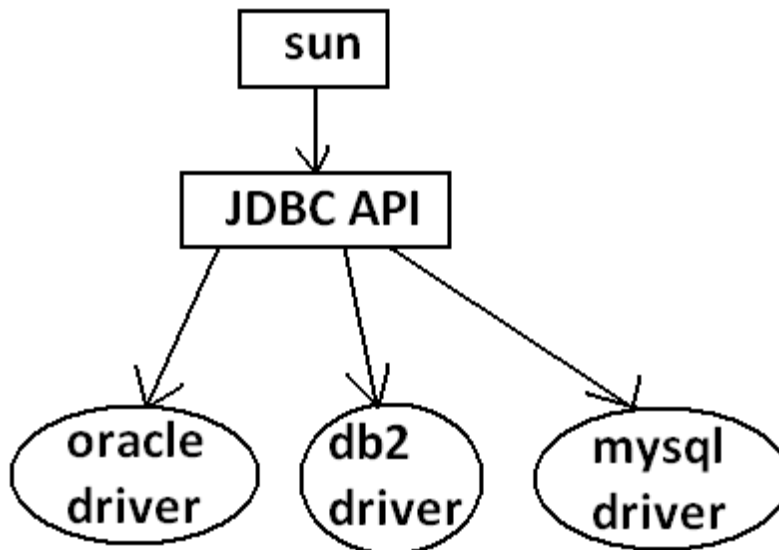


Interfaces

Def1: Any service requirement specification (srs) is called an interface.

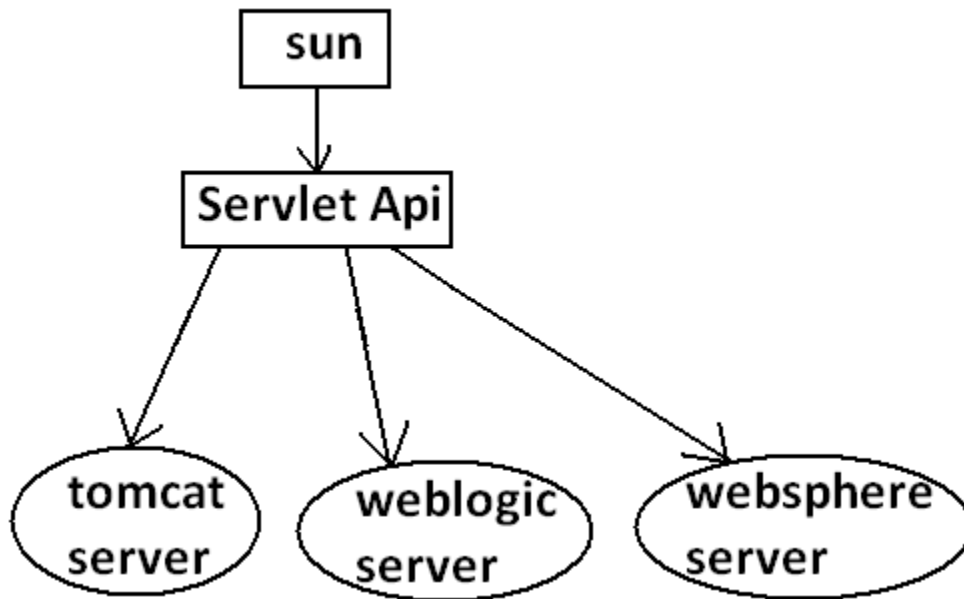
Example1: Sun people responsible to define JDBC API and database vendor will provide implementation for that.

Diagram:



Example2: Sun people define Servlet API to develop web applications web server vendor is responsible to provide implementation.

Diagram



Def2: From the client point of view an interface defines the set of services what is expecting. From the service provider point of view an interface defines the set of services what is offering. Hence an interface is considered as a contract between client and service provider.

