Interface:

1. An Interface is a Blueprint
2. There can be only Abstract Methods in the Interface, not Method Body.
3. Interface represents the IS-A Relationship.
4. Interface can be implemented by using implements Keyword.
5. If a Class implements more than One Interface, then Interfaces are speared by comma. { Class A implements B, C}
6. An Interface can extend Multiple Interfaces. {public interface ClientFiveThree extends ClientFiveOne, ClientFiveTwo}
7. In Interface all variables are by default public static final and all methods are public and abstract
8. Since Java 8, we can have default and static methods in an interface.
9. Since Java 9, we can have private methods in an interface.

In Interface we declare all abstract methods by default they are public and abstract
All the interface methods need to implemented in child class

```java
package com.dl.one;

public interface ClientOne {

public void m1();
public void m2();
public void m3();
public void m4();

}
```

```java
package com.dl.one;

public class ClientOneImpl implements ClientOne {

@Override
public void m1() {
System.out.println("M1 Method");
}

@Override
public void m2() {
System.out.println("M2 Method");
}

@Override
public void m3() {
System.out.println("M3 Method");
}

@Override
public void m4() {
System.out.println("M4 Method");
}
}
```

```java
package com.dl.one;

public class MainMethod extends ClientOneImpl {

public static void main(String[] args) {

//ClientOne c = new ClinetOne(); // ClinetOne cannot be resolved to a type

ClientOne one = new ClientOneImpl();
one.m1();
one.m2();
one.m3();
one.m4();

ClientOneImpl impl = new ClientOneImpl();
impl.m1();
impl.m2();
impl.m3();
impl.m4();


}
}
```

In Interface all the variables must be public static final by default and we can implement them in different ways as below

```java
package com.dl.two;

public interface ClientTwo {

public static final int x = 10;
public static final int y = 20;

}
```

```java
package com.dl.two;

public class ClientTwoImpl implements ClientTwo {

public static void main(String[] args) {

System.out.println(ClientTwo.x); //10
System.out.println(ClientTwo.y); //20

System.out.println(ClientTwoImpl.x); //10
System.out.println(ClientTwoImpl.y); //20
}
}
```

If a Class implements more than One Interface, then Interfaces are speared by comma. { Class A implements B, C}

```java
package com.dl.three;

public interface ClientThree {

public abstract void m1();
public abstract void m2();

}
```

```java
package com.dl.three;

public interface ClientThreePlus {

public abstract void m3();
public abstract void m4();

}
```

```java
package com.dl.three;

public class ClientThreeImpl implements ClientThree, ClientThreePlus {

@Override
public void m3() {
System.out.println("M3 Method");
}

@Override
public void m4() {
System.out.println("M4 Method");
}

@Override
public void m1() {
System.out.println("M1 Method");
}

@Override
public void m2() {
System.out.println("M2 Method");
}
}
```

```java
package com.dl.three;

public class MainMethod {

public static void main(String[] args) {

// Interface i = new Class();
ClientThree three = new ClientThreeImpl();
three.m1();
three.m2();

// Interface i = new Class();
ClientThreePlus plus = new ClientThreeImpl();
plus.m3();
plus.m4();

// Class c = new Class();
ClientThreeImpl impl = new ClientThreeImpl();
impl.m1();
impl.m2();
impl.m3();
impl.m4();
}

}
```

In some case we don't want to implement all the abstract methods in Interface, then we can implement the expected method in next abstract class and remaining methods in normal class

```java
package com.dl.four;

public interface ClientFour {
public abstract void  m1();
public abstract void  m2();
public abstract void  m3();
public abstract void  m4();
}
```

```java
package com.dl.four;

public abstract class ClientFourImpl implements ClientFour {

@Override
public void m1() {
System.out.println("M1 Method Implementation in Abstract Class");
}
}
```

```java
package com.dl.four;

public class ClientFourPlus extends ClientFourImpl {

@Override
public void m2() {
System.out.println("M2 Method");
}

@Override
public void m3() {
System.out.println("M3 Method");
}

@Override
public void m4() {
System.out.println("M4 Method");
}
}
```

```java
package com.dl.four;

public class MainMethod {

public static void main(String[] args) {

ClientFourImpl c = new  ClientFourPlus();
c.m1();
c.m2();
c.m3();
c.m4();

}
}
```

In Java for normal classes we cannot extend multiple classes at a time, so there is no multiple inheritance
It is possible in Interface

```java
package com.dl.five;

public interface ClientFiveOne {
public abstract void m1();
}
```

```java
package com.dl.five;

public interface ClientFiveTwo {
public abstract void m2();
}
```

```java
package com.dl.five;

public interface ClientFiveThree extends ClientFiveOne, ClientFiveTwo{
public void m3();
}
```

```java
package com.dl.five;

public class MainMethod {
public static void main(String[] args) {
ClientImpl impl = new ClientImpl();
impl.m1();
impl.m2();
impl.m3();
}
}
```

```java
package com.dl.five;

public class ClientImpl implements ClientFiveThree {

@Override
public void m2() {
System.out.println("M2 Method");
}

@Override
public void m1() {
System.out.println("M1 Method");
}

@Override
public void m3() {
System.out.println("M3 Method");
}

}
```

In Java we can extend a class and implement a Interface in a same class

```java
package com.dl.six;

public interface ClientSixI {
public abstract void m1();
public abstract void m2();
}
```

```java
package com.dl.six;

public class ClientSixC {
public void m3() {
System.out.println("M3 Method");
}
}
```

```java
package com.dl.six;

public class ClassSixImpl extends ClientSixC implements ClientSixI {
@Override
public void m1() {
System.out.println("M1 Method");
}

@Override
public void m2() {
System.out.println("M2 Method");
}
}
```

```java
package com.dl.six;

public class MainMethod {

public static void main(String[] args) {

// ClassName c = new ClassName
ClassSixImpl plus = new ClassSixImpl();
plus.m1();
plus.m2();
plus.m3();

// Interface i = new ClassName();
ClientSixI c = new ClassSixImpl();
c.m1();
c.m2();

// ClassName c = new ClassName
ClientSixC c2 = new ClientSixC();
c2.m3();

}

}
```

Since Java 8, we can have default and static methods in an interface.

```
package com.dl.seven;

public interface ClientSeven {

default void m1() {
System.out.println("M1 Method");
}

public static  void m2() {
System.out.println("M2 Method");
}

public static void main(String[] args) {
ClientSeven.m2(); // M2 Method
}
}
```

```
package com.dl.seven;

public class ClientSevenImpl implements ClientSeven {

public static void main(String[] args) {

new ClientSevenImpl().m1(); // M1 Method
}
}
```

Differences of classes, abstract classes, interfaces

In **class** we can we can create **concrete methods**
We can **create instance of class** and **refrence variable**

```
public class A{

}
```

In **abstract class** we can create **abstract methods and concrete methods**
We cant **create instance of abstract class** and can use **refrence variable**

```
public abstract class A{

}
```

In **interfaces** we can create only **abstract methods**
We cant **create instance of interfaces** and can use **refrence variable**

```
public interface A{

}
```