## 7) Difference between Extends vs implements

(1) Extends :- only one class Extends only one class at a time.

Eg:

```
class one {

}
class Two extends One {

}
```

(2) Implements :- one class implements ~~two~~ any no. of interfaces. at a time.

Eg:

```
interface IOne {
    Public void MethodOne()
    {
    }
}
interface ITwo {
    Public Void MethodTwo()
    {
    }
}
Class Demo implements IOne, ITwo {
    Public Void MethodOne() { }
    "  "  MethodTwo() { }.
}
```

Case ②:- A class can extend a class and can implement any no. of interfaces.

Eg:-

```
interface One {
    Void m1() {
    }
}
Class Two {
    Public Void m2() {
    }
}
Class Three extends Two implements One {
    @override
    Public Void m1() {
    }
    @override
    Public Void m2() {
    }
}
```

① Reusability -> extends
② interface -> implementation

-> Reusability followed by implementation

-> class & class -> extends
-> class & interface -> implements
-> class extends class implements interface.

Case ③ :- An interface can extend any no. of interfaces at a time.

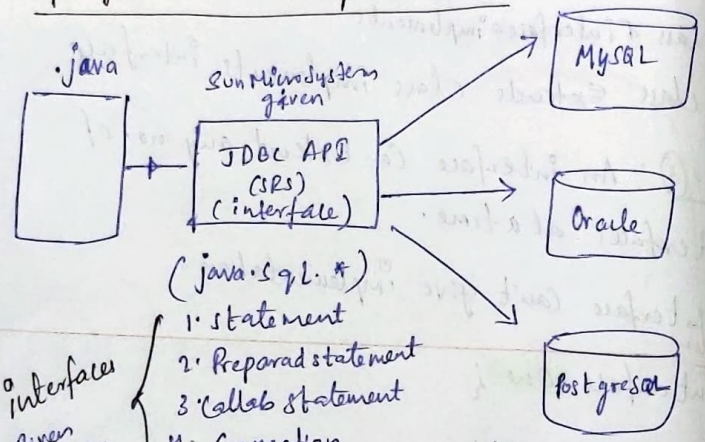
-> interface can't give implementation.

Code:-

```
interface One {
    Void m1();
}
interface Two {
    Void m2();
}
interface Three extends One, Two {
    Void m3();
}
class SampleImpl implements Three
{
    Public Void m1() {
    }
    Public Void m2() {
    }
    Public Void m3() {
    }
}
```

All implementations for these Methods are given here.

## ⑧ Specification and implementation




→ specification given by Sun Microsystem team.

→ implementations for all interfaces given by [redacted] respective service providers should give.

→ Service providers → SQL, Oracle, PostgreSQL gives implementation for (java.sql.*) (or) Interfaces.

→ Sun Micro Systems gives only interfaces (or SRS

→ implementation is given by Service providers (MySQL, Oracle, Postgresql)

→ All Methods present in interface are public and abstract.

→ ① Public → to Make the Method available for Every implementation

② abstract - implementation class is responsible for providing the implementation

[redacted] ⇒ Jdbc API

→ implementation should be given by

ⓐ MySQL
ⓑ Oracle
ⓒ Postgresql

→ Since the Methods present inside the interface is Public and abstract we can't use the Modifiers like static, Private, Protected, Strictfp, Synchronized, native final..

## ⑨ Interface Variable

① Inside the interface we can define Variables.

② Inside the interface Variables defines requirement level Constants

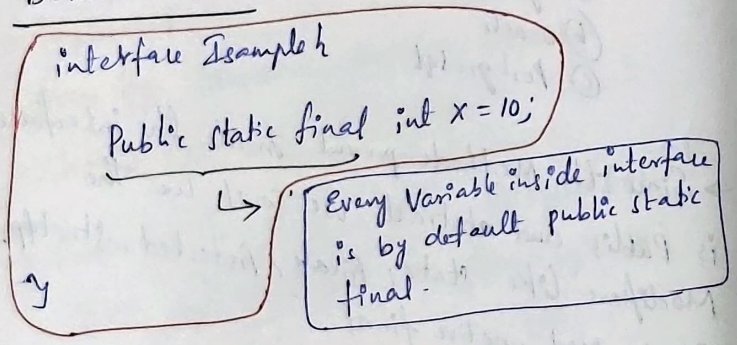
③ Every variables inside the interface is by default public static final.

Eg:- Interface Isample {

    int x = 10;

}

Behind Scene




Public :- To make it available for implementation class object.

Static :- To access it without using implementation class Name.

final :- implementation class can access the value without any modification

Static → Complete information. Should be provided.

→ Method inside interface (incompleteness).
↳ Methods can't be static inside interface
↳ Method inside interface are abstract
↳ They are incomplete.

→ ④ Variables inside interface are public, static and final.

