***Abstraction***

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

Example- Real life scenario is car.

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.
* 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 those methods as abstract.
* If any method is abstract in a class then that class must be declared as abstract
* We cannot create the object of abstract class.

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** a1(); // this is abstract method a1

}

Note>>

Here, If we declared the method is an abstract then class should be abstract only as per below example

**package** com.abstraction;

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

**abstract** **void** a1(); // 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** a1(); // abstract method a1

**abstract** **void** a2(); // abstract method a2

}

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** a1() {

System.***out***.println("this is the a1 method..");

}

@Override

**void** a2() {

System.***out***.println("this is the a2 method..");

}

}

**package** com.abstraction;

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

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

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

example.a1();

example.a2();

}

}

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

**The main advantages of Abstraction are:**

1. We can achieve security as we are not highlighting our internal implementation. (i.e., outside person doesn't aware our internal implementation.)

2. Enhancement will become very easy because without effecting end user we can able to perform any type of changes in our internal system.

3. It provides more flexibility to the end user to use system very easily.

4. It improves maintainability of the application.

5. It improves modularity of the application.

6. It improves easiness to use our system.

*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 must 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.

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

6. 1.8 Declare the default & static method with body in interface.

7. 1.9 we can define the private methods in interface also.

8. We cannot create the object of interface.

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

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

11. Interface does not have constructor.

**Syntax**

interface interface\_name {

}

Example-1

**public** **interface** Student {

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

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

}

Example-2

**public** **interface** Test {

**public** **abstract** **void** m1(); // allowed

**public** **void** m2(); // allowed

**abstract** **void** m3();// allowed

**void** m4(); // 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** Test {

**public** **abstract** **void** m1(); // allowed

}

**package** com.abstraction;

//this is the implementation class

**public** **class** Demo **implements** Test {

@Override

**public** **void** m1() {

System.***out***.println("Test-m1 method");

}

}

**package** com.abstraction;

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

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

Demo demo= **new** Demo();

demo.m1();

}

}

Output

Test-m1 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();

}

}

Output

Test-x1 method

Below is the list of possible scenarios 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 class 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 Transaction {

void withdrawAmt(int amtToWithdraw) ;

}

class TransactionalImpl implements Transaction {

void withdrawAmt(int amtToWithdraw) {

//logic of withdraw

//SBI DB connection and updating in their DB

}

}

Now how amazon will do this as below as-

Transaction ti = new TransactionalImpl ();

ti.withdrawAmt(500);

In this case, both applications can achieve their aims.

**Difference between Interface and Abstract class:**

|  |  |
| --- | --- |
| Interface | Abstract class |
| If we don't know anything about  implementation just we have requirement specification then we should go for interface. | If we are known about implementation  but not completely (partial  implementation) then we should go for  abstract class. |
| Every method present inside interface is always public and abstract whether we are declaring or not. (java 1.8 and 1.9 onwards some exception in this, already mentioned above) | Every method presents inside abstract  class need not be public and abstract. |
| We can't declare interface methods with the  modifiers private, protected, final, static,  synchronized, native, strictfp. | There are no restrictions on abstract  class method modifiers. |
| Every interface variable is always public static final whether we are declaring or not. | Every abstract class variable need not be public static final. |
| Every interface variable is always public static final we can't declare with the following modifiers.  Private, protected, transient, volatile. | There are no restrictions on abstract  class variable modifiers. |
| For the interface variables compulsory, we should perform initialization at the time of declaration otherwise we will get compile time error because they are final. | It is not required to perform initialization  for abstract class variables at the time of declaration. |
| Inside interface we can't have static and  instance blocks. | Inside abstract class we can have both  static and instance blocks. |
| Inside interface we don’t have constructor. | Inside abstract class we can have  constructor. |