Introduction to Java and OOP

Dr. M. G. Abbas Malik
Assistant Professor
Faculty of Computing & Information Technology
King Abdulaziz University, North Jeddah Branch
mgmalik@kau.edu.sa
Course Book

Slides Modified for:

“Advanced Programming”
CPIT305

“Java – How to Program”, 9th Edition, Paul Deitel & Harvey Deitel

Topics - Course Module CPIT-305

- Object Oriented Programming – Review
- Exception Handling – A Deeper Look
- Files, Streams and Object Serialization
- Generic Collections
- Multithreading
- Networking
- Accessing Database with JDBC
Course

- Title and Code: CPIT 305 “Advanced Programming”
- Instructor: Dr. Muhammad Ghulam Abbas Malik
- Course Timing: 0800 to 0920 every Sunday and Tuesday
- Course Home: www.sanlp.org/cpit305/ap.html

Course Lab

- Instructor: Maha Alaslani
- Timing: 1300 to 1420 every Sunday
Outline

1.1  Introduction to Java
1.3  Data Hierarchy
1.6  Introduction to Object Technology
1.9  Java and Typical Development Environment
1.1 Introduction

- Java is the **most widely used** software development language in the world.
  - As of 2010 97% of enterprise desktops, three billion handsets, and 80 million television devices run Java.
- Java has become the language of choice for implementing **Internet-based** applications and software for devices that communicate over a network.
- You’ll learn **object-oriented programming**—today’s key programming methodology.
- You’ll learn **advanced features** of Java programming language such as exception handling, generic collections, files and streams, multithreading, networking etc.
1.3 Data Hierarchy

- Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from bits to characters to fields, and so on.
  - The smallest data item in a computer can assume the value 0 or the value 1 called a **bit**
  - Digits, letters and special symbols are known as **characters**
  - A **field** is a group of characters or bytes that conveys meaning
  - Several related fields can be used to compose a **record**
  - A **file** is a group of related records
1.6 Introduction to Object Technology

- **Objects** are essentially reusable software components.
  - Almost any noun can be reasonably represented as a software object in terms of attributes (e.g., name, color and size) and behaviors (e.g., calculating, moving and communicating).

- **Object-Oriented (OO)** design and implementation approach can make software-development groups much more productive than was possible with earlier popular techniques like “structured programming”

- Object-Oriented programs are often easier to understand, correct and modify.
The **Automobile** as an Object

Let’s begin with a simple analogy. Suppose you want to **drive** a car and make it go faster by pressing its accelerator pedal and stop it by pressing brake pedal.

Before you can drive a car, someone has to **design** it.

A car typically begins as **engineering drawings**, similar to the blueprints that describe the design of a house. Then based on these drawings, a car is manufactured.

Pedal **hides from the driver the complex mechanisms** that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel “hides” the mechanisms that turn the car.

Enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.
1.6 Introduction to Object Technology (Cont.)

- Methods and Classes
  - The method consists of program statements that actually perform desired tasks.
  - In Java, we create a program unit called a class which consists of a set of methods that perform the class’s tasks.
  - A class is similar in concept to a car’s engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.
1.6 Introduction to Object Technology (Cont.)

- Instantiation
  - Just as someone has to **build** a car from its engineering drawings before you can actually drive a car, you must build an object of a class before a program can perform the tasks that the class’s methods define.
  - An object is then referred to as an **instance** of its class.
1.6 Introduction to Object Technology (Cont.)

- **Reuse**
  - Just as a car’s engineering drawings can be reused many times to build many cars, you can reuse a class many times to build many objects.
  - Reuse of existing classes when building new classes and programs saves time and effort.
  - Reuse also helps you build more reliable and effective systems, because existing classes and components often have gone through extensive testing, debugging and performance tuning.
Messages and Methods Calls

When you drive a car, pressing its gas pedal sends a message to the car to perform a task—that is, to go faster.

Similarly, you send messages to an object.

