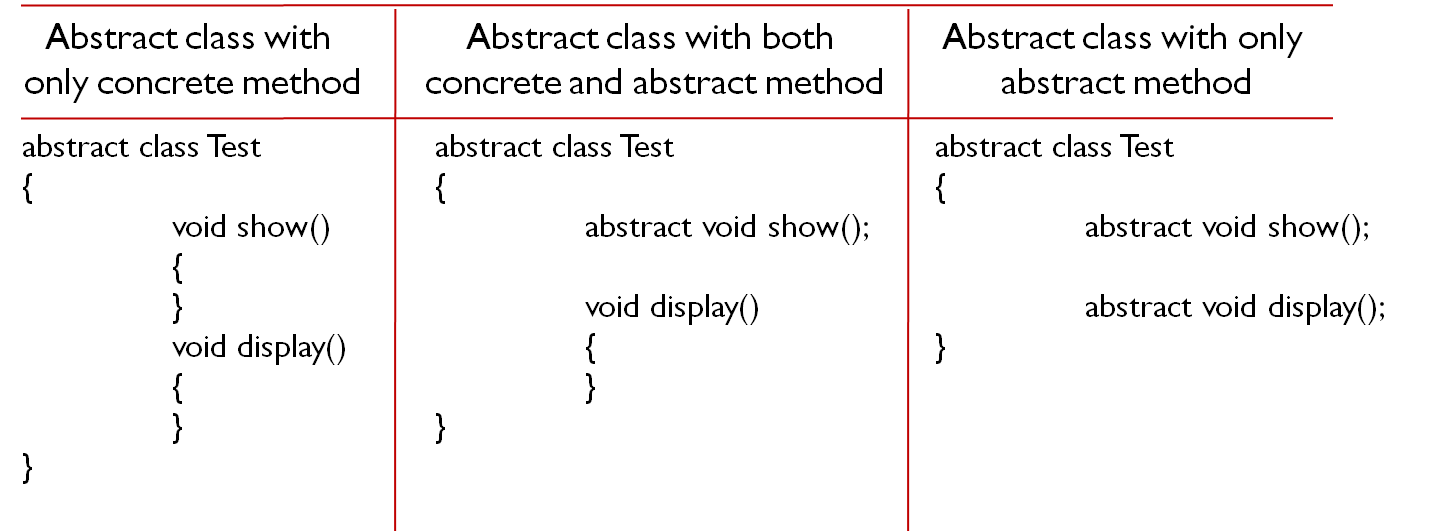


**Abstract Class**

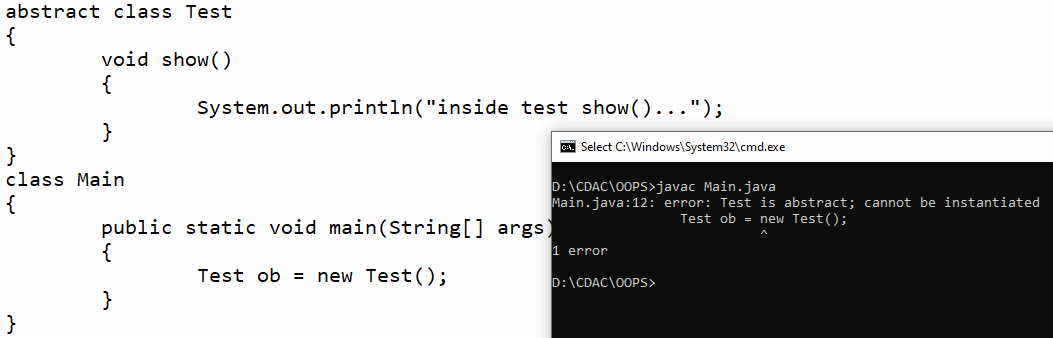
* Abstract class is a java class which contains both concrete and abstract method.
* abstract modifier is used to create abstract class.



**Abstract Class cannot be Instantiated**

* Abstract classes are partially implemented classes hence object creation is not possible.

**Example#1**



**Example#2**

abstract class Test{

void show(){

System.out.println("inside test show()...");

}

}

class Best extends Test{

}

class Main

{

public static void main(String[] args)

{

Best ob = new Best();

ob.show();

}

}

**Example#3**

abstract class Test{

void show(){

System.out.println("inside test show()...");

}

abstract void display();

}

class Best extends Test{

void display(){

System.out.println("inside best display()...");

}

}

class Main

{

public static void main(String[] args)

{

Best ob = new Best();

ob.show();

ob.display();

}

}

**Note:**

* If abstarct class contains abstract method then we need to override in it’s child class otherwise we have to make child class as abstract.

