Java OOPS Concept

Learn Automation Testing from Scratch

Trainer – Haradhan Pal

Agenda

- Inheritance
- Method Overloading
- Method Overriding
- super keyword
- Abstraction/Abstract Class
- Interface
- static keyword
- final keyword

Inheritance in Java

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPS (Object Oriented Programming System).
- Using Inheritance we can create Classes that are built in upon existing classes.
- When we Inherit from an existing class, then we can reuse Methods and fields from Parent class and we can add new methods and fields.
- The class where the class members are getting Inherited is called as Super class/Parent class/Base class.
- The class to which the class members are getting Inherited is called as Sub class/Child class/Derived class.
- The Inheritance between Super class and Sub class is achieved using "extends" keyword.

Example:

1) Single Inheritance

Example:

```
public Class ClassB extends Class A {
}
```

2) Multi level Inheritance

Example:

```
public Class ClassB extends ClassA {
public Class ClassC extends ClassB {
}
}
```

Compile Time Polymorphism in Java

- Polymorphism in Java is a concept by which we can perform a single action in different ways.
- Polymorphism derived from two Greek words, Poly-means Many and Morphs - means ways; So polymorphism means many ways.
- There are two types of Polymorphism is available in Java.

a) Compile Time Polymorphism (**Method Overloading)**

b) Run-time Polymorphism (**Method Overriding)**

- Method Overloading: Two are more methods having same name in the same class but they differ in following ways.

a) Number of Arguments

b) Type of Arguments

Example for Method Overloading:

```
public class MethodOverLoading {
public void add(int a, int b){
System.out.println(a+b);
}
public void add(int a, int b, int c){
System.out.println(a+b+c);
}
public void add(double a, double b){
System.out.println(a+b);
}
public static void main(String[] args) {
MethodOverLoading obj = new MethodOverLoading();
obj.add(100, 200);
obj.add(15, 25, 35);
obj.add(101.234, 23.456);
}
```

Run-time Polymorphism in Java

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
- When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.
- Constructors cannot be overridden.
- A method declared final cannot be overridden.
- Static method cannot be overridden. It is because the static method is bound with class whereas instance method is bound with an object.
- Lets take a simple example to understand this. We have two classes: A child class Girl and a parent class Human. The Girl class extends Human class. Both the classes have a common method void eat(). Girl class is giving its own implementation to the eat() method or in other words it is overriding the eat() method.

Example:

```
class Human{
  //Overridden method
  public void eat()
  {
    System.out.println("Human is eating");
  }}
class Girl extends Human{
  //Overriding method
  public void eat(){
    System.out.println("Girl is eating");
  }
  public static void main( String args[]) {
    Girl obj = new Girl();
    //This will call the child class version of eat()
    obj.eat();
  }
} //Output: Girl is eating
```

Super keyword

- The super keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
- Super keyword can be used at variable, method and constructor level.
- Private methods of the super-class cannot be called. Only public and protected methods can be called by the super keyword.

Example:

```
class Animal{
String Name="Elephant";
}
class Dog extends Animal{
String Name="Bullet";
void printName(){
System.out.println(Name);//prints Name of Dog class
System.out.println(super.Name);//prints Name of Animal class
}}
class TestSuper{
public static void main(String args[]){
Dog d=new Dog();
d.printName();
}}
```

Abstraction in Java

- Abstraction is a process of hiding implementation details and showing only functionality to the user.
- In another way it shows important things to the user and hides internal details, for example, sending SMS where you type the text and send the message. One don't know the internal processing about the message delivery.
- Abstraction focuses on what the Object does instead of how it does.
- A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).
- From Class (Concrete class or Abstract Class) to Class we use "extends" keyword

Example:

```
abstract class Animal{
abstract void run();
}
class Tiger extends Animal{
void run()
{System.out.println("Tiger is running");
}
public static void main(String args[]){
Tiger obj = new Animal();
obj.run();
} }
```

Interface in Java

- Interface is a Java type definition block which is 100% abstract
- All the Interface methods by default public and abstract.
- static and final modifiers are not allowed for interface methods.
- In Interfaces variables are public, static, and final by default.
- Interface can not be instantiated, and it does not contain any constructors.
- Interface can be used using "implements" keyword for a Class.
- A class that implements an interface must implement all the methods declared in the interface.
- From Interface to Interface, we use "extends" keyword

Example:

```
public interface Bike{
public void engine();
public void wheels();
public void seat();
}
```

Static keyword

- The static keyword in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class.
- When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.
- It is possible to create a class members (variables and methods) that can be accessed and invoked without creating an instance of the class. In order to create such a member, we have to use static keyword while declaring a class member.

Example:

```
public class TestStaticVariable {
static int m = 10; // static variable
int n = 5; // non static variable
    public static void main(String[] args) {
        TestStaticVariable tsv = new TestStaticVariable();
        System.out.println(tsv.n);
        System.out.println(m);
    }
}
```

Final keyword

- final keyword is used in different contexts. First of all, final is a non-access modifier applicable only to a variable, a method or a class.
- Following are different contexts where final is used.
  - Final Variables -> To create constant variables
  - Final Methods -> Prevent Method Overriding
  - Final Classes -> Prevent Inheritance
- The only difference between a normal variable and a final variable is that we can re-assign value to a normal variable, but we cannot change the value of a final variable once assigned. Hence final variables must be used only for the values that we want to remain constant throughout the execution of program.
- A final method cannot be overridden by any subclasses. The final modifier prevents a method from being modified in a subclass. The main intention of making a method final would be that the content of the method should not be changed by any outsider.
- The main purpose of using a class being declared as final is to prevent the class from being subclassed. If a class is marked as final, then no class can inherit any feature from the final class.

Example:

```
class Test
{
    public static void main(String args[])
    {
        final int i; // local final variable
        i = 20;
//i=50; It will throw error
        System.out.println(i);
    }
}
```