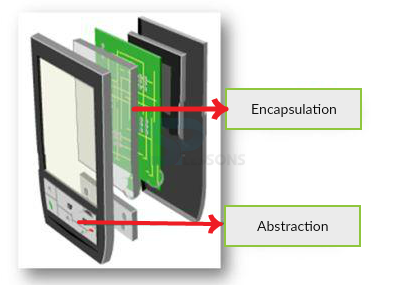
### Abstraction: Showing only essential/necessary features

### Related image



### Image result for java abstraction

### Abstraction in Java

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

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

### Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

Abstract class in Java

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

### Abstract class in Java

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

### Example abstract class

1. **abstract** **class** A{}

# Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

## Why use Java interface?

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

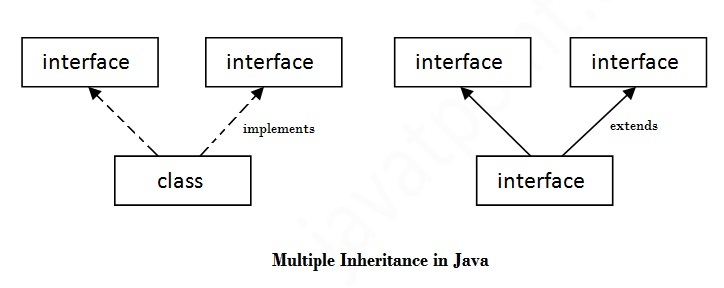
interface in java

#### Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.

## relationship between class and interfaceMultiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) **Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

**Examples Of Abstraction:**

Lets now understand abstraction concept using real life examples of different sounds created by animals. For example Cat does Meow and Lion Does Roar. We will display different sounds using Abstraction in JAVA.

**Step 1:** First create a new project in Eclipse and create a abstract class Sound. Below is the code of Sound.java

abstract class Sound {

//Non-abstract method i.e. method with implementation

    public void soundmessage(){

        System.out.print("Animal Sound");

    }

//Abstract method i.e. without body/implementation

    abstract void sound();

}

**Step 2:** Now create another class name Cat which extends Sound class. Here we will implement abstract sound() method for Cat. Below is the code of Cat.java

class Cat extends Sound{

void sound(){

soundmessage();

System.out.println(" of Cat: Meow");

}

}

**Step 3:** In the same way we are creating another Lion class which extends Sound class. Here also we will implement the sound() method but for Lion. Below is the code of Lion.java:

class Lion extends Sound {

void sound(){

soundmessage();

System.out.println(" of Lion: Roar");

}

}

**Step 4:** Now create AnimalSound main class. Here first we will create object of Cat & Lion class and then we will call sound() method on these objects. Below is the code of AnimalSound.java

public class AnimalSound {

public static void main(String[] args) {

Cat cat = new Cat();

cat.sound();

Lion lion = new Lion();

lion.sound();

}

}

**Output:**

Now run the program and you will see sounds of Cat and Lion printed.

Animal Sound of Cat: Meow

Animal Sound of Lion: Roar

**Conclusion:** So you can see how useful is Abstraction. We just define mixture of abstract and non-abstract methods in Abstract class and then implement abstract method in child class (i.e. sub class) according to requirement. In the end same method gives different result depending on Objects of which sub-class. Remember we can’t instantiate Abstract class as discussed earlier.

## When to use abstract class and when to use interfaces

Always remember that **choice between interface or abstract class** is not either/or scenario, where choosing anyone without proper analysis would yield same results. A choice must be made very intelligently after understanding the problem in hand. Lets try to put some intelligence here:

**Abstract classes let you define some behaviors; it makes them excellent candidates inside of application frameworks.**

Lets take example of HttpServlet. It is the main class you must inherit, if you are developing an web application using Servlets technology. As we know, each servlet has a definite life cycle phases, i.e. initialization, service and destruction. What if each servlet we create, we have to write same peace of code regarding initialization and destruction again and again. Surely, it will be a big pain.

JDK designers solved this by making HttpServlet abstract class. It has all basic code already written for initialization of a servlet and destruction of it also. You only need to override certain methods where you write your application processing related code, an that’s all. Make sense, right !!

Can you add above feature using interface? No, even if you can, design will be like a hell for most of innocent programmers.

