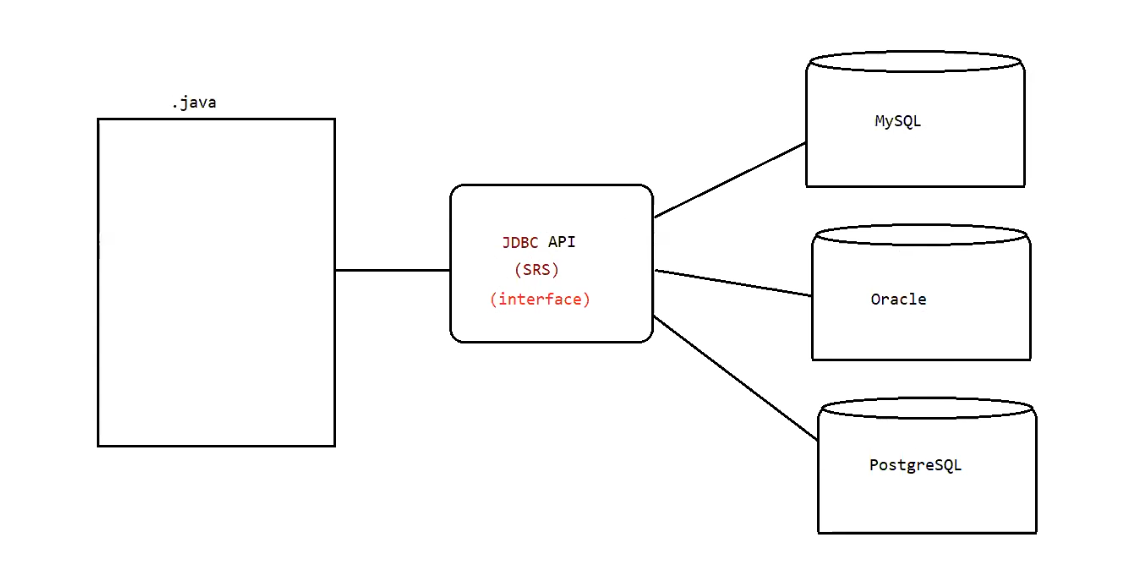
Interface:

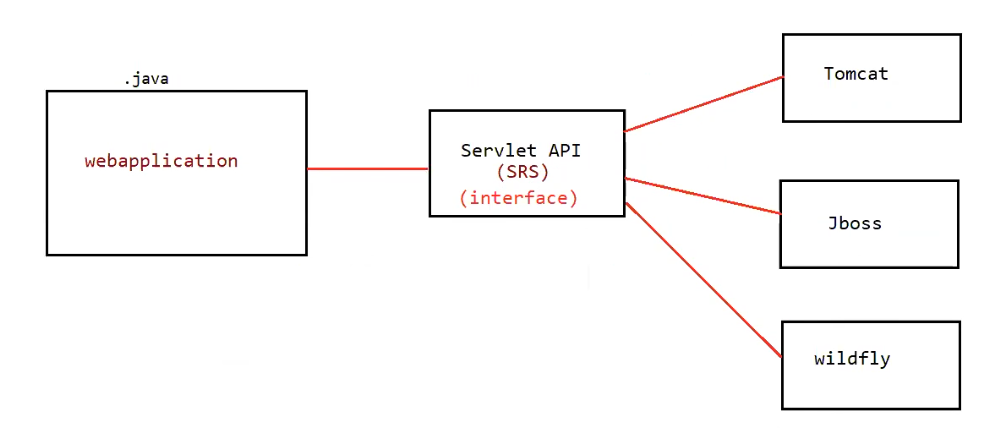
Any SRS (Software Requirement Specification) is called an interface



Explanation:

