|  |  |
| --- | --- |
| **Name** | Shubhan Singh |
| **UID no.** | 2022300118 |
| **Experiment No.** | 8-C |

|  |  |
| --- | --- |
| **PROBLEM STATEMENT :** | Aayush has deposited Rs. 10000 in SBI Bank,  Rs. 12500 in ICICI Bank, and Rs. 20000 in AXIS bank respectively for a particular month.  You need to print the money he will get by applying the rate of interest as per the bank and days.  Create a class 'Bank' with a method 'get\_rate\_of\_interest' which returns 2%.  Make three subclasses named  SBI\_Bank, ‘ICICI\_Bank’ and 'AXIS\_bank' with a method with the same name 'get\_rate\_of\_interest' which returns the rate of interest.  Also, give the final amount Ayush will get from that particular bank by applying the rate of interest and period. Use Calendar Class to count the number of days and amount he will get after maturity with the date of Maturity, if he deposits today. |
| **THEORY:** | Overriding in Java:  In Java, overridden functions play a crucial role in object-oriented programming, enabling a subclass to provide its own implementation of a method inherited from its superclass. When a subclass overrides a method, it means that it provides a specialized implementation of that method, tailored to its own needs.  Here are some key points about overridden functions in Java:   1. Inheritance Hierarchy: In Java, classes are organized in an inheritance hierarchy, where a subclass inherits properties and methods from its superclass. When a subclass overrides a method, it means that it replaces the superclass's implementation with its own implementation. 2. Method Signature: To override a method, the subclass must provide a method with the same name, return type, and parameter types as the method in the superclass. The method signature acts as a contract, ensuring that the overridden method has the same interface as the original method. 3. @Override Annotation: It is considered good practice to use the **@Override** annotation when overriding a method. This annotation informs the compiler that the method is intended to override a superclass method. It helps to catch any mistakes in method signature and provides better readability and maintainability. 4. Access Modifiers: When overriding a method, the access modifier of the overriding method should not be more restrictive than the access modifier of the overridden method. For example, if the overridden method is public, the overriding method can be public or protected, but not private. 5. Polymorphism: Overridden methods contribute to achieving polymorphism in Java. Polymorphism allows objects of different classes to be treated as objects of a common superclass, enabling them to be used interchangeably. Through polymorphism, you can invoke overridden methods on objects of the subclass, and the correct implementation will be executed based on the actual type of the object at runtime. 6. Covariant Return Types: Starting from Java 5, covariant return types are allowed when overriding methods. This means that an overriding method in a subclass can have a return type that is a subclass of the return type in the superclass. This feature helps in writing more expressive and flexible code. 7. Super Keyword: Within an overridden method, the **super** keyword can be used to invoke the superclass's implementation of the method. This is helpful when you want to extend the functionality of the superclass's method without completely replacing it. |