Example: ATM GUI screen describes the set of services what bank people offering, at the same time the same GUI screen the set of services what customer is expecting hence this GUI screen acts as a contract between bank and customer.

Def3: Inside interface every method is always abstract whether we are declaring or not hence interface is considered as 100% pure abstract class.

Summary def: Any service requirement specification (SRS) or any contract between client and service provider or 100% pure abstract classes is considered as an interface.

Note1:

Whenever we are implementing an interface compulsory for every method of that interface we should provide implementation otherwise we have to declare class as abstract in that case child class is responsible to provide implementation for remaining

methods.

Note2:

Whenever we are implementing an interface method compulsory it should be declared

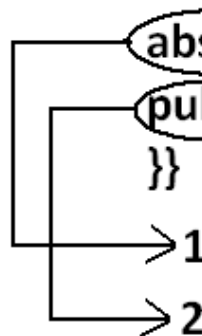
as public otherwise we will get compile time error.

Example:

```
interface Interf
```

```
{  
void methodOne();  
void methodTwo();  
}
```

```
abstract class ServiceProvider implements Interf{  
    public void methodOne(){  
    }  
}
```



1
2

```
class SubServiceProvider extends ServiceProvider
```

```
{  
}
```

Output:

Compile time error.

D:\Java>javac SubServiceProvider.java

SubServiceProvider.java:1:

SubServiceProvider is not abstract and does not override
abstract method methodTwo() in Interf

class SubServiceProvider extends ServiceProvider

Extends vs implements:

A class can extend only one class at a time.

Example:

```
class One{  
    public void methodOne(){  
    }  
}  
  
class Two extends One{  
}
```

A class can implement any no. Of interfaces at a time.

Example:

```
interface One{  
    public void methodOne();  
}  
  
interface Two{  
    public void methodTwo();  
}  
  
class Three implements One,Two{  
    public void methodOne(){  
    }  
    public void methodTwo(){  
    }  
}
```

A class can extend a class and can implement any no. Of interfaces simultaneously.

```
interface One{  
    void methodOne();  
}
```

```
}  
class Two  
{  
public void methodTwo(){  
}  
}  
class Three extends Two implements One{  
public void methodOne(){  
}  
}
```

An interface can extend any no. Of interfaces at a time.

Example:

```
interface One{  
void methodOne();  
}  
interface Two{  
void methodTwo();  
}  
interface Three extends One,Two  
{  
}
```

Which of the following is true?

1. A class can extend any no. Of classes at a time.
2. An interface can extend only one interface at a time.
3. A class can implement only one interface at a time.
4. A class can extend a class and can implement an interface but not both simultaneously.

5. An interface can implement any no.Of interfaces at a time.
6. None of the above.

Ans: 6

Consider the expression X extends Y for which of the possibility of X and Y this expression is true?

1. Both x and y should be classes.
2. Both x and y should be interfaces.
3. Both x and y can be classes or can be interfaces.
4. No restriction.

Ans: 3

X extends Y, Z ?

X, Y, Z should be interfaces.

X extends Y implements Z ?

X, Y should be classes.

Z should be interface.

X implements Y, Z ?

X should be class.

Y, Z should be interfaces.

X implements Y extend Z ?

Example:

```
interface One{  
}  
class Two {  
}  
class Three implements One extends Two{  
}
```

Output:

Compile time error.

D:\Java>javac Three.java

Three.java:5: '{' expected

class Three implements One extends Two{

Interface method:

Every method present inside interface is always public and abstract whether we are declaring or not. Hence inside interface the following method declarations are equal.

void methodOne();
public Void methodOne();
abstract Void methodOne(); Equal
public abstract Void methodOne();

public: To make this method available for every implementation class.

abstract: Implementation class is responsible to provide implementation .

As every interface method is always public and abstract we can't use the following modifiers for interface methods.

Private, protected, final, static, synchronized, native, strictfp.

Inside interface which method declarations are valid?