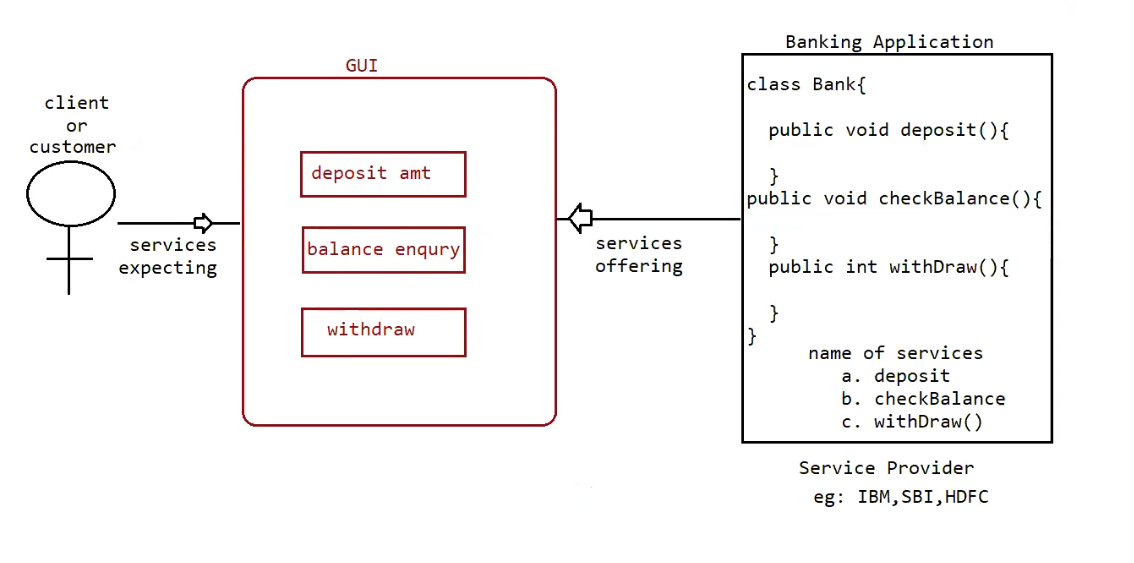
* We write a java program ( .java file ) to connect with the database
* There are different database’s in the market some of them are MySQL, Oracle, PlSQL. To connect with these 3 databases we do not need 3 different java programs , only a single java program is enough to connect with the 3 databases
* To promote this type of specification sun micro system team had given one API( Application Programming Interface , API is collection of .class files , API means someone would write the code we will use it ) JDBC .
* So this can be said as SRS (service requirement specification).
* In java to represent this srs we use Interface.

Example: 2



* Here this .java file is a web application, it does not run on jvm, this application runs on the server. There are no of servers, some of them are Tomcat, Jboss, Wildfly.
* To promote this type of specification java team had given one API called Servlet API
* So this can be said as an SRS
* In java to represent srs we use interface.

Example: 3



Def:

From the client point of view an interface define the set of services what is expecting.

From the service provider point of view an interface define the set of services what is offering

So interface acts as a contract (for that contract only we will write the code) between the client and service provider

Eg:: GUI screen of ATM defines the set of services what the customer is expecting,

Bank offered the same set of services what customer is expecting.

Through class we can have abstract methods and concrete methods (methods that have logic), so 100% abstraction is not possible through class.

Inside Interface every method is always abstract whether we are declaring with abstract keyword or not.

But through Interface 100% abstraction is possible.

Eg:

Interface Account {

// it is 100% abstract class

// the methods are by default (no need to specify manually) “abstract and public “

void withDraw();

void deposit();

void checkBalance();

}

Note: In abstract class if we write void withDraw() its access modifier is default.

Interface corresponds to Service Requirement Specification (SRS) or contract between service provider and the client / customer (or) 100% pure abstract class.

If any one of this requirement you are getting in your java code then use Interface.

