**Abstract class in Java**

A class that is declared with abstract keyword, is known as **abstract class**. An abstract class is one which is containing some defined method and some undefined method. In java programming undefined methods are known as un-Implemented, or abstract method.

# Syntax

abstract class className

{

......

}

# Example

abstract class A

{

.....

}

If any class have any abstract method then that class become an abstract class.

# Example

class Vehicle

{

abstract void Bike();

}

Class Vehicle is become an abstract class because it have abstract Bike() method.

# Make class as abstract class

To make the class as abstract class, whose definition must be preceded by a abstract keyword.

# Example

abstract class Vehicle

{

......

}

# Abstract method

An abstract method is one which contains only declaration or prototype but it never contains body or definition. In order to make any undefined method as abstract whose declaration is must be predefined by abstract keyword.

# Syntax

abstract ReturnType methodName(List of formal parameter)

# Example

abstract void sum(); abstract void diff(int, int);

# Example of abstract class

abstract class Vehicle

{

abstract void speed(); // abstract method

}

class Bike extends Vehicle

{

void speed()

{

System.out.println("Speed limit is 40 km/hr..");

}

public static void main(String args[])

{

Vehicle obj = new Bike(); //indirect object creation obj.speed();

}

}

# Output

Speed limit is 40 km/hr..

# Create an Object of abstract class

An object of abstract class cannot be created directly, but it can be created indirectly. It means you can create an object of abstract derived class.

# Example

Vehicle obj = new Bike(); //indirect object creation

# Important Points about abstract class

* Abstract class of Java always contains common features.
* Every abstract class participates in inheritance.
* Abstract class definitions should not be made as final because abstract classes always participate in inheritance classes.
* An object of abstract class cannot be created directly, but it can be created indirectly.
* All the abstract classes of Java makes use of polymorphism along with method overriding for business logic development and makes use of dynamic binding for execution logic.

# Advantage of abstract class

* Less memory space for the application
* Less execution time
* More performance

# Example of abstract class having method body

abstract class Vehicle

{

abstract void speed(); void mileage()

{

System.out.println("Mileage is 60 km/ltr..");

}

}

class Bike extends Vehicle

{

void speed()

{

System.out.println("Speed limit is 40 km/hr..");

}

public static void main(String args[])

{

Vehicle obj = new Bike(); obj.speed(); obj.mileage();

}

}

# Output

Mileage is 60 km/ltr.. Speed limit is 40 km/hr..

# Example of abstract class having constructor, data member, methods

abstract class Vehicle

{

int limit=40; Vehicle()

{

System.out.println("constructor is invoked");

}

void getDetails()

{

System.out.println("it has two wheels");

}

abstract void run();

}

class Bike extends Vehicle

{

void run()

{

System.out.println("running safely..");

}

public static void main(String args[])

{

# Output

Vehicle obj = new Bike(); obj.run();

obj.getDetails();

System.out.println(obj.limit);

}

}

constructor is invoked running safely..

it has two wheels 40

**Interface in Java**

**Interface** is similar to class which is collection of public static final variables (constants) and abstract methods.

The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

# Use of Interface:

* It is used to achieve fully abstraction.
* By using Interface, you can achieve multiple inheritance in java.
* It can be used to achieve loose coupling.

# Properties of Interface

* It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
* Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
* Methods in an interface are implicitly public.
* All the data members of interface are implicitly public static final.

# How interface is different from class:

* You cannot instantiate an interface. It does not contain any constructors.
* All methods in an interface are abstract.
* Interface cannot contain instance fields. Interface only contains public static final variables.
* Interface is cannot extended by a class; it is implemented by a class.
* Interface can extend multiple interfaces. It means interface support multiple inheritance

# Behavior of compiler with Interface program

In the above image when we compile any interface program, by default compiler added public static final before any variable and public abstract before any method.

Because **Interface** is design for fulfill universal requirements and to achieve fully abstraction.

# Declaring Interfaces:

The **interface** keyword is used to declare an interface.

# Example

interface Person

{

datatype variablename=value;

//Any number of final, static fields

returntype methodname(list of parameters or no parameters)

//Any number of abstract method declarations

}

**Explanations**

In the above syntax **Interface** is a keyword interface name can be user defined name the default signature of variable is public static final and for method is public abstract. JVM will be added implicitly public static final before data members and public abstract before method.

# Example

public static final datatype variable name=value; > for data member

public abstract returntype methodname(parameters) > for method

**Implementing Interfaces:**

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

# Example

interface Person

{

void run();

}

class Employee implements Person

{

public void run()

{

System.out.println("Run fast");

}

}

# When we use abstract and when we use Interface

If we do not know about any things about implementation just we have requirement specification then we should be go for **Interface**

If we are talking about implementation but not completely (partially implemented) then we should be go for **abstract**

# Rules for implementation interface

* A class can implement more than one interface at a time.
* A class can extend only one class, but implement many interfaces.
* An interface can extend another interface, similarly to the way that a class can extend another class.

# Difference between Abstract class and Interface

|  |  |  |
| --- | --- | --- |
|  | **Abstract class** | **Interface** |
| 1 | It is collection of abstract method and  concrete methods. | It is collection of abstract method. |
| 2 | There properties can be reused  commonly in a specific application. | There properties commonly usable in any  application of java environment. |
| 3 | It does not support multiple  inheritance. | It support multiple inheritance. |
| 4 | Abstract class is preceded by abstract  keyword. | It is preceded by Interface keyword. |
| 5 | The default access specifier of abstract  class methods are default. | There default access specifier of interface  method are public. |
| 6 | These class properties can be reused in  other class using extend keyword. | These properties can be reused in any  other class using implements keyword. |
| 7 | Inside abstract class we can take  constructor. | Inside interface we can not take any  constructor. |
| 8 | For the abstract class there is no restriction like initialization of variable  at the time of variable declaration. | For the interface it should be compulsory to initialization of variable at the time of  variable declaration. |
| 9 | There are no any restriction for abstract class variable. | For the interface variable can not declare  variable as private, protected, transient, volatile. |
| 10 | There are no any restriction for abstract class method modifier that means we  can use any modifiers. | For the interface method can not declare method as strictfp, protected, static,  native, private, final, synchronized. |

**Example of Interface**

interface Person

{

void run(); // abstract method

}

class A implements Person

{

public void run()

{

System.out.println("Run fast");

}

public static void main(String args[])

{

A obj = new A();

# Output

Run fast

obj.run();

}

}

# Multiple Inheritance using interface Example

interface Developer

{

void disp();

}

interface Manager

{

void show();

}

# Output

class Employee implements Developer, Manager

{

public void disp()

{

System.out.println("Hello Good Morning");

}

public void show()

{

System.out.println("How are you ?");

}

public static void main(String args[])

{

Employee obj=new Employee(); obj.disp();

obj.show();

}

}

Hello Good Morning How are you ?