**Abstract class**

* There is a situation in which you want to define a super class that declares the structure of a given abstraction without providing a complete implementation of every method.
* That is sometimes you want to create a super class or parent class that only defines a generalized method and that method you want to share by all of its sub-classes or child classes, leaving it to be each sub class to implement method.
* Abstract class determines the nature of the methods that the **sub-classes must implement.**
* **Abstract class** means a class which is declared with the “**abstract**” keyword and abstract class may or may not include **abstract method**.
* **Abstract methods** means a method which are only declared but not defined (means abstract method does not have body).
* **Abstract class needs to be extended and its method implemented. Means the abstract methods which are there in super class must be override in sub-class otherwise it will give you error message.**
* We can’t create an instance of an abstract class because it is an incomplete class.

**Points to Remember**

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.



Eg:

abstract Shape

public abtract void draw();

Triangle

Square

Rectangle

In this example, Shape is the abstract class, and its implementation is provided by the Triangle Rectangle and Square classes.

|  |
| --- |
| **Syntax to create abstract class:** |
| <abstract> <class> <calss\_name)  {  //body  } |

|  |
| --- |
| **Syntax to create abstract method:** |
| <access-modifier> <abstract> <retruen-type> <method\_name>()  {  } |

abstract class Shape{

abstract void draw();

}

//In real scenario, implementation is provided by others i.e. unknown by end user

class Triangle extends Shape{

void draw(){System.out.println("drawing triangle");}

}

class Rectangle extends Shape{

void draw(){System.out.println("drawing rectangle");}

}

class Square extends Shape{

void draw(){System.out.println("drawing square");}

}

//In real scenario, method is called by programmer or user

class TestAbstraction1{

public static void main(String args[]){

Triangle t=new Triangle();

t.draw();

Rectangle r=new Rectangle();

r.draw();

}

}

Eg:

abstract class Bank{

abstract int getRateOfInterest();

}

class SBI extends Bank{

int getRateOfInterest(){return 7;}

}

class PNB extends Bank{

int getRateOfInterest(){return 8;}

}

class TestBank{

public static void main(String args[]){

Bank b;

b=new SBI();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

b=new PNB();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

}}

**Abstract class having constructor, data member and methods**

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor.

//Example of an abstract class that has abstract and non-abstract methods

**abstract class Bike**{

Bike(){System.out.println("bike is created");}

abstract void run();

void changeGear(){System.out.println("gear changed");}

}

//Creating a Child class which inherits Abstract class

**class Honda extends Bike**{

void run(){System.out.println("running safely..");}

}

//Creating a Test class which calls abstract and non-abstract methods

class Test{

public static void main(String args[]){

Bike obj = new Honda();

obj.run();

obj.changeGear();

}

}

**Output:**

bike is created

running safely..

gear changed

**Question:** Define an abstract class Staff with members’ names and addresses and abstract method i.e. accept and display. Define two sub-classes of this class – “FullTimeStaff” (department, salary) and “PartTimeStaff” (number-of-hours, rate-per hour). Appropriate method. Create n objects which could be of either FullTimeStaff or PartTimeStaff class by asking the user’s choice. Display details of all “FullTimeStaff” objects and all “PartTimeStaff” objects.

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  abstract class Staff{   String name,address;  public abstract void accept();  public abstract void display();   }  class FullTimeStaff extends Staff{   String department;   double salary;   public void accept() throws IOException{    System.out.println("Enter the name, address, department and salary: ");    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));    name=br.readLine();    address=br.readLine();    department=br.readLine();    salary=Double.parseDouble(br.readLine());   }   public void display(){    System.out.println("Name: "+name);    System.out.println("Address: "+address);    System.out.println("Department: "+department);    System.out.println("Salary: "+salary);    System.out.println("----------------------");   }  }  class PartTimeStaff extends Staff{   int hours, rate;   public void accept() throws IOException{    System.out.println("Enter the name, address, No of working hours and rate per hour: ");    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));    name=br.readLine();    address=br.readLine();    hours=Integer.parseInt(br.readLine());    rate=Integer.parseInt(br.readLine());   }   public void display(){    System.out.println("Name: "+name);    System.out.println("Address: "+address);    System.out.println("No of Working Hours: "+hours);    System.out.println("Rate per hour: "+rate);    System.out.println("----------------------");   }  }    public class Menu {   public static void main(String [] args) throws IOException{    int i;    System.out.println("Select Any One: ");    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));    System.out.println("1.Full Time Staff");    System.out.println("2.Part Time Satff");    int ch=Integer.parseInt(br.readLine());    switch(ch){    case 1:     System.out.println("Enter the number of Full Time Staff: ");     int n=Integer.parseInt(br.readLine());    FullTimeStaff [] l=new FullTimeStaff[n];     for(i=0;i<n;i++){      l[i]=new FullTimeStaff();      l[i].accept();     }     for(i=0;i<n;i++){      l[i].display();     }     break;    case 2:     System.out.println("Enter the number of Part Time Staff: ");     int m=Integer.parseInt(br.readLine());     PartTimeStaff [] h=new PartTimeStaff[m];     for(i=0;i<m;i++){       h[i]=new PartTimeStaff();       h[i].accept();      }     for(i=0;i<m;i++){      h[i].display();     }     break;       }    }  } |

|  |
| --- |
| **Q: created an abstract class CalculatorOperation with the following abstract method i.e add(), subs(), multi() and div().**  **Create a menu driven program in Calculator class such as 1 for Addition, 2 for Subtract, 3 for Multiplication, 4 for Division and 5 for Exit. In order to implement the class Calculator use abstract CalculatorOperation  class.** |

|  |
| --- |
| **Q. Write a JAVA program which has  i. An Abstract class for Stack Operations**  **ii. A Class that implements the abstract Stack class and creates a fixed length Stack.** |