Abstraction in Java

Abstraction is one of the four fundamental principles of object-oriented programming (OOP) and plays a crucial role in the Java programming language. Abstraction is the process of simplifying complex reality by modelling classes based on real-world objects and their behaviours while hiding the unnecessary details.

In Java, abstraction is achieved through the following mechanisms:

1. Abstract Classes: An abstract class is a class that cannot be instantiated and is meant to serve as a blueprint for other classes. It can contain abstract methods (methods without a body) that must be implemented by concrete (non-abstract) subclasses. Abstract classes are defined using the abstract keyword.

Example of an abstract class –

```
1   package p2;
2
3   public class task5 {
4       public static void main(String[] args){
5           searchAll a= new searchAll();
6           searchImage i = new searchImage();
7           searchVideo v = new searchVideo();
8           a.search();
9           a.message();
10          i.search();
11          i.message();
12          }
13      }
```

```java
package p2;
abstract class google {
    2 usages  3 implementations
    abstract void search();
    2 usages
    void message() {
        System.out.println("Thank you for using google");
    }
}
2 usages
class searchAll extends google {
    2 usages
    void search() {
        System.out.println("All search results");
    }
}
2 usages
class searchImage extends google {
    2 usages
    void search() {
        System.out.println("image search results");
    }
}
2 usages
class searchVideo extends google {
    2 usages
    void search() {
        System.out.println("video search results");
    }
}

```

Output –

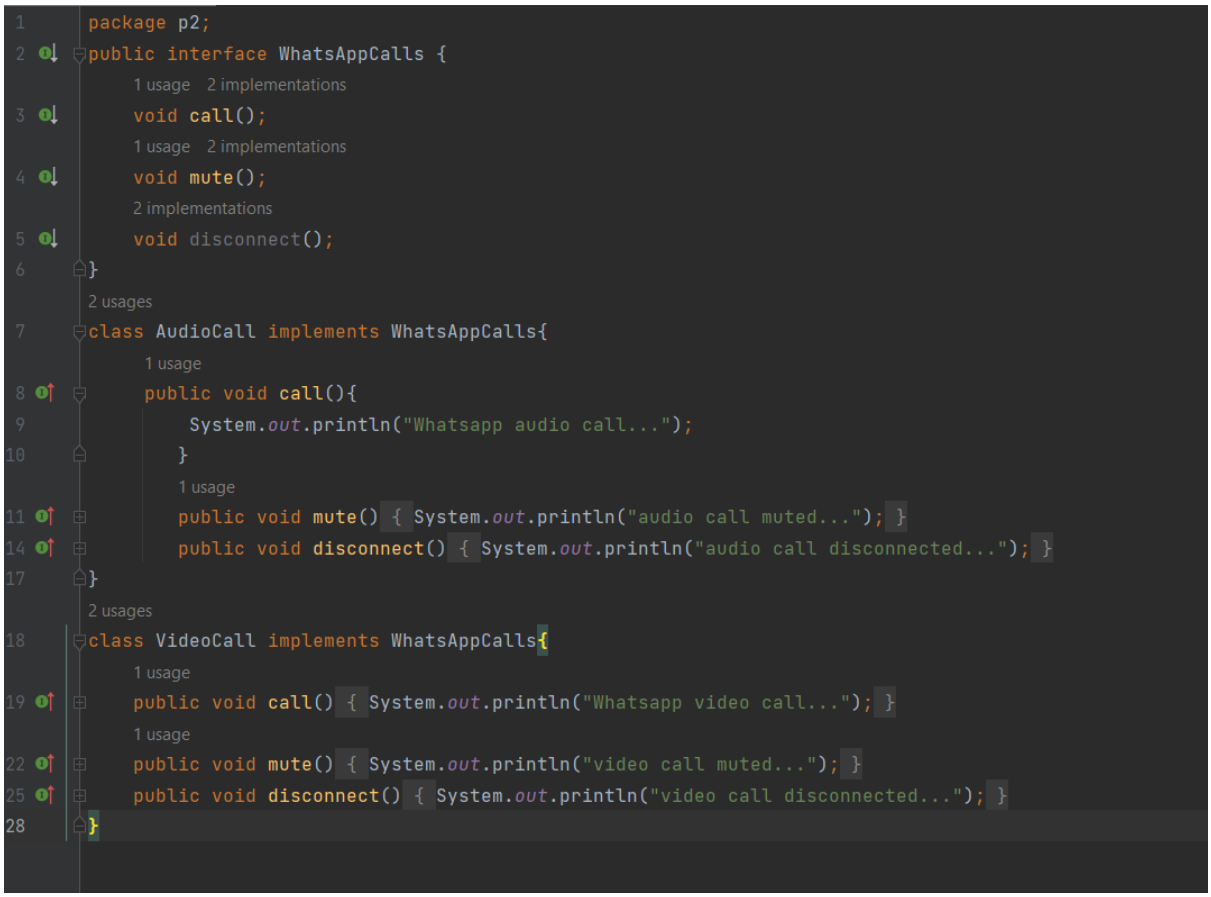
```
All search results
Thank you for using google
image search results
Thank you for using google

Process finished with exit code 0
```

