Module 1

Introduction

Adapted from Absolute Java, Rose Williams, *Binghamton University*

Language Paradigms

- Major Programming Language Paradigms
  - Procedural
    - Imperative
    - Object-Oriented
  - Declarative
    - Functional
    - Logic Programming
- More Concepts
  - Concurrency
  - Exception Handling
  - Persistency
- Other Paradigms
  - Constraint, Rule-Based, Pattern, Scripting, Visual Language Paradigms

The Object Oriented Paradigm

- Programming methodology that views a program as consisting of objects that interact with one another by means of actions (called methods)
- Objects of the same kind are said to have the same type or be in the same class

The Object Oriented Paradigm

- Paradigm Evolution
  - Procedural-Oriented – 1950s-1970s (procedural abstraction)
  - Data-Oriented – early 1980s (data abstraction, called *object-based*)
  - Object-Oriented – late 1980s (Inheritance and dynamic binding)

The Object Oriented Paradigm

- Categories of languages that support OOP:
  - *OOP support is added to an existing language*
    - C++ (also supports procedural and data-oriented programming)
    - Ada 95 (also supports procedural and data-oriented programming)
    - CLOS (also supports functional programming)
    - Scheme (also supports functional programming)
  - *Support OOP, but have the same appearance* and use the basic structure of earlier imperative languages
    - Eiffel (not based directly on any previous language)
    - Java (based on C++)
  - Pure OOP languages
    - Smalltalk

Language Implementation

- Implementation Methods
  - Compilation (Executable Images)
    - Machine Code
  - Pure Interpretation
  - Hybrid Implementation
    - Intermediate Code:
      Machine Language/Assembly Language

Computer Language Levels

- *High-level language*: A language that people can read, write, and understand
  - A program written in a high-level language must be translated into a language that can be understood by a computer before it can be run
- *Machine language*: A language that a computer can understand
- *Low-level language*: Machine language or any language similar to machine language
- *Compiler*: A program that translates a high-level language program into an equivalent low-level language program
  - This translation process is called *compiling*

Byte-Code and the Java Virtual Machine

- The compilers for most programming languages translate high-level programs directly into the machine language for a particular computer
  - Since different computers have different machine languages, a different compiler is needed for each one
- In contrast, the Java compiler translates Java programs into *byte-code*, a machine language for a fictitious computer called the *Java Virtual Machine*
  - Once compiled to *byte-code*, a Java program can be used on any computer, making it highly portable

Byte-Code and the Java Virtual Machine

- *Interpreter:* The program that translates a program written in Java byte-code into the machine language for a particular computer when a Java program is executed
  - The interpreter translates and immediately executes each byte-code instruction, one after another
  - Translating byte-code into machine code is relatively easy compared to the initial compilation step

The Unified Modeling Language (UML)

- Pseudocode is a way of representing a program in a linear and algebraic manner
  - It simplifies design by eliminating the details of programming language syntax
- Graphical representation systems for program design have also been used
  - *Flowcharts* and *structure diagrams* for example
- *Unified Modeling Language* (*UML*) is yet another graphical representation formalism
  - UML is designed to reflect and be used with the OOP philosophy

Introduction to Java

- Most people are familiar with Java as a language for Internet applications
- We will study Java as a general purpose programming language
  - The syntax of expressions and assignments will be similar to that of other high-level languages
  - Details concerning the handling of strings and console output will probably be new

Origins of the Java Language

- Created by Sun Microsystems team led by James Gosling (1991)
- Originally designed for programming home appliances
  - Difficult task because appliances are controlled by a wide variety of computer processors
  - Team developed a two-step translation process to simplify the task of compiler writing for each class of appliances

Origins of the Java Language

- Significance of Java translation process
  - Writing a compiler (translation program) for each type of appliance processor would have been very costly
  - Instead, developed intermediate language that is the same for all types of processors : Java *byte-code*
  - Therefore, only a small, easy to write program was needed to translate byte-code into the machine code for each processor

