**Abstraction-**

It is the process of hiding the certain details and showing the important information to the end user called as “Abstraction”.

Example-

1. Car

2. ATM Machine

3. TV Remote

How to achieve the Abstraction in java?

There are two ways to achieve the abstraction in java.

1. Abstract class
2. Interface

**Abstract class**

* Abstract class have constructor
* It contains abstract methods or concrete methods or empty class or combination of both methods.
* We cannot create the object of abstract class.
* To use abstract method of class, we should extends the abstract class and use that methods.
* If we don't want to implement or override that method, make that class as abstract.
* If any method is abstract in a class then that class must be declared as abstract

Note- Multiple inheritances are not allowed in abstract class but allowed in interfaces

Example-1

**package** com.abstraction;

**public** **class** Test { // this is abstract class

**abstract** **void** x1(); // this is abstract method

}

Here, method is the abstract then class should be abstract only as per below example

**package** com.abstraction;

**public** **abstract** **class** Test { // this is abstract class

**abstract** **void** x1(); // this is abstract method

}

Example-2- we can write multiple abstract method into abstract class as per below

**package** com.abstraction;

**public** **abstract** **class** Test {

**abstract** **void** x1(); // abstract method

**abstract** **void** x2();

}

How to implement abstract methods?

We need to create the class which extends from abstract class as shown in below.

**package** com.abstraction;

//this is the implementation class

**public** **class** Example **extends** Test {

@Override

**void** x1() {

System.***out***.println("x1 method..");

}

@Override

**void** x2() {

System.***out***.println("x2 method..");

}

}

**package** com.abstraction;

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

Example example = **new** Example();

example.x1();

example.x2();

}

}

Note- Suppose in the sub class, I don’t want to override the abstract methods then make that subclass as abstract.

**Interface-**

1. It contains public abstract methods and public static final variables by default.
2. We must follow I to C design principle in java. It means every class should be implemented by some interfaces.
3. In company, Team Lead or Manager level people can design the interface then give it to developer for implementing it.

4. We cannot create the object of interface.

5. In interface, we can just define the method only but implemented those methods into implemented class.

6. Before 1.7, interface does not have any method body.

7. In 1.8 Declare the default & static method with body in interface.

8. In 1.9 we can define the private methods in interface also.

9. Java supports multiple inheritance in the terms of interfaces but not classes.

10. Interface does not have constructor.

**Syntax**

interface interface\_name {

}

Example-1

**public** **interface** Demo {

**public** **abstract** **void** x1();

**public** **static** **final** **int** ***a***=5;

}

Example-2

**public** **interface** A {

**public** **abstract** **void** x1(); // allowed

**public** **void** x2(); // allowed

**abstract** **void** x3();// allowed

**void** x4(); // allowed

}

Note- if we don’t write public or abstract in interface then JVM will insert it automatically.

}

Example-3

**package** com.abstraction;

**public** **interface** A {

**public** **abstract** **void** x1(); // allowed

}

**package** com.abstraction;

//this is the implementation class

**public** **class** Test **implements** A {

@Override

**public** **void** x1() {

System.***out***.println("Test-x1 method");

}

}

**package** com.abstraction;

**public** **class** TestExample {

**public** **static** **void** main(String[] args) {

Test test= **new** Test();

test.x1();

}

}

Output

Test-x1 method

Example-4

**package** com.abstraction;

**public** **interface** A {

}

**package** com.abstraction;

**public** **interface** B {

}

**package** com.abstraction;

**public** **interface** C **extends** A,B {

}

This is allowed in java.

Example-5

**package** com.abstraction;

**public** **interface** A {

**public** **abstract** **void** x1(); // allowed

}

**package** com.abstraction;

**public** **interface** B {

**public** **abstract** **void** x1(); // allowed

}

**package** com.abstraction;

**public** **class** Test **implements** A,B {

@Override

**public** **void** x1() {

System.***out***.println("Test-x1 method");

}

}

**package** com.abstraction;

**public** **class** TestExample {

**public** **static** **void** main(String[] args) {

Test test= **new** Test();

test.x1();

}

}

**package** com.abstraction;

**public** **class** TestExample {

**public** **static** **void** main(String[] args) {

Test test= **new** Test();

test.x1();

}

}

Output

Test-x1 method

Below are the list of possible scenario regarding the interface and

Note- Try this from your end on laptop or desktop.

* interface can extend interface1 and interface2
* Interface can extends interface
* Interface can extends the multiple interface
* class extends class implements interface
* class implements interface
* class extends cl  
  ass implements interface1 and interface2

Why interface?

Suppose there is a requirement for Amazon to integrate SBI bank code into their shopping cart. Their customers want to make payment for products they purchased.

Let's say SBI develops code like below:

class Transaction {

void withdrawAmt(int amtToWithdraw) {

//logic of withdraw

// SBI DB connection and updating in their DB

}

}

Amazon needs this class so they request SBI bank for the same. The problem with SBI is that if they give this complete code to amazon they risk exposing everything of their own database to them as well as their logic, which cause a security violation.

Now the solution is for SBI to develop an Interface of Transaction class as shown below:

interface Transactioni {

void withdrawAmt(int amtToWithdraw) ;

}

class TransactionImpl implements Transactioni {

void withdrawAmt(int amtToWithdraw) {

//logic of withdraw

//SBI DB connection and updating in their DB

}

}

Now how amazon will do this as below as-

Transactioni ti = new TransactionImpl();

ti.withdrawAmt(500);

In this case, both application can achieve their aims.