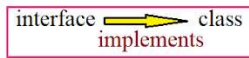


Java Interface:

- 1) Interface is not a class, it is a blue print of a class.
- 2) **Syntax:**

```
<AccessModifier>interface<InterfaceName>
{
}
```
- 3) Every method in interface is by default **public abstract**
- 4) Every variable in interface is by default **public static final**
- 5) From Java 1.8v onwards we can write **default** methods inside an Interface
- 6) From Java 1.8v onwards we can write **static** methods also inside an Interface. [including main()]
- 7) From Java 1.9v onwards we can write **private** methods also inside an Interface
- 8) Inside an interface we cannot write constructors because interface is not a class.
- 9) Just like a abstract class we cannot instantiate an interface, means we cannot create an object for interface
- 10) If we want to **inherit an interface into a class** we need to use the keyword **implements**.



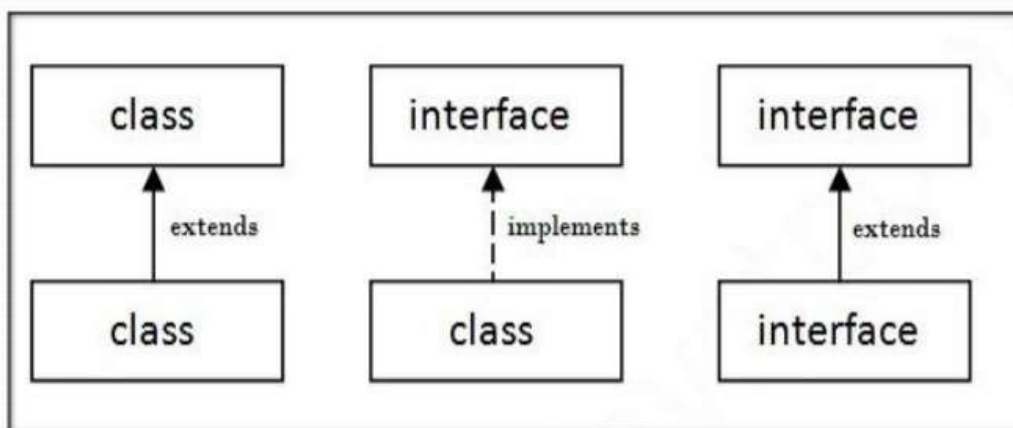
- 11) If we are inheriting an interface into a class 100% in the child class we need to provide implementation (by using Method Overriding) for all the abstract methods which are present in the interface, otherwise we will be getting a compile time error.
- 12) If we don't want to provide implementation for all the abstract methods present in the interface then we need to make our child class also as abstract. **WE CAN ACHIEVE MULTIPLE INHERITANCE IN JAVA BY USING INTERFACES**

Understanding Interfaces

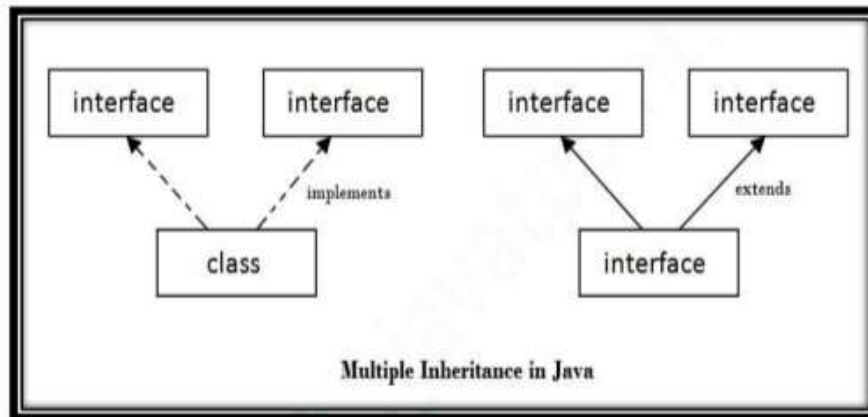
An interface in the Java programming language is an abstract type that is used to specify a behavior that classes must implement.

