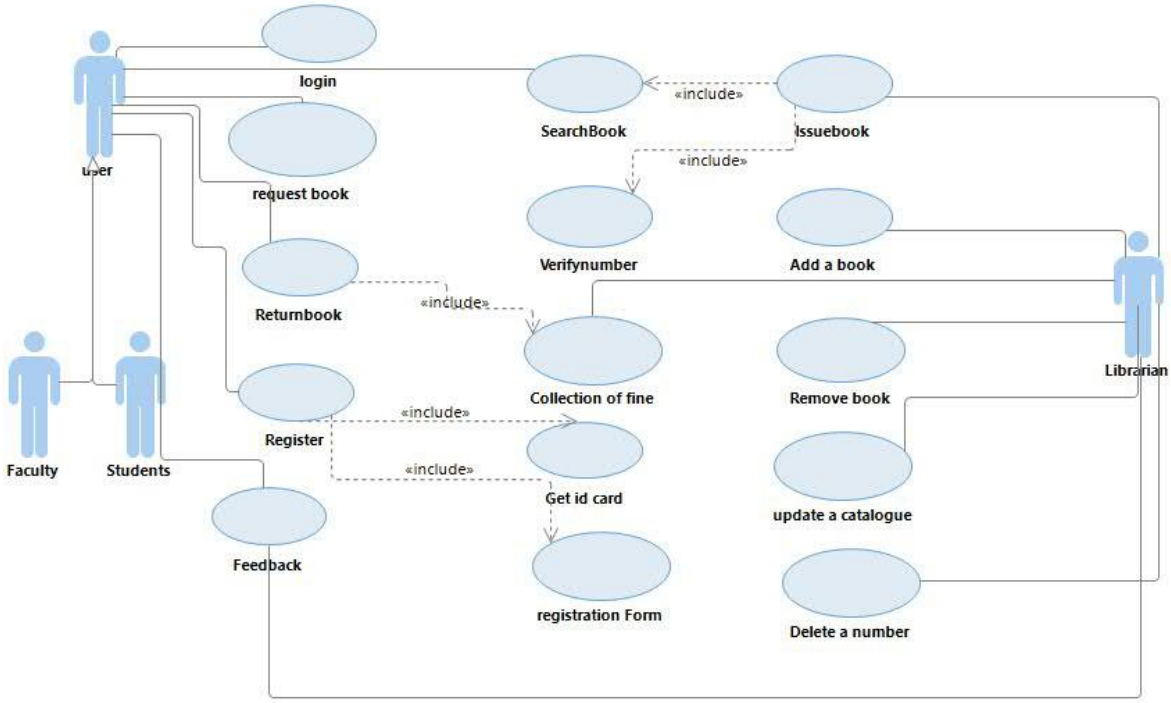
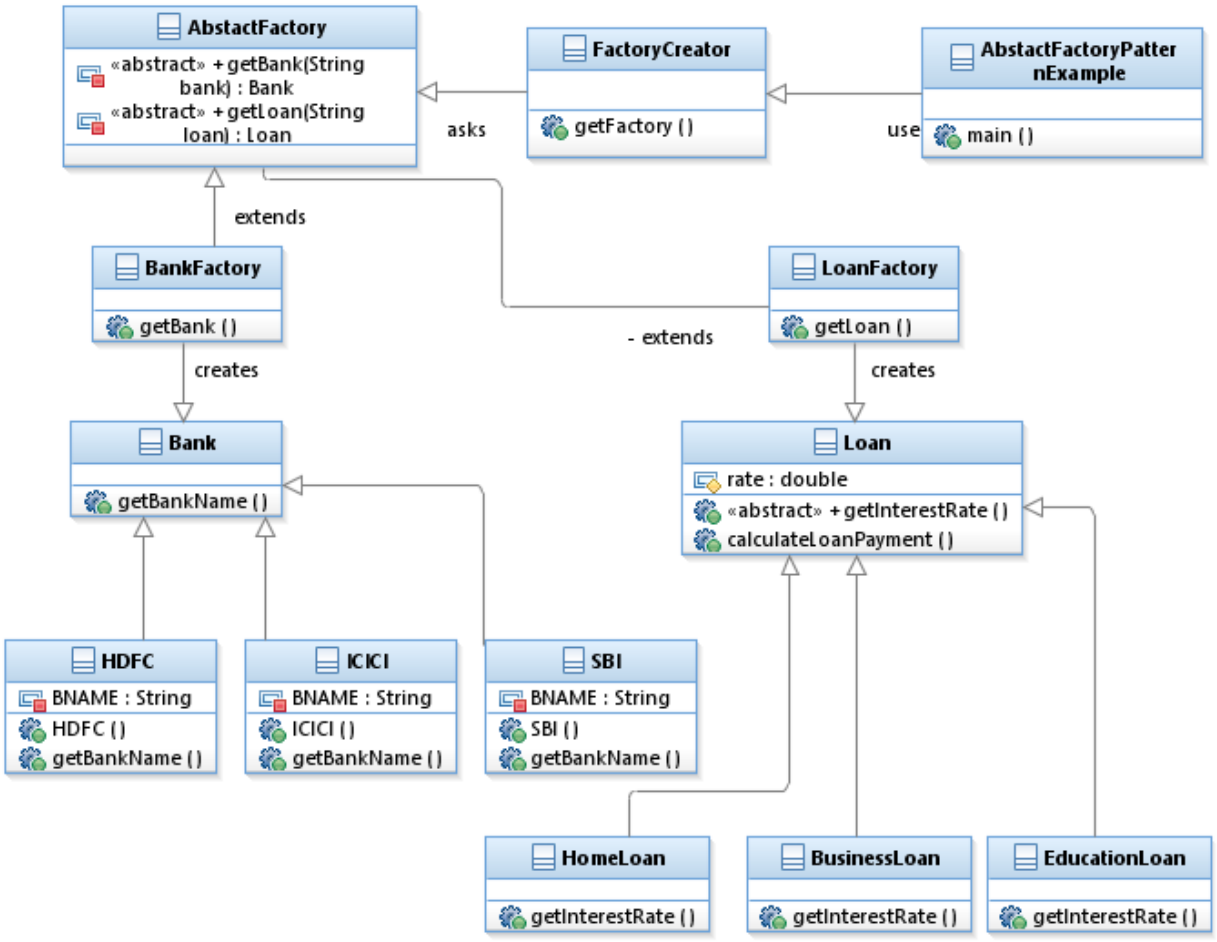
DESIGN PATTERNS LAB PROGRAMS

1.USECASE DIAGRAM:librarian



2.Abstract Factory




Program:

```
import java.io.*;

interface Bank{

      String getBankName();

}

class HDFC implements Bank{

       private final String BNAME;

       public HDFC(){

            BNAME="HDFC BANK";

      }
```

```
        public String getBankName() {
                return BNAME;
        }
}
class ICICI implements Bank{
        private final String BNAME;
        ICICI(){
                BNAME="ICICI BANK";
        }
        public String getBankName() {
                return BNAME;
        }
}
class SBI implements Bank{
      private final String BNAME;
      public SBI(){
                BNAME="SBI BANK";
        }
        public String getBankName(){
                return BNAME;
        }
}
abstract class Loan{
   protected double rate;
   abstract void getInterestRate(double rate);
```

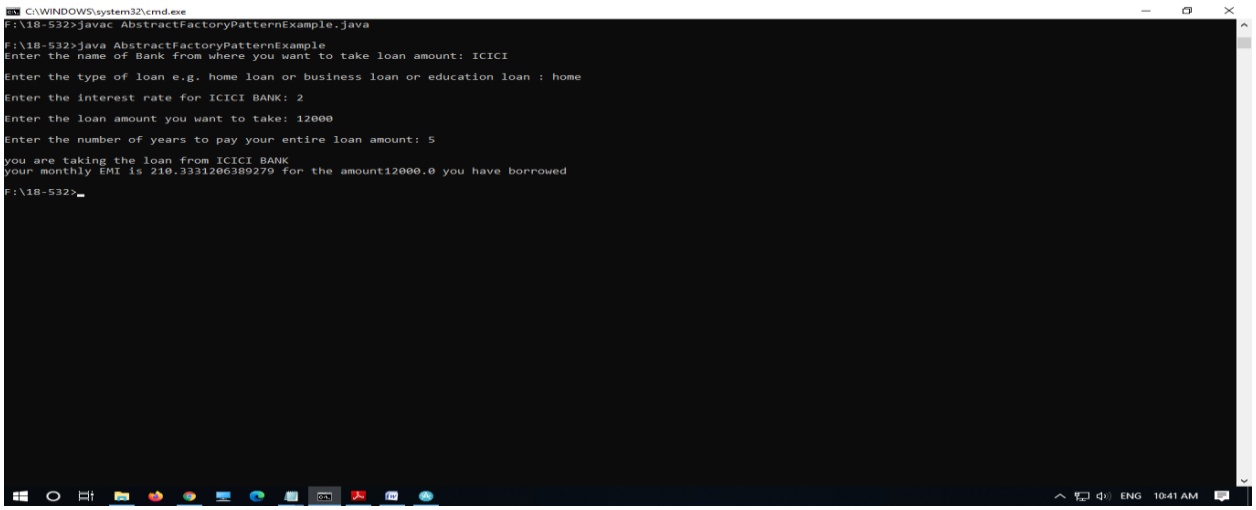
```
  public void calculateLoanPayment(double loanamount, int years)
  {
        double EMI;
        int n;

        n=years*12;
        rate=rate/1200;
        EMI=((rate*Math.pow((1+rate),n))/((Math.pow((1+rate),n))-1))*loanamount;
System.out.println("your monthly EMI is "+ EMI +" for the amount"+loanamount+" you have
borrowed");
 }
}// end of the Loan abstract class.
class HomeLoan extends Loan{
    public void getInterestRate(double r){
        rate=r;
   }
}//End of the HomeLoan class.
class BussinessLoan extends Loan{
   public void getInterestRate(double r){
         rate=r;
    }

}//End of the BusssinessLoan class.
class EducationLoan extends Loan{
    public void getInterestRate(double r){
      rate=r;
```

```
 }
}//End of the EducationLoan class.
abstract class AbstractFactory{
  public abstract Bank getBank(String bank);
  public abstract Loan getLoan(String loan);
}
class BankFactory extends AbstractFactory{
   public Bank getBank(String bank){
      if(bank == null){
         return null;
      }
      if(bank.equalsIgnoreCase("HDFC")){
         return new HDFC();
      } else if(bank.equalsIgnoreCase("ICICI")){
         return new ICICI();
      } else if(bank.equalsIgnoreCase("SBI")){
         return new SBI();
      }
      return null;
   }
  public Loan getLoan(String loan) {
      return null;
   }
}//End of the BankFactory class.
```

```
class LoanFactory extends AbstractFactory{
          public Bank getBank(String bank){
              return null;
          }

     public Loan getLoan(String loan){
      if(loan == null){
        return null;
      }
      if(loan.equalsIgnoreCase("Home")){
        return new HomeLoan();
      } else if(loan.equalsIgnoreCase("Business")){
        return new BussinessLoan();
      } else if(loan.equalsIgnoreCase("Education")){
        return new EducationLoan();
      }
      return null;
   }

}
class FactoryCreator {
     public static AbstractFactory getFactory(String choice){
      if(choice.equalsIgnoreCase("Bank")){
        return new BankFactory();
      } else if(choice.equalsIgnoreCase("Loan")){
```

```
        return new LoanFactory();
      }
      return null;
   }
}//End of the FactoryCreator.

public class AbstractFactoryPatternExample {
      public static void main(String args[])throws IOException {

      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

      System.out.print("Enter the name of Bank from where you want to take loan amount: ");
      String bankName=br.readLine();

System.out.print("\n");
System.out.print("Enter the type of loan e.g. home loan or business loan or education loan : ");

String loanName=br.readLine();
AbstractFactory bankFactory = FactoryCreator.getFactory("Bank");
Bank b=bankFactory.getBank(bankName);

System.out.print("\n");
System.out.print("Enter the interest rate for "+b.getBankName()+ ": ");

double rate=Double.parseDouble(br.readLine());
```

```
System.out.print("\n");

System.out.print("Enter the loan amount you want to take: ");


double loanAmount=Double.parseDouble(br.readLine());

System.out.print("\n");

System.out.print("Enter the number of years to pay your entire loan amount: ");

int years=Integer.parseInt(br.readLine());


System.out.print("\n");

System.out.println("you are taking the loan from "+ b.getBankName());


AbstractFactory loanFactory = FactoryCreator.getFactory("Loan");

        Loan l=loanFactory.getLoan(loanName);

        l.getInterestRate(rate);

        l.calculateLoanPayment(loanAmount,years);

 }

}//End of the  AbstractFactoryPatternExample
```