**Example#4**

abstract class Test{

abstract void show();

abstract void display();

}

class Best extends Test{

void show(){

System.out.println("inside show()...");

}

void display(){

System.out.println("inside display()...");

}

}

class Main

{

public static void main(String[] args)

{

Best ob = new Best();

ob.show();

ob.display();

}

}

**Partial Implementation**

* If the child class is unable to provide the implementation of all abstract methods of parent class then declare that class with abstract and take one more child class to complete the implementation of remaining abstract methods.

**Exaample#6**

abstract class Test

{

abstract void show1();

abstract void show2();

}

abstract class Best extends Test

{

void show1()

{

System.out.println("inside best show1()...");

}

}

class Nest extends Best

{

void show2()

{

System.out.println("inside best show2()...");

}

}

class Main

{

public static void main(String[] args)

{

Nest ob = new Nest();

ob.show1();

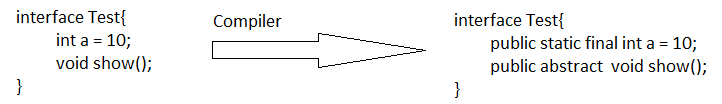
ob.show2();

}

}

**Interface**

* Interface is also one type of java class which contains only abstract method.
* interface keyword is used to create interface.
* Interface cannot be instantiated.
* Every interface methods are by default ***public abstract*** and variables are ***public static*** *and* ***final***.



**Example#1**

interface Test

{

int a = 10;

void show();

}

class Best implements Test{

public void show(){

System.out.println("inside show()...");

}

}

class Main

{

public static void main(String[] args)

{

Best ob = new Best();

ob.show();

}

}

**Nested interfaces**

* Declaring interface inside the class or abstract class or interface is called nested interface.

**Example#1**

class Test

{

interface intf1

{

void show();

}

}

class Best implements Test.intf1

{

public void show()

{

System.out.println("show method..");

}

}

class Main

{

public static void main(String[] args)

{

Best ob = new Best();

ob.show();

}

}

**Example#2**

interface Test

{

interface intf1

{

void show();

}

}

class Best implements Test.intf1

{

public void show()

{

System.out.println("show method..");

}

}

class Main

{

public static void main(String[] args)

{

Best ob = new Best();

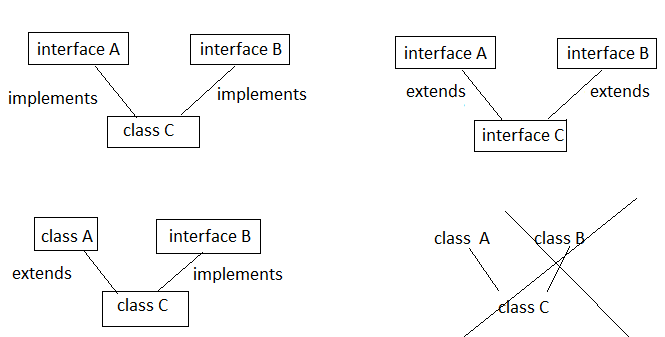
ob.show();

}

}

**Multiple Inheritance using Interface**

* An interface can extend more than one parent interface. The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

****

**Example#1**

interface A{

void show1();

}

interface B{

void show2();

}

class C implements A,B{

public void show1(){

System.out.println("show1 method...");

}

public void show2(){

System.out.println("show2 method...");

}

}

class Main{

public static void main(String args[]){

C ob = new C();

ob.show1();

ob.show2();

}

}

**Example#2**

interface A{

void show1();

}

interface B{

void show2();

}

interface C extends A, B{

}

class D implements C{

public void show1(){

System.out.println("show1 method...");

}

public void show2(){

System.out.println("show2 method...");

}

}

class Main{

public static void main(String args[]){

D ob = new D();

ob.show1();

ob.show2();

}

}

**Example#3**

class A{

void show1(){

System.out.println("show1 method...");

}

}

interface B{

void show2();

}

class C extends A implements B {

public void show2(){

System.out.println("show2 method...");

}

}

class Main{

public static void main(String args[]){

C ob = new C();

ob.show1();

ob.show2();

}

}

**Marker interface**

* An interface that has no members (methods and variables) is called as marker interface.
* Marker interface is used to inform the JVM that the classes implementing them will have some special behavior.
* User defined empty interfaces are not a marker interfaces, only predefined empty interfaces are marker interfaces.

**New interface features (Java 8)**

**Default Method**

Default method allows the developers to add new methods to the interfaces without affecting the classes that implements these interfaces.

**Static Method**

Static methods cannot be override in the classes that implements these interfaces.

**Example#1**

interface A{

void show1();

default void show2(){

System.out.println("show2...");

}

static void show3(){

System.out.println("show3...");

}

}

class B implements A{

public void show1(){

System.out.println("show1...");

}

}

class Main{

public static void main(String args[]){

B ob = new B();

ob.show1();

ob.show2();

A.show3();

}

}