➢ Interface in Java

- ❖ An interface in Java is a blueprint of a class. It has static constants and abstract methods.
- ❖ The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
- ❖ In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.
- ❖ Java Interface also represents the IS-A relationship.
- ❖ It cannot be instantiated just like the abstract class.
- ❖ Since Java 8, we can have default and static methods in an interface.
- ❖ Since Java 9, we can have private methods in an interface.

☞ Why use Java interface?

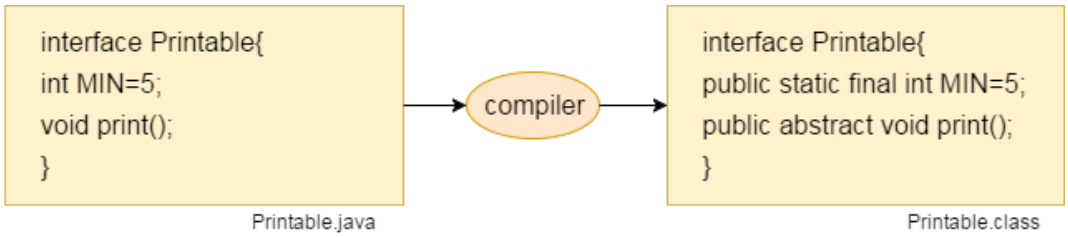
- ❖ There are mainly three reasons to use interface. They are given below.
  - It is used to achieve abstraction.
  - By interface, we can support the functionality of multiple inheritance.
  - It can be used to achieve loose coupling. (Classes are Independent from each other)

☞ How to declare an interface?

- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

```
interface <interface_name>
{
   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

- The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.
- In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



☞ Abstract Methods:

- ❖ Interfaces can declare abstract methods without providing an implementation.
- ❖ All methods declared in an interface are implicitly public and abstract.

Example:1

```
interface In1
{
    void m1()
    {

    }
}
```

//In1.java:4: error: interface abstract methods cannot have body

Note: In interface the methods declared are by default public abstract and interface abstract methods cannot have body.

```
interface In1
{
    void m1();
}
```

//code will compile happily

Example:2

```
interface In1
{
    void m1();
}
class Test implements In1
{

}
```

// In1.java:5: error: Test is not abstract and does not override abstract method m1() in In1

Note: In interface if the child class is not abstract then you must have to override all the abstract methods of interface.

```
interface In1
{
    void m1();
}
class Test implements In1
{
    void m1()
    {

    }
}
/*In1.java:7: error: m1() in Test cannot implement m1() in In1
void m1()
^
attempting to assign weaker access privileges; was public*/
```

Note: In interface the methods declared are by default public abstract so we must have to use public access modifier in the overriding method

```
interface In1
{
    void m1();
}
class Test implements In1
{
    public void m1()
    {

    }
}

//code will compile happily
```

☞ Object Creation in Interfaces:

- ❖ Interfaces cannot be instantiated directly with the new keyword.
- ❖ Objects of interfaces are created indirectly through the instantiation of classes that implement those interfaces.

Example: 3

```
interface Vehicle
{
   void ignition();
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
    }
}
class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Vehicle();
    }
}

/*In3.java:16: error: Vehicle is abstract; cannot be instantiated
        Vehicle v=new Vehicle(); */
```

Note: No object can be created for interface but we can use the reference of interface

```
interface Vehicle
{
   void ignition();
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
    }
}
class Main
{
    public static void main(String[] args)
    {
      //reference of interface object of implemented class
```

```
        Vehicle v=new Twowheeler();
        v.ignition();
    }
}

/* Output:
   kick Start */
```

☞ Creation of Constructors, Instant Block and Static Block in interface:

- ❖ Interfaces in Java cannot have constructors.
- ❖ Instance initializer blocks (non-static blocks) are not allowed in interfaces.
- ❖ Static initializer blocks are not allowed in interfaces as well.

Example: 4

```
interface Vehicle
{
    Vehicle()
    {
       //constructor
    }
   void ignition();
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
    }
}
class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Twowheeler();
        v.ignition();
    }
}
/*In3.java:3: error: <identifier> expected
    Vehicle() */
```

Note: Cannot create constructors, Static Block and Instance Block in interface

☞ Interface fields

- ❖ Interfaces can contain fields (public, static, final fields), which are implicitly public, static, and final.
- ❖ Example: public static final int MY_CONSTANT = 10;

Example: 5