3.Adapter

Program:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
 interface CreditCard {
   public void giveBankDetails();
   public String getCreditCard();
}
 class BankDetails{
   private String bankName;
   private String accHolderName;
   private long accNumber;

   public String getBankName() {
      return bankName;
   }
   public void setBankName(String bankName) {
this.bankName = bankName;
   }
   public String getAccHolderName() {
      return accHolderName;
   }
   public void setAccHolderName(String accHolderName) {
this.accHolderName = accHolderName;
   }
```

```
    public long getAccNumber() {
        return accNumber;
    }
    public void setAccNumber(long accNumber) {
this.accNumber = accNumber;
    }
}
 class BankCustomer extends BankDetails implements CreditCard {
 public void giveBankDetails(){
try{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.print("Enter the account holder name :");
   String customername=br.readLine();
System.out.print("\n");
System.out.print("Enter the account number:");
   long accno=Long.parseLong(br.readLine());
System.out.print("\n");
System.out.print("Enter the bank name :");
   String bankname=br.readLine();
setAccHolderName(customername);
setAccNumber(accno);
setBankName(bankname);
}catch(Exception e){
e.printStackTrace();
   }
```

```
 }
 public String getCreditCard() {
  long accno=getAccNumber();
  String accholdername=getAccHolderName();
  String bname=getBankName();
  return ("The Account number "+accno+" of "+accholdername+" in "+bname+ "bank is valid and authenticated for issuing the credit card. ");
 }
}
public class AdapterPatternDemo {
 public static void main(String args[]){
CreditCard targetInterface=new BankCustomer();
targetInterface.giveBankDetails();
System.out.print(targetInterface.getCreditCard());
 }
}
```

Output:

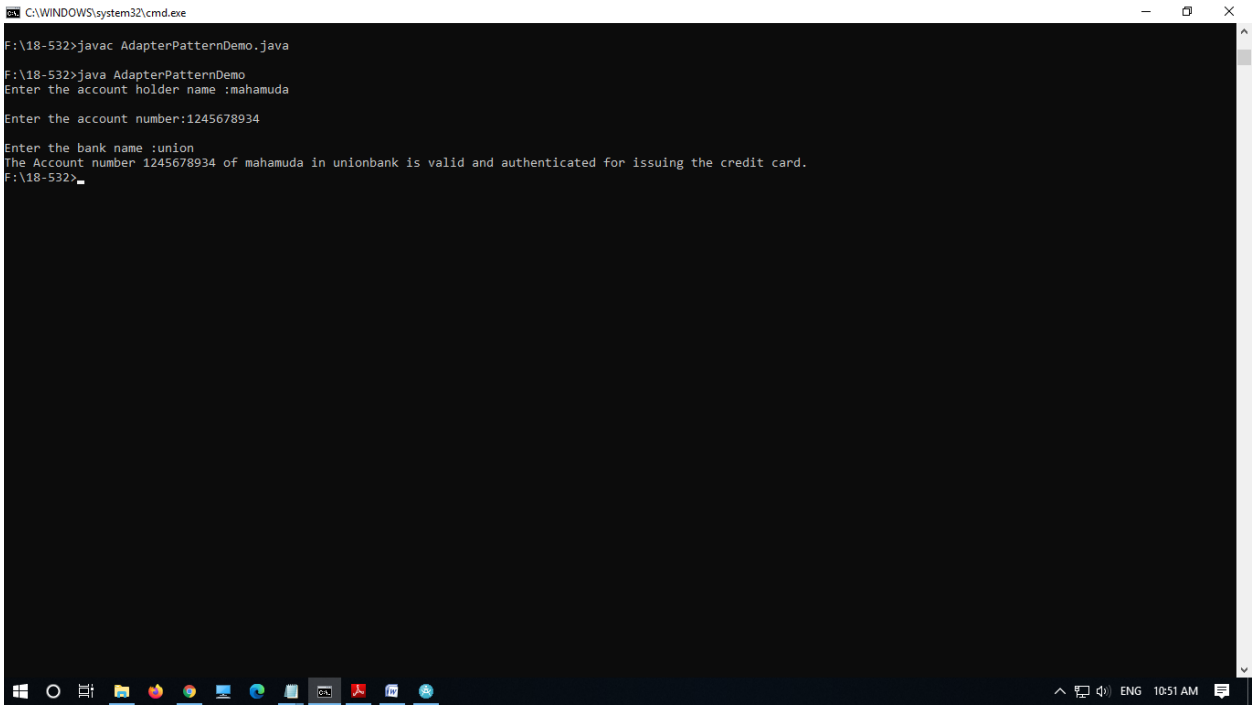
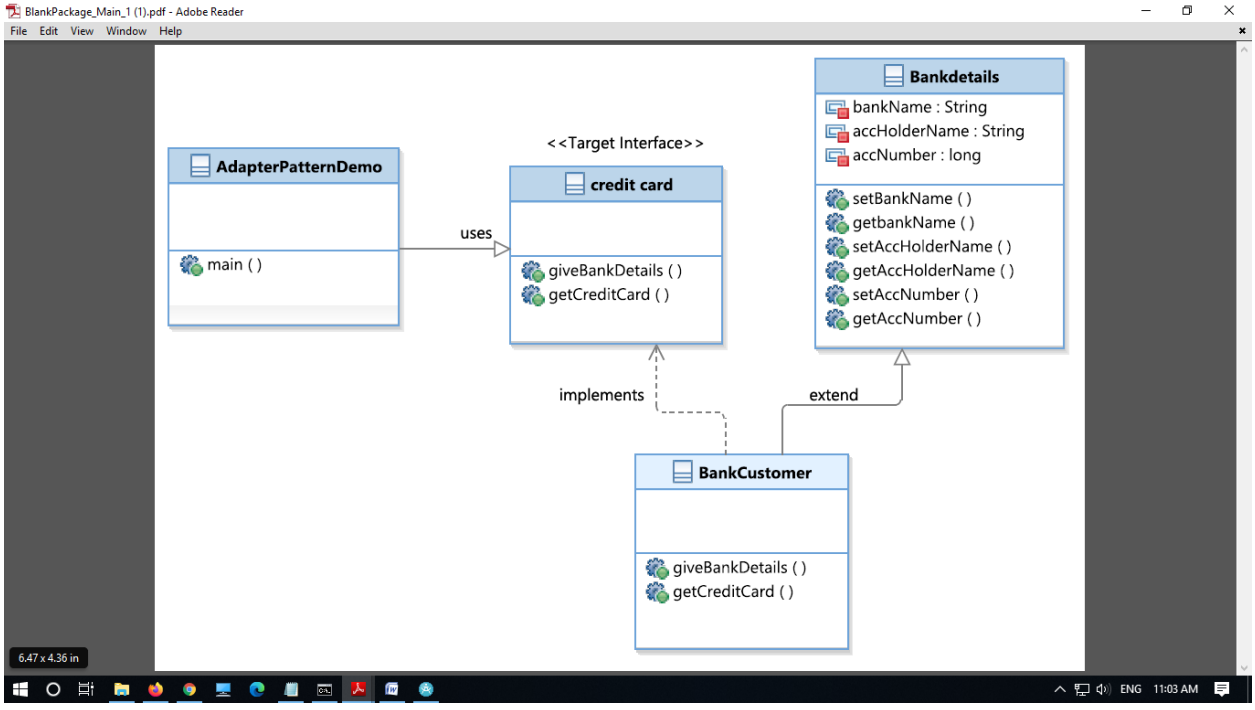



Diagram:



5.Strategy

Program:

```
interface Strategy {
  public int doOperation(int num1, int num2);
}
class OperationAdd implements Strategy{
  @Override
  public int doOperation(int num1, int num2) {
    return num1 + num2;
  }
}
class OperationSubstract implements Strategy{
  @Override
  public int doOperation(int num1, int num2) {
    return num1 - num2;
  }
}
class OperationMultiply implements Strategy{
  @Override
  public int doOperation(int num1, int num2) {
    return num1 * num2;
  }
}
class Context {
  private Strategy strategy;
```

