COMP231

Advanced Programming

Compiled By: Dr. Majdi Mafarja
Fall Semester 2017/2018

Course Description

In this course, you will learn some of the concepts, fundamental syntax, and thought processes behind true **O**bject-**O**riented **P**rogramming (**OOP**)

Course Description

- Upon completion of this course, you'll be able to:
  - Demonstrate understanding of classes, constructors, objects, and instantiation.
  - Access variables and modifier keywords.
  - Develop methods using parameters and return values.
  - Build control structures in an object-oriented environment.
  - Convert data types using API methods and objects.
  - Design object-oriented programs using scope, inheritance, and other design techniques.
  - Create an object-oriented application using Java packages, APIs. and interfaces, in conjunction with classes and objects.

Logistics

- Instructor: **Majdi Mohamad Mafarja**
  - **Office:** Masri318.
  - Office Hours: S(1-2), TR (11:30 – 12:30), W(11-12)
- Text book:
  - Introduction To JAVA Programming, 10th edition.
  - Author: Y. Daniel Liang.
  - Publisher: Prentice Hall.
- Lab Manual:
  - LABORATORY WORK BOOK (COMP231 – **2017**)

Grading Criteria

- Midterm exam 30%
- 4 Assignments 10%
- 4 Quizzes 15%
- Final Practical Exam 10%
- Final exam 35%

Special Regulations

- **Assignments**:
  - All assignments are **individual** efforts any duplicated copies will be treated as a cheating attempt which lead to **ZERO** mark.
  - Using code from the internet will be treated as cheating as well.
  - The assignments should be **submitted through Ritaj** within the specified deadline.
  - No late submissions are accepted even by **1 minute** after the deadline.

Special Class Regulations

- **Attendance** is mandatory. University regulations will be **strictly** enforced.
- **Mobile**: Keep it off during the class. If your mobile ring you have to leave the classroom quickly, quietly and don't come back.
- **Late**: you are expected to be in the classroom before the teacher arrival. After **5** minutes you will not allowed entering the classroom.

Course Outline

Topics	Chapter	# of lectures
Introduction to Java	1-8	**6**
Objects and Classes	9	**3**
Strings	4.4, 10.10, 10.11	**2**
Thinking in Objects	10	**2**
Inheritance and Polymorphism	11	**3**
Midterm Exam (30%)		
Abstract Classes and Interfaces	13	**3**
Exception Handling and Text I/O	12	**3**
JavaFX Basics	14	**3**
JavaFX UI Controls	16	**3**
Event-Driven Programming	15	**3**
Final Exam (35%)		

Lab Outline

Lab #	Title	Quizzes
1	Program structure in Java	
2	Structure Programming - Revision	
3	Methods	
4	Arrays and Object Use	**Q1**
5	Object-Oriented Programming	
6	String I	
7	String II	**Q2**
8	Inheritance and Polymorphism	
9	Abstract classes and Interfaces	
10	Text I/O	**Q3**
11	GUI	
12	Event-Driven Programming	**Q4**
Practical Final Exam (10%)		

Chapter 1 Introduction to Computers, Programs, and Java

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, so you have to enter binary codes for various instructions. Program with native machine language is a tedious process. Moreover the programs are highly difficult to read and modify. For example, to add two numbers, you might write an instruction in binary like this:

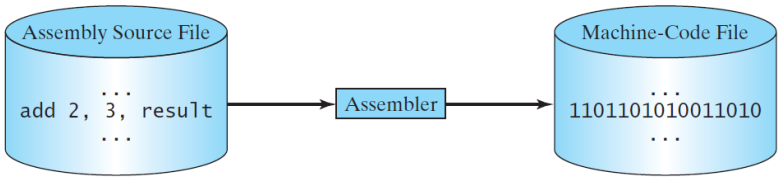
```
1101101010011010
```


Programming Languages

Machine Language Assembly Language High-Level Language

Assembly languages were developed to make programming easy. Since the computer cannot understand assembly language, however, a program called 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:

```
ADDF3 R1, R2, R3
```