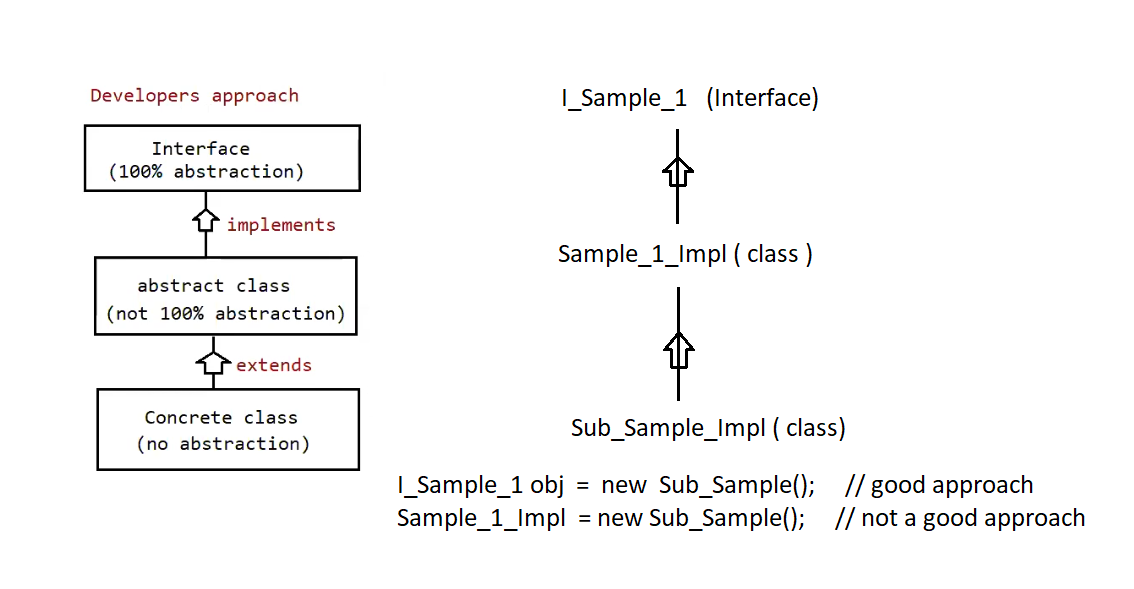
* Interface name should be preceded with "I", to differentiate b/w class name and interface name
* since interface methods needs to be overridden , the class name that overrides should be same name as Interface, but prefix “I” should be removed and, “Impl” should be added at postfix
* “implements” keyword is used after the class name for the implementation for the interface
* Class implementing an interface must implement all the methods of that interface, otherwise the class must be declared as abstract.
* If a class contain atleast one abstract method, then the class should also be abstract.
* If all the methods in interface are overridden manually in the class then overridden rules must be followed, and “public” access modifier should be preceded with overridden methods, because if we don’t write access modifier in the class it is marked with “default” modifier but the in interface “public” modifier is used by default. so visibility decreases which is not the rule of overriding, to avoid this overridden methods should also be public.
* Whenever we are implementing an interface compulsorily for every method of that interface, we should provide implementation otherwise we have to declare class as abstract class in that case child class is responsible to provide implementation for remaining methods.
* Whenever we are implementing interface method it should be marked as public otherwise it would result in compile time error.

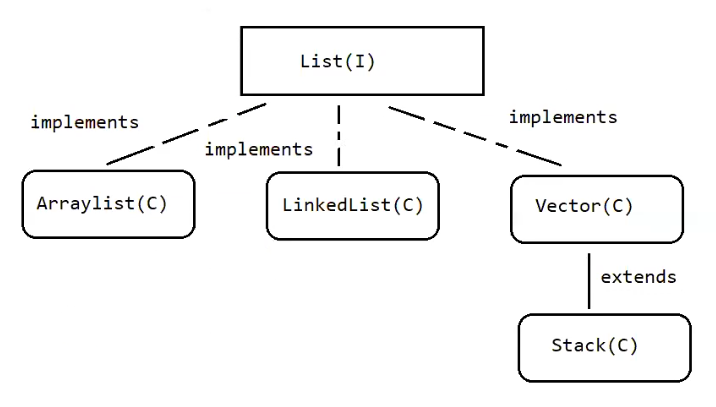
<https://www.refreshjava.com/java/interface-key-points>