class	interface
We can instantiate a class, i.e. we can create an object reference for a class	We can't instantiate an interface.
A class is declared using a keyword 'class' class <class name> { }	An interface is declared by using a keyword 'interface'. interface <interface name> { }
The members of a class can have the access modifiers like public, private, protected .	The members of an interface are always public. Up to (1.7v)
Inside a class you can have method implementation.	In interfaces we can't write method body, because all the methods are by default public abstract methods. Up to (1.7v)
Multiple inheritance is not possible	Multiple inheritance is possible
We can have constructor.	We can't have constructors

- An interface is a blueprint of a class.
- Interface is a mechanism to achieve fully abstraction in java.
- There can be only abstract methods in the interface (up-to 1.7v).
- It is used to achieve fully abstraction and multiple inheritance in Java.
- Interface **fields are public, static and final by default**, and **methods are public and abstract**.
- For variables present in the interface we should provide initialization at the time of declaration only.
- **Interfaces should have only abstract methods**. (we can check the internal implementation by using `javap .className`).
- We can declare a class inside the interface.
- We can write main method in interface from **jdk 1.8v**
- If we want to inherit an interface from a class we need to use the keyword '**implements**' not '**extends**'.
- If we want to inherit an interface from another interface we need to use the keyword '**extends**' not '**implements**'.



- We can achieve multiple inheritance in java by using interfaces.



- We can extends multiple classes from a class at a time (T/F)
 - We can implement any number of interfaces from a interface at a time (T/F)
 - We can implement only one interface from a class at a time (T/F)
- When implementing an interface , if a class cant provide the implementation of all the abstracts methods present in that interface then make that class as **abstract class**.

Marker Interface:

- A marker interface in Java is an interface with no fields or methods. To be precise, an empty interface in Java is called a marker interface.
- Examples of marker interfaces are Serializable, Cloneable etc


```

3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default meth2() called");
14     }
15     static void meth3() // Java 1.8v
16     {
17         System.out.println("InterfaceA static meth3() called");
18     }
19     private void meth4() // Java 1.9v
20     {
21         System.out.println("InterfaceA private meth4() called");
22     }
23     public static void main(String[] args) // Java 1.8v
24     {
25         System.out.println("InterfaceA main() called");
26     }
}

```

```

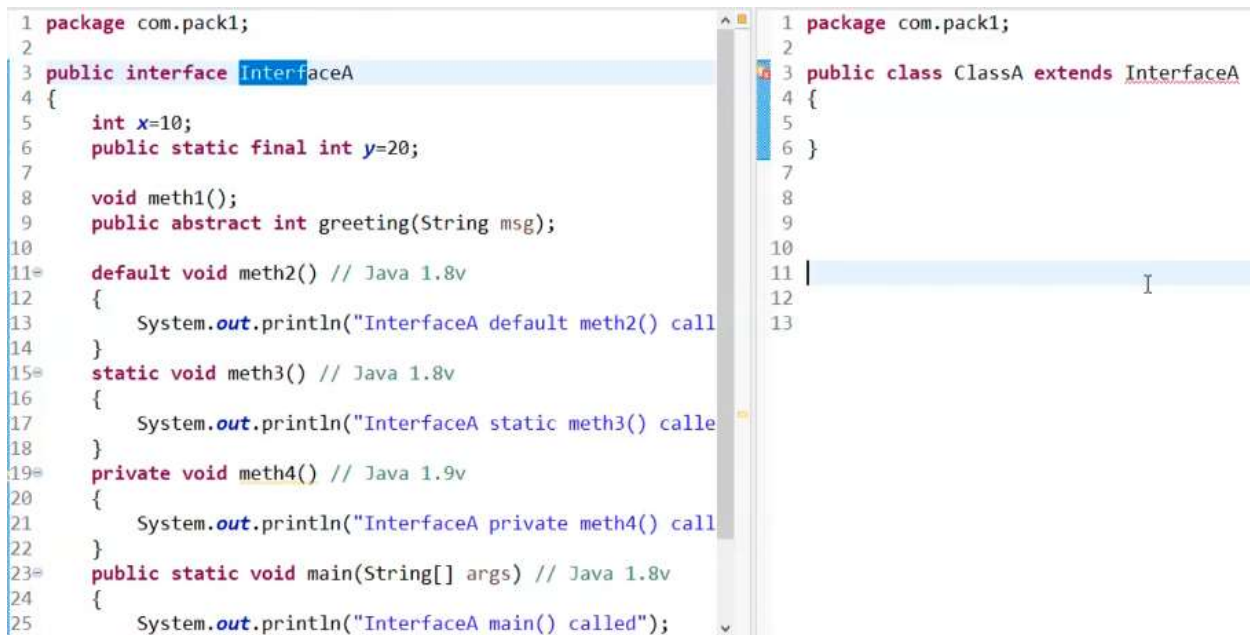
3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default meth2() called");
14     }
15     static void meth3() // Java 1.8v
16     {
17         System.out.println("InterfaceA static meth3() called");
18     }
19     private void meth4() // Java 1.9v
20     {
21         System.out.println("InterfaceA private meth4() called");
22     }
23     public static void main(String[] args) // Java 1.8v
24     {
25         System.out.println("InterfaceA main() called");
26         InterfaceA.meth3();
27     }
}