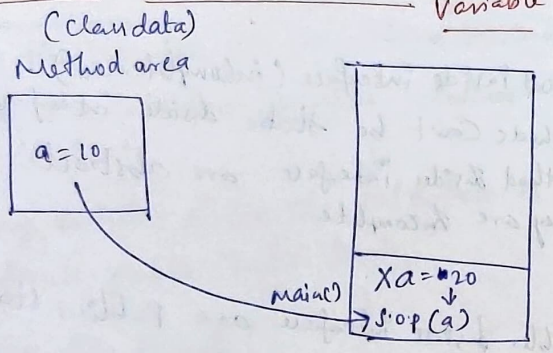
⑤ access modifiers illegal combination
- (a) Private
- (b) Protected
- (c) transient
- (d) Volatile.

→ When variable is final always we have to give value to Variable.

⑥ Since the variable is static and final Compulsorily it should be initialized at the time of declaration otherwise it would be result in Compile time Error.

⑦ local Variable wins over static and final Variable.

⇒ local variable wins over static and final variable




Memore Map.

Static Variable → class data
↳ it is available in Method area

→ if a is not present in main it
Pull a from Method area

Code:
```
interface Iexample {
    int a=10; // Public static final.
}
Public Test App implements Iexample {
    public static Main (String[] args) {
        int a = 20 → local variable
        S.o.p(a) // output : 20
    }
}
```

## ⑩ Interface Naming Conflicts

Case①: if 2 interfaces contain a Method with same signature and return type in the implementation class only one Method implementation is Enough.

Eg:-
```
interface Demo1
{
    Void m1();
}
interface Demo2
{
    Void m2();
}
class CommonImple Implements IDemo1, IDemo2
{ @override
  Public Void m1(){ // valid.
  }
}
```

Case-2 :- if two interfaces contain a method with same name but different arguments in the implementation class we have to provide implementation for both methods and these methods acts as overload methods.

Code :- interface11.java (refer github)

```
interface IDemo11
{
  Void m1();
}
interface IDemo12
{
  Void m2(int i);
}
class CommonImpl implements IDemo11, IDemo12
{
  @Override
  Public Void m1() { → Method with no argument
                     // Valid
  }
  @Override
  Public Void m2(int i) {
                     → Method with argument
                     // Valid
  }
}
```

→ ~~Compiler said that wrong with signature~~

Case-3 :- if two interfaces contains a method with same signature but different return type then it is not possible to implement both interface simultaneously.

Code :- interface12.java (refer github)

```
interface IDemo1 {
    Void m1();
}
interface IDemo2 {
    int m1();
}
class CommonImple Implements IDemo1, IDemo2
{
  @Override
  Public Void m1() {
  }
  @Override
  Public int m1() {
  }
}
```

invalid case Return dtype of M1 Method int 2 interface are different

→ Compiler says you can't write two Method with different return type.

① Can a Java class implements 2 interfaces simultaneously?

Ⓐ Yes possible, Except if two interface Contains a Method with same Signature but different return type.

few interfaces → type in Cmd

① Javap java.lang.Runnable → 1 abstract Method inside it

② Javap java.io.Serializable

③ Javap java.lang.clonable

Note:-

① ~~inside interface the methods are by default~~

② inside interface the variables are by default "public static and final"

③ inside interface the Methods are by default "Public and abstract".

④ We can also write an interface without any variable (or abstract Method.

Eg interface Serializable {

}

class SampleEmple implements Serializable

{

}

interface clonable {

}

class SimpleEmpl implements clonable

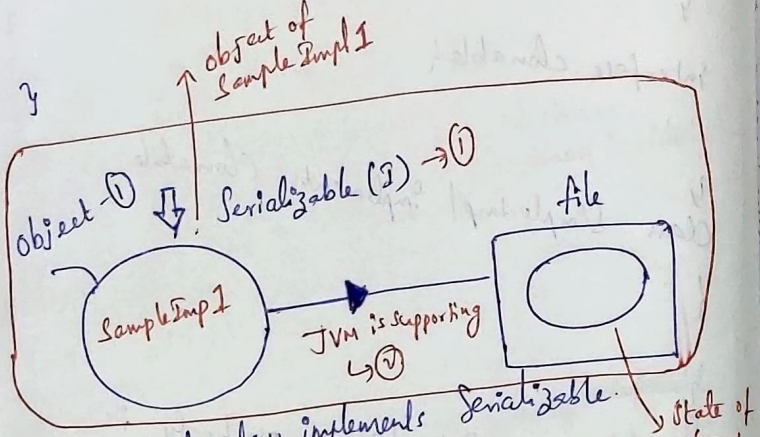
{

}

→ No Methods in interface so No body is given for them. then why we need this?

## (11) Marker interface

```
interface Serializable {
}
class SampleImpl1 implements Serializable
{
}
```




→ SampleImple class implements Serializable.

→ object get facility of storing data on file.

→ Serializable → interface implemented by obj. ①

→ if this object(obj) implements Serializable Automatically that object will get facility of transporting over the Network to Store the state of object in a file. — ②

→ Marking class with Serializable.

→ there are few interfaces where for which a body is not given for those object automatically JVM will supply some functionality.

