Name	Lekh Sanatan Nayak
UID no.	2023800068
Experiment No.	8

AIM:	Implement a Program to demonstrate Abstraction using interfaces
Program 1	
PROBLEM STATEMENT :	Consider two interfaces, Volume and SurfaceArea with methods getVolume() and getSurfaceArea() respectively. Class 'Cylinder' implements both Volume and SurfaceArea and implements their methods. The class contains their required dimensions as data members. Write a program which inputs its dimensions and prints its volume and surface area.

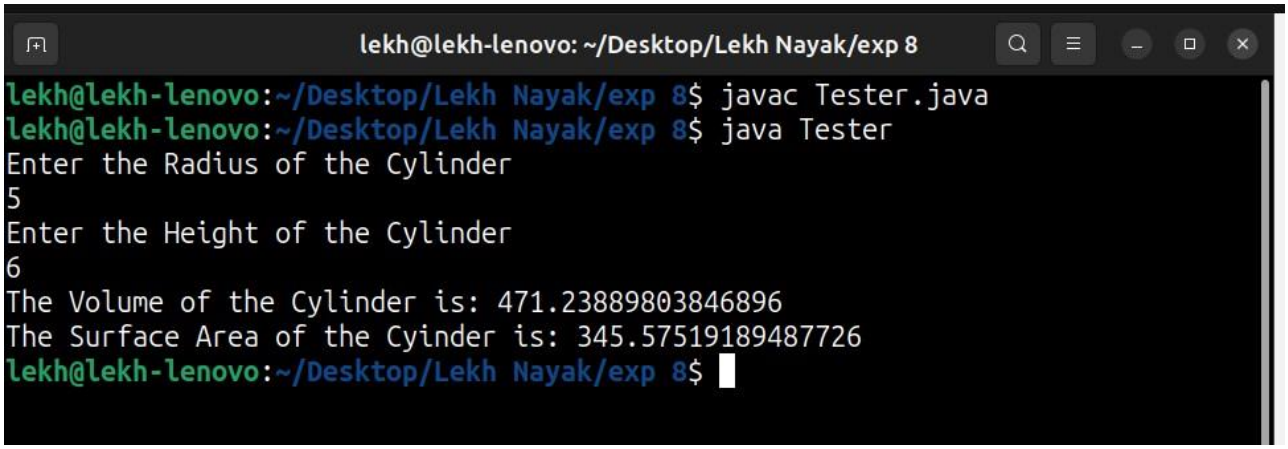
PROGRAM:	import java.util.*;
	
	//creating interface volume
	interface Volume {
	  //declaring abstract method getVolume()
	  double getVolume();
	}
	
	//creating interface SurfaceArea
	interface SurfaceArea {
	  //declaring abstract method getSurfaceArea()
	  double getSurfaceArea();
	}
	
	class Cylinder implements Volume, SurfaceArea {
	  private double radius;
	  private double height;
	
	  //Declaring constructor
	  public Cylinder(double radius, double height) {
	  this.radius = radius;
	  this.height = height;
	  }

```
   //inheriting the method getVolume()
   public double getVolume() {
      return Math.PI * radius * radius * height;
   }

   //inheriting the method getVolume()
   public double getSurfaceArea() {
      return 2 * Math.PI * radius * (radius + height);
   }
}

//creating class Tester containing the Main method
public class Tester {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter the Radius of the Cylinder");
      double radius = sc.nextDouble();

      System.out.println("Enter the Height of the Cylinder");
      double height = sc.nextDouble();

      //Creating the object for Cylinder Class
      Cylinder cy = new Cylinder(radius, height);

      System.out.println("The Volume of the Cylinder is: " +
cy.getVolume());
      System.out.println("The Surface Area of the Cyinder is: " +
cy.getSurfaceArea());
   }
}
```

/*Consider two interfaces, Volume and SurfaceArea with methods getVolume() and getSurfaceArea() respectively.
Class â€˜Cylinderâ€™ implements both Volume and SurfaceArea and implements their methods. The class contains their required dimensions as data members.
Write a program which inputs its dimensions and prints its volume and

	surface area.*/

RESULT:



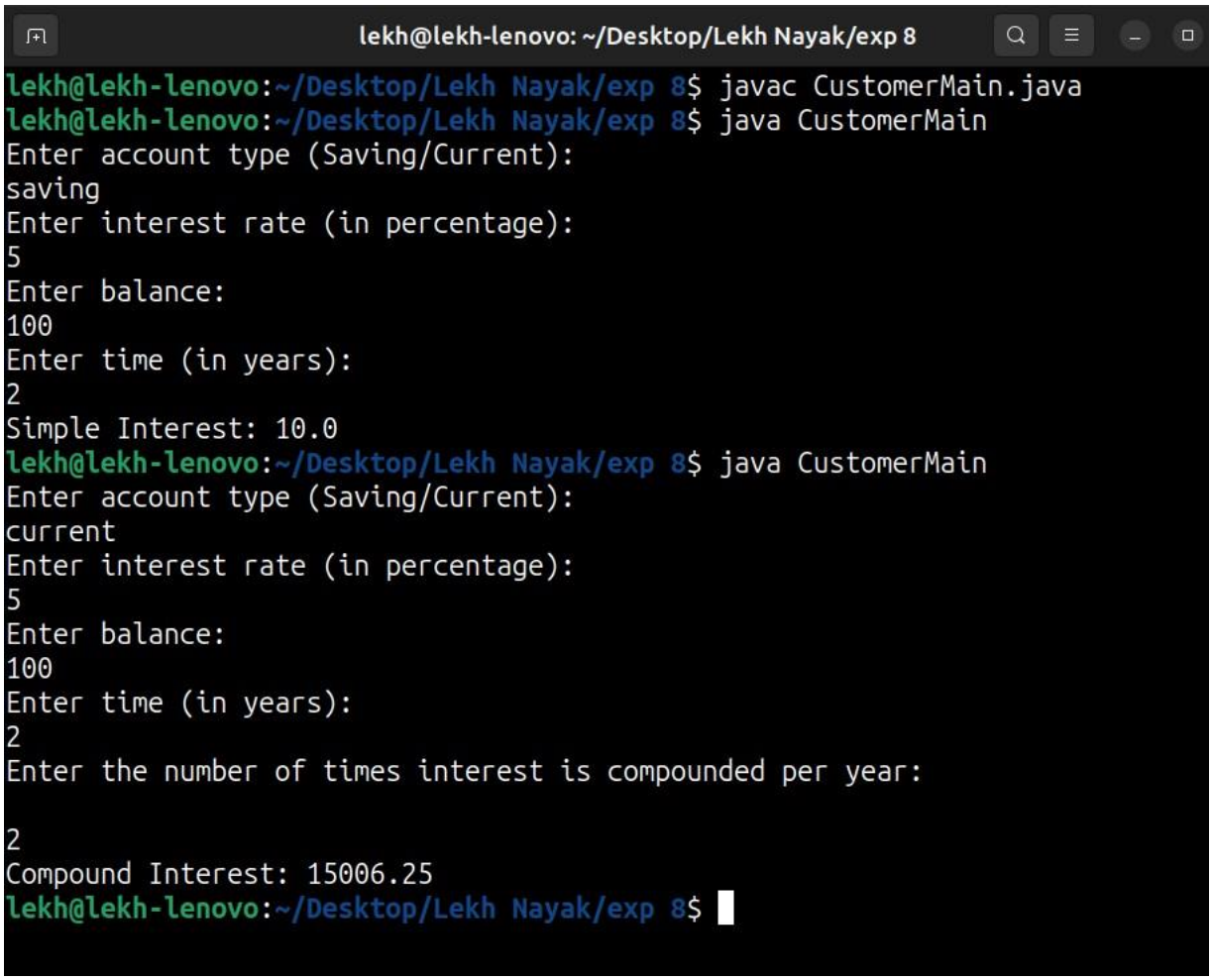
Program 2	
PROBLEM STATEMENT :	A banking system has two interfaces SavingAccount and CurrentAccount. The SavingAccount account has method getSimpleInterest() and CurrentAccont has method getCompoudInterest(). Both these interfaces are implemented by class Customer. Customer have data members: account type, interest rate and balance. The class then calculates interest on balance and prints it.