```
 public Context(Strategy strategy){
    this.strategy = strategy;
  }
 public int executeStrategy(int num1, int num2){
    return strategy.doOperation(num1, num2);
  }
}
public class StrategyPatternDemo {
  public static void main(String[] args) {
    Context context = new Context(new OperationAdd());
    System.out.println("10 + 5 = " + context.executeStrategy(10, 5));

    context = new Context(new OperationSubstract());
    System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

    context = new Context(new OperationMultiply());
    System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
  }
}
```

Output:

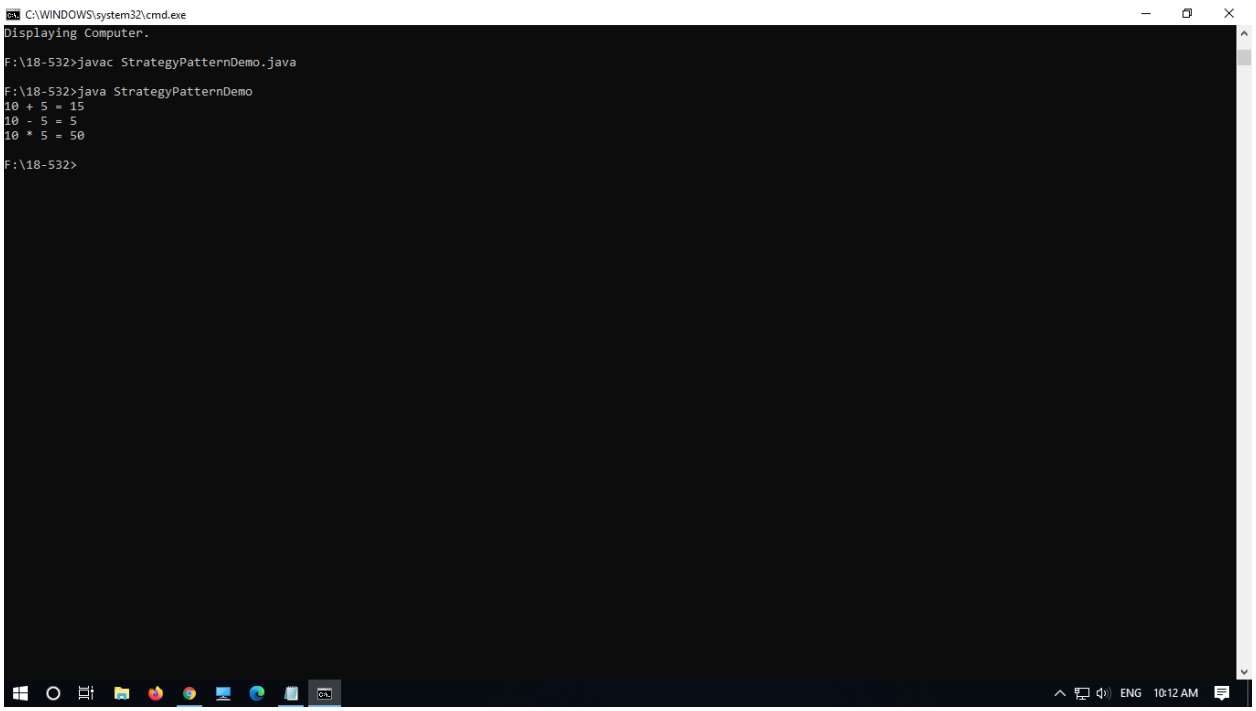
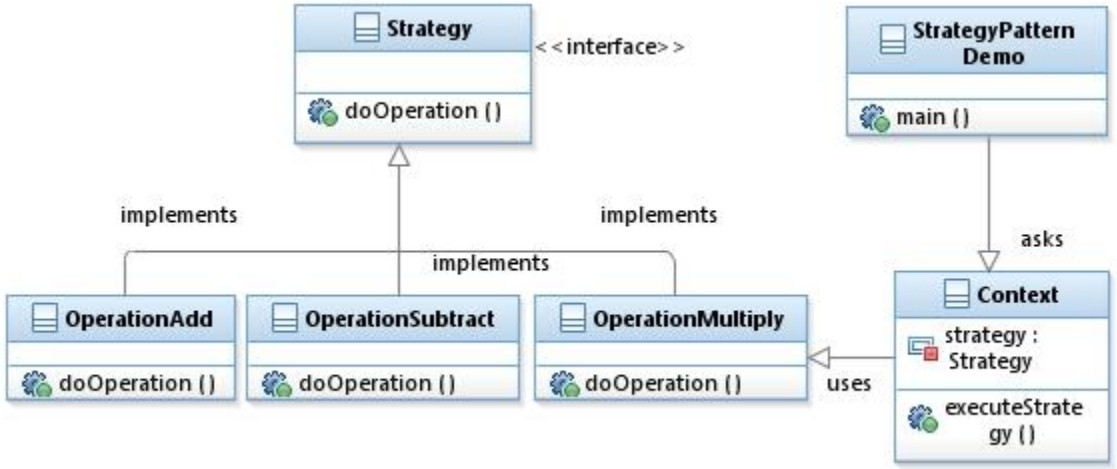



Diagram:




6.Builder:

Program:

```
import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.ArrayList;
```

```
import java.util.List;
interface Item
{
  public String name();
   public String size();
   public float price();
}// End of the interface Item.
abstract class Pizza implements Item
{
        public abstract float price();
}
abstract class ColdDrink implements Item
{
        public abstract float price();
}
abstract class VegPizza extends Pizza
{
   public abstract float price();

   public abstract  String name();

   public abstract  String size();
}// End of the abstract class VegPizza.
abstract class NonVegPizza extends Pizza
{
```

```
    public abstract float price();


    public abstract String name();
    public abstract String size();
}// End of the abstract class NonVegPizza.
class SmallCheezePizza extends VegPizza
{
    public float price()
            {
       return 170.f;
    }
    public String name()
            {
       return "Cheeze Pizza";
    }
    public String size()
            {
      return "Small size";
    }
}// End of the SmallCheezePizza class.
class MediumCheezePizza extends VegPizza{
    public float price()
            {
      return  220.f;
    }
```

```
    public String name()
            {
       return "Cheeze Pizza";
    }
    public String size()
            {
     return "Medium Size";
            }
}// End of the MediumCheezePizza class.

class LargeCheezePizza extends VegPizza{
    public float price()
            {
       return 260.0f;
    }
    public String name()
            {
       return "Cheeze Pizza";
    }
    public String size()
            {
       return "Large Size";
    }
}// End of the LargeCheezePizza class.
class ExtraLargeCheezePizza extends VegPizza{
```

```
    public float price()
            {
       return 300.f;
    }
    public String name()
            {
       return  "Cheeze Pizza";
    }
    public String size()
            {
       return "Extra-Large Size";
    }
}// End of the ExtraLargeCheezePizza class.
class SmallOnionPizza extends VegPizza
{
    public float price()
            {
       return 120.0f;
    }
    public String name()
            {
       return  "Onion Pizza";
    }
    public String size()
            {
```

```
      return  "Small Size";
   }
}// End of the SmallOnionPizza class
class MediumOnionPizza extends VegPizza
{
   public float price()
          {
       return 150.0f;
   }
   public String name()
          {
       return  "Onion Pizza";
   }
   public String size()
          {
      return  "Medium Size";
   }
}// End of the MediumOnionPizza class.
class LargeOnionPizza extends  VegPizza
{
   public float price()
          {
       return 180.0f;
   }
   public String name()
```

```
            {
        return "Onion Pizza";
    }
    public String size()
            {
      return  "Large size";
    }
}// End of the LargeOnionPizza class.
class ExtraLargeOnionPizza extends VegPizza
{
    public float price()
            {
       return 200.0f;
    }
    public String name()
            {
        return  "Onion Pizza";
    }
    public String size()
            {
      return  "Extra-Large Size";
    }
}// End of the ExtraLargeOnionPizza class
class SmallMasalaPizza extends VegPizza
{
```

```
    public float price()
            {
       return 100.0f;
    }
    public String name()
            {
       return  "Masala Pizza";
    }
    public String size()
            {
      return  "Samll Size";
    }
}// End of the SmallMasalaPizza class
class MediumMasalaPizza extends VegPizza
{
    public float price()
            {
       return 120.0f;
    }
    public String name()
            {
       return  "Masala Pizza";


    }
    public String size()
```

```
            {
      return  "Medium Size";
   }
}
class LargeMasalaPizza extends  VegPizza
{
   public float price()
            {
       return 150.0f;
   }
   public String name()
            {
       return  "Masala Pizza";
   }
   public String size()
            {
      return  "Large Size";
   }
} //End of the LargeMasalaPizza class
class ExtraLargeMasalaPizza extends VegPizza
{
   public float price()
            {
       return 180.0f;
   }
```

```
   public String name()
           {
      return  "Masala Pizza";
   }
   public String size()
           {
     return  "Extra-Large Size";
   }
}// End of the ExtraLargeMasalaPizza class
class SmallNonVegPizza extends NonVegPizza
{
   public float price()
           {
      return 180.0f;
   }
   public String name()
           {
     return "Non-Veg Pizza";
   }
   public String size()
           {
      return "Samll Size";
   }
}
class MediumNonVegPizza extends NonVegPizza
```

```
{
   public float price()
          {
      return 200.0f;
   }
   public String name()
          {
     return "Non-Veg Pizza";
   }
   public String size()
          {
      return "Medium Size";
   }
}
class LargeNonVegPizza extends NonVegPizza
{
   public float price()
          {
      return 220.0f;
   }
   public String name()
          {
     return "Non-Veg Pizza";
   }
   public String size()
```

```
            {
        return "Large Size";
    }
}// End of the LargeNonVegPizza class
class ExtraLargeNonVegPizza extends NonVegPizza
{
    public float price()
            {
        return 250.0f;
    }
    public String name()
            {
       return "Non-Veg Pizza";
    }
    public String size()
            {
        return "Extra-Large Size";
    }
}
abstract class Pepsi extends ColdDrink
{
    public abstract  String name();
    public abstract  String size();
    public abstract  float price();
}// End of the Pepsi class
```

```
abstract class Coke  extends ColdDrink
{
   public abstract  String name();
   public abstract  String size();
   public abstract  float price();
}// End of the Coke class
class SmallPepsi  extends Pepsi
{
   public String name()
           {
      return "300 ml Pepsi";
   }
   public float price()
           {
       return 25.0f;
   }
   public String size()
           {
       return "Small Size";
   }
}// End of the SmallPepsi class

class MediumPepsi extends Pepsi
{
   public String name()
```

```
        {
      return "500 ml Pepsi";
   }
   public String size()
        {
       return "Medium Size";
   }
   public float price()
        {
       return 35.0f;
   }
}// End of the MediumPepsi class
class LargePepsi extends Pepsi
{
   public String name()
        {
      return "750 ml Pepsi";
   }
   public String size()
        {
       return "Large Size";
   }
   public float price()
        {
       return 50.0f;
```

```
    }
}// End of the LargePepsi class
class SmallCoke extends Coke
{
    public String name()
            {
         return "300 ml Coke";
    }
    public String size() {
        return "Small Size";
    }
    public float price()
            {


        return  25.0f;
    }
 }// End of the SmallCoke class
class MediumCoke extends Coke
{
    public String name()
            {
         return "500 ml Coke";
    }
    public String size()
            {
```

```
        return "Medium Size";
    }
    public float price()
            {
        return  35.0f;
            }
}// End of the MediumCoke class


class LargeCoke extends Coke
{
    public String name()
            {
         return "750 ml Coke";
    }
    public String size()
            {
        return "Large Size";
    }
    public float price()
            {
        return  50.0f;
    }
}// End of the LargeCoke class
class OrderedItems
{
```

```
   List<Item> items=new ArrayList<Item>();
   public void addItems(Item item)
          {
items.add(item);
   }
   public float getCost()
          {
      float cost=0.0f;
                  for (Item item : items)
                  {
                        cost+=item.price();
                  }
      return cost;
   }
   public void showItems()
          {
      for (Item item : items)
                  {
System.out.println("Item is:" +item.name());
System.out.println("Size is:" +item.size());
System.out.println("Price is: " +item.price());
      }
   }
}// End of the OrderedItems class
```