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;
```

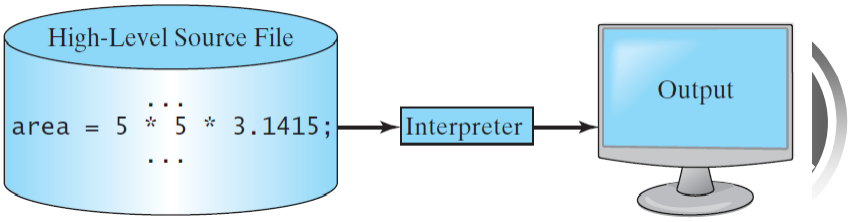
Examples: Basic, VB, C, C++, Java, Python, …

Interpreting/Compiling Source Code

A program written in a high-level language is called a *source program* or ***source code***. Because a computer cannot understand a source program, a source program must be translated into machine code for execution. The translation can be done using another programming tool called an *interpreter* or a *compiler*.

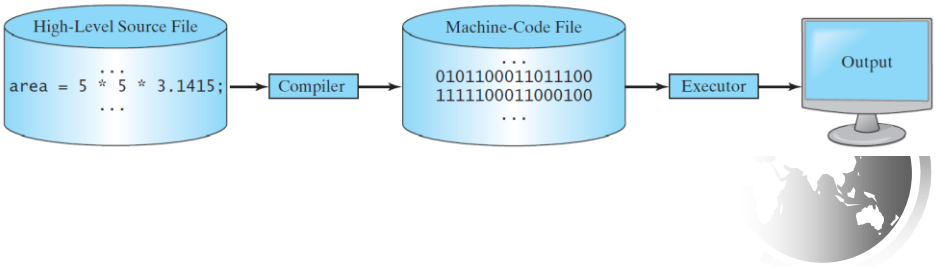
Interpreting Source Code

An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away, as shown in the following figure. Note that a statement from the source code may be translated into several machine instructions.



Compiling Source Code

A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed, as shown in the following figure.



Why Java?

- Java is a general purpose programming language.
- Java is the Internet programming language.

Java, Web, and Beyond

- Java can be used to develop standalone applications.
- Java can be used to develop applications running from a browser.
- Java can also be used to develop applications for hand-held devices.
- Java can be used to develop applications for Web servers.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

www.cs.armstrong.edu/liang/JavaCharacteristics.pdf

JDK Versions

- JDK 1.02 (1995)
- 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 (2011) a. k. a. JDK 7 or Java 7
- JDK 1.8 (2014) a. k. a. JDK 8 or Java 8
  - **Java 8 Update 144 Limited Update / July 26, 2017**
-

The Java Language Specification, API, JDK, and IDE

- *Java syntax is defined in the Java language specification.*
- *Application Program Interface (API),*
  - *Java library.*
- *Java Development Toolkit (JDK) is the software for developing and running Java programs.*
- *Integrated Development Environment (IDE) for rapidly developing programs.*
- (e.g., NetBeans, Eclipse, and TextPad)

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- 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 such as cell phones.

This book uses J2SE to introduce Java programming.

Popular Java IDEs

IDE ➔ **I**ntegrated **D**evelopment **E**nvironment

A Simple Java Program

Listing 1.1

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

Welcome

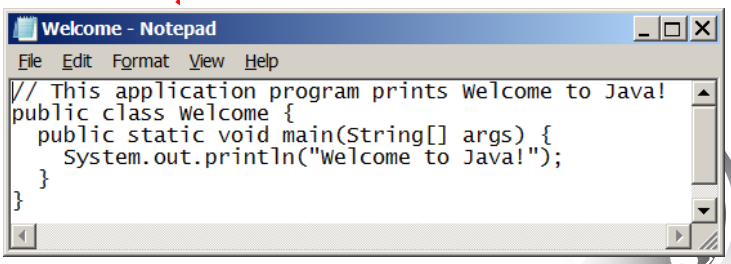
Run

Note: Clicking the green button displays the source code with interactive animation. You can also run the code in a browser. Internet connection is needed for this button.

Note: Clicking the blue button runs the code from Windows. If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc.

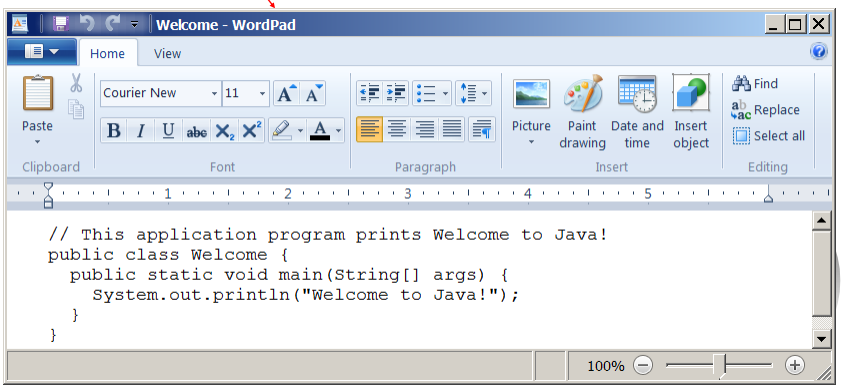
Creating and Editing Using NotePad

To use NotePad, type
notepad Welcome.java
from the DOS prompt.

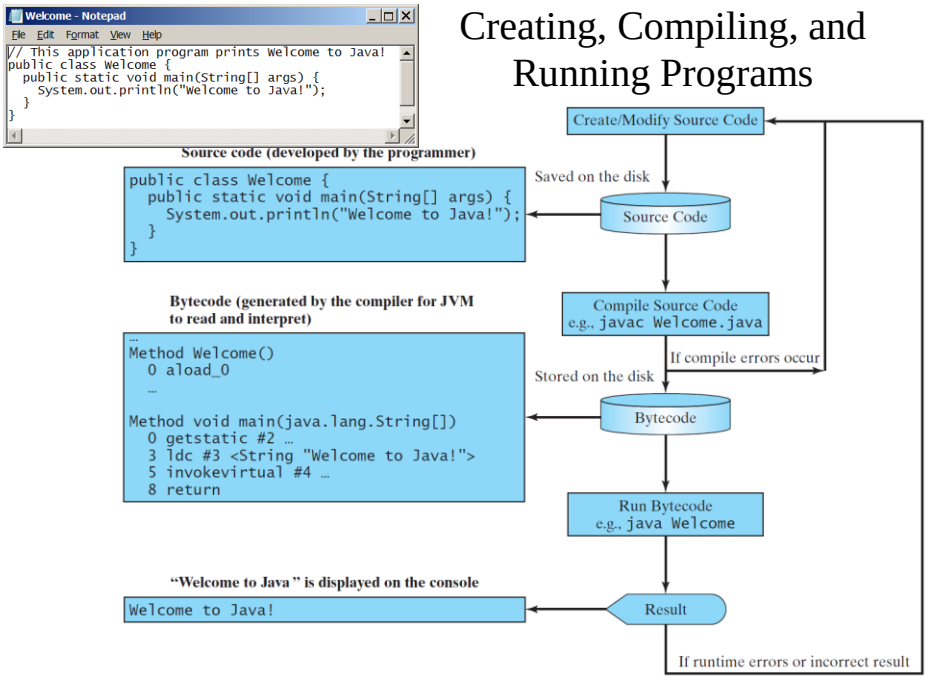


Creating and Editing Using WordPad

To use WordPad, type
write Welcome.java
from the DOS prompt.

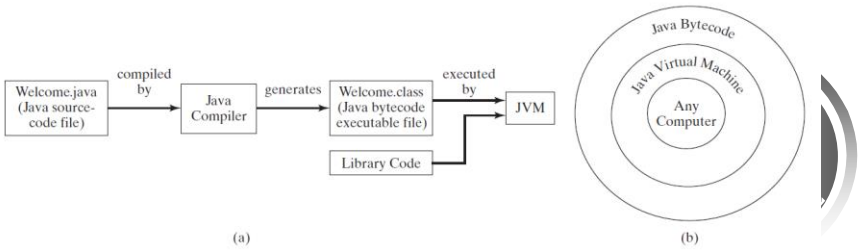


Creating, Compiling, and Running Programs



Compiling Java Source Code

You can port a source program to any machine with appropriate compilers. The source program must be recompiled, however, because the object program can only run on a specific machine. Nowadays computers are networked to work together. Java was 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, as shown below. Java Virtual Machine is a software that interprets Java bytecode.



animation

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!");
  }
}
```