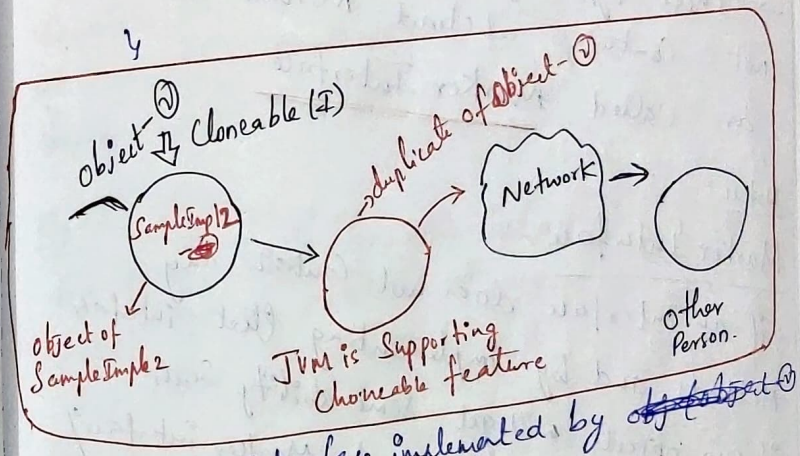
Eg ③: interface Cloneable {

}

class SampleImple2 implements Cloneable

{

}




→ Cloneable - interface implemented by ~~obj~~ object ③

→ Automatically object get duplicated. There duplicated can send it over network to other person.

→ JVM is supporting Cloneable feature.

↳ Let sampleImple implement Cloneable.

## Conclusion of Above discussion

An interface which does not Contain any abstract Method those ~~methods~~ also Supported by JVM in Many ways.

Eg :- ① Serializable
② Cloneable

→ These type of interfaces which does not Contains abstract Methods ~~there~~ are Called Marker Interface.

Note:

## Marker Interface :-

→ if an interface does not Contain any Methods and by implementing that interface if our objects will get Some ability Such type of interface is Called "Marker interface" (or Tag Interface or Ability interface).

Examples
① Serizable
② Cloneable
③ SingleThreadModel.

↳ Pre-defined interface
↳ Meaning of these interfaces known by JVM.

Example-①:-
by implementing Serizable interface we can Send that object across the network and we can Save State of an object into the file.

Example-②
By implementing SingleThreadModel interface Servelet can process only one client request at a time so that we can get "Thread Safety".

Eg-③
By implementing cloneable interface our object is a position to provide Exactly duplicate cloned object.

Questions
① Without having any methods in Marker interface how objects will get ability?
Ⓐ JVM is reponsible to provide required ability.
② Why JVM is providing the required ability to Marker interfaces?
Ⓐ To reduce the Complexity of the programming
③ Can we create our own Market interface?
Ⓐ Yes, it is possible but we need to Customize JVM.

## ⑥ Creating own Marker Interface

① JVM → Customize the JVM by writing lines of code for a Marker interface

⑫ **Adapter class:** it is design pattern allowed to solve the problem of direct implementation of interface Methods)

→ it is a simple java class that implements an interface only with Empty implementation for every Method.

→ if we implement an interface Compulsorily we should give the body for all the Methods whether it required or not.

→ this approach increases the length of the code and reduces readability.

refer git repo Interfaces

Eg

```
interface X {
    Void m1()
    Void m2()
}
Class Test implements X {
    Public Void m2() {
        s.o.p("I am from m3()");
    }
    Public Void m1() { }
}
```

→ In the above approach Even though we want only m3(), still we need to give body for all the abstract Methods, which increase the length of code, to reduce this we need to use "Adapter class".

→ Instead of implementing the interface directly we opt for "Adapter class".

→ Adapter class are such classes which implements the interface and gives dummy implementation for all the abstract Methods of interface.

→ so if we Extends Adapter classes then we can easily give body only for those Methods which are interested in giving the body.

refer

Eg Interface16.java (github repo)

```
Interface IDemo3 {
    Void m1();
    Void m2();
}
abstract class Adapterclass implements IDemo3 {
    Public Void m1() { }
    Public Void m2() { }
}
Class TestApp Exteds Adapter class {
    Public Void m3() {
        s.o.p("implementing m3");
    }
}
```

## Adapter class Code :- Interface16.java

```
Interface Demo {
    Void m1( );
    Void m2( );
    Void m3( );
    Void m4( );
}
// AdapterClass give dummy implementation
   for all Methods.
abstract class Adapterclass implements Demo
{
    Public Void m1( );
    Public Void m2( );
    Public Void m3( );
    Public Void m4( );
}
// give implementation for only m3 Method
class TestApp Extends Adapterclass
{
    Public Void m3( ){
        S.o.p(" give implementation to m3 ");
    }
}
Public class interface16 {
        Public static Void main(String[] aq)
    Demo demo = new TestApp( );
        demo.m3( );
}
```

## Real life Example

Servlet (Interface)
↓ Implements
GenericServlet (abstract class) → implements Servlet
↓ Extends
HttpServlet (Abstract class)
↓ Extends
MyServlet (class)

→ Generic Servlet also have some limitations