```

InterfaceA main() called
InterfaceA static meth3() called

We cannot write constructor in an interface, why because constructor name should be same as class name here interface is not a class.

We can write constructors in an abstract class.



```
1 package com.pack1;
2
3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default meth2() call
14     }
15     static void meth3() // Java 1.8v
16     {
17         System.out.println("InterfaceA static meth3() calle
18     }
19     private void meth4() // Java 1.9v
20     {
21         System.out.println("InterfaceA private meth4() call
22     }
23     public static void main(String[] args) // Java 1.8v
24     {
25         System.out.println("InterfaceA main() called");
26     }
27 }
```

```
1 package com.pack1;
2
3 public class ClassA extends InterfaceA
4 {
5
6 }
7
8
9
10
11
12
13
```

We can inherit a class in class by extends, interface into interface by extends, interface into class by implements
But not class into interface, because you cannot create an object for an interface.

```

1 package com.pack1;
2
3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default meth2() call");
14     }
15     static void meth3() // Java 1.8v
16     {
17         System.out.println("InterfaceA static meth3() called");
18     }
19     private void meth4() // Java 1.9v
20     {
21         System.out.println("InterfaceA private meth4() called");
22     }
23     public static void main(String[] args) // Java 1.8v
24     {
25         System.out.println("InterfaceA main() called");
26     }
27 }

```

```

1 package com.pack1;
2
3 public class ClassA implements InterfaceA
4 {
5     // Implementation of InterfaceA methods
6 }

```

Whenever we are inheriting an interface into a class if the interface has any abstract methods 100%, we need to provide implementations for all those abstract methods otherwise we are getting a compile time error.

```

1 package com.pack1;
2
3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default meth2() call");
14     }
15     static void meth3() // Java 1.8v
16     {
17         System.out.println("InterfaceA static meth3() called");
18     }
19     private void meth4() // Java 1.9v
20     {
21         System.out.println("InterfaceA private meth4() called");
22     }
23     public static void main(String[] args) // Java 1.8v
24     {
25         System.out.println("InterfaceA main() called");
26     }
27 }

```

```

1 package com.pack1;
2
3 public abstract class ClassA implements InterfaceA
4 {
5     // Implementation of InterfaceA methods
6 }

```

Not use full because we cannot create an object for abstract class.

```

1 package com.pack1;
2
3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7     void meth1();
8     public abstract int greeting(String msg);
9
10    default void meth2() // Java 1.8v
11    {
12        System.out.println("InterfaceA default");
13    }
14    static void meth3() // Java 1.8v
15    {
16        System.out.println("InterfaceA static");
17    }
18    private void meth4() // Java 1.9v
19    {
20        System.out.println("InterfaceA private");
21    }
22    public static void main(String[] args) //
23    {
24        System.out.println("InterfaceA main");
25    }
26 }

```

```

1 package com.pack1;
2
3 public class ClassA implements InterfaceA
4 {
5     void meth1()
6     {
7         System.out.println("InterfaceA meth1() overridden");
8     }
9 }