PROGRAM:	import java.util.*;
	
	interface SavingAccount {
	  double getSimpleInterest();
	}
	
	interface CurrentAccount {
	  double getCompoundInterest();

```
}

class Customer implements SavingAccount, CurrentAccount {
   private String AccType;
   private double InterestRate;
   private double balance;
   private double time;
   private int n;

   public Customer(String AccType, double InterestRate, double balance,
double time) {
      this.AccType = AccType;
      this.InterestRate =  InterestRate;
      this.balance = balance;
      this.time = time;
   }

   public Customer(String AccType, double InterestRate, double balance,
double time, int n) {
      this.AccType = AccType;
      this.InterestRate =  InterestRate;
      this.balance = balance;
      this.time = time;
      this.n = n;
   }

   public double getSimpleInterest() {
      double amount = (balance * InterestRate * time)/100 ;
      return amount ;
   }

   public double getCompoundInterest() {
      double amount = balance * Math.pow(1 + (InterestRate / n), n * time);
      return amount ;
   }
}

public class CustomerMain {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
```

```
        System.out.println("Enter account type (Saving/Current): ");
        String accType = scanner.next();

        System.out.println("Enter interest rate (in percentage): ");
        double interestRate = scanner.nextDouble();

        System.out.println("Enter balance: ");
        double balance = scanner.nextDouble();

        System.out.println("Enter time (in years): ");
        double time = scanner.nextDouble();

        Customer customer1 = new Customer(accType, interestRate, balance,
time);

        if(accType.equalsIgnoreCase("saving")) {
            System.out.println("Simple Interest: " +
customer1.getSimpleInterest());
        }
        else if (accType.equalsIgnoreCase("current")) {
            System.out.println("Enter the number of times interest is
compounded per year: ");
            int n = scanner.nextInt();
            Customer customer2 = new Customer(accType, interestRate,
balance, time, n);
            System.out.println("Compound Interest: " +
customer2.getCompoundInterest());
        }
        scanner.close();
    }
}
```

RESULT:



Program 3	
PROBLEM STATEMENT:	A fast food centre has two interfaces EatItHere and TakeAway. They have methods like deliverOrder() and dispatchOrder(). the class Consumer inherits both of them. If Consumer orders a takeaway he will be charged a convenience fee of Rs. 45/- or 5% of order price whichever is more. If he chooses to eat in a restaurant then he will be charged Rs. 500/- service charge or 10% of order price whichever is lower. The class consumer has data members as order number and order price.

PROGRAM:	import java.util.*;
	
	interface EatItHere {
	  void deliverOrder(int orderNumber, double orderPrice);
	}

```
interface TakeAway {
   void dispatchOrder(int orderNumber, double orderPrice);
}

class Consumer implements EatItHere, TakeAway {
   private int orderNumber;
   private double orderPrice;

   public Consumer(int orderNumber, double orderPrice) {
      this.orderNumber = orderNumber;
      this.orderPrice = orderPrice;
   }

   public void deliverOrder(int orderNumber, double orderPrice) {
      double ServiceCharge = Math.min(500, orderPrice * 0.10);
      double totalAmount = orderPrice + ServiceCharge;
      System.out.println("Order Number: " + orderNumber);
      System.out.println("Total Amount (Eat It Here): " + totalAmount);
   }

   public void dispatchOrder(int orderNumber, double orderPrice) {
      double ServiceCharge = Math.max(45, orderPrice * 0.05);
      double totalAmount = orderPrice + ServiceCharge;
      System.out.println("Order Number: " + orderNumber);
      System.out.println("Total Amount (Take Away): " + totalAmount);
   }
}

public class ConsumerMain {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter order number: ");
      int orderNumber = sc.nextInt();

      System.out.println("Enter order price: ");
      double orderPrice = sc.nextDouble();

      Consumer obj1 = new Consumer(orderNumber, orderPrice);
      int option;
```

```
        do {
            System.out.println("Choose an option : ");
            System.out.println("1. Eat it Here");
            System.out.println("2. Take Away");
            System.out.println("0. EXIT THE PROGRAM");
            option = sc.nextInt();

            switch (option) {
                case 1:
                    obj1.deliverOrder(orderNumber, orderPrice);
                    break;
                case 2:
                    obj1.dispatchOrder(orderNumber, orderPrice);
                    break;
                case 0:
                    System.exit(0);
                default:
                    System.out.println("Invalid option.");
            }
        }
        while (option < 3);
        sc.close();
    }
}
```