Program terminology

- *Code*: A program or a part of a program
- *Source code* (or *source program*): A program written in a high-level language such as Java
  - The input to the compiler program
- *Object code*: The translated low-level program
  - The output from the compiler program, e.g., Java byte-code
  - In the case of Java byte-code, the input to the Java byte-code interpreter

Class Loader

- Java programs are divided into smaller parts called ***classes***
  - Each class definition is normally in a separate file and compiled separately
- *Class Loader*: A program that connects the byte-code of the classes needed to run a Java program
  - In other programming languages, the corresponding program is called a *linker*

Java Application Programs

- There are two types of Java programs: *applications* and *applets*
- A Java *application program* or "regular" Java program is a class with a method named **`main`**
  - When a Java application program is run, the *run-time system* automatically invokes the method named **`main`**
  - All Java application programs start with the **`main`** method

Applets

- A Java *applet* (*little Java application*) is a Java program that is meant to be run from a Web browser
  - Can be run from a location on the Internet
  - Can also be run with an applet viewer program for debugging
  - Applets always use a windowing interface
- In contrast, application programs may use a windowing interface or console (i.e., text) I/O

A Sample Java Application

Display 1.1 A Sample Java Program

Name of class (program)

The `main` method

```
1  public class FirstProgram
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("Hello reader.");
6          System.out.println("Welcome to Java.");
7          System.out.println("Let's demonstrate a simple calculation.");
8          int answer;
9          answer = 2 + 2;
10         System.out.println("2 plus 2 is " + answer);
11     }
12 }
```

SAMPLE DIALOGUE 1

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```

Syntax and Semantics

- *Syntax*: The arrangement of words and punctuations that are legal in a language, the *grammar rules* of a language
- *Semantics*: The meaning of things written while following the syntax rules of a language
- Compilation can uncover syntax errors but not semantic ones

Comments

- A *line comment* begins with the symbols `//`, and causes the compiler to ignore the remainder of the line
  - This type of comment is used for the code writer or for a programmer who modifies the code
- A *block comment* begins with the symbol pair `/*`, and ends with the symbol pair `*/`
  - The compiler ignores anything in between
  - This type of comment can span several lines
  - This type of comment provides documentation for the users of the program

Program Documentation

- Java comes with a program called **`javadoc`** that will automatically extract documentation from block comments in the classes you define
  - As long as their opening has an extra asterisk (**`/**`**)
- Ultimately, a well written program is self-documenting
  - Its structure is made clear by the choice of identifier names and the indenting pattern
  - When one structure is nested inside another, the inside structure is indented one more level

@ Tags

- @ tags should be placed in the order found below
- If there are multiple parameters, each should have its own `@param` on a separate line, and each should be listed according to its left-to-right order on the parameter list
- If there are multiple authors, each should have its own `@author` on a separate line

```
@param Parameter_Name Parameter_Description
@return Description_Of_Value_Returned
@throws Exception_Type Explanation
@deprecated
@see Package_Name.Class_Name
@author Author
@version Version_Information
```

Compiling a Java Program or Class

- Each class definition must be in a file whose name is the same as the class name followed by `.java`
  - The class `FirstProgram` must be in a file named `FirstProgram.java`
- Each class is compiled with the command `javac` followed by the name of the file in which the class resides

```
javac FirstProgram.java
```

  - The result is a byte-code program whose filename is the same as the class name followed by `.class`

```
FirstProgram.class
```

- For now, your program and all the classes it uses should be in the same directory or folder

Running a Java Program

- A Java program can be given the *run command* (**`java`**) after all its classes have been compiled
  - Only run the class that contains the **`main`** method (the system will automatically load and run the other classes, if any)
  - The **`main`** method begins with the line:

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

  - Follow the run command by the name of the class only (no **`.java`** or **`.class`** extension)

```
java FirstProgram
```