Course Name: Object Oriented Programming
Course code: CS36

Unit-1

The Object Model Foundations of the Object Model: Object-Oriented Programming, Object-Oriented Design, Object-Oriented Analysis Elements of the Object Model: Abstraction, Encapsulation, Modularity, Hierarchy, Typing, Concurrency, Persistence Applying the Object Model. Introduction to Java Programming: Java Buzzwords, Overview of Java Datatypes, Variables, arrays, Control statements.

This notes is meant for only referential purpose, kindly refer to the syllabus proposed and prepare as per the prescribed textbooks listed, Lecture notes/PPT shared to you.

- **Abstraction: Types of Abstraction: Example of Abstraction**

Abstraction: An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. An abstraction focuses on the outside view of an object and so serves to separate an object's essential behavior from its implementation.

Kinds of Abstractions:

1. **Entity abstraction:** An object that represents a useful model of a problem domain or solution domain entity
2. **Action abstraction:** An object that provides a generalized set of operations, all of which perform the same kind of function
3. **Virtual machine abstraction:** An object that groups operations that are all used by some superior level of control, or operations that all use some junior-level set of operations

Example:

- Abstraction shows only important things to the user and hides the internal details, for example, when we ride a bike, we only know about how to ride bikes but can not know about how it work? And also we do not know the internal functionality of a bike.
- Another real life example of Abstraction is ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know internal details about ATM.

- **Five metrics that measure the quality of Abstraction:**

1. **Coupling:** coupling as "the measure of the strength of association established by a connection from one module to another. Complexity can be reduced by designing systems with the weakest possible coupling between modules
2. **Cohesion:** cohesion measures the degree of connectivity among the elements of a single module. Consider a class comprising the abstractions of dogs and spacecraft, whose behaviors are quite unrelated. The most desirable form of cohesion is functional cohesion,

in which the elements of a class or module all work together to provide some well-bounded behavior

3. **Sufficiency:** sufficiency implies a minimal interface, a complete interface is one that covers all aspects of the abstraction
4. **Completeness:** A complete class or module is thus one whose interface is general enough to be commonly usable to any client
5. **Primitiveness:** Providing all meaningful operations for a particular abstraction overwhelms the user and is generally unnecessary since many high-level operations can be composed from low-level ones. For this reason, we also suggest that classes and modules be primitive.

- **Multiple Inheritance: Solution to implement Multiple Inheritance**

Multiple Inheritance: When one class inherits the properties of 2 or more classes.
Key Difficulty with Multiple Inheritance:

1. Name Collisions from different super classes
2. Handling Repeated Inheritance

Resolving Name Collisions:

1. Language semantics might regard such a clash as illegal and reject the compilation of the class.
2. Language semantics might regard the same name introduced by different classes as referring to the same attribute.
3. Language semantics might permit the clash but require that all references to the name fully qualify the source of its declaration

Resolving Repeated Inheritance:

1. We can treat occurrences of repeated inheritance as illegal.
2. We can permit duplication of super classes but require the use of fully qualified names to refer to members of a specific copy.
3. We can treat multiple references to the same class as denoting the same class.

- **Object Oriented Programming: Features: Factors that make OO**

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.
There are three important parts to this definition

- Object-oriented programming uses objects, not algorithms, as its fundamental logical building blocks.
- Each object is an instance of some class.
- Classes may be related to one another via inheritance relationships.

The important features of object–oriented programming are:

1. Bottom–up approach in program design

2. Programs organized around objects, grouped in classes
3. Focus on data with methods to operate upon object's data
4. Interaction between objects through functions
5. Reusability of design through creation of new classes by adding features to existing classes.

A language is object-oriented if and only if it satisfies the following requirements:
1. It supports objects that are data abstractions with an interface of named operations and a hidden local state.
2. Objects have an associated type [class].
3. Types [classes] may inherit attributes from supertypes [superclasses].

Object Oriented Analysis: Primary Tasks:

"Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain." Traditional structured analysis techniques focus on the flow of data within a system. Object-oriented analysis (OOA) emphasizes the building of real-world models, using an object-oriented view of the world.
The primary tasks in object-oriented analysis (OOA) are:

1. Identifying objects
2. Organizing the objects by creating object model diagram
3. Defining the internals of the objects, or object attributes
4. Defining the behavior of the objects, i.e., object actions
5. Describing how the objects interact.

- **Object Oriented Design:**

Object-oriented design is a method of design encompassing the process of object oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design
There are two important parts to this definition: object-oriented design
(1) leads to an object-oriented decomposition.
(2) uses different notations to express different models of the logical (class and object structure) and physical (module and process architecture) design of a system.

- The support for object-oriented decomposition is what makes object oriented design quite different from structured design.
- The former uses class and object abstractions to logically structure systems, and the latter uses algorithmic abstractions.
- The term object-oriented design is used to refer to any method that leads to an object-oriented decomposition.

The implementation details generally include:
1. Restructuring the class data (if necessary)
2. Implementation of methods, i.e., internal data structures and algorithms
3. Implementation of control

4. Implementation of associations.

- **Major and minor elements of object model:**

Major elements of object model	Minor elements of object model
1. Abstraction 2. Encapsulation 3. Modularity 4. Hierarchy	1. Typing 2. Concurrency 3. Persistence

- **Uses of Object Oriented language:**
  - It helps in faster development of software.
  - It is easy to maintain. ...
  - It supports relatively hassle-free upgrades.
  - It enables reuse of objects, designs, and functions.
  - It reduces development risks, particularly in integration of complex systems.

