Abstraction-

It is the process of hiding the certain details and showing the important information to the end user called as "Abstraction". Or Hide internal implementation and just highlight the set of services, is called abstraction.
Example- By using ATM GUI screen bank people are highlighting the set of services what they are offering without highlighting internal implementation.

How to achieve the Abstraction in java?

There are two ways to achieve the abstraction in java.

1. Abstract class
2. Interface

Abstract class

- For any java class if we are not allow to create an object such type of class we have to declare with abstract modifier that is for abstract class instantiation is not possible.
- 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.
- Even though class doesn't contain any abstract methods still we can declare the class as abstract that is an abstract class can contain zero no. of abstract methods also.

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

Example-

```
public class Test {
    abstract void demo ();
}
```

Here, method is the abstract then class should be abstract only.

```
public abstract class Test {
      abstract void test();
      void demo(){
       //concrete methods here
       }
```

```
package com.abstraction;

public abstract class Test {

    abstract void example(); // abstract method

    abstract void demo();
}
```

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- If in subclass I don’t want to override the parent class abstract methods then make that subclass also as abstract.

Interface-

1. Any service requirement specification (SRS) or any contract between client and service provider or 100% pure abstract classes is considered as an interface.
2. Every method in interface is public abstract methods and public static final variables by default.
3. We must follow I to C design principle in java. It means every class must implement some interfaces.
4. 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. From 1.8 we can 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-

```
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.

Note:

We can’t create an object from abstract class and interface still abstract class has a constructor and interface doesn’t. The reason for this is in abstract class variables can be non final and hence to provide the initialization of this global variables abstract class will need constructors while creating object of child class.
But in interface, global variables are by default final and when we write a variable as final then you have to initialize that variable at the same time when we declare it so interface doesn’t require constructors for initialization of their global variables and hence interface doesn’t have constructor.

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 know 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 present 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 require 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.