Each message is implemented as a method call that tells a method of the object to perform its task.
1.6 Introduction to Object Technology (Cont.)

- Attributes and Instance Variables
  - A car has attributes
    - Color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading).
  - Every car maintains its own attributes.
  - Each car knows how much gas is in its own gas tank, but not how much is in the tanks of other cars.
  - Attributes are specified by the class’s instance variables
Encapsulation

- Classes encapsulate (i.e., wrap) attributes and methods into objects—an object’s attributes and methods are intimately related.

- Objects may communicate with one another, but they’re normally not allowed to know how other objects are implemented—implementation details are hidden within the objects themselves.

- Information hiding, as we’ll see, is crucial to good software engineering.
1.6 Introduction to Object Technology (Cont.)

- **Inheritance**
  - A new class of objects can be created quickly and conveniently by *inheritance*
  - The new class *absorbs* the characteristics of an existing class, possibly *customizing* them and *adding* unique characteristics of its own
Object-Oriented Analysis and Design (OOAD)

- How will you create the code (i.e., the program instructions) for your programs?
  - Follow a detailed **analysis** process for determining your project’s **requirements** (i.e., defining what the system is supposed to do)
  - Develop a **design** that satisfies them (i.e., deciding how the system should do it).
  - Carefully **review** the design (and have your design reviewed by other software professionals) before writing any code.
Analyzing and designing your system from an object-oriented point of view is called an object-oriented analysis and design (OOAD) process.

Languages like Java are object oriented.

Object-oriented programming (OOP) allows you to implement an object-oriented design as a working system.
1.6 Introduction to Object Technology (Cont.)

- The **UML** (Unified Modeling Language)
  - The Unified Modeling Language (UML) is the most widely used graphical scheme for modeling object-oriented systems.
1.9 Java and a Typical Java Development Environment

- Java Class Libraries
  - Rich collections of existing classes and methods also known as the Java APIs (Application Programming Interfaces).

- Java programs normally go through five phases
  - edit
  - compile
  - load
  - verify
  - execute.
1.9 Java and a Typical Java Development Environment (Cont.)

- **Phase 1** consists of editing a file with an Editor
  - Type a Java program (source code) using the editor.
  - Make any necessary corrections.
  - Save the program.
  - A file name ending with the `.java extension` indicates that the file contains Java source code.
1.9 Java and a Typical Java Development Environment (Cont.)

- **Phase 2: Compiling a Java Program into Bytecodes**
  - Use the command `javac` (the Java compiler) to compile a program. The compiler produces a `.class` file that contains the compiled version of the program.
1.9 Java and a Typical Java Development Environment (Cont.)

- Bytecodes are executed by the **Java Virtual Machine (JVM)** – a software application that simulates a computer
  - Hides the underlying operating system and hardware from the programs that interact with it.
- Bytecodes are **platform independent** and are **portable**
- The JVM is invoked by the **java** command.
Phase 3: Loading a Program into Memory

- The JVM places the program in memory to execute it—this is known as **loading**.
- **Class loader** takes the `.class` files containing the program’s bytecodes and transfers them to primary memory.
Phase 4: Bytecode Verification

- As the classes are loaded, the bytecode verifier examines their bytecodes, ensures that they’re valid and do not violate Java’s security restrictions.
- Java enforces strong security to make sure that Java programs arriving over the network do not damage your files or your system (as computer viruses and worms might).
Phase 5: Execution

- The JVM executes the program’s bytecodes.
- JVMs typically execute bytecodes using a combination of interpretation and so-called just-in-time (JIT) compilation.
- A just-in-time (JIT) compiler—known as the Java HotSpot compiler—translates the bytecodes into the underlying computer’s machine language.
When the JVM encounters these compiled parts again, the faster machine-language code executes.

Java programs go through two compilation phases:
- One in which source code is translated into bytecodes (for portability across JVMs on different computer platforms) and
- A second in which, during execution, the bytecodes are translated into machine language for the actual computer on which the program executes.