1. public void methodOne(){}
2. private void methodOne();
3. public final void methodOne();
4. public static void methodOne();
5. public abstract void methodOne();

Ans: 5

Interface variables:

- ❑ An interface can contain variables
- ❑ The main purpose of interface variables is to define requirement level constants.
- ❑ Every interface variable is always public static and final whether we are declaring or not.

Example:

```
interface interf
{
int x=10;
}
```

public: To make it available for every implementation class.

static: Without existing object also we have to access this variable.

final: Implementation class can access this value but cannot modify.

Hence inside interface the following declarations are equal.

```
int x=10;
public int x=10;
static int x=10;
final int x=10; Equal
public static int x=10;
public final int x=10;
static final int x=10;
public static final int x=10;
```

- ❑ As every interface variable by default public static final we can't declare with the following modifiers.

- o Private

- o Protected
- o Transient
- o Volatile

□ For the interface variables compulsory we should perform initialization at the time of declaration only otherwise we will get compile time error.

Example:

```
interface Interf
```

```
{  
int x;  
}
```

Output:

Compile time error.

D:\Java>javac Interf.java

Interf.java:3: = expected

```
int x;
```

Which of the following declarations are valid inside interface ?

1. int x;
2. private int x=10;
3. public volatile int x=10;
4. public transient int x=10;
5. public static final int x=10;

Ans: 5

Interface variables can be access from implementation class but cannot be modified.

Example:

```
interface Interf
```

```
{  
int x=10;  
}
```

Example 1:**Example**

```
class Test implements Interf
{
    public static void main(String args[]){
        x=20;
        System.out.println("value of x"+x);
    }
}
```

output:

compile time error.

D:\Java>javac Test.java

Test.java:4: cannot assign a value to final variable x
x=20;

Example 2:

```
class Test implements Interf
```

```
{
    public static void main(String args[]){
        int x=20;
```

```
//here we declaring the variable x.
```

```
System.out.println(x);
```

```
}
```

```
}
```

Output:

D:\Java>javac Test.java

D:\Java>java Test

20\

Interface naming conflicts:

Method naming conflicts:

Case 1:

If two interfaces contain a method with same signature and same return type in the implementation class only one method implementation is enough.

Example 1:

```
interface Left
{
    public void methodOne();
}
```

Example 2:

```
interface Right
{
    public void methodOne();
}
```

Example 3:

```
class Test implements Left,Right
{
    public void methodOne()
    {
    }}
}}
}}
}}
```

Output:

D:\Java>javac Left.java

D:\Java>javac Right.java

D:\Java>javac Test.java

Case 2:

50, Lakhanpur Housing Society, Near Lucky Restaurant, Vikas Nagar 208024

Contact: 9794687277, 8707276645

Web: www.bytecode.co.in, Email: bytecodeitsolutions@gmail.com

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 a overloaded methods

Example 1:

```
interface Left
{
    public void methodOne();
}
```

Example 2:

```
interface Right
{
    public void methodOne(int i);
}
```

Example 3:

```
class Test implements Left,Right
{
    public void methodOne()
    {
    }
    public void methodOne(int i)
    {
    }
}
```

Output:

```
D:\Java>javac Left.java
```

```
D:\Java>javac Right.java
```

```
D:\Java>javac Test.java
```

Case 3:

If two interfaces contain a method with same signature but different return types then it

is not possible to implement both interfaces simultaneously.

Example 1:

```
interface Left
{
    public void methodOne();
}
```

Example 2:

```
interface Right
{
    public int methodOne(int i);
}
```

We can't write any java class that implements both interfaces simultaneously.

Is a java class can implement any no. Of interfaces simultaneously ?

Yes, except if two interfaces contains a method with same signature but different return types.

Variable naming conflicts:

Two interfaces can contain a variable with the same name and there may be a chance variable naming conflicts but we can resolve variable naming conflicts by using interface names.

