used to develop applications for hand-held devices such as Palm and cell phones

JDK Versions

- JDK 1.02 (1995) ***(Java Development Kit)***
- JDK 1.1 (1996)
- JDK 1.2 (1998)
- JDK 1.3 (2000)
- JDK 1.4 (2002)
- JDK 1.5 (2004) a. k. a. JDK 5 or Java 5
- JDK 1.6 (2006) a. k. a. JDK 6 or Java 6
- JDK 1.7 (2010) a. k. a. JDK 7 or Java 7
- JDK 1.8 (2014) a. k. a. JDK 8 or Java 8
- JDK 1.9 (2017) a. k. a. JDK 9 or Java 9

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets. (2 refers to *Java 2 platform*)
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

- <u>The recommended textbook uses J2SE to introduce Java programming.</u>

Popular Java IDEs

- NetBeans Open Source by Sun
- Eclipse Open Source by IBM
- JBuilder by Borland
- MetroWerks CodeWarrior
- BlueJ
- JGRASP (we'll use this IDE in this course, download it from http://www.jgrasp.org/)

A Simple Java Program

Welcome.java

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

Syntax and Semantics

- The ***syntax rules*** of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The ***semantics*** of program statements define the meaning/logic of the statements (program).
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

Errors

- A program can have <u>three types of errors</u>
- The compiler will find syntax errors and other basic problems (*<u>compile-time errors or syntax errors</u>*)
  - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to <u>terminate abnormally</u> (*<u>run-time errors</u>*)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*<u>logical errors</u>*)

Syntax Errors

- Did you make any mistakes when you typed in the examples?
  - If you use the <u>wrong case</u> it won’t work

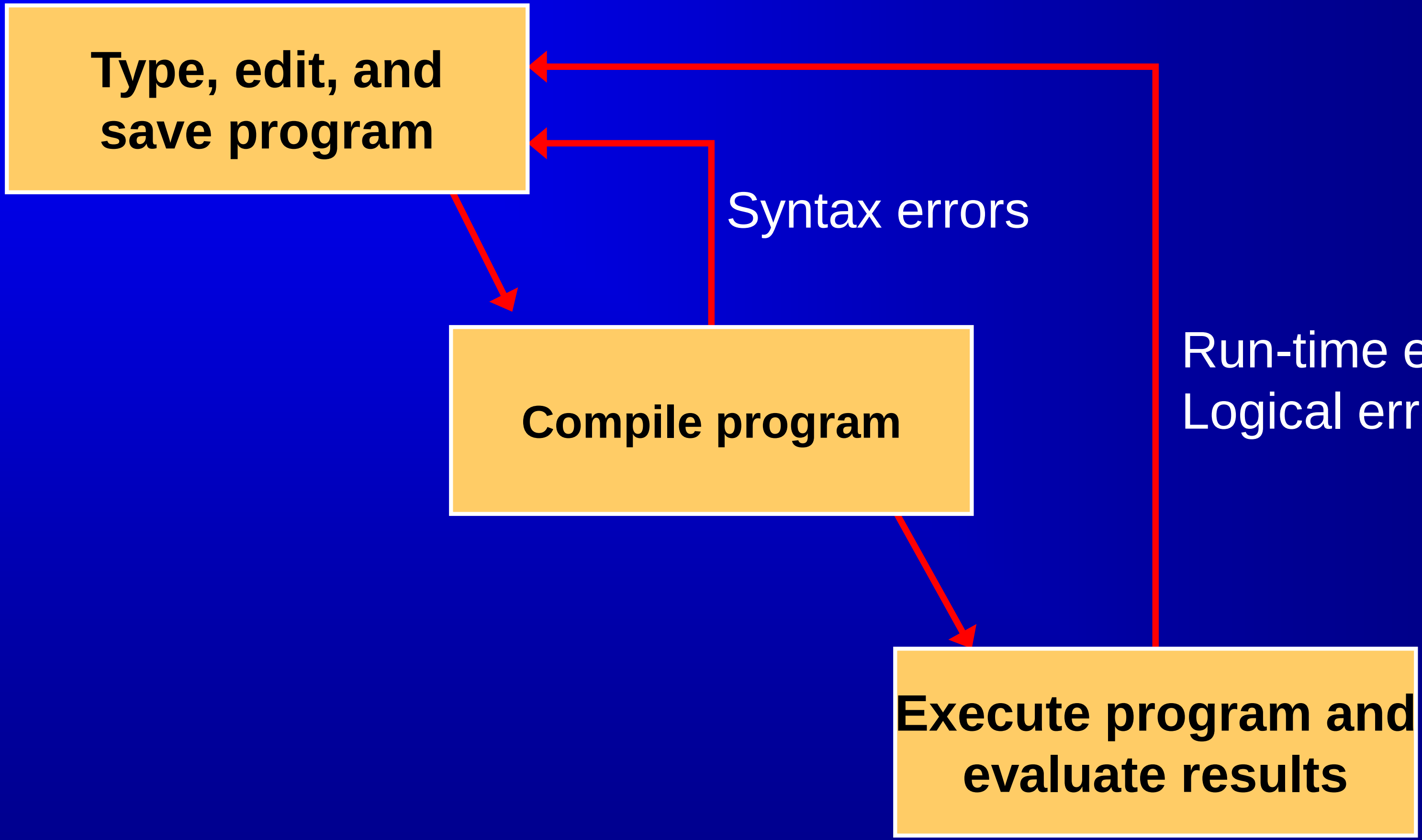
    > math.abs(-3) → Error: Undefined class 'math‘