```
class OrderBuilder
{
   public OrderedItems OrderedItemspreparePizza() throws IOException
          {
OrderedItems itemsOrder=new OrderedItems();
BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
System.out.println(" Enter the choice of Pizza ");
System.out.println("============================");
System.out.println("       1. Veg-Pizza       ");
System.out.println("       2. Non-Veg Pizza   ");
System.out.println("       3. Exit            ");
System.out.println("============================");
int pizzaandcolddrinkchoice=Integer.parseInt(br.readLine());
      switch(pizzaandcolddrinkchoice)
      {
        case 1:
                        {
System.out.println("You ordered Veg Pizza");
System.out.println("\n\n");
System.out.println(" Enter the types of Veg-Pizza ");
System.out.println("-----------------------------");
System.out.println("       1.Cheeze Pizza       ");
System.out.println("       2.Onion Pizza        ");
System.out.println("       3.Masala Pizza       ");
System.out.println("       4.Exit           ");
```

```
System.out.println("------------------------------");
int vegpizzachoice=Integer.parseInt(br.readLine());
                switch(vegpizzachoice)
                {
                    case 1:
                        {
System.out.println("You ordered Cheeze Pizza");
System.out.println("-----------------------------------");
System.out.println("    1. Small Cheeze Pizza ");
System.out.println("    2. Medium Cheeze Pizza ");
System.out.println("    3. Large Cheeze Pizza ");
System.out.println("    4. Extra-Large Cheeze Pizza ");
System.out.println("-----------------------------------");
int cheezepizzasize=Integer.parseInt(br.readLine());
                            switch(cheezepizzasize)
                              {
                                  case 1:
itemsOrder.addItems(new SmallCheezePizza());
                                      break;
                                  case 2:
itemsOrder.addItems(new MediumCheezePizza());
                                      break;
                                  case 3:
itemsOrder.addItems(new LargeCheezePizza());
                                      break;
```

```
                                case 4:
itemsOrder.addItems(new ExtraLargeCheezePizza());
                                    break;
                                                                                    }
                                                                                }
                        case 2:
                            {
System.out.println("You ordered Onion Pizza");
System.out.println("Enter the Onion pizza size");
System.out.println("---------------------------------");
System.out.println("    1. Small Onion Pizza ");
System.out.println("    2. Medium Onion Pizza ");
System.out.println("    3. Large Onion Pizza ");
System.out.println("    4. Extra-Large Onion Pizza ");
System.out.println("---------------------------------");
int onionpizzasize=Integer.parseInt(br.readLine());
                                switch(onionpizzasize)
                                    {
                                        case 1:
itemsOrder.addItems(new SmallOnionPizza());
                                        break;
                                        case 2:
itemsOrder.addItems(new MediumOnionPizza());
                                        break;
                                        case 3:
```

```
itemsOrder.addItems(new LargeOnionPizza());
                                    break;
                                    case 4:
itemsOrder.addItems(new ExtraLargeOnionPizza());
                                    break;
                                }
                            }
                        break;
                    case 3:
                        {
System.out.println("You ordered Masala Pizza");
System.out.println("Enter the Masala pizza size");
System.out.println("-----------------------------------");
System.out.println("    1. Small Masala Pizza ");
System.out.println("    2. Medium Masala Pizza ");
System.out.println("    3. Large Masala Pizza ");
System.out.println("    4. Extra-Large Masala Pizza ");
System.out.println("-----------------------------------");
int masalapizzasize=Integer.parseInt(br.readLine());
                                switch(masalapizzasize)
                                    {
                                    case 1:
itemsOrder.addItems(new SmallMasalaPizza());
                                    break;
                                    case 2:
```

```
itemsOrder.addItems(new MediumMasalaPizza());
                                    break;
                                   case 3:
itemsOrder.addItems(new LargeMasalaPizza());
                                    break;
                                   case 4:
itemsOrder.addItems(new ExtraLargeMasalaPizza());
                                    break;
                                }
                        }
                        break;
                 }
                }
                break;// Veg- pizza choice completed.
            case 2:
                {
              System.out.println("You ordered Non-Veg Pizza");
              System.out.println("\n\n");
                        System.out.println("Enter the Non-Veg pizza size");
                        System.out.println("-----------------------------------");
                        System.out.println("    1. Small Non-Veg  Pizza ");
                        System.out.println("    2. Medium Non-Veg  Pizza ");
                        System.out.println("    3. Large Non-Veg  Pizza ");
                        System.out.println("    4. Extra-Large Non-Veg  Pizza ");
                        System.out.println("-----------------------------------");
```

```
int nonvegpizzasize=Integer.parseInt(br.readLine());
                switch(nonvegpizzasize)
                   {
                      case 1:
itemsOrder.addItems(new SmallNonVegPizza());
                         break;
                      case 2:
itemsOrder.addItems(new MediumNonVegPizza());
                         break;
                      case 3:
itemsOrder.addItems(new LargeNonVegPizza());
                         break;
                      case 4:
itemsOrder.addItems(new ExtraLargeNonVegPizza());
                         break;
                    }
                }
                 break;
        default:
        {
            break;
        }
                    }
System.out.println(" Enter the choice of ColdDrink ");
System.out.println("==========================");
```

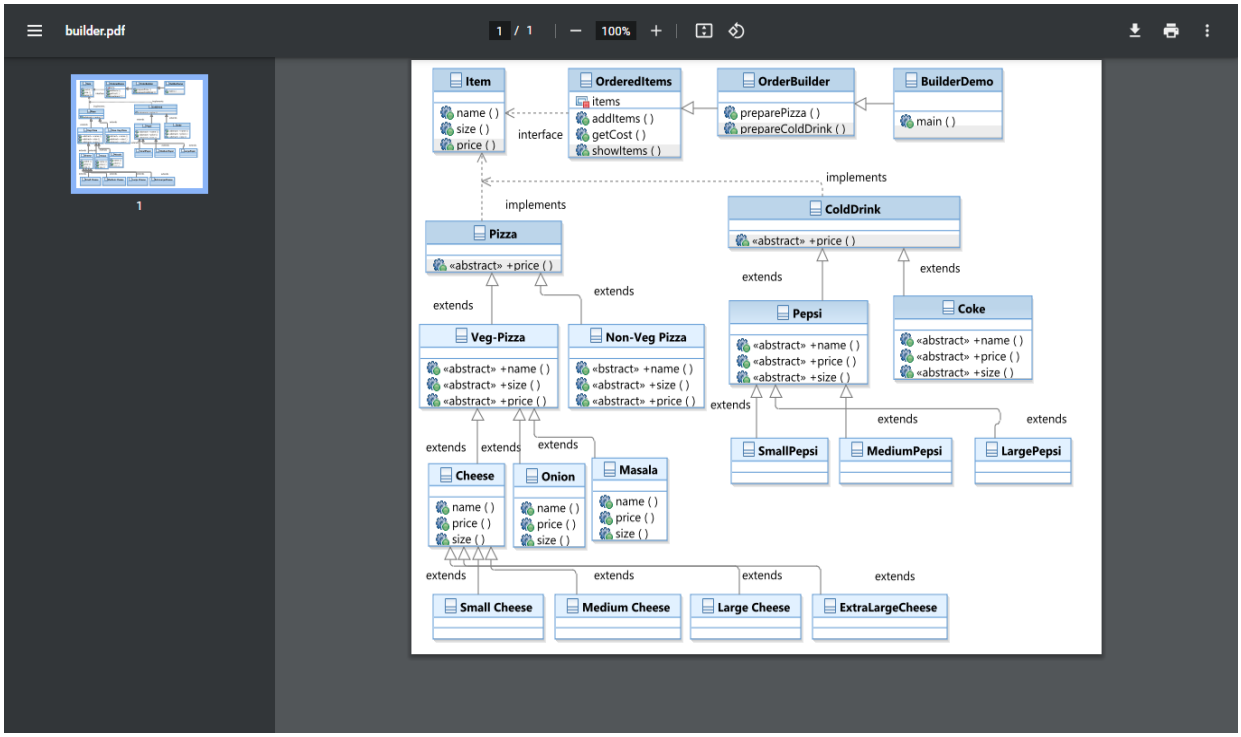
```
System.out.println("       1. Pepsi          ");
System.out.println("       2. Coke           ");
System.out.println("       3. Exit          ");
System.out.println("============================");
int coldDrink=Integer.parseInt(br.readLine());
      switch (coldDrink)
        {
          case 1:
                                        {
                                                System.out.println("You ordered Pepsi ");
                                                System.out.println("Enter the  Pepsi Size ");
                                                System.out.println("------------------------");
                                                System.out.println("    1. Small Pepsi ");
                                                System.out.println("    2. Medium Pepsi ");
                                                System.out.println("    3. Large Pepsi ");
                                                System.out.println("------------------------");
                                                int pepsisize=Integer.parseInt(br.readLine());
                                                switch(pepsisize)
                                                {
                                                case 1:
                                                        itemsOrder.addItems(new SmallPepsi());
                                                                        break;
case 2:
itemsOrder.addItems(new MediumPepsi());
                                                                        break;
```

```
                                        case 3:
                                                itemsOrder.addItems(new LargePepsi());
                                                        break;
}
                                    }
                                    break;
                        case 2:
                                    {
                                        System.out.println("You ordered Coke");
                                        System.out.println("Enter the Coke Size");
                                        System.out.println("------------------------");
                                        System.out.println("    1. Small Coke ");
                                        System.out.println("    2. Medium Coke  ");
                                        System.out.println("    3. Large Coke  ");
                                        System.out.println("    4. Extra-Large Coke ");
                                        System.out.println("------------------------");
                                        int cokesize=Integer.parseInt(br.readLine());
                                        switch(cokesize)
                                        {
case 1:
itemsOrder.addItems(new SmallCoke());
break;
case 2:
itemsOrder.addItems(new MediumCoke());
break;
```

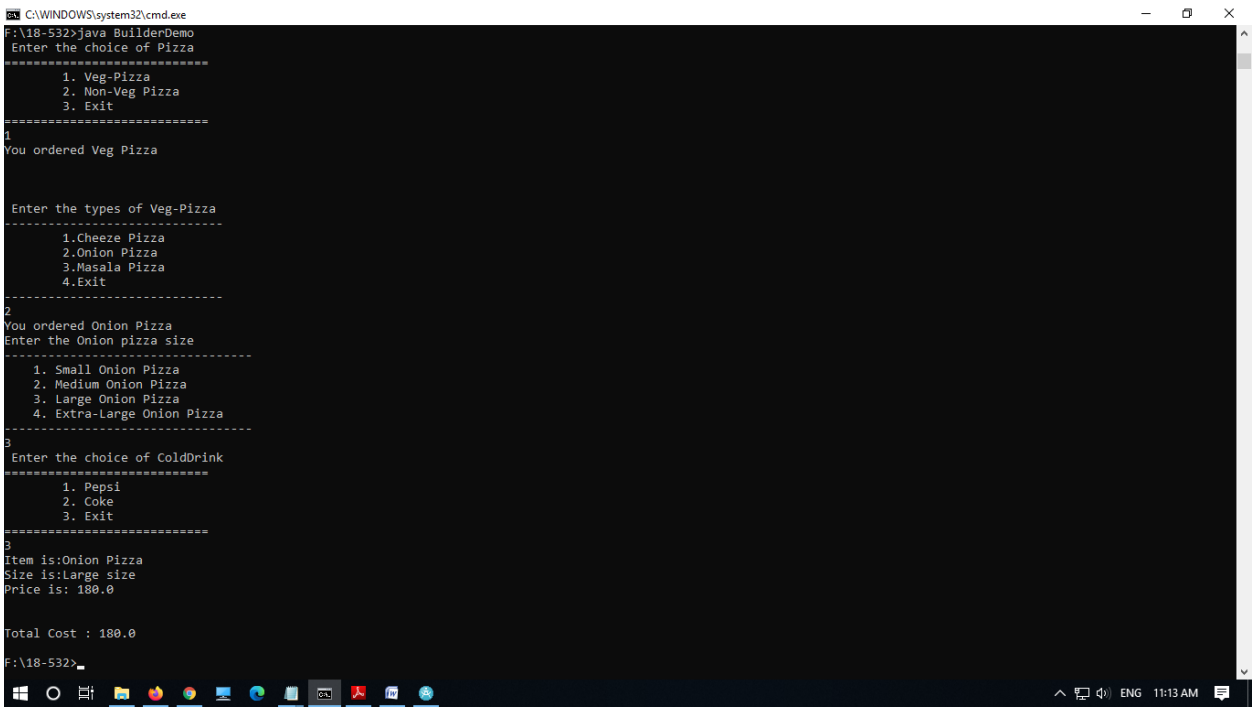
```
case 3:
itemsOrder.addItems(new LargeCoke());
break;
                                                                }
                                        }
                                        break;
            default:
                                        {
                                                break;
                                        }