RESULT:

```
lekh@lekh-lenovo:~/Desktop$ javac SortMain.java
lekh@lekh-lenovo:~/Desktop$ java SortMain
Enter the number of elements in the array: 5
Enter the elements of the array:
 9 8 7 5 6
Select sorting algorithm:
1. Bubble Sort
2. Selection Sort
1
Sorted array:
5 6 7 8 9
lekh@lekh-lenovo:~/Desktop$ java SortMain
Enter the number of elements in the array: 5
Enter the elements of the array:
 9 8 7 5 6
Select sorting algorithm:
1. Bubble Sort
2. Selection Sort
2
Sorted array:
5 6 7 8 9
lekh@lekh-lenovo:~/Desktop$
```

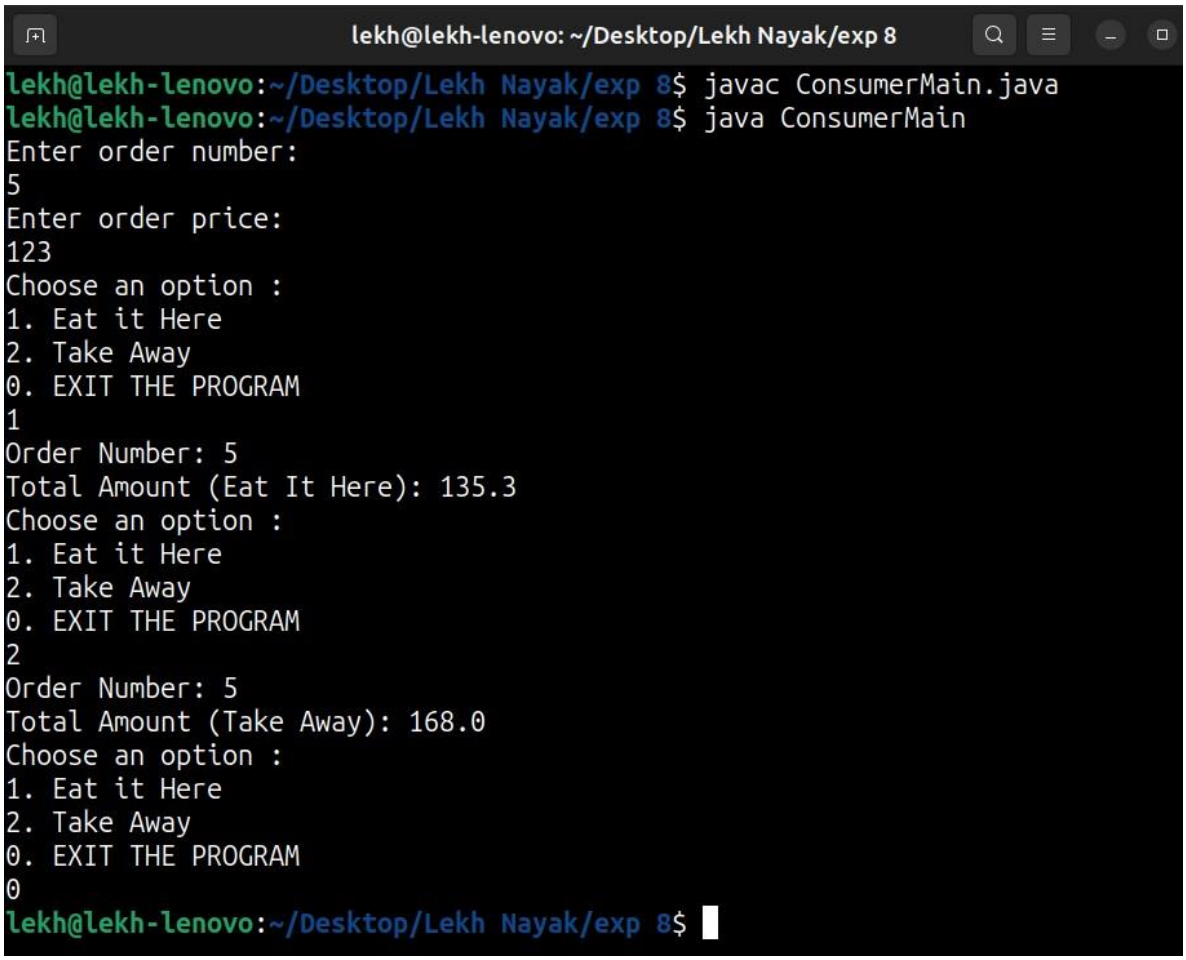
Program 4	
PROBLEM STATEMENT:	Write a Java program to create an interface Sortable with a method sort() that sorts an array of integers in ascending order. Create two classes BubbleSort and SelectionSort that implement the Sortable interface and provide their own implementations of the sort() method.