animation

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!");
  }
}
```

animation

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!");
  }
}
```

print a message to the console

Two More Simple Examples

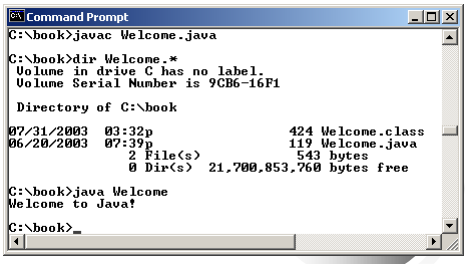
WelcomeWithThreeMessages Run

ComputeExpression Run

Companion Website

Compiling and Running Java from the Command Window

- Set path to JDK bin directory
  - set path=c:\Program Files\java\jdk1.8.0\bin
- Set classpath to include the current directory
  - set classpath=.
- Compile
  - javac Welcome.java
- Run
  - java Welcome



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.

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

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!");
  }
}
```

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`.

Proper Indentation and Spacing

- Indentation
  - Indent two spaces.
- Spacing
  - Use blank line to separate segments of the code.

Block Styles

Use end-of-line style for braces.

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");
  }
}
```


Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result

Syntax Errors

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

ShowSyntaxErrors Run

Runtime Errors

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

ShowRuntimeErrors Run

Logic Errors

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

ShowLogicErrors Run

Displaying Text in a Message Dialog Box

you can use the showMessageDialog method in the JOptionPane class. JOptionPane is one of the many predefined classes in the Java system, which can be reused rather than “reinventing the wheel.”

WelcomeInMessageDialogBox Run

The showMessageDialog Method

```
JOptionPane.showMessageDialog(null,
  "Welcome to Java!",
  "Display Message",
  JOptionPane.INFORMATION_MESSAGE);
```

Two Ways to Invoke the Method

There are several ways to use the showMessageDialog method. For the time being, all you need to know are two ways to invoke it.

One is to use a statement as shown in the example:

```
JOptionPane.showMessageDialog(null, x,
  y, JOptionPane.INFORMATION_MESSAGE);
```

where x is a string for the text to be displayed, and y is a string for the title of the message dialog box.

The other is to use a statement like this:

```
JOptionPane.showMessageDialog(null, x);
```

where x is a string for the text to be displayed.

Implicit Import and Explicit Import

```
java.util.* ; // Implicit import

java.util.JOptionPane; // Explicit Import
```

No performance difference