```
interface Vehicle
{
    int engineCapacity;
    void ignition();
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
    }
}
class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Twowheeler();
        v.ignition();
    }
}
```

Note: Interface fields are public, static and final by default so it must be initialized

```
interface Vehicle
{
    int engineCapacity=100;
    void ignition();
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
        System.out.println("engine capacity= "+ engineCapacity+ "
cc");
    }
}
```

```
class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Twowheeler();
        v.ignition();
    }
}
/* Output
kick Start
engine capacity= 100 cc */
```

☞ Default Methods in interface:

- ❖ Java 8 introduced default methods in interfaces, allowing the addition of new methods without breaking existing implementations.
- ❖ Default methods are declared using the default keyword.
- ❖ Default methods are allowed only in interfaces

Example: 6

```
interface Vehicle
{
    int engineCapacity=100;
    void ignition();
    default void myDefaultmethod()
    {
        System.out.println("Default method of interface");
    }
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
        System.out.println("engine capacity= "+ engineCapacity+ "
cc");
    }
}

class Main
{
    public static void main(String[] args)
    {
```

```
        Vehicle v=new Twowheeler();
        v.ignition();
        v.myDefaultmethod();
    }
}
/* Output
    kick Start
    engine capacity= 100 cc
    Default method of interface
*/
```

Example: 7

```
interface Vehicle
{
    int engineCapacity=100;
    void ignition();
    default void myDefaultmethod()
    {
        System.out.println("Default method of interface");
    }
}
class Twowheeler implements Vehicle
{
    public default void myDefaultmethod()
    {
        System.out.println("Default method of interface in
child");
    }
    public void ignition()
    {
        System.out.println("kick Start");
        System.out.println("engine capacity= "+ engineCapacity+ "
cc");
    }
}

class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Twowheeler();
        v.ignition();
        v.myDefaultmethod();
    }
```

```
}
/*PS E:\SEM-II\JAVA-2 Notes\Interface> javac in1.java
in1.java:12: error: modifier default not allowed here
    public default void myDefaultmethod()*/
```

Example: 7

```
interface Vehicle
{
    int engineCapacity=100;
    void ignition();
    default void myDefaultmethod()
    {
        System.out.println("Default method of interface");
    }
}
class Twowheeler implements Vehicle
{
    public  void myDefaultmethod()
    {
        System.out.println("Default method of interface in
child");
    }
    public void ignition()
    {
        System.out.println("kick Start");
        System.out.println("engine capacity= "+ engineCapacity+ "
cc");
    }
}

class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Twowheeler();
        v.ignition();
        v.myDefaultmethod();
    }
}
/*kick Start
engine capacity= 100 cc
Default method of interface in child  */
```

☞ Static Methods in interface:

- ❖ Java 8 also introduced static methods in interfaces.
- ❖ Static methods are declared using the static keyword. But we cannot override it.
- ❖ Can call the static methods of interface by the name of interface.

Example: 8

```
interface Vehicle
{
    int engineCapacity=100;
    void ignition();
    static void myStaticmethod()
    {
        System.out.println("Static method of interface");
    }
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
        System.out.println("engine capacity= "+ engineCapacity+ "
cc");
    }
}

class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Twowheeler();
        v.ignition();
        Vehicle.myStaticmethod();
    }
}
/* Output
    kick Start
    engine capacity= 100 cc
    Static method of interface
*/
```

☞ Private Methods in interface:

- ❖ In Java 9, we can create private methods inside an interface. Interface allows us to declare private methods that help to share common code between non-abstract methods.
- ❖ Before Java 9, creating private methods inside an interface cause a compile time error.

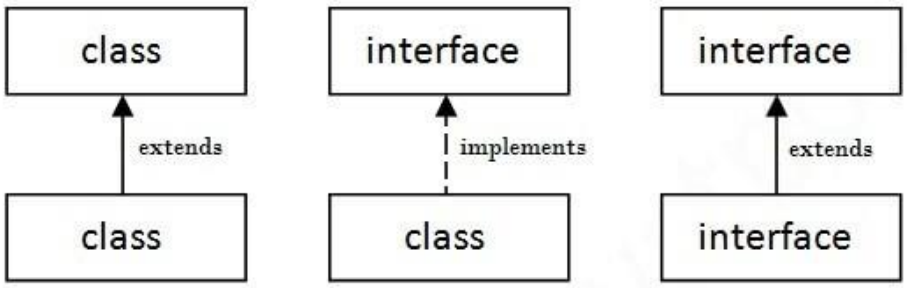
Example: 8

