SEC: Programming in Java

OOPs Principles

Mahesh Kumar
(maheshkumar@andc.du.ac.in)

Course Web Page
(www.mkbhandari.com/mkwiki)

Outline

1. Object
2. Class
3. Encapsulation
4. Abstraction
5. Inheritance
6. Polymorphism

Object Oriented Programming Languages

- Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.
  - → Objects interacts among themselves to implement program logic.
- The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.
- Simula is considered the first object-oriented programming language.
- Smalltalk is considered the first truly object-oriented programming language.
- The popular object-oriented languages are Java, Python, C++, C#, PHP, etc.

Object Oriented Programming Languages

- Object-Oriented programming has several advantages over procedural programming:

  → OOPs makes development and maintenance easier where as in Procedure-Oriented Programming language it is not easy to manage if code grows as project size grows. *The core concept of the object-oriented approach is to break complex problems into smaller objects.*

  → OOPs provides data hiding whereas in Procedure-Oriented Programming language a global data can be accessed from anywhere.

  → OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

  → OOP makes it possible to create full reusable applications with less code and shorter development time

Class

- In Java, every small or large program can only be created using a template called class.

- A class consists of data(fields) and code(methods) that operate on that data.
  - → The class defines the values the object can contain and the operations that can be performed on the object.

- A class is used to create and define objects.
  - → Class is a template/blueprint/framework/model from which objects are created. For example: College, Faculty, Student, Car, Bike, City, Laptop, Mobile, etc.
  - → It is a logical entity. It can't be physical.
  - → Once a class is created/defined, any number of objects can be created from it.
  - → A class is a group of objects which have common properties.

Class

- We can create a class in Java using the class keyword. For example:

```
class ClassName {

    // fields

    // methods

}
```

- Here, fields (variables) and methods represent the state and behavior of the object respectively.
- *fields* (variables) are used to store data
- *methods* are used to perform some operations

Class

- We can create a class in Java using the class keyword. For example:

```
class ClassName {

    // fields

    // methods

}
```

```
class Student {

    // state or field
    private int rollNumber;

    // behavior or method
    public void attemptQuiz( ) {
        System.out.println("Attempting a Java Quiz");

    }
}
```

Object

- Object are basic building blocks in an object oriented programming languages.
- An object is a real-world entity that has state and behavior. It can be physical or logical (tangible and intangible).
- An object is a runtime entity and has three characteristics:
  1. **State**: represents the data (value) of an object.
  2. **Behavior**: represents the behavior (functionality) of an object such as deposit, withdraw, post, share, download, etc.
  3. **Identity**: represents unique ID not visible to the external user, used internally by the JVM to identify each object uniquely.
- An object is an instance of a class.
  - → A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object

- We can create an Object of a Class in Java as follows:

```
className object = new className( );


// for Student class
Student s1 = new Student( );

Student s2 = new Student( );
```

Encapsulation

- Encapsulation is a protective mechanism by which member of a class (methods and variables) are prevented from being accessed by members of other classes.

- A class is an example of encapsulation because a class binds its variables and methods into a single unit and hides their complexity from other classes.

- *Variables* (fields or data) in a class are generally marked private to prevent other classes from accessing them.

- *Methods* are created as public that other classes can access.
  - → By accessing these public methods, variables can be accessed.
  - → This is the only way of interacting with the variables declared private in a class.

Encapsulation - Implementation

```
class Student{
        private String name;
        private String course;
        public void info(String n, String c){
                name=n;
                course=c;
                System.out.println(name+ ”doing Graduation in :” +course);
        }
}

public class EncapsulationDemo{
        public static void main(String args[ ]){
                Student s1 = new Student( );
                s1.info(“Mark Zuckerberg”, “B.Sc.(P) Computer Science”);
        }
}
```

Abstraction

- Abstraction is the concept of object-oriented programming that "shows" only essential attributes and "hides" unnecessary information.

- The advantage of abstraction is that the user can work only with the needed data and is not required to view the unwanted data.

- In OOPs, you need to know about the specific classes or methods that are called to implement specific program logic, without knowing how these classes or methods function.

Abstraction - Implementation

```
class Customer{
        private int accNo;
        private String cName;
        private double balance, salary, credit
        public void display(int anum, Stiring name, String bal){
                accNo=anum;
                cName=name;
                balance=bal;
                System.out.println(”Account Number=”+accNo);
                System.out.println(”Customer Name =”+cName);
                System.out.println(”Total Balance =”+balance);
        }
}
public class AbstractionDemo{
        public static void main(String args[ ]){
                Customer c1= new Customer( );
                c1.display(101,“Sundar Pichai”,60000000);
        }
}
```

Inheritance

- Inheritance in Java is a mechanism in which object of one class acquires all the properties and behaviors of another existing class(parent object). *For example*: the BallPointPen class is a subclass of the Pen class.
- It is an important part of OOPs (Object Oriented programming system).
- It gives the concept (idea) of reusability.
- A class that is inherited is called a superclass and the class that inherits the superclass is called a subclass. (*hierarchical classification*)
- ***Note***: Multiple inheritance is not supported in Java, which means a subclass can extend only one superclass in Java.

Inheritance - Implementation

```
class Person{
        private String name;
        public void setName(Stiring n){
                name=n;
        }
        public void getName( ){
                System.out.println(”Name : ” +name);\
        }
}

public class SportsPerson extends Person{
        private String sport;
        public void setSport(Stiring sp){
                sport = sp;
        }
        public void getSport( ){
                System.out.println(”Sport : ” +sport);
        }
```

Inheritance - Implementation

```
// main method
public static void main(String[ ] args){

        SportsPerson sp = new SportsPerson( );

        sp.setName("Rafael Nadal");
        sp.setSport("Lawn Tennis");

        sp.getName( );
        sp.getSpor( );

}
```

Polymorphism

- Polymorphism is the ability of an object to take on many forms.
- In OOPs, Polymorphism allows you to perform various operations by using methods with the same name.
- In Java, Polymorphism is performed by changing the implementation of the method.
- Polymorphism can be divided into the following two types:
  1. **Static (compile time) polymorphism**: - The behavior of the method is decided during compilation. (*method overloading*)
  2. **Dynamic (run time) polymorphism:** - The behavior of the method is decided during runtime. (*method overriding*)
- The type of Polymorphism depends on how the method is invoked in a class.

Implementing the *static* Polymorphism

```
class StaticPoly{

        void product(int x, int y){
                System.out.println(”Product of two numbers: ” +(x*y) );
        }

        void product(int x, int y, int z){
                System.out.println(”Product of three numbers: ” +(x*y*z) );
        }

        public static void main(String args[ ]){

                StaticPoly obj=new StaticPoly( );
                obj.product(10,20);
                obj.product(10,20,30);

        }
}
```

Implementing the *dynamic* Polymorphism

```
class A{

        static void calc(double x){
                System.out.println("Square of the given value: " +(x*x) );
        }
}
class B extends A{

        static void calc(double x){
                double y=5;
                System.out.println("Area of the rectangle: " +(x*y) );
        }
}
public class DynamicPoly{
        public static void main(String args[ ]){
                A a=new B( );
                a.calc(5);
                B b=new B( );
                b.calc(6);
        }
}
```

References

- **Reference for this topic**

1. **Book:** Java: The Complete Reference, Ninth Edition: Herbert Schildt

2. **Web:** https://www.tutorialspoint.com/java/index.htm

3. **Web:** https://www.javatpoint.com/java-tutorial