- **Modularity: Example:**

Modularity:

- "The act of partitioning a program into individual components can reduce its complexity to some degree.
- "Modularization consists of dividing a program into modules which can be compiled separately, but which have connections with other modules.
- Deciding on the right set of modules for a given problem is almost as hard a problem as deciding on the right set of abstractions.
- Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.

Example:

- Suppose we decide to use a commercially available workstation where the user can control the system's operation.
- At this workstation, an operator could create new growing plans, modify old ones, and follow the progress of currently active ones.
- Since one of our key abstractions here is that of a growing plan, we might therefore create a module whose purpose is to collect all of the classes associated with individual growing plans (e.g., FruitGrowingPlan, GrainGrowingPlan).

- **Java Buzzwords:**
  - Simple
  - Secure
  - Portable
  - Object-oriented

- Robust
- Architecture-neutral (or) Platform Independent
- Multi-threaded
- Interpreted
- High performance
- Distributed
- Dynamic

➢ **Display all the duplicate values with the frequency of their occurrence:**

```
import java.io.*;
class GFG {
   public static void main(String[] args)
   {
      int A[] = { 1, 6, 4, 6, 4, 8, 2, 4, 1, 1 };
      int max = Integer.MIN_VALUE;
      for (int i = 0; i < A.length; i++) {
         if (A[i] > max)
            max = A[i];
      }
      int B[] = new int[max + 1];
      for (int i = 0; i < A.length; i++) {

         // increment in array B for every integer
         // in A.
         B[A[i]]++;
      }
      for (int i = 0; i <= max; i++) {
         // output only if element is more than
         // 1 time in array A.
         if (B[i] > 1)
            System.out.println(i + "-" + B[i]);
      }
   }
}
```

➢ **Static and Dynamic Typing:**

Static and Dynamic Typing:

- In Java static typing refers to the execution of a program where type of object is determined/known at compile time i.e when compiler executes the code it know the type of object or class to which object belongs. While in case of dynamic typing the type of object is determined at runtime.

- Also static typing uses type of class to bind while dynamic typing uses type of object as the resolution happens only at runtime because object only created during runtime due to which dynamic typing becomes slower than in case of static typing.
- As private,final and static modifiers binds to the class level so methods and variables uses static typing and bonded by compiler while the other methods are bonded during runtime based upon runtime object.
- In general we can say that overloaded methods are bonded using static typing while overridden methods are bonded using dynamic typing.

➢ **Illustrate the use of continue statement and break statement:**

```
//Java Program to illustrate the use of continue statement
//with label inside an inner loop to break outer loop
public class BreakExample3 {
public static void main(String[] args) {
        aa:
        for(int i=1;i<=3;i++){
            bb:
            for(int j=1;j<=3;j++){
               if(i==2&&j==2){
                  //using break statement with label
                  break aa;
               }
               System.out.println(i+" "+j);
            }
        }
}
}
```

//Java Program to demonstrate break statement

```
class Main {
  public static void main(String[] args) {

    // outer loop is labeled as first
    first:
    for (int i = 1; i < 6; ++i) {

      // inner loop
      for (int j = 1; j < 5; ++j) {
        if (i == 3 || j == 2)

          // skips the current iteration of outer loop
          continue first;
```

```
            System.out.println("i = " + i + "; j = " + j);
          }
        }
      }
    }
```

- **Just-In-Time compiler:**

Java uses Just-In-Time compiler to enable high performance. Just-In-Time compiler is a program that turns Java bytecode, which is a program that contains instructions that must be interpreted into instructions that can be sent directly to the processor.

- **Java Bytecode:**

Java bytecode is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.

When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code. When we wish to run this .class file on any other platform, we can do so. After the first compilation, the bytecode generated is now run by the Java Virtual Machine and not the processor in consideration. This essentially means that we only need to have basic java installation on any platforms that we want to run our code on. Resources required to run the bytecode are made available by the Java Virtual Machine, which calls the processor to allocate the required resources. JVM's are stack-based so they stack implementation to read the codes.

- **Variable length array in Java:**

An example of variable size array can be declaring in the following format –

```
int x[][]=new int[3][];
x[0]=new int[2];
x[1]=new int[4];
x[2]=new int[3];
```

ii.

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

- **JVM: Tasks of JVM:**

JVM is a virtual machine or a program that provides run-time environment in which java byte code can be executed. JVMs are available for many hardware and software platforms. The use of the same byte code for all JVMs on all platforms make java platform independent.

Main task of JVM:

1. Search and locate the required files.
2. Convert byte code into executable code.
3. Allocate the memory into ram
4. Execute the code.
5. Delete the executable code

- **WORA: Platform Independence:**

Write once, run anywhere (WORA) – that means java program can run anywhere and on any platform. When java code is compiled it is converted into byte code. Now only this byte code is needed to run using JVM, no need of source code and recompilation.

- **Hello world Program: With explanation:**

Refer Slide 34 and 35 of Unit-1

- **Create and compile Java Program:**

Refer Slide 37 to 40 of Unit-1

- **Ragged arrays in Java:**

Refer Slide 53 and 54 of Unit-1