Eg: Interface\_Eg1

Eg: Interface\_Eg2

// go through the code

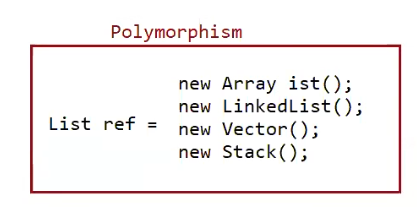




At runtime I don’t know the object what is coming I only know all of them are the implementation classes of list interface

So reference should be of type list interface = object is of list implementation class

Assume at the left side reference of type Arraylist and object at right is of LinkedList (or) Stack then that would result in the error.



To avoid errors we need reference of parent type.

extends:: One class can extend only one class at a time.

Eg:: class One{}

class Two extends One{}

implements:: One class can implement any no of interface at a time

Eg:: interface One{

public void methodOne();

}

Interface Two{

public void methodTwo();

}

Class Demo implements One,Two{

public void methodOne() {}

public void methodTwo(){}

}

Case2:

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

Eg: interface One{

public void methodOne();

}

interface Two{

public void methodTwo();

}

class Three{

}

class Three extends Three implements One,Two{

@Override

public void methodOne(){

….

}

@override

Public void methodTwo(){

….

}

}

Note: whenever we are writing an application we always try to get the code written by someone, so reusability nature is the first priority we give, if reusability nature does not exist that is when we try to look for srs for which we try to give an implementation.

That is why first it is always extends followed by implements (in case2)

Note:

class & class extends

class & interface implements

class extends class & implements interface

interface & interface extends

(because interface can’t give implementation, interface refers to 100% abstract class. so one interface extends another interface)

Case 3:

An interface can extend any no of interface at a time

Eg:: interface One{

public void methodOne();

}

interface Two{

public void methodTwo();

}

interface Three extends One, Two{

public void methodThree();

}

class SampleImpl implements Three{

public void methodOne() { }

public void methodTwo() { }

public void methodThree() { }

}

// since class SampleImpl implements interface Three it should provide implementation for methodOne(), methodTwo(), methodThree(), because interface One, Two methods are a part of interface Three .

Answer the following?

Which of the following is true?

1. A class can extend any no of class at a time.

// class can extend only one class at a time

1. An interface can extend only one interface at a time

// interface can extend any no of interface at a time

1. A class can implement only one interface at a time

// A class can implement any no of interface at a time

1. A class can extend a class and can implement an interface but not both simultaneously

// A classs can extend a class and implement an interface simultaneously

1. An interface can implement any no of interfaces at a time

// an interface can extend any no of interface at a time

1. None of the above

Ans: f

Consider the expression X extends Y which of the possibility of X and Y 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

Equation: X extends Y è true

class extends class => true

interface extends interface => true

Ans: 3

Predict X,Y,Z

1. X extends Y,Z ?
2. X extends Y implements Z ?
3. X implements Y,Z ?
4. X implements Y extends Z ?

Explanation:

X extends Y,Z

X => interface

Y=> interface

Z => interface

X extends Y implements Z

X => class

Y => class

Z=> interface

X implements Y,Z

X=> class

Y=> interface

Z=> interface

X implements Y extends Z

// illegal combination

Interface Methods:

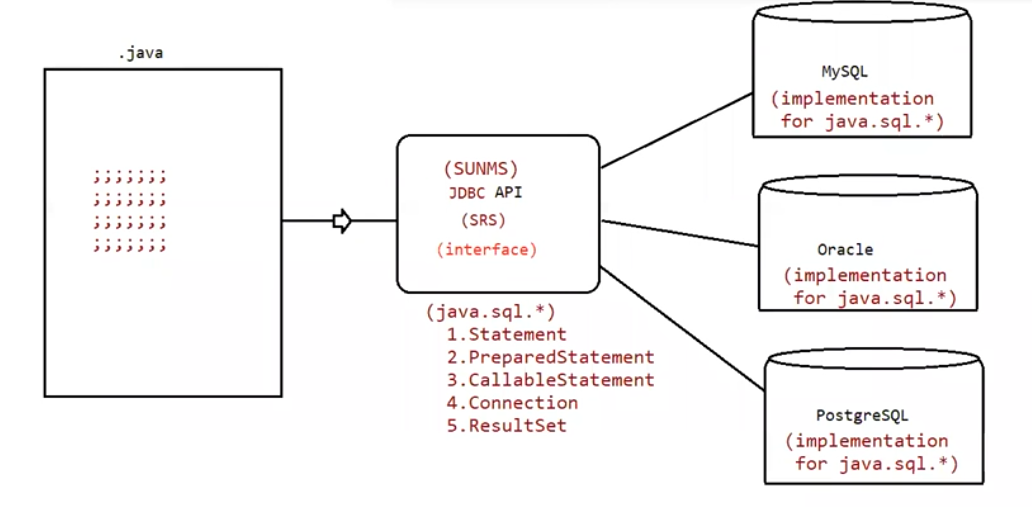
Every method present inside the interface is public

Every method present inside the interface is abstract

How many valid declarations?

1. Void methodOne?
2. public void methodOne?
3. abstract void methodOne?
4. public abstract void methodOne();

Ans: All are valid



Java code will be present in the .java file that code will work for all the three databases (MySQL, Oracle, PostgreSQL). sun micro systems has given the JDBC API and it consists of java.sql\* package, The package consists of above 5 interfaces

1. Statement
2. PreparedStatment
3. CallableStatment
4. Connection
5. ResultSet

Sun micro system team gave the above 5 interfaces. they had just gave the specification but they had not gave the implementation.

The implementation of these 5 interfaces should be given by the database vendors (mysql , oracle , plsql ), because if java team writes the implementation for these interfaces they it would be database specific and may cause errors while connection with other database .

So the implementation is left for the database vendors.

1. To make the method available for the implementation class they are given public
2. Implementation class is responsible for providing the implementation so they are given abstract

Because of the above two properties method is visible, and implementation can be given by the DB vendors.

We divide modifiers into two groups:

* **Access Modifiers** - controls the access level
* **Non-Access Modifiers** - do not control access level, but provides other functionality

Access Modifiers

For **classes**, you can use either public or *default*:

|  |  |  |
| --- | --- | --- |
| public | The class is accessible by any other class |  |
| default | The class is only accessible by classes in the same package. This is used when you don't specify a modifier. |  |

For **attributes, methods and constructors**, you can use the one of the following:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** |  |
| public | The code is accessible for all classes |  |
| private | The code is only accessible within the declared class |  |
| default | The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the [Packages chapter](https://www.w3schools.com/java/java_packages.asp) |  |
| protected | The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the [Inheritance chapter](https://www.w3schools.com/java/java_inheritance.asp) |  |

## Non-Access Modifiers

For **classes**, you can use either final or abstract:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** |  |
| final | The class cannot be inherited by other classes |  |
| abstract | The class cannot be used to create objects (To access an abstract class, it must be inherited from another class.) |  |

For **attributes and methods**, you can use the one of the following:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| final | Attributes and methods cannot be overridden/modified |
| static | Attributes and methods belongs to the class, rather than an object |
| abstract | Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example **abstract void run();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the [Inheritance](https://www.w3schools.com/java/java_inheritance.asp) and [Abstraction](https://www.w3schools.com/java/java_abstract.asp) chapters |
| transient | Attributes and methods are skipped when serializing the object containing them |
| synchronized | Methods can only be accessed by one thread at a time |
| volatile | The value of an attribute is not cached thread-locally, and is always read from the "main memory" |