         }//End of the Cold-Drink switch
             return itemsOrder;
       } //End of the preparePizza() method
}
public class BuilderDemo
{
   public static void main(String[] args) throws IOException
           {
OrderBuilder builder=new OrderBuilder();
OrderedItems orderedItems=builder.OrderedItemspreparePizza();
orderedItems.showItems();
System.out.println("\n");
System.out.println("Total Cost : "+ orderedItems.getCost());
   }
```

}

Diagram:




Output:



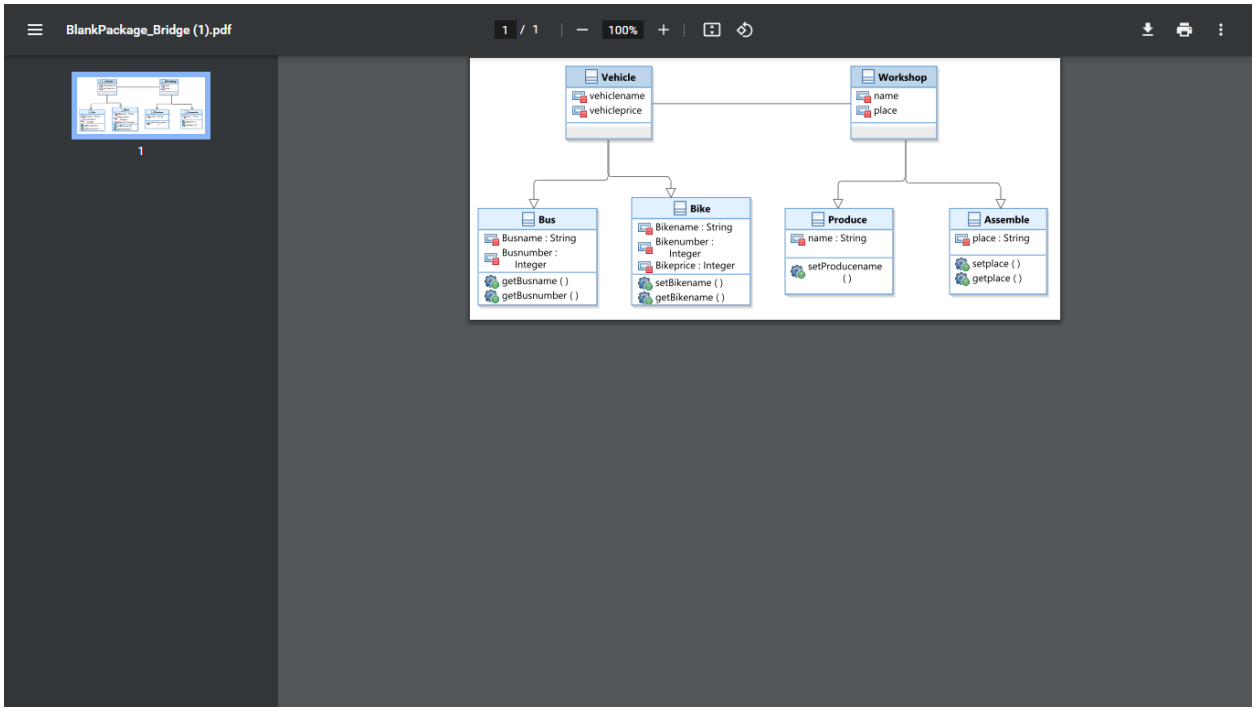

7.Bridge:

Program:

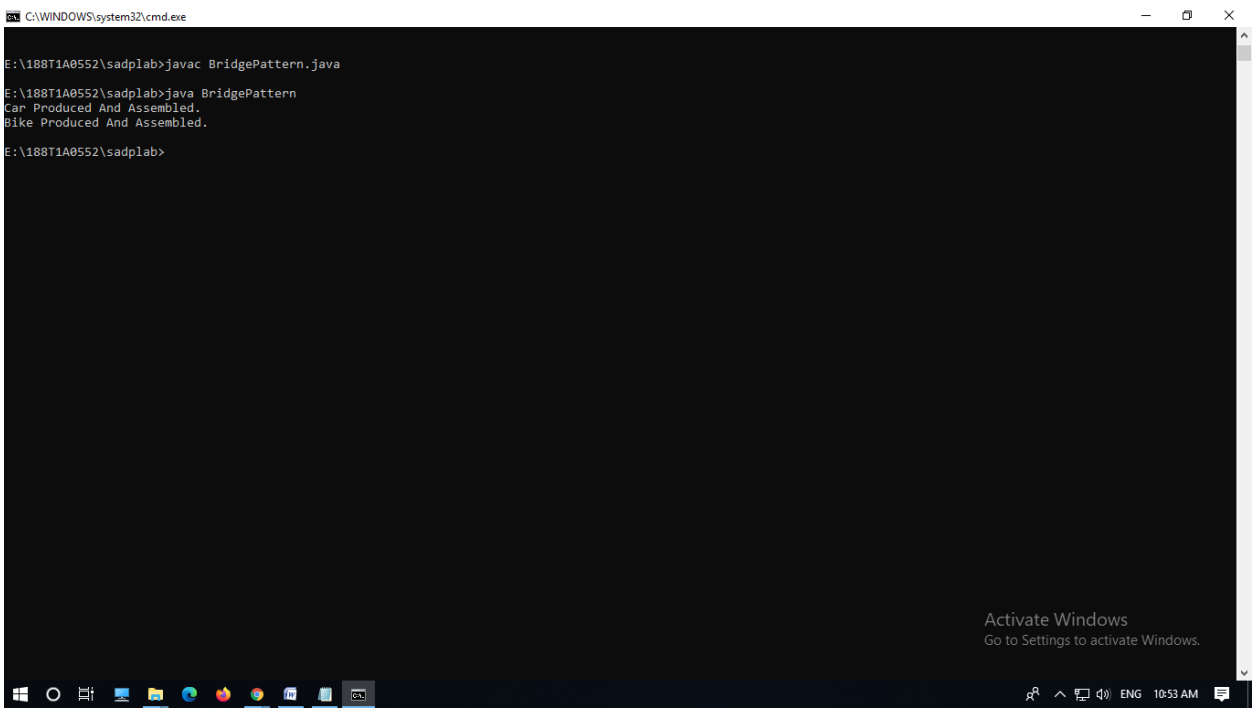
```
abstract class Vehicle {
   protected Workshop workShop1;
   protected Workshop workShop2;
   protected Vehicle(Workshop workShop1, Workshop workShop2)
   {
this.workShop1 = workShop1;
this.workShop2 = workShop2;
   }
   abstract public void manufacture();
}
class Car extends Vehicle {
   public Car(Workshop workShop1, Workshop workShop2)
   {
super(workShop1, workShop2);
   }
   public void manufacture()
   {
System.out.print("Car ");
      workShop1.work();
      workShop2.work();
   }
}
class Bike extends Vehicle {
```

```
    public Bike(Workshop workShop1, Workshop workShop2)
    {
super(workShop1, workShop2);
    }
    public void manufacture()
    {
System.out.print("Bike ");
        workShop1.work();
        workShop2.work();
    }
}
interface Workshop
{
    abstract public void work();
}
class Produce implements Workshop {
    public void work()
    {
System.out.print("Produced");
    }
}
class Assemble implements Workshop {
    public void work()
    {
System.out.print(" And");
System.out.println(" Assembled.");
    }
}
public class BridgePattern{
    public static void main(String[] args)
    {
        Vehicle vehicle1 = new Car(new Produce(), new Assemble());
        vehicle1.manufacture();
        Vehicle vehicle2 = new Bike(new Produce(), new Assemble());
        vehicle2.manufacture();
    }
}
```

diagram:




Output:



8.Decorator

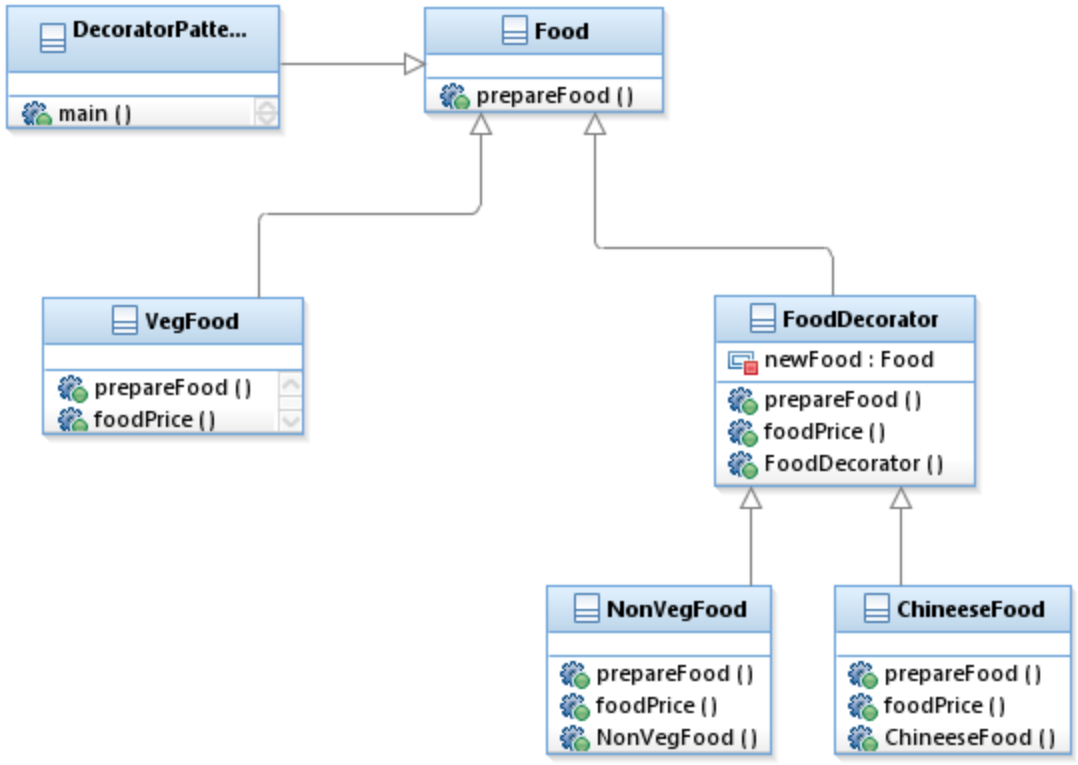
Program:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
interface Food {
   public String prepareFood();
   public double foodPrice();
}// End of the Food interface.

 class VegFood implements Food {
   public String prepareFood(){
       return "Veg Food";
   }

      public double foodPrice(){
      return 100.0;
   }
}

 abstract class FoodDecorator implements Food{
   private Food newFood;
   public FoodDecorator(Food newFood)  {
      this.newFood=newFood;
   }
   @Override
   public String prepareFood(){
      return newFood.prepareFood();
   }
   public double foodPrice(){
      return newFood.foodPrice();
   }
}

class NonVegFood extends FoodDecorator{
   public NonVegFood(Food newFood) {
      super(newFood);
   }
   public String prepareFood(){
      return super.prepareFood() +" With Roasted Chiken and Chiken Curry  ";
   }
   public double foodPrice()   {
      return super.foodPrice()+200.0;
   }
}
```

