Book 3

Chapter One

Understanding Object-Oriented Programming

Object-Oriented Programming (OOP)

Creating objects that contain both data and methods

Objects have characteristics:

1. **Identity**
    - Every occurrence of the object (i.e. instance) can be distinguished from all other occurrences of the same type.
    - Two objects of the same type have their own memory locations – allowing the computer to distinguish one object from another.
    - Each object has a hash code – int value automatically generated, representing the object's identity.
2. **Type**
    - Assign names to different kinds of objects
    - Objects created from a type = classes
    - E.g. Invoice 1 = new Invoice() therefore id = 1, type = Invoice
3. **State**
    - Combo of all the values for all attributes = object's state
    - E.g. Object: car; Attributes: Model and Colour; Values: Model – BMW, Colour – Red
    - State of an object represented by class variables = fields. i.e. public
4. **Behaviour**
    - Provided by its method
    - Behaviour is not different for each instance of a type – calculations are done the same way
    - Interface of a class is a set of methods and fields that the class makes public – accessible by other objects

Life Cycle of an object

1. Load class
    a. Java runtime locates class and reads it into memory
    b. Searches for static initializers – intialise static fields (belong to the class itself and shared by all objects created from the class)
    c. Run main method of a class, class is initialized because main method is static
2. Create object from class using *new* keyword
    a. Class initialized through object being allocated memory and reference for the Java runtime to keep track of it. Constructor responsible for all processing
3. Object provides access to public methods and fields

4. Object will be removed from memory and internal reference dropped
    a. Garbage collector takes care of destroying objects no longer in use

Working with Related Classes

Java has 2 object-oriented programming features designed to handle classes that are related – inheritance and interfaces.

1. **Inheritance:**
    a. Where one class acquires the properties (methods and fields) of another
    b. Class that inherits the properties of another is known as a subclass
    c. Class whose properties are inherited is known as superclass
    d. Make use of key word "extends"
        i. E.g. public class Super {
        …
        …
        }
        Class Sub extends Super {
        …
        …
        }

2. **Interfaces**
    a. Set of methods and fields that a class must provide to implement the interface
    b. Set of public methods and field declarations
    c. Code not provided to implement the methods – only declarations
    d. CLASS that implements the interface provides code for each method

3. **Designing a Program with Objects**
    a. Decide which classes the application requires and the public interface to those classes
    b. Plan classes properly for application to run smoothly
    c. Divide application into layers/tiers
        i. Presentation
        ii. Logic
        iii. Database

Diagramming Classes with UML

Draw class diagrams that illustrate relationship among the classes that make up an application

Make use of UML – unified modeling language