PROGRAM:	import java.util.*;
	
	interface SavingAccount {
	  double getSimpleInterest();
	}

```
interface CurrentAccount {
   double getCompoundInterest();
}

class Customer implements SavingAccount, CurrentAccount {
   private String AccType;
   private double InterestRate;
   private double balance;
   private double time;
   private int n;

   public Customer(String AccType, double InterestRate, double balance,
double time) {
      this.AccType = AccType;
      this.InterestRate =  InterestRate;
      this.balance = balance;
      this.time = time;
   }

   public Customer(String AccType, double InterestRate, double balance,
double time, int n) {
      this.AccType = AccType;
      this.InterestRate =  InterestRate;
      this.balance = balance;
      this.time = time;
      this.n = n;
   }

   public double getSimpleInterest() {
      double amount = (balance * InterestRate * time)/100 ;
      return amount ;
   }

   public double getCompoundInterest() {
      double amount = balance * Math.pow(1 + (InterestRate / n), n * time);
      return amount ;
   }
}
```

```
public class CustomerMain {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.println("Enter account type (Saving/Current): ");
      String accType = scanner.next();

      System.out.println("Enter interest rate (in percentage): ");
      double interestRate = scanner.nextDouble();

      System.out.println("Enter balance: ");
      double balance = scanner.nextDouble();

      System.out.println("Enter time (in years): ");
      double time = scanner.nextDouble();

      Customer customer1 = new Customer(accType, interestRate, balance,
time);

      if(accType.equalsIgnoreCase("saving")) {
         System.out.println("Simple Interest: " +
customer1.getSimpleInterest());
      }
      else if (accType.equalsIgnoreCase("current")) {
         System.out.println("Enter the number of times interest is
compounded per year: ");
         int n = scanner.nextInt();
         Customer customer2 = new Customer(accType, interestRate,
balance, time, n);
         System.out.println("Compound Interest: " +
customer2.getCompoundInterest());
      }
      scanner.close();
   }
}
```

RESULT:



Program 5	
PROBLEM STATEMENT:	Write a class that implements the CharSequence interface. Your implementation should return the string backwards. Select one of the sentences to use as the data. Write a small main method to test your class.

PROGRAM:	public class BackwardsCharSequence implements CharSequence {
	private String originalString;
	
	public BackwardsCharSequence(String originalString) {
	this.originalString = originalString;
	}
	
	public int length() {
	return originalString.length();
	}

```
    public char charAt(int index) {
        return originalString.charAt(length() - 1 - index);
    }

    public CharSequence subSequence(int start, int end) {
        return new BackwardsCharSequence(originalString.substring(length()
- end, length() - start));
    }

    public String toString() {
        StringBuilder reversed = new StringBuilder(originalString);
        return reversed.reverse().toString();
    }

    public static void main(String[] args) {
        String sentence = "Implementing CharSequence interface backwards.";
        BackwardsCharSequence backwardsSequence = new
BackwardsCharSequence(sentence);

        // Testing length method
        System.out.println("Length: " + backwardsSequence.length());

        // Testing charAt method
        System.out.println("Character at index 0: " +
backwardsSequence.charAt(0));
        System.out.println("Character at index 10: " +
backwardsSequence.charAt(10));
        System.out.println("Character at index 20: " +
backwardsSequence.charAt(20));

        // Testing subSequence method
        System.out.println("Subsequence from index 5 to 15: " +
backwardsSequence.subSequence(5, 15));

        // Testing toString method
        System.out.println("Reversed string: " + backwardsSequence);
    }
}
```

RESULT:

```
Length: 46
Character at index 0: .
Character at index 10:
Character at index 20:
Subsequence from index 5 to 15: wkcab ecaf
Reversed string: .sdrawkcab ecafretni ecneuqeSrahC gnitnemelpmI


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION:	Implementing a program utilizing interfaces makes it evident that abstraction effectively encapsulates behavior, allowing for modular, flexible, and simplified software design.