How to Achieve Abstraction in Java?

Object-oriented programming has four pillars that are an abstraction, **polymorphism, encapsulation,** and **inheritance**. In this section, we will discuss one of them, **abstraction**. Along with this, we can also learn **how to achieve abstraction in Java.**

Abstraction

Abstraction is a feature of OOPs. The feature allows us to **hide** the implementation detail from the user and shows only the functionality of the programming to the user. Because the user is not interested to know the implementation. It is also safe from the security point of view.

Let's understand the abstraction with the help of a real-world example. The best example of abstraction is a car. When we derive a car, we do not know **how is the car moving** or **how internal components are working?** But we know **how to derive a car**. It means it is not necessary to know how the car is working, but it is important how to derive a car. The same is an abstraction.

The same principle (as we have explained in the above example) also applied in Java programming and any OOPs. In the language of programming, the code implementation is hidden from the user and only the necessary functionality is shown or provided to the user. We can achieve abstraction in two ways:

- Using Abstract Class
- Using Interface

Using Abstract Class

Abstract classes are the same as normal Java classes the difference is only that an abstract class uses abstract keyword while the normal Java class does not use. We use the abstract keyword before the class name to declare the class as abstract.


Note: Using an abstract class, we can achieve 0-100% abstraction.

Remember that, we cannot instantiate (create an object) an abstract class. An abstract class contains abstract methods as well as concrete methods. If we want to use an abstract class, we have to inherit it from the base class.

If the class does not have the implementation of all the methods of the interface, we should declare the class as abstract. It provides complete abstraction. It means that fields are public static and final by default and methods are empty.

The syntax of abstract class is:

```
public abstract class ClassName
{
public abstract methodName();
}
```

It is used to define generic types of behavior at the top of an OOPs class hierarchy and use its subclasses to provide implementation details of the abstract class.

Let's see an example of an abstract class.

MainClass.java

```
abstract class Demo
{
//abstract method
abstract void display();
}
//extends the abstract class
public class MainClass extends Demo
{
//defining the body of the method of the abstract class
void display()
{
System.out.println("Abstract method called.");
}
public static void main(String[] args)
{
MainClass obj = new MainClass ();
//invoking abstract method
obj.display();
}
}
```

Output:

```
Abstract method called.
```

Let's see another example of the abstract class in which we have used a non-abstract method.

Owner.java

```
abstract class Car
{
//abstract method
abstract void start();
//non-abstract method
public void stop()
{
System.out.println("The car engine is not started.");
}
}
//inherit abstract class
public class Owner extends Car
{
//defining the body of the abstract method of the abstract class
void start()
{
System.out.println("The car engine has been started.");
}
public static void main(String[] args)
{
Owner obj = new Owner();
//calling abstract method
obj.start();
```

```
obj.stop();
}
}
```

Output:

```
The car engine has been started.
The car engine is not started.
```

Using Interface

In Java, an interface is similar to Java classes. The difference is only that an interface contains empty methods (methods that do not have method implementation) and variables. In other words, it is a collection of abstract methods (the method that does not have a method body) and static constants. The important point about an interface is that each method is **public** and **abstract** and does not contain any constructor. Along with the abstraction, it also helps to achieve multiple inheritance. The implementation of these methods provided by the clients when they implement the interface.


Note: Using interface, we can achieve 100% abstraction.

Separating interface from implementation is one way to achieve abstraction. The **Collection framework** is an excellent example of it.

Features of Interface:

- We can achieve total abstraction.
- We can use multiple interfaces in a class that leads to multiple inheritance.
- It also helps to achieve loose coupling.

Syntax:

```
public interface XYZ
{
public void method();
}
```

To use an interface in a class, Java provides a keyword called **implements**. We provide the necessary implementation of the method that we have declared in the interface.

Let's see an example of an interface.

Car.java

```
interface CarStart
{
void start();
}
interface CarStop
{
void stop();
}
public class Car implements CarStart, CarStop
```

```
public void start()
{
System.out.println("The car engine has been started.");
}
public void stop()
{
System.out.println("The car engine has been stopped.");
}
public static void main(String args[])
{
Car c = new Car();
c.start();
c.stop();
}
}
```

Output:

```
The car engine has been started.
The car engine has been stopped.
```

← Prev Next →