Abstraction in java

**Abstraction-**

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

Or

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Realtime Examples of Abstraction in Java

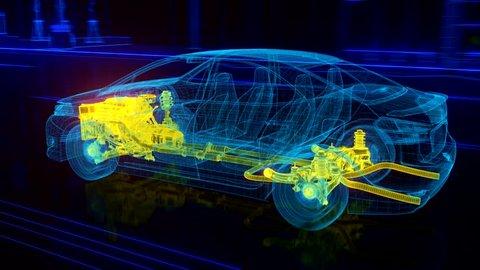
1. Let’s first take ATM machine as a real-time example. We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc. in our daily life.

But we don’t know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.



2. When you need to send an SMS from your mobile, you only type the text and send the message. But you don’t know the internal processing of the message delivery.

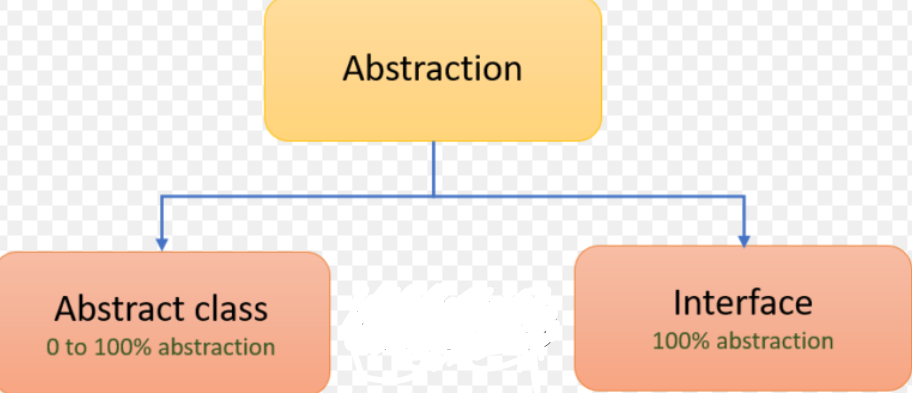
3. Car is also an example of abstraction where we don’t know the internal working of the car but we are using the car.



**How to achieve Abstraction in Java?**

There are two ways to achieve or implement abstraction in java program. They are as follows:

1. Abstract class (0 to 100%)
2. Interface (100%)



**Abstract Class in Java**

An abstract class in Java is a class, which is declared with an abstract keyword. It has the following features.

* 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 extend the abstract class and use those 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.
* Abstract class is not a pure abstraction in java.

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

**Syntax for abstract class**

<Access specifier>abstract class <Class\_name> {

// Class Body

}

Note: Abstract is a non-access modifier in java which is applicable for classes, interfaces, methods, and inner classes. It represents an incomplete class that depends on subclasses for its implementation. Creating subclass is compulsory for abstract class.

Note: An abstract concept is not applicable to variables.

**When to use Abstract class in Java?**

An abstract class can be used when we need to share the same method to all non-abstract subclasses with their own specific implementations.

**Abstract Method in Java**

A method that is declared with abstract modifier in an abstract class and has no implementation (means no body) is called abstract method in java. It does not contain any body.

Syntax for abstract method

<Access specifiers> <abstract><Returntype>method\_name (Arguments);

There are the following uses of abstract method in Java.

They are as follows:

1. An abstract method can be used when the same method has to perform different tasks depending on the object calling it.

2. A method can be used as abstract when you need to be overridden in its non-abstract subclasses.

**package** com.abstraction;

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

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

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

}

**How to implement that methods?**

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

**package** com.abstraction;

**public** **class** C **extends** Test {

@Override

**void** example() {

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

}

@Override

**void** demo() {

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

}

}

**package** com.abstraction;

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

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

C c= **new** C();

c.demo();

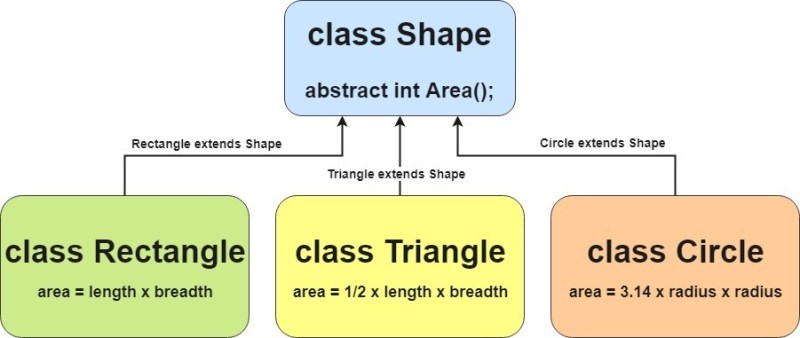
c.example();

}

}

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

Example :



Note :

If we don’t create an object of abstract class then why do we need constructor inside abstract class?

Because, the constructor can be used to initialize the instance variables present in the abstract class.

**Interface-**

1. It contain 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.
4. We can not 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. 1.8 Declare the default & static method with body in interface.
8. 1.9 we can define the private methods in interface also.
9. We cannot create the object of interface.
10. In interface, we can just define the method only but implemented that methods into implemented class.
11. Java supports multiple inheritance in the terms of interfaces but not classes.
12. Interface does not have constructor.

Syntax

interface interface\_name {

}

Example-

public interface Demo {

public abstract void m1();

public static final int a=5;

}

Example-

package com.abstra.interf;

 interface A {

public abstract void demo();

public abstract void example();

}

interface A{

   public abstract void demo ();  //allowed

   public void demo (); //allowed

   void demo (); //allowed

   abstract void demo (); //allowed

}

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

}

package com.abstra.interf;

 interface A {

public abstract void demo();

public abstract void example();

}

package com.abstra.interf;

public class Z implements A {

@Override

public void demo() {

System.out.println("this is demo method");

}

@Override

public void example() {

System.out.println("this is example method");

}

}

package com.abstra.interf;

public class TestMain {

public static void main(String[] args) {

Z z = new Z();

z.demo();

z.example();

//A a= new A();

}

}

Example-2

interface A {

}

interface B {

}

Interface C extends A, B {

}

This is allowed in java.

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 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 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.

Difference between Interface & Abstract class

|  |  |
| --- | --- |
| Interface | Abstract class |
| If we don’t know anything about implementation & we just have requirement specification then we should go for the interface. | If we are talking about implementation but not completely (partial  implementation) then we should go for the abstract class. |
| Inside interface every method is  always public & abstract hence it is  also considered as 100% pure  abstract class. | Inside Abstract class, every method  need not to be public & abstract. We  can have concrete method also in  abstract class. |
| 100% abstraction level can be  achieved. | 0 to 100% abstraction level can be  achieved. |
| We can’t declare the interface  method with following modifiers  Private, protected, final. Static. It  gives compile time error. | There are no restrictions on  modifiers in the case of abstract  class methods. |
| Every variable inside interface is  always public static final. | Variable present is abstract class  need not to be public static final. |
| We can’t declare the variables with  private & protected modifiers in  case of interface. | There is no restriction on variable  in case of abstract class. |
| For the interface variables compulsorily, we should perform initialization at the time of declaration otherwise it will give  compile time error. | Here abstract class variables don’t  require initialization at the time of  declaration. |
| Inside interface we can’t declare  instance blocks & static blocks. | We can declare instance blocks &  static blocks in case of abstract  class. |
| Inside interface we can’t declare  constructors. | Inside Abstract class we can declare constructor which will be executed  or called at the time of child object  creation. |