Now, lets look at usage of interfaces.**An interface only provide contracts and it is responsibility of implementing classes to implement each and every single contract provided to it**.

An interface is a best fit for cases where you want to **define only the characteristics of class**, and you want force all implementing entities to implement those characteristics.

I would like to take example of Map interface in collections framework. It provides only rules, how a map should behave in practice. e.g. it should store key-value pair, value should be accessible using keys etc.  
These rules are in form of abstract methods in interface.

All implementing classes ( such as HashMap, HashTable, TreeMap or WeakHashMap) implements all methods differently and thus exhibit different features from rest.

Also, interfaces can be used in defining the separation of responsibilities. For example, HashMap implements 3 interfaces: Map, Serializable and Cloneable. Each interface defines separate responsibilities and thus an implementing class choose what it want to implement, and so will provide that much limited functionality.

# Interfaces In Java

* Interfaces are declared with keyword ‘**interface**‘ and interfaces are implemented by the class using ‘**implements**‘ keyword.

interface InterfaceClass

{

//Some Abstract methods

}

class AnyClass implements InterfaceClass

{

//Use 'implements' while implementing Interfaces

//Don't use 'extends'

}

* Interfaces should contain only abstract methods. Interfaces should not contain a single concrete method.

interface InterfaceClass

{

abstract void abstractMethodOne(); //abstract method

abstract void abstractMethodTwo(); //abstract method

void concreteMethod()

{

//Compile Time Error.

//Concrete Methods are not allowed in interface

}

}

* Interface can have two types of members.  **1) Fields     2) Abstract Methods.**

interface InterfaceClass

{

int i = 0; //Field

abstract void abstractMethodOne(); //abstract method

abstract void abstractMethodTwo(); //abstract method

}

* By default, Every field of an interface is public, static and final (we will discuss about final keyword Later). You can’t use any other modifiers other than these three for a field of an interface.

interface InterfaceClass

{

int i = 0;

//By default, field is public, static and final

//Following statements give compile time errors

private double d = 10;

protected long l = 15;

//You can't use any other modifiers other than public, static and final

}

* You can’t change the value of a field once they are initialized. Because they are static and final. Therefore, sometimes fields are called as **Constants**. (We will discuss this feature in detail while covering ‘final’ keyword)

interface InterfaceClass

{

int i = 0;

}

class AnyClass implements InterfaceClass

{

void methodOne()

{

//Following statement gives compile time error.

InterfaceClass.i = 10;

//final field can not be re-assigned

}

}

* By default, All methods of an interface are public and abstract.

interface InterfaceClass

{

void abstractMethodOne(); //Abstract method

void abstractMethodTwo(); //Abstract Method

//No need to use abstract keyword,

//by default methods are public and abstract

}

* Like classes, for every interface .class file will be generated after compilation.
* While implementing any interface methods inside a class, that method must be declared as public. Because, according to method overriding rule, you can’t reduce visibility of super class method. By default, every member of an interface is public and while implementing you should not reduce this visibility.

interface InterfaceClass

{

void methodOne();

}

class AnyClass implements InterfaceClass

{

void methodOne()

{

//It gives compile time error.

//Interface methods must be implemented as public

}

}

* By default, Interface itself is not public but by default interface itself is abstract like below,

abstract interface InterfaceClass

{

//By default interface is abstract

//No need to use abstract keyword

}

* SIB – Static Initialization Block and IIB – Instance Initialization Block are not allowed in interfaces.

interface InterfaceClassOne

{

static

{

//compile time error

//SIB's are not allowed in interfaces

}

{

//Here also compile time error.

//IIB's are not allowed in interfaces

}

void methodOne(); //abstract method

}

* As we all know that, any class in java can not extend more than one class. But class can implement more than one interfaces. This is how **multiple inheritance** is implemented in java.

interface InterfaceClassOne

{

void methodOne();

}

interface InterfaceClassTwo

{

void methodTwo();

}

class AnyClass implements InterfaceClassOne, InterfaceClassTwo

{

public void methodOne()

{

//method of first interface is implemented

}

//method of Second interface must also be implemented.

//Otherwise, you have to declare this class as abstract.

public void methodTwo()

{

//Now, method of Second interface is also implemented.

//No need to declare this class as abstract

}

}