```

because of access modifiers

in interface by default access modifiers for method1() is public

in class the access modifiers is default for method1()

child class method access modifier should not be more restricted than parent class access modifiers

private > default > protected > public

parent class

child class

private invalid(inheritance)

public

public

protected

protected & public

default

default, protected & public

```

1 package com.pack1;
2
3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default");
14     }
15     static void meth3() // Java 1.8v
16     {
17         System.out.println("InterfaceA static");
18     }
19     private void meth4() // Java 1.9v
20     {
21         System.out.println("InterfaceA private");
22     }
23     public static void main(String[] args) // Java 1.9v
24     {
25         System.out.println("InterfaceA main()");
26     }
27 }

```

```

1 package com.pack1;
2
3 public class ClassA implements InterfaceA
4 {
5     @Override
6     public void meth1()
7     {
8         System.out.println("InterfaceA meth1() overridden");
9     }
10    @Override
11    public int greeting(String data)
12    {
13        System.out.println("data : "+data);
14        return 100;
15    }
16 }

```

```

1 package com.pack1;
2
3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default");
14     }
15     static void meth3() // Java 1.8v
16     {
17         System.out.println("InterfaceA static");
18     }
19     private void meth4() // Java 1.9v
20     {
21         System.out.println("InterfaceA private");
22     }
23     public static void main(String[] args) // Java 1.9v
24     {
25         System.out.println("InterfaceA main()");
26         InterfaceA aobj = new ClassA();
27         aobj.meth1();
28         System.out.println("====>" + aobj.greeting("Java is awesome!!!"));
29         aobj.meth2();
30         //aobj.meth4(); // C.E because of private access modifier
31     }
32 }

```

```

1 package com.pack1;
2
3 public class ClassA implements InterfaceA
4 {
5     @Override
6     public void meth1()
7     {
8         System.out.println("InterfaceA meth1() overridden");
9     }
10    @Override
11    public int greeting(String data)
12    {
13        System.out.println("data : "+data);
14        return 100;
15    }
16    public static void main(String[] args)
17    {
18        InterfaceA aobj = new ClassA();
19        aobj.meth1();
20        System.out.println("====>" + aobj.greeting("Java is awesome!!!"));
21        aobj.meth2();
22        //aobj.meth4(); // C.E because of private access modifier
23    }
24 }

```

```

<terminated> ClassA [Java Application]
InterfaceA meth1() overridden
data : Java is awesome!!!
====>100
InterfaceA default

```



```

3 public interface InterfaceA
4 {
5     int x=10;
6     public static final int y=20;
7
8     void meth1();
9     public abstract int greeting(String msg);
10
11     default void meth2() // Java 1.8v
12     {
13         System.out.println("InterfaceA default meth2() called");
14         this.meth4();
15     }
16     static void meth3() // Java 1.8v
17     {
18         System.out.println("InterfaceA static meth3() called");
19         //this.meth4();// C.E we cant use 'this' keyword inside a static area
20     }
21     private void meth4() // Java 1.9v
22     {
23         System.out.println("InterfaceA private meth4() called");
24     }
25     public static void main(String[] args) // Java 1.8v
26     {
27         System.out.println("InterfaceA main() called");
28         InterfaceA.meth3();
29     }
30 }

```

```

1 package com.pack1;
2
3 public interface InterfaceB
4 {
5     void display();
6 }
7
8
9

```

ClassB.java

```

1 package com.pack1;
2
3 public class ClassB
4 {
5     void show()
6     {
7         System.out.println("ClassB show() called");
8     }
9 }
10
11
12

```

```

1 package com.pack1;
2
3 public class ClassA extends ClassB implements InterfaceA,InterfaceB
4 {
5     @Override
6     public void meth1()
7     {
8         System.out.println("InterfaceA meth1() overridden");
9     }
10     @Override
11     public int greeting(String data)
12     {
13         System.out.println("data : "+data);
14         return 100;
15     }
16     @Override
17     public void display()
18     {
19         System.out.println("InterfaceB display() overridden");
20     }
21     public static void main(String[] args)
22     {
23         InterfaceA aobj=new ClassA();
24         aobj.meth1();
25         System.out.println("==>" +aobj.greeting("Java is awesome!!!"));

```

```
26     aobj.meth2();
27     //aobj.meth4(); // C.E because of private access modifier
28
29     InterfaceB bobj=new ClassA();
30     bobj.display();
31
32     new ClassA().show();
33 }
34 }
```