In the above example, an abstract class on Google was created using an abstract keyword. An abstract method search is declared using an abstract keyword.

The abstract method does not contain any body. An abstract class cannot be instantiated, so we create a child class to call the methods in the abstract class.

2. Interfaces: An interface is a pure abstraction mechanism in Java. It defines a contract for classes that implement it. All methods in an interface are implicitly abstract and public, and implementing classes must provide concrete implementations for all interface methods.

Example of an interface –



```
package p2;
public interface WhatsAppCalls {
    1 usage  2 implementations
    void call();
    1 usage  2 implementations
    void mute();
    2 implementations
    void disconnect();
}
2 usages
class AudioCall implements WhatsAppCalls{
     1 usage
     public void call(){
         System.out.println("Whatsapp audio call...");
        }
        1 usage
        public void mute() { System.out.println("audio call muted..."); }
        public void disconnect() { System.out.println("audio call disconnected..."); }
}
2 usages
class VideoCall implements WhatsAppCalls{
    1 usage
    public void call() { System.out.println("Whatsapp video call..."); }
    1 usage
    public void mute() { System.out.println("video call muted..."); }
    public void disconnect() { System.out.println("video call disconnected..."); }
}
```

```java
package p2;

public class tsk5 {
    public static void main(String[] args){
        AudioCall a = new AudioCall();
        VideoCall v = new VideoCall();

        a.call();
        a.mute();
    }
}

```

Output –

```
Whatsapp audio call...
audio call muted...
```

In the above example, abstraction is achieved through interfaces. Here, all the methods defined in the interface class will be abstract; there is no need to specify an abstract keyword. Interfaces cannot be instantiated. So we create a subclass to extract methods from the superclass. All the methods mentioned in the superclass should be present in the child class.

Encapsulation in Java

Binding of data(Attributes and Methods) in a class in to a single unit is called Encapsulation. For example methods and variables can be encapsulated in class or object. To achieve this encapsulation variables must be declared as private so it can only be accessed by the same class. We can view or modify variables by public

Getter(read-only) and Setter(write-only) methods. This is called data hiding.

Advantage of Encapsulation –

- Increase data security.
- Increase readability and flexibility.
- Reusability.
-

Example –

```java
package p2;
public class employee {
    2 usages
    private String name;
    2 usages
    private int salary;
    1 usage
    public String getName() {
        return name;
    }
    1 usage
    public void setName(String name) {
        this.name = name;
    }
    1 usage
    public int getSalary() {
        return salary;
    }
    1 usage
    public void setSalary(int salary) {
        this.salary = salary;
    }
}
```

```java
package p2;

public class tk5 {
    public static void main(String[] args){
        employee e = new employee();
        e.setName("Rohit");
        e.setSalary(100000);
        System.out.println("Name - "+ e.getName());
        System.out.println("Salary - "+ e.getSalary());
    }
}
```

Output –

```
Name - Rohit
Salary - 100000
```

The above program shows encapsulation. Here getter and setter are used to view and modify the variables.

Inheritance in Java

Inheritance is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Java Inheritance Types

Below are the different types of inheritance which are supported by Java.

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

1. Single Inheritance

In single inheritance, subclasses inherit the features of one superclass.

2. Multilevel Inheritance

In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes.

3. Hierarchical Inheritance

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass.

Example –

```java
package p2;
class Animal{
    1 usage
    void eat(){System.out.println("eating...");}
}
1 usage  1 inheritor
class Dog extends Animal{
    1 usage
    void bark(){System.out.println("barking...");}
}
2 usages
class BabyDog extends Dog{
    1 usage
    void weep(){System.out.println("weeping...");}
}
class t5{
    public static void main(String args[]){
        BabyDog d=new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }}
```

The above example shows multilevel inheritance, in which class Animal is the parent class, class Dog is a child class of Animal, and class BabyDog is a child class of Dog.