| **PROGRAM:** | import java.time.LocalDate; import java.time.Period; import java.util.Scanner;  class bank{  float rate\_of\_interest;  float amount\_in\_bank;  int no\_of\_days;   public void setRate\_of\_interest(float *rate\_of\_interest*) {  this.rate\_of\_interest = *rate\_of\_interest*;  }   public float getRate\_of\_interest() {  return 2;  }   public float getAmount\_in\_bank() {  return amount\_in\_bank;  }  public float get\_total\_amount(){  return (float) ((float)amount\_in\_bank\*Math.**pow**((1+rate\_of\_interest/100),((double) no\_of\_days /365)));  } } class SBI\_bank extends bank{  SBI\_bank(int *days*,float *amount*){  amount\_in\_bank=*amount*;  no\_of\_days=*days*;  set\_rate\_SBI();  }  void set\_rate\_SBI(){  if(no\_of\_days>=7 && no\_of\_days<=14){  rate\_of\_interest= 3.0F;  }  else if (no\_of\_days>14 && no\_of\_days<=30) {  rate\_of\_interest=3.0F;  }  else if (no\_of\_days>30 && no\_of\_days<=45) {  rate\_of\_interest=3.0F;  }  else if(no\_of\_days>45 && no\_of\_days<=90){  rate\_of\_interest=4.05F;  }  else if(no\_of\_days>90 && no\_of\_days<=120){  rate\_of\_interest=4.1F;  }  else{  rate\_of\_interest=4.1F;  }  }  } class Axis\_bank extends bank{  Axis\_bank(int *days*,float *amount*){  amount\_in\_bank=*amount*;  no\_of\_days=*days*;  set\_rate\_Axis();  }  void set\_rate\_Axis(){  if(no\_of\_days>=7 && no\_of\_days<=14){  rate\_of\_interest= 3.15F;  }  else if (no\_of\_days>14 && no\_of\_days<=30) {  rate\_of\_interest=3.15F;  }  else if (no\_of\_days>30 && no\_of\_days<=45) {  rate\_of\_interest=3.45F;  }  else if(no\_of\_days>45 && no\_of\_days<=90){  rate\_of\_interest=4.05F;  }  else if(no\_of\_days>90 && no\_of\_days<=120){  rate\_of\_interest=4.7F;  }  else{  rate\_of\_interest=5F;  }  } } class ICICI\_bank extends bank{  ICICI\_bank(int *days*,float *amount*){  amount\_in\_bank=*amount*;  no\_of\_days=*days*;  set\_rate\_ICICI();  }  void set\_rate\_ICICI(){  if(no\_of\_days>=7 && no\_of\_days<=14){  rate\_of\_interest= 3.1F;  }  else if (no\_of\_days>14 && no\_of\_days<=30) {  rate\_of\_interest=3.2F;  }  else if (no\_of\_days>30 && no\_of\_days<=45) {  rate\_of\_interest=3.5F;  }  else if(no\_of\_days>45 && no\_of\_days<=90){  rate\_of\_interest=4.5F;  }  else if(no\_of\_days>90 && no\_of\_days<=120){  rate\_of\_interest=4.7F;  }  else{  rate\_of\_interest=4.9F;  }  } } public class Bank\_interest {  public static void main(String[] *args*) {  Scanner sc = new Scanner(System.in);  //assigning today's date to d1  LocalDate d1 = LocalDate.**now**();   //asking the user the amount he has deposited in different banks  System.out.println("Enter the amount you have deposited in");  System.out.print("SBI Bank: ");  int sbi = sc.nextInt();  System.out.print("ICICI Bank: ");  int icici = sc.nextInt();  System.out.print("AXIS Bank: ");  int axis = sc.nextInt();   //asking the user when does he want to end his maturity  System.out.println("Enter the date of maturity.");  System.out.print("Year: ");  int year = sc.nextInt();  System.out.print("Month: ");  int month = sc.nextInt();  System.out.print("Day: ");  int day = sc.nextInt();  //assigning the date of maturity to d2  LocalDate d2 = LocalDate.**of**(year,month,day);   //calculating the gap between d1 and d2 in terms of years, months and days  Period gap = Period.**between**(d1,d2);  //converting duration into days  int duration = gap.getYears()\*365+gap.getMonths()\*30+gap.getDays();  System.out.print("\nStart date: " + d1 + "\nEnd date: " + d2);  System.out.printf("\nThe duration for which amount is kept in the bank is %d days.\n",duration);  SBI\_bank sbibank =new SBI\_bank(duration,sbi);  Axis\_bank axisBank=new Axis\_bank(duration,axis);  ICICI\_bank iciciBank=new ICICI\_bank(duration,icici);  System.out.printf("The amount received after maturity from SBI is %.2f\nThe amount received from Axis bank is %.2f\n" +  "The amount received from ICICI bank after maturity is %.2f\n",sbibank.get\_total\_amount(),axisBank.get\_total\_amount(),iciciBank.get\_total\_amount());  } } |
| **RESULT:** | |