  - If you <u>misspell</u> something it won’t work

    > Mat.abs(-3) → Error: Undefined class 'Mat'

    > Math.ab(-3) → Error: No 'ab' method in 'java.lang.Math'

Basic Program Development



Type, edit, and save program
Syntax errors
Compile program
Run-time errors
Logical errors
Execute program and evaluate results

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.

```
// 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.

A program/class ***will not*** run without a 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 statement System.out.println("Welcome to Java!") in the program in Listing 1.1 is a statement to display the greeting "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!");
   System.out.println("Again, 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!");
  }
}
```

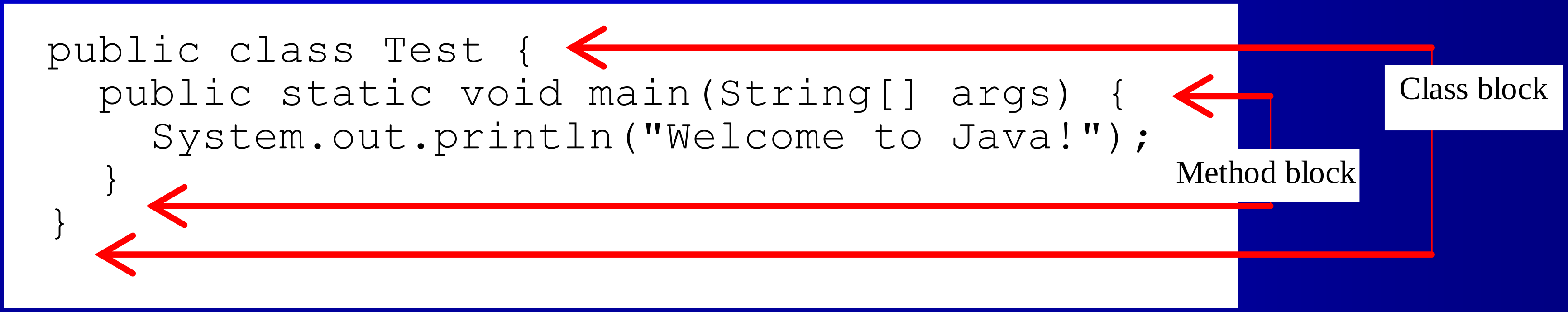
Reserved Words

Java reserved words:

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

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!");
  }
}
```

Special Symbols

Character	Name	Description
{}	Braces	Denotes a block to enclose statements.
()	Parentheses	Used with methods.
[]	Brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	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!");
  }
}
```

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 your name, class section, instructor, 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, the class name `ComputeExpression`.
- Method names:
  - Lowercase name (one word) or capitalize the first letter of every other word in the name. For example, method `main(String[] args)` method `computeAverageGarde(int[] grades)`

Proper Indentation and Spacing

- Indentation
  - Indent two spaces.

    Note: JGrasp has a function that does indentation automatically (called CSD style).

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

Block Styles

Use either style, just be consistent!

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 Chapter 1