7. INTERFACES

1. Definition:

An interface in Java is a blueprint of a class. It has static constants and abstract methods only. The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and indirectly multiple inheritance in Java.
Java Interface also **represents IS-A relationship**.

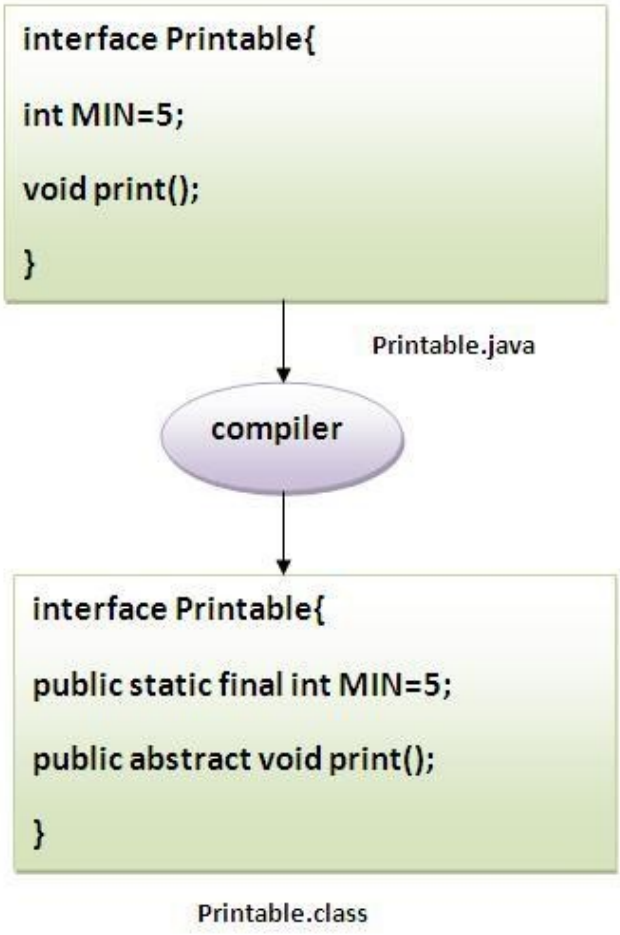
2. Use of interfaces:

There are mainly two reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.

3. Structure of interface:

```
interface Printable
	{
		int MIN=5;
		void print();
	}
```



Interface variables are by default **public static and final**. We have to initialize those in interface only. Interface methods are by default **public and abstract**. Those all methods should be

overridden in class implementing interface either or that class should be declared as as abstract.

Simple example of Java interface:
In this example, Printable interface have only one method, its implementation is provided in the PrintDemo class.

```
interface printable
{
        void print();
}

public class PrintDemo implements printable{

        public void print() {
                System.out.println("Hello");
        }

        public static void main(String args[]) {
                PrintDemo  obj=new PrintDemo();
                obj.print();
        }
}
```

Output: Hello

4. Properties of Interface:

- A class implements the interface and overrides the method of interface.
- A class can extend a class and also can implement interface.
- An interface can not implement an interface.
- A interface can extends two or more interfaces.
- A interface cannot extend a class.
- A class cannot extend multiple class.
- A class implements interface but one interface can extend another interface .
- A class must either implemet all methods of its interface to be implement or it should be declared abstract.
- Interface variables are by default public static final
- Interface methods are by default public and abstract. We can not use any other modifier.
- While overridding methods of interface in class, methods should be defined as public.

5. Extended interfaces:

Interface can be extended by another interface. But the class implemening child interface must implement all the methods of both parent or child interface or it should be declared as abstract class.

Example:

```
interface Printable {
        void print();
    }
interface Showable extends Printable {
        void show();
    }
public class ExtendedInterfaceDemo implements Showable
{
    // this class must override methods of both sub interface and super interface

    public void print() {
        System.out.println("Hello");
    }
    public void show() {
        System.out.println("Welcome");
    }
    public static void main(String args[]) {
        ExtendedInterfaceDemo obj=new ExtendedInterfaceDemo();
        obj.print();
        obj.show();
    }
}
```

Output :

Hello
Welcome.