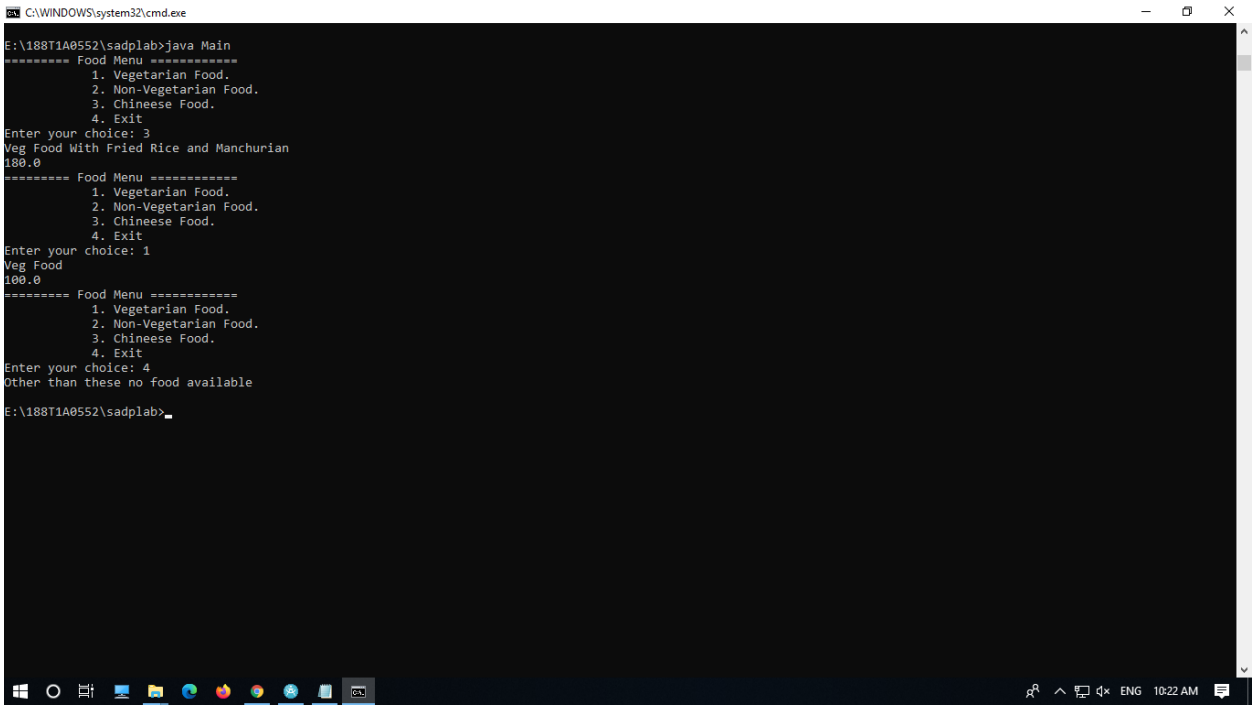
```
class ChineeseFood extends FoodDecorator{
  public ChineeseFood(Food newFood)    {
       super(newFood);
  }
    public String prepareFood(){
       return super.prepareFood() +" With Fried Rice and Manchurian  ";
    }
    public double foodPrice()   {
       return super.foodPrice()+80.0;
       }
}

public class DecoratorPatternCustomer {
    private static int  choice;
    public static void main(String args[]) throws NumberFormatException, IOException    {
      do{
       System.out.print("========= Food Menu ============ \n");
       System.out.print("          1. Vegetarian Food.   \n");
       System.out.print("          2. Non-Vegetarian Food.\n");
       System.out.print("          3. Chineese Food.        \n");
       System.out.print("          4. Exit                  \n");
       System.out.print("Enter your choice: ");
       BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
       choice=Integer.parseInt(br.readLine());
       switch (choice) {
       case 1:{
              VegFood vf=new VegFood();
            System.out.println(vf.prepareFood());
            System.out.println( vf.foodPrice());
          }
          break;

             case 2:{
             Food f1=new NonVegFood((Food) new VegFood());
                System.out.println(f1.prepareFood());
             System.out.println( f1.foodPrice());
       }
          break;
     case 3:{
           Food f2=new ChineeseFood((Food) new VegFood());
                 System.out.println(f2.prepareFood());
                System.out.println( f2.foodPrice());
            }
          break;
```

```
        default:{
           System.out.println("Other than these no food available");
         }
    return;
     }//end of switch

}while(choice!=4);
    }
}
```

diagram:



Output:




9.Chain of responsibility:

Program:

```
abstract class AbstractLogger {
  public static int INFO = 1;
  public static int DEBUG = 2;
  public static int ERROR = 3;

  protected int level;
  protected AbstractLogger nextLogger;

  public void setNextLogger(AbstractLogger nextLogger){
    this.nextLogger = nextLogger;
  }

  public void logMessage(int level, String message){
    if(this.level <= level){
      write(message);
    }
    if(nextLogger !=null){
      nextLogger.logMessage(level, message);
    }
  }
```

```
   abstract protected void write(String message);

}
class ConsoleLogger extends AbstractLogger {

   public ConsoleLogger(int level){
      this.level = level;
   }

   @Override
   protected void write(String message) {
      System.out.println("Standard Console::Logger: " + message);
   }
}
class ErrorLogger extends AbstractLogger {

   public ErrorLogger(int level){
      this.level = level;
   }

   @Override
   protected void write(String message) {
      System.out.println("Error Console::Logger: " + message);
   }
}
class FileLogger extends AbstractLogger {

   public FileLogger(int level){
      this.level = level;
   }

   @Override
   protected void write(String message) {
      System.out.println("File::Logger: " + message);
   }
}
public class ChainPatternDemo {

   private static AbstractLogger getChainOfLoggers(){

      AbstractLogger errorLogger = new ErrorLogger(AbstractLogger.ERROR);
      AbstractLogger fileLogger = new FileLogger(AbstractLogger.DEBUG);
      AbstractLogger consoleLogger = new ConsoleLogger(AbstractLogger.INFO);

      errorLogger.setNextLogger(fileLogger);
      fileLogger.setNextLogger(consoleLogger);
```

```
    return errorLogger;
  }

  public static void main(String[] args) {
    AbstractLogger loggerChain = getChainOfLoggers();

    loggerChain.logMessage(AbstractLogger.INFO,
      "This is an information.");

    loggerChain.logMessage(AbstractLogger.DEBUG,
      "This is an debug level information.");

    loggerChain.logMessage(AbstractLogger.ERROR,
      "This is an error information.");
  }
}
```

output:

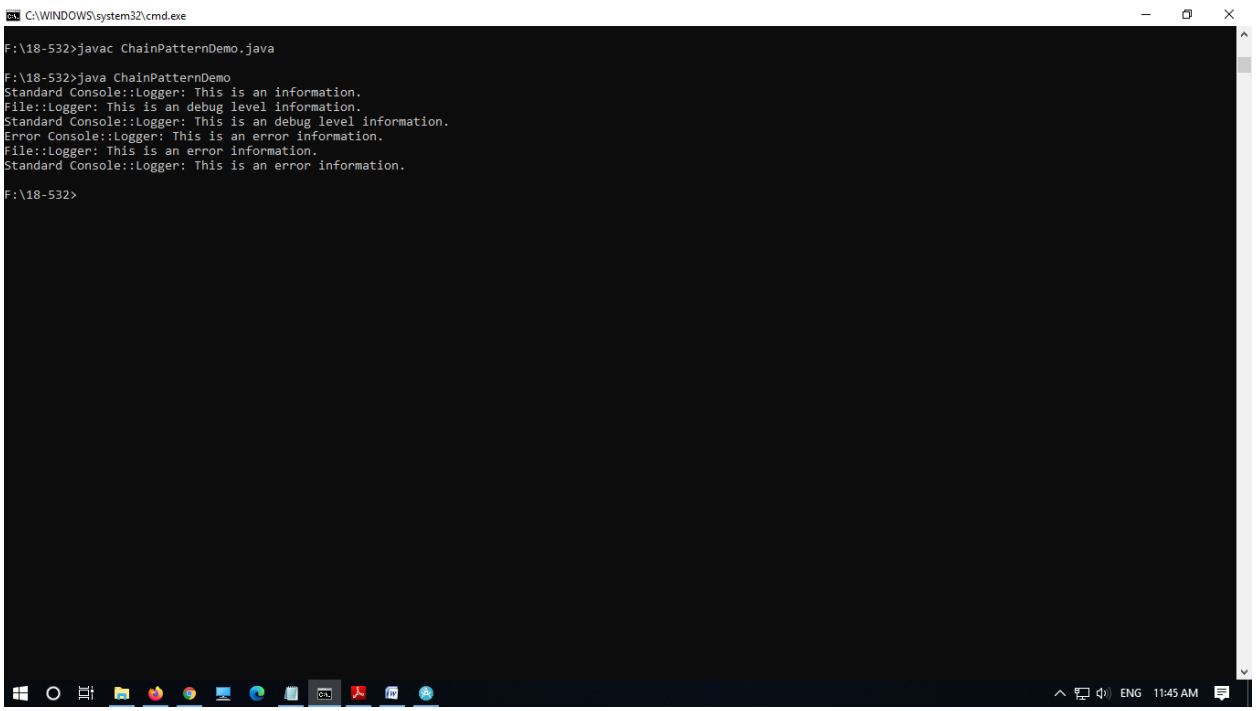
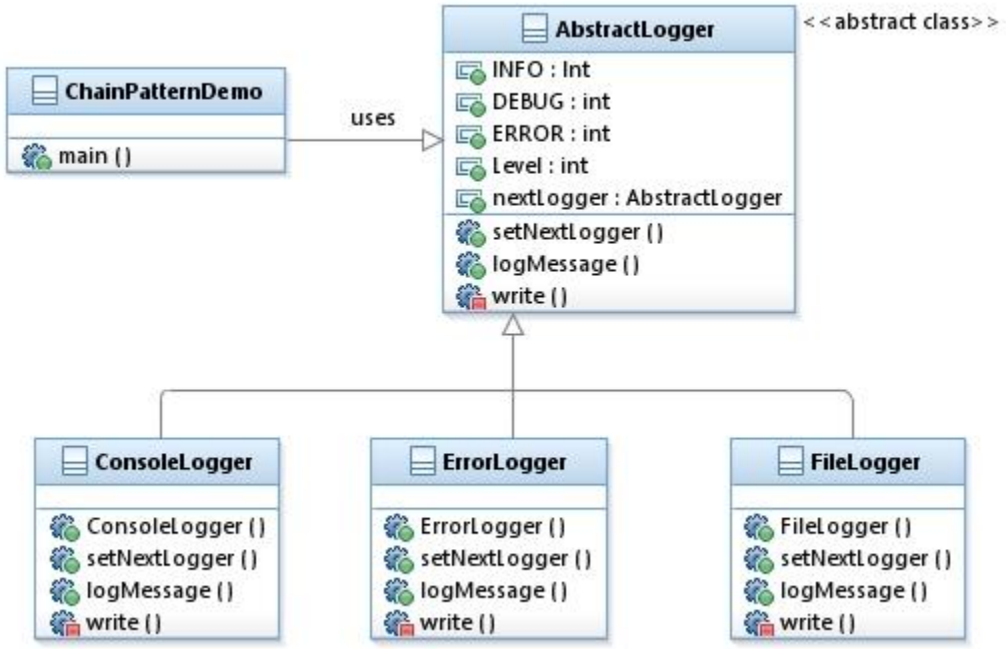


