|  |  |
| --- | --- |
| **Name** | Shubhan Singh |
| **UID no.** | 2022300118 |
| **Experiment No.** | 9-B |

|  |  |
| --- | --- |
| **PROBLEM STATEMENT :** | *Write a program to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B.*    *Create an abstract class 'Marks' with an abstract method 'getPercentage'.*    *It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students.*    *The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B.*    *Input to be taken from the user. Display the marks in highest order of student A and B.* |
| **THEORY:** | **Abstract Keyword:**  The `abstract` keyword in Java is used to declare classes and methods as abstract. It is a fundamental element of abstraction in object-oriented programming. Here's a brief note on how the `abstract` keyword is used:  1. Abstract Classes:  - An abstract class is declared using the `abstract` keyword. It serves as a blueprint for other classes and cannot be instantiated directly.  - Abstract classes can have both abstract and non-abstract methods.  - Abstract methods are declared without a body and are meant to be implemented by the subclasses.  - Abstract classes can also contain concrete methods with a body that can be inherited by the subclasses.  - Subclasses of an abstract class must either provide an implementation for all the abstract methods or be declared as abstract themselves.  - Abstract classes are useful when you want to define a common interface and provide some default implementations.  2. Abstract Methods:  - An abstract method is declared using the `abstract` keyword and does not have a method body.  - Abstract methods are meant to be overridden by the subclasses, providing their own implementation.  - Classes that have one or more abstract methods must be declared as abstract classes.  - Abstract methods define a contract that the subclasses must follow.  The `abstract` keyword allows you to create abstract classes and methods that serve as a foundation for creating specialized subclasses. It helps in achieving abstraction, encapsulation, and modularity in your Java programs. |
| **PROGRAM:** | import java.util.\*; //creating an abstract class abstract class Marks {  public abstract double getPercentage(); }  class A extends Marks {  private int m1,m2,m3;  public A(int *m1*, int *m2*, int *m3*) {  this.m1 = *m1*;  this.m2 = *m2*;  this.m3 = *m3*;  }   *@Override* public double getPercentage() {  return (m1+m2+m3)/3.0;  }   //method prints the marks of student A from highest to lowest  public void displayMarks() {  int[] m = {m1,m2,m3};  //sorting the array  Arrays.**sort**(m);  //using collection framework to reverse an array  Collections.**reverse**(Arrays.**asList**(m));  System.out.println("\nMarks of student A");  for (int j : m) {  System.out.printf("%d ", j);  }  } }  class B extends Marks {  private int m1,m2,m3,m4;   public B(int *m1*, int *m2*, int *m3*,int *m4*) {  this.m1 = *m1*;  this.m2 = *m2*;  this.m3 = *m3*;  this.m4 = *m4*;  }   *@Override* public double getPercentage() {  return (m1+m2+m3+m4)/4.0;  }   //method prints the marks of student A from highest to lowest  public void displayMarks() {  int[] m = {m1,m2,m3,m4};  Arrays.**sort**(m);  Collections.**reverse**(Arrays.**asList**(m));  System.out.println("\nMarks of student B");  for (int j : m) {  System.out.printf("%d ", j);  }  } }  public class percentmarks {  public static void main(String[] *args*) {  int[] m = new int[4];  Scanner sc = new Scanner(System.in);  //inputting the marks of Student A  System.out.println("Enter the marks of student A.");  for(int i=0;i<3;i++) {  System.out.printf("Subject %d: ",i+1);  m[i] = sc.nextInt();  }  A a = new A(m[0],m[1],m[2]);   //inputting the marks of Student B  System.out.println("\nEnter the marks of student B.");  for(int i=0;i<4;i++) {  System.out.printf("Subject %d: ",i+1);  m[i] = sc.nextInt();  }  B b = new B(m[0],m[1],m[2],m[3]);   //printing how much %age they got  System.out.printf("\nA secured %.2f%c\n",a.getPercentage(),'%');  System.out.printf("B secured %.2f%c\n",b.getPercentage(),'%');   //printing their marks in descending order  a.displayMarks();  b.displayMarks();  } } |
| **RESULT:** | |