6. Nested interfaces:

An interface which is defined inside another interface or another class is know as Nested Interface. It will get protected from outside world. It will become member of outer interface or class.

It can be declared private, default, protected and public. But it will be accessed via outer interface name.

Example:

```
    interface Printable {
        interface Showable  {
            void show();
            void print();
        }
    }
```

```
public class NestedInterfaceDemo implements Printable.Showable {
        // override methods Showable interface
}
```

7. Functional Interfaces:

The interfaces with only one method signature are known as functional interfaces.
For e.g. Runnable interface- it has a method signature **void run();**

8. Marker Interfaces:

There are some interfaces in java which has no body. Means no any method signatures or variables. Such interfaces are empty and known as marker interfaces. These interfaces only will mark the class as special class which is implementing marker interface. The classes implementing marker interfaces treated as special classes that will behave different than other classes. These will get some extra priviledges over other classes.

Example: Cloneable interface, Serializable interface

Cloneable Interface:
If a class implements Cloneable interface we can create clone (copy) of the object using clone method. Java clone method is of Object class.
Its syntax is:

public Object clone() throws CloneNotSupportedException.

This exception is checked exception u have to check it at compile time using either try catch or throws.

Serializable Interface:
If a class implements Serializable interface then we can write the state of an object into file in terms of byte and transfer it over internet securely.
For this writing state of an object, writeObject method is used. For reading the state from file, readObject method is used. These methods are of ObjectOutputStream and ObjectInputStream class repectively.
The Syntax is:
public final Object readObject()
public final void writeObjet(objectofserializedclass)

These methods throw following exceptions:
ClassNotFoundException -- Class of a serialized object cannot be found.
InvalidClassException -- Something is wrong with a class used by serialization.
StreamCorruptedException -- Control information in the stream is inconsistent.

OptionalDataException -- Primitive data was found in the stream instead of objects.
IOException -- Any of the usual Input/Output related exceptions.

9. Difference between Abstract classes and Interfaces:

Sr. No.	Abstract class	Interface
1.	May contain concrete methods	Never contain Concrete methods
2.	May be partially implemented	Full absract
3.	Can have instance variables and constructors	Never contain instance variables and construcotrs
4.	Can have final, static and static final variables	Only contain public static final variables
5.	We can not create instance of abstract class directly but it will get created through subclass	Interface instance never get created
6.	Class absract methods explicitly declared as absract	Interface methods are by default abstract
7.	Here Absract methods can be declared as protected, default and public.	Here abstract methods are bydefault public so others access modifiers can not be used
8.	Class extends abstract class	Class implements interface
9.	A class can extend only one absract class	A class can implement more than one interfaces
10.	Variables must be explicitly declared public, final or static if needed.	Interface variables are by default public static final.

10. New features added in JDK 1.8:

From JDK 1.8, you may have default and static methods in interface

Default methods: If a class implements any interface and later if interface get modified by adding the method then in this case class has to implement the newly added method and for that we need to change the existing code of class.
Solution : u can declare default method in interface.
Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces.

You specify that a method definition in an interface is a default method with the default keyword at the beginning of the method signature. All method declarations in an interface, including default methods, are implicitly public, so you can omit the public modifier.

7. INTERFACES

Static methods: You can define static methods in interfaces. You can keep static methods specific to an interface in the same interface rather than in a separate class.
These are also by default public.

Example:

```
public interface TimeClient {
   // ...
   static  ZoneId getZoneId (String zoneString) {
      try {
         return ZoneId.of(zoneString);
      } catch (DateTimeException e) {
         System.err.println("Invalid time zone: " + zoneString +
            "; using default time zone instead.");
         return ZoneId.systemDefault();
      }
   }

   default  ZonedDateTime getZonedDateTime(String zoneString) {
      return ZonedDateTime.of(getLocalDateTime(), getZoneId(zoneString));
   }
}
```

Thus whenever you need full abstraction, go for interfaces. Through the full abstraction you can show your overview of functionality to outside world but your implementation will get hidden.

ASSIGNMENTS

1. Define the interface which shows the abstract functionality of Car. Implement this interface in Car. Define the final shared veriable in interface.
2. Show the use of Extended interface, Nested interface, Functional interface and Marker interface in above example.