Example 1:

```
interface Left
{
    int x=888;
}
```

Example 2:

```
interface Right
```

```
{  
    int x=999;  
}
```

Example 3:

```
class Test implements Left,Right
```

```
{  
    public static void main(String args[]){  
        //System.out.println(x);  
        System.out.println(Left.x);  
        System.out.println(Right.x);  
    }  
}
```

Output:

```
D:\Java>javac Left.java
```

```
D:\Java>javac Right.java
```

```
D:\Java>javac Test.java
```

```
D:\Java>java Test
```

888

999

Marker interface:

If an interface doesn't contain any methods and by implementing that interface if our objects will get some ability such type of interfaces are called Marker interface (or) Tag

interface (or) Ability interface.

Example:

50, Lakhanpur Housing Society, Near Lucky Restaurant, Vikas Nagar 208024

Contact: 9794687277, 8707276645

Web: www.bytecode.co.in, Email: bytecodeitsolutions@gmail.com

Serializable

Cloneable

RandomAccess These are marked for some ability

SingleThreadModel

.
.
.
.
.

Example 1:

By implementing Serializable interface we can send that object across the network and

we can save state of an object into a file.

Example 2:

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

Example 3:

By implementing Cloneable interface our object is in a position to provide exactly duplicate cloned object.

Without having any methods in marker interface how objects will get ability ?

Internally JVM is responsible to provide required ability.

Why JVM is providing the required ability in marker interfaces ?

To reduce complexity of the programming.

Is it possible to create our own marker interface ?

Yes, but customization of JVM must be required.

Ex : Sleepable , Jumpable ,

Adapter class:

- Adapter class is a simple java class that implements an interface only with empty implementation for every method.
- If we implement an interface directly for each and every method compulsory we should provide implementation whether it is required or not. This approach increases length of the code and reduces readability.

Example 1:

```
interface X{  
void m1();  
void m2();  
void m3();  
void m4();  
//.  
//.  
//.  
//.  
void m5();  
}
```

Example 2:

```
class Test implements X{  
public void m3(){  
System.out.println("m3() method is called");  
}  
public void m1(){}  
public void m2(){}  
public void m4(){}  
public void m5(){}  
}
```


- We can resolve this problem by using adapter class.
- Instead of implementing an interface if we can extend adapter class we have to provide implementation only for required methods but not for all methods of that interface.
- This approach decreases length of the code and improves readability.

Example 1:

```
abstract class AdapterX implements X{  
    public void m1(){  
    public void m2(){  
    public void m3(){  
    public void m4(){  
    //.  
    //.  
    //.  
    public void m1000(){  
    }  
}
```

Example 2:

```
public class Test extend AdapterX{{  
    public void m3(){  
    }}  
}
```

Example:

Generic

Servlet(I)
GenericServlet(AC)

```
import javax.servlet.*;
class MyServlet implements Servlet{
public void init(ServletConfig config){}
public void destroy(){ }
public void service(ServletRequest request,ServletResponse response){}
public String getServletInfo(){return null;}
public ServletConfig getServletConfig(){return null;}
}
```

```
import javax.servlet.*;
class MyServlet1 extends GenericServlet{
public void service(ServletRequest request,ServletResponse response){}
}
```

Generic Servlet simply acts as an adapter class for Servlet interface.

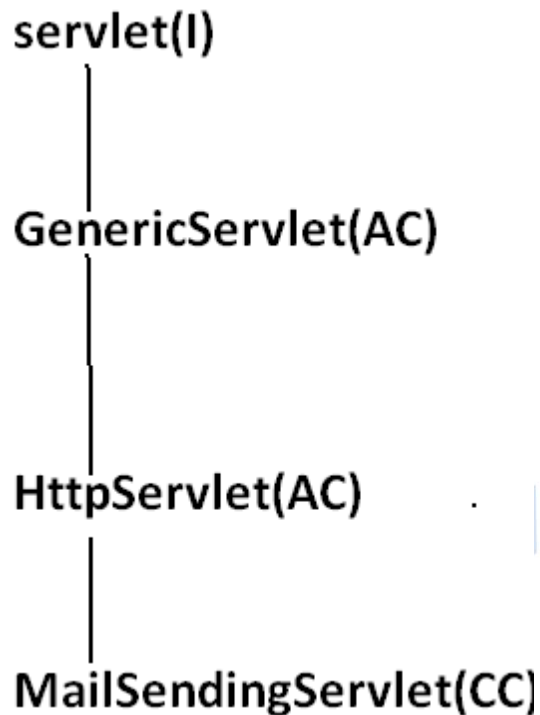
Note : marker interface and Adapter class are big utilities to the programmer to simplify programming.

What is the difference between interface,abstract class and concrete class?

When we should go for interface,abstract class and concrete class?

- ☐ If we don't know anything about implementation just we have requirement specification then we should go for interface.
- ☐ If we are talking about implementation but not completely (partial implementation) then we should go for abstract class.
- ☐ If we are talking about implementation completely and ready to provide service then we should go for concrete class.

Example:



What is the Difference between interface and abstract class ?

Interface-

If we don't know anything about implementation just we have requirement specification then we should go for interface.

Every method present inside interface is always public and abstract whether we are declaring or not.

We can't declare interface methods with the modifiers private, protected, final, static, synchronized, native, strictfp.

Every interface variable is always public static final whether we are declaring or not.

Every interface variable is always public static final we can't declare with the following modifiers. Private, protected, transient, volatile.

For the interface variables compulsory we should perform initialization at the time of declaration otherwise we will get compile time error.

Inside interface we can't take static and instance blocks.

Inside interface we can't take constructor.

Abstract class-

If we are talking about implementation but not completely (partial implementation) then we should go for abstract class.

Every method present inside abstract class need not be public and abstract.

There are no restrictions on abstract class method modifiers.

Every abstract class variable need not be public static final.

There are no restrictions on abstract class variable modifiers.

It is not require to perform initialization for abstract class variables at the time of declaration.

Inside abstract class we can take both static and instance blocks.

Inside abstract class we can take constructor.

We can't create object for abstract class but abstract class can contain constructor what is the need ?

abstract class constructor will be executed when ever we are creating child class object to perform initialization of child object.

Example:

```
class Parent{
    Parent()
    {
        System.out.println(this.hashCode());
    }
}

class child extends Parent{
    child(){
        System.out.println(this.hashCode());
    }
}
```

```
}  
class Test{  
public static void main(String args[]){  
child c=new child();  
System.out.println(c.hashCode());  
}  
}
```

Note : We can't create object for abstract class either directly or indirectly.

Every method present inside interface is abstract but in abstract class also we can take only abstract methods then what is the need of interface concept ?

We can replace interface concept with abstract class. But it is not a good programming practice. We are misusing the roll of abstract class. It may create performance problems also.

(this is just like recruiting IAS officer for sweeping purpose)

Example :

```
interface X {  
    -----  
    -----  
}  
class Test implements X {  
    -----  
    -----  
}
```

Test t=new Test();

- //takes 2 sec*
1) performance is high
2) while implementing X
we can extend some
other classes

```
abstract class X {  
    -----  
    -----  
}  
class Test extends X {  
    -----  
    -----  
}
```

Test t=new Test();

- //takes 20sec*
1) performance is low
2) while extending X we can't
extend any other classes

If every thing is abstract then it is recommended to go for interface.

Why abstract class can contain constructor where as interface doesn't contain constructor ?

The main purpose of constructor is to perform initialization of an object i.e., provide values for the instance variables, Inside interface every variable is always static and there is no chance of existing instance variables. Hence constructor is not required for interface.

But abstract class can contains instance variable which are required for the child object

to perform initialization for those instance variables constructor is required in the case of abstract class.