```
interface Vehicle
{
    int engineCapacity=100;
    void ignition();
    default void myDefaultmethod()
    {
        System.out.println("Default method of interface");
        Myprivate();
    }
    private void Myprivate()
    {
        System.out.println("Private method of interface");
    }
}
class Twowheeler implements Vehicle
{
    public void ignition()
    {
        System.out.println("kick Start");
        System.out.println("engine capacity= "+ engineCapacity+ "
cc");
    }
}
class Main
{
    public static void main(String[] args)
    {
        Vehicle v=new Twowheeler();
        v.ignition();
        v.myDefaultmethod();
    }
}
```

/*kick Start
engine capacity= 100 cc
Default method of interface
Private method of interface*/

☞ Extending Interface:

- ❖ As shown in the figure given below, a class extends another class, an interface extends another interface, but a class implements an interface.



\

Example: 8

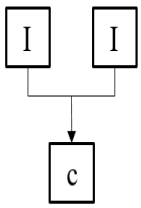
```
interface In1
{
    void display();
}
interface In2 extends In1
{
    void show();
}
class Test implements In2
{
    public void show()
    {

    }
    public void display()
    {

    }
}

class Main
{
    public static void main(String[] args)
    {
```

```
            Test t=new Test();
        }
    }
```

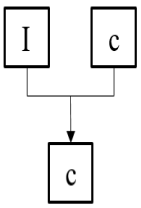
Note: Here interface In2 extends In1 and class Test implements In2. so, class Test is child of In2 and In2 is child of In1, So compulsory we have to implements abstract method display() of In1 and show() of In2 in Test class.

☞ Multiple Inheritance:

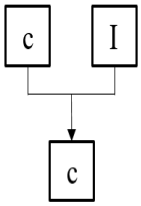
- ❖ Multiple inheritance in java is the capability of creating a single class with multiple superclasses.



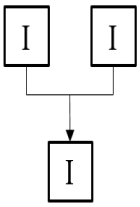

```
interface I1
{
}
interface I2
{
}
class C1 implements I1,I2
{
}
```




```
interface I1
{
}
class C1
{
}
class C2 extends C1 implementsI2
{
}
```




```
class C1
{
}

interface I1
{
}
class C2 extends C1 implementsI2
{
}
```




```
interface I1
{
}
interface I2
{
}
interface I3 extends I1,I2
{
}
```

☞ Default Method Calling:

```
interface In1
{
    default void show()
    {
        System.out.println("In1");
    }
}
interface In2
{
    default void show()
```

```
        {
            System.out.println("In2");
        }
    }
    class Test implements In1,In2
    {
        public void show()
        {
            System.out.println("Hi Test");
        }
        public void showIn1()
        {
            In1.super.show();
        }
        public void showIn2()
        {
            In2.super.show();
        }
    }

    class Main
    {
        public static void main(String[] args)
        {
           Test t=new Test();
           t.show();
           t.showIn1();
           t.showIn2();
           In1 i1=new Test();
           i1.show();
           In2 i2=new Test();
           i2.show();
        }
    }
```

Program:

```
    /*Level:1:-Interface :Full abstraction
      Level:2:-Abstract class: Partial implementation
      Level:3:-Implementation Class:Partial implementation
      Level:3:-Final Code :Main Method
      */

    interface Bank
    {
```

```
    void deposite();
    void withdraw();
    void loan();
    void balanceCheck();
}
abstract class VillageBank implements Bank
{
    public void deposite()
    {
        System.out.println("deposite");
    }
    public void withdraw()
    {
        System.out.println("Withdraw");
    }
}
abstract class CityBank extends VillageBank
{
    public void loan()
    {
        System.out.println("Loan");
    }
}
class FinalBank extends CityBank
{
    public void balanceCheck()
    {
        System.out.println("Balance Check");
    }
}

class Main
{
    public static void main(String[] args)
    {
        FinalBank fb=new FinalBank();
        fb.deposite();
        fb.withdraw();
        fb.loan();
        fb.balanceCheck();

    }
}
```