Interface in Java

Dr. Partha Pratim Sarangi

Interface

- Like Abstract class, Java provides another concept named Interface.
- Unlike abstract class, Interface contains static constants and abstract methods.
- The interface in Java is a mechanism to achieve pure abstraction and multiple inheritance.
- Java interface also represents the IS-A relationship.
- It cannot be instantiated just like the abstract class.
- Interfaces are implemented rather than being extended in the inheritance.

Contd...

- Initially, interface contains only abstract methods and static constants both are public by default.
- In Java 8, two more methods are included such as default and static methods in an interface.
- In java 9 onwards, we can define private methods in an interface.
- Writing an interface is similar to writing an abstract class that uses interface keyword before name of the interface. In abstract class contains both abstract and non-abstract (concrete) methods. Interface contains abstract, default and private methods from which default and private methods are defined inside the interface.

An interface is different from an abstract class

- Interfaces promote loose coupling between the classes.
- An interface does not contain any constructors.
- By default all of the methods in an interface are abstract.
- The only attributes that can appear in an interface must be declared both static and final by default.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces as well as a class can implement multiple interfaces.

Declaration of Interface

Both are used to achieve abstraction and can never be instantiated

Abstract class declaration

```
abstract class AbstractDemo
{
    abstract void show( ); //Abstract method

    void print( )  // Concrete method
    {




    }
}
```

Interface declaration

```
Interface InterfaceDemo extends I1, I2
{
    Attributes; // public static final by default
    void show( );  //Abstract method by default
    default void print( ) //Concrete method
    {

    }
    static void display( )  //Static method
    {

    }
}
```

Properties of Interface

- An interface is implicitly abstract. It is not necessary to use the keyword abstract before interface declaration.
- Similarly, all methods inside the body of the interface are implicitly abstract.
- All attributes or fields are public static and final.
- Methods in an interface are implicitly public.
- An interface can extend multiple interfaces.
- Similarly, a class can extend one base class but implement multiple interfaces.
- Using interface, Java can support multiple inheritance.

Implementing an interface

- When a class implements an interface, we can think of the class as signing a contract, agreeing to define all abstract methods present in the interface, otherwise compilation error will be generated.
- This compilation error can be eliminated, by using an abstract class that implements the interface as a replacement for the normal class.
- Furthermore, an abstract class cannot be instantiated. Hence, we have to use another subclass to extends the abstract class and defining the remaining methods which have not defined in the abstract class.

Dynamic method lookup

- In Java, dynamic method lookup is achieved using interfaces.
- When a method is called on an object that implements an interface, the Java runtime looks up the implementation of that method at runtime. For example:

```
interface Animal {
        void makeSound();
}
```

```
class Dog implements Animal {
    public void makeSound() {
        System.out.println("Woof!");
    }
}
```

```
class Cat implements Animal {
        public void makeSound() {
                System.out.println("Meow!");
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        Animal myCat = new Cat();
        myDog.makeSound(); // Output: "Woof!"
        myCat.makeSound(); // Output: "Meow!"
    }
}
```

Advantages of an Interface in Java

- Use interface to achieve 100% abstraction
- Interface supports the functionality of multiple inheritances in Java.
- We can use interface for gaining loose coupling that promotes to reduce the inter-dependence between unrelated classes.
- Loose coupling between the classes can make our code more modular and easier to maintain.

➢Loose coupling means the classes in a system are not tightly dependent on each other and can be modified or replaced without affecting the rest of the system.

When to use an abstract class

- Abstract classes are used when we want to define a base class that provides some common functionality and properties to its related subclasses.
- Abstract classes can have constructors, instance variables, and concrete methods (i.e., methods with implementation), as well as abstract methods (i.e., methods without implementation).
- When we want to define a common contract for a group of related classes.

When to use an interface

- An interface can be used to define a contract behavior and it can also act as a contract between two software components to interact,
- Interfaces are best suited for providing a common functionality to unrelated classes.
- Abstract classes are mainly used to share common behavior with related subclasses, it means that all child classes should have to perform the same functionality.
- The use of an interface separates system design from implementation details to achieve 100% abstraction.
- As we cannot inherit from multiple abstract classes, it is possible to implement more than one interface in order to achieve multiple inheritance.
- If we are designing specific and concise functionality, use interfaces. If we are designing large functional units, use an abstract class.

How to create an object of an interface

As interface is implicitly abstract in nature, so it is not possible to construct an object of an interface. If the programmer wants to construct an object of an interface then the programmer is bound to make an interface as an anonymous inner class.

```
interface In
{
   void show();
}
public class Inter
{
  public static void main(String[] args)
  {
```

```
In ii = new In()
{
    public void show()
    {
       System.out.println("Java Interface");
     }
};
    ii.show();
    }
}
```