Diagram:




10.Flyweight:

Program:

```
import java.util.HashMap;

interface Shape {
   void draw();
}

class Circle implements Shape {
   private String color;
   private int x;
   private int y;
   private int radius;


   public Circle(String color){
```

```
    this.color = color;
  }

  public void setX(int x) {
    this.x = x;
  }

  public void setY(int y) {
    this.y = y;
  }

  public void setRadius(int radius) {
    this.radius = radius;
  }

  @Override
  public void draw() {
    System.out.println("Circle: Draw() [Color : " + color + ", x : " + x + ", y :" + y + ", radius :" + radius);
  }
}
class ShapeFactory {
  private static final HashMap circleMap = new HashMap();
   public static Shape getCircle(String color) {
    Circle circle = (Circle)circleMap.get(color);
     if(circle == null) {
```

```
      circle = new Circle(color);
      circleMap.put(color, circle);
      System.out.println("Creating circle of color : " + color);
    }
    return circle;
  }
}
public class FlyweightPatternDemo{
  private static final String colors[] = { "Red", "Green", "Blue", "White", "Black" };
  public static void main(String[] args) {

    for(int i=0; i < 20; ++i) {
      Circle circle = (Circle)ShapeFactory.getCircle(getRandomColor());
      circle.setX(getRandomX());
      circle.setY(getRandomY());
      circle.setRadius(100);
      circle.draw();
    }
  }
  private static String getRandomColor() {
    return colors[(int)(Math.random()*colors.length)];
  }
  private static int getRandomX() {
    return (int)(Math.random()*100 );
  }
```

```
  private static int getRandomY() {
     return (int)(Math.random()*100);
  }
}
```

Output

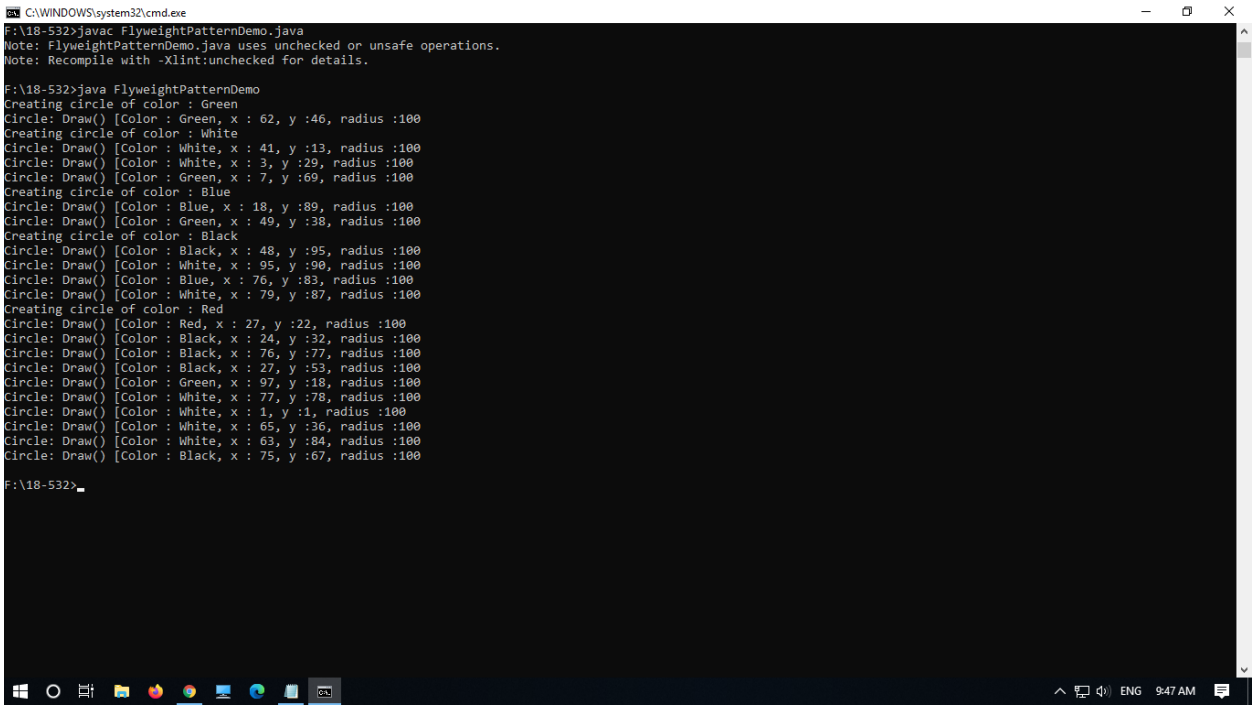
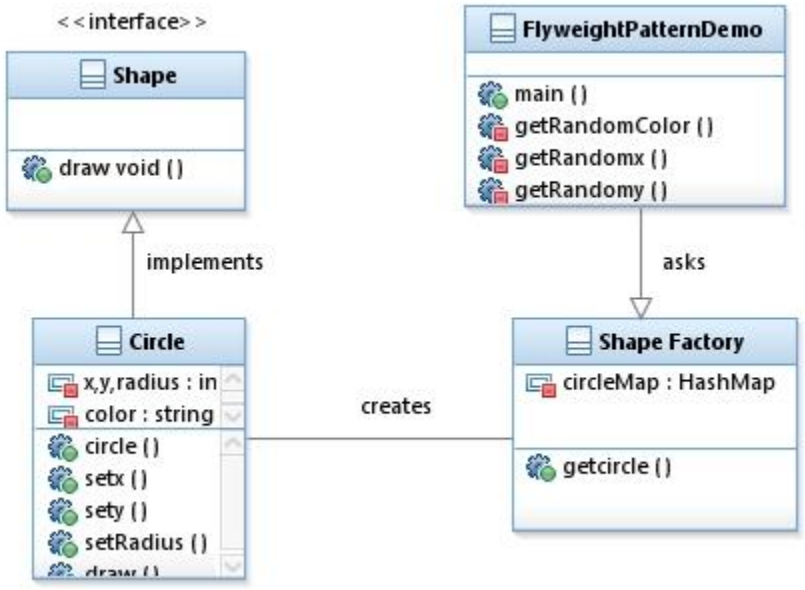


Diagram:




11.Facade Design Pattern

Program:

```
import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;


interface MobileShop {

   public void modelNo();

   public void price();

}

class Iphone implements MobileShop {

   @Override

   public void modelNo() {

      System.out.println(" Iphone 6 ");
```

```
    }
    @Override
    public void price() {
    System.out.println(" Rs 65000.00 ");
    }
}
class Samsung implements MobileShop {
    @Override
    public void modelNo() {
    System.out.println(" Samsung galaxy tab 3 ");
    }
    @Override
    public void price() {
        System.out.println(" Rs 45000.00 ");
    }
}
class Blackberry implements MobileShop {
    @Override
    public void modelNo() {
    System.out.println(" Blackberry Z10 ");
    }
    @Override
    public void price() {
        System.out.println(" Rs 55000.00 ");
    }
```

```
}
class ShopKeeper {
   private MobileShop iphone;
   private MobileShop samsung;
   private MobileShop blackberry;

   public ShopKeeper(){
      iphone= new Iphone();
      samsung=new Samsung();
      blackberry=new Blackberry();
   }
   public void iphoneSale(){
      iphone.modelNo();
      iphone.price();
   }
      public void samsungSale(){
      samsung.modelNo();
      samsung.price();
   }
  public void blackberrySale(){
   blackberry.modelNo();
   blackberry.price();
      }
}
```

```
public class FacadePatternClient {
   private static int  choice;
   public static void main(String args[]) throws NumberFormatException, IOException{
      do{
         System.out.print("========= Mobile Shop ============ \n");
         System.out.print("          1. IPHONE.            \n");
         System.out.print("          2. SAMSUNG.           \n");
         System.out.print("          3. BLACKBERRY.        \n");
         System.out.print("          4. Exit.              \n");
         System.out.print("Enter your choice: ");

         BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
         choice=Integer.parseInt(br.readLine());
         ShopKeeper sk=new ShopKeeper();

         switch (choice) {
         case 1:
            {
             sk.iphoneSale();
               }
            break;
      case 2:
            {
             sk.samsungSale();
               }
```

```
            break;
        case 3:
                    {
                    sk.blackberrySale();
                    }
                break;
            default:
            {
               System.out.println("Nothing You purchased");
            }
               return;
            }

      }while(choice!=4);
   }
}
```

Output:

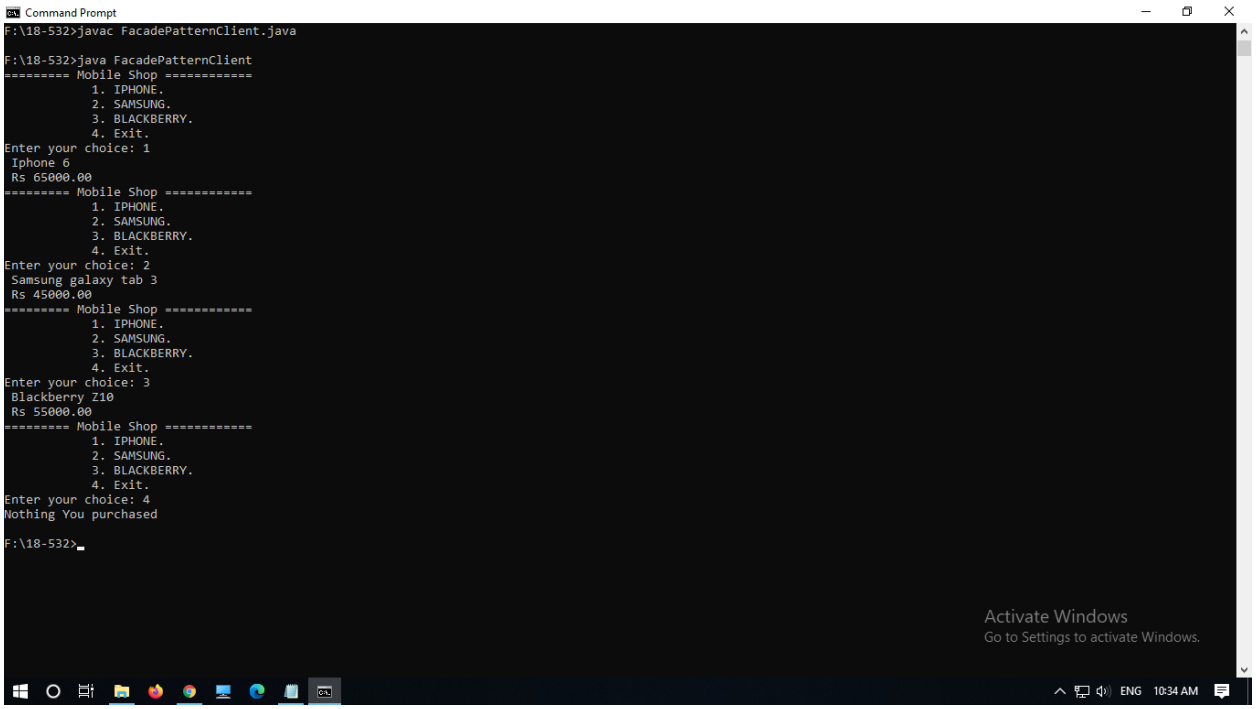
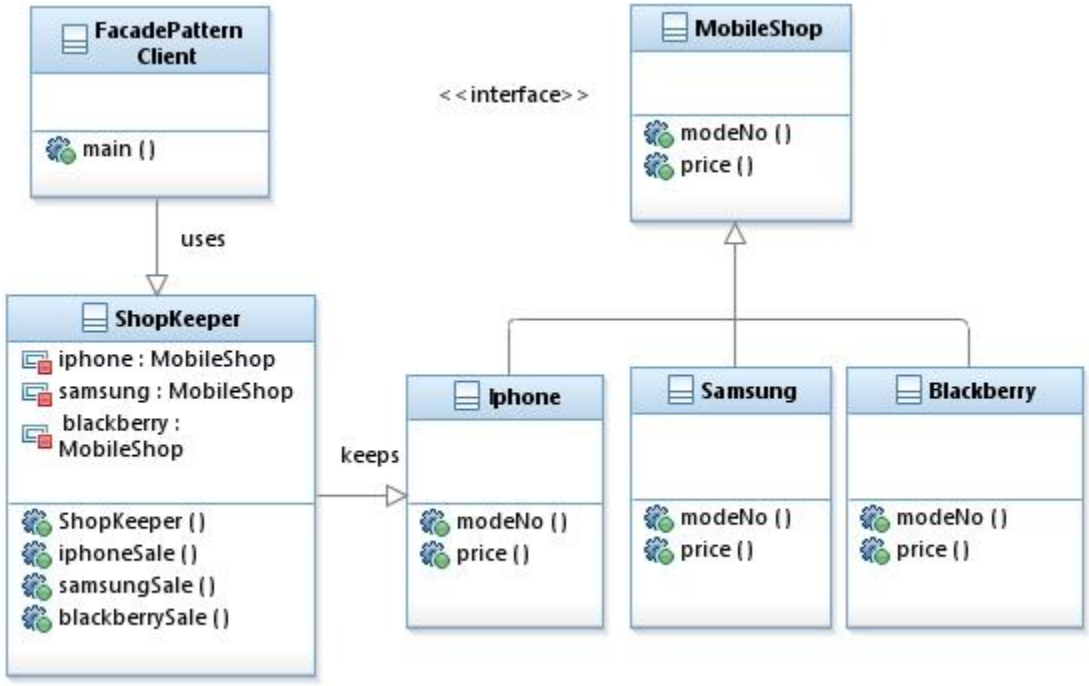



Diagram:



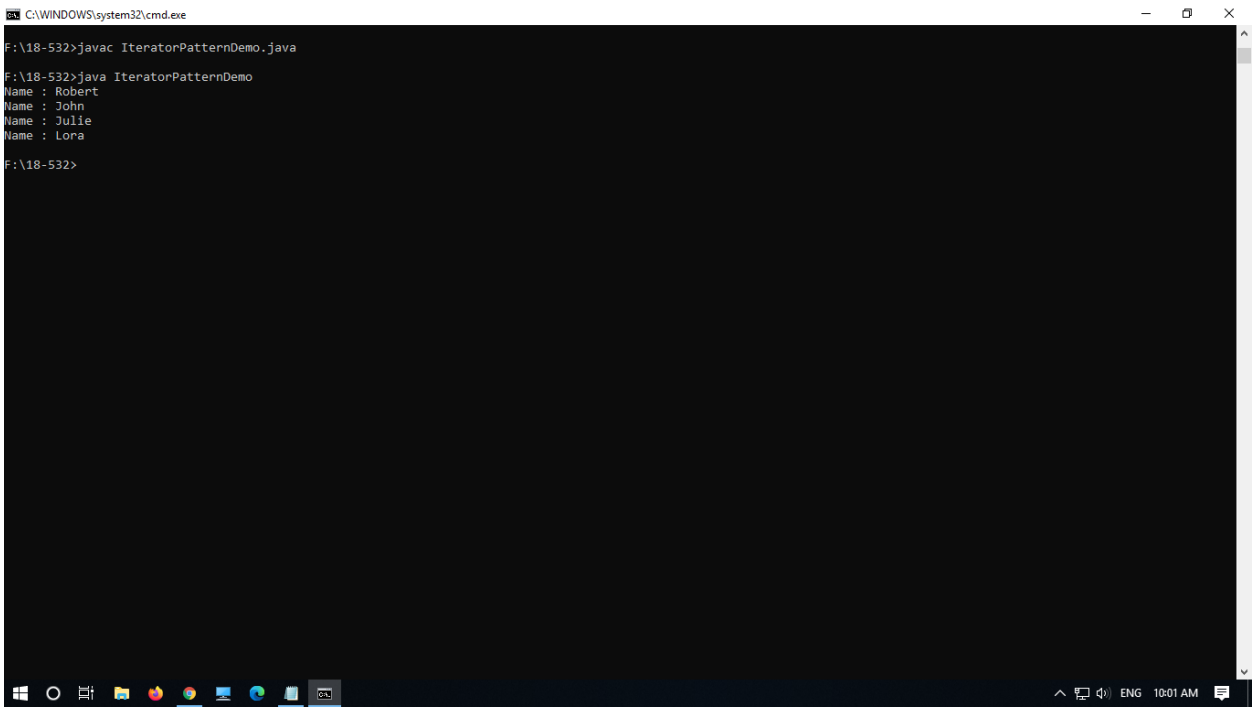
12.Iteratordesign pattern

Program:

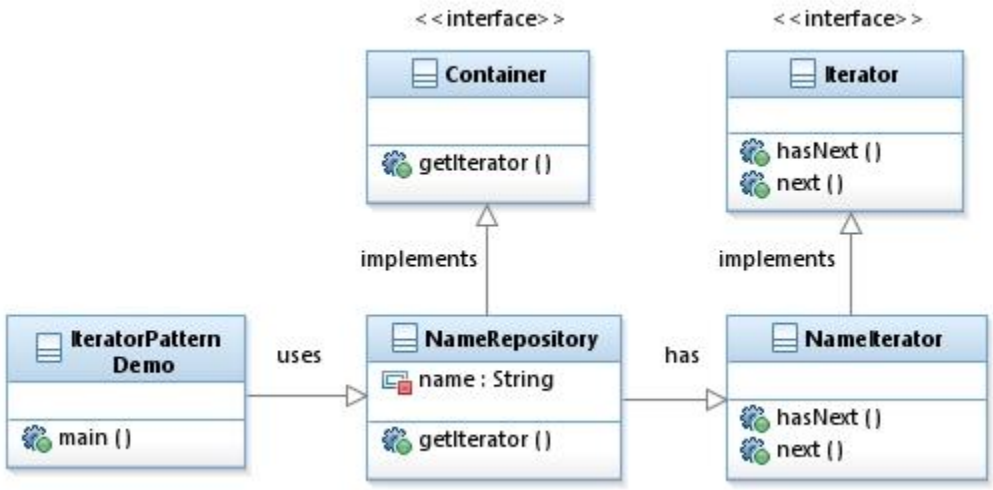
```
interface Iterator {
  public boolean hasNext();
  public Object next();
}
interface Container {
  public Iterator getIterator();
}
class NameRepository implements Container {
  public String names[] = {"Robert" , "John" ,"Julie" , "Lora"};

  @Override
  public Iterator getIterator() {
    return new NameIterator();
  }

  private class NameIterator implements Iterator {
   int index;
   @Override
    public boolean hasNext() {
    if(index < names.length){
        return true;
      }
      return false;
```

```
    }
    @Override
    public Object next() {
    if(this.hasNext()){
        return names[index++];
      }
      return null;
    }
  }
}
public class IteratorPatternDemo {
          public static void main(String[] args) {
    NameRepository namesRepository = new NameRepository();

    for(Iterator iter = namesRepository.getIterator(); iter.hasNext();){
      String name = (String)iter.next();
      System.out.println("Name : " + name);
    }
  }
}
```

Output:




Diagrams:




13.Mediator:

Program:

```
import java.util.Date;

class ChatRoom {
```

```
  public static void showMessage(User user, String message){
     System.out.println(new Date().toString() + " [" + user.getName() + "] : " + message);
  }
}
class User {
  private String name;
public String getName() {
     return name;
  }
public void setName(String name) {
     this.name = name;
  }
public User(String name){
     this.name  = name;
  }
public void sendMessage(String message){
     ChatRoom.showMessage(this,message);
  }
}
public class MediatorPatternDemo {
  public static void main(String[] args) {
     User robert = new User("Robert");
     User john = new User("John");
     robert.sendMessage("Hi! John!");
     john.sendMessage("Hello! Robert!");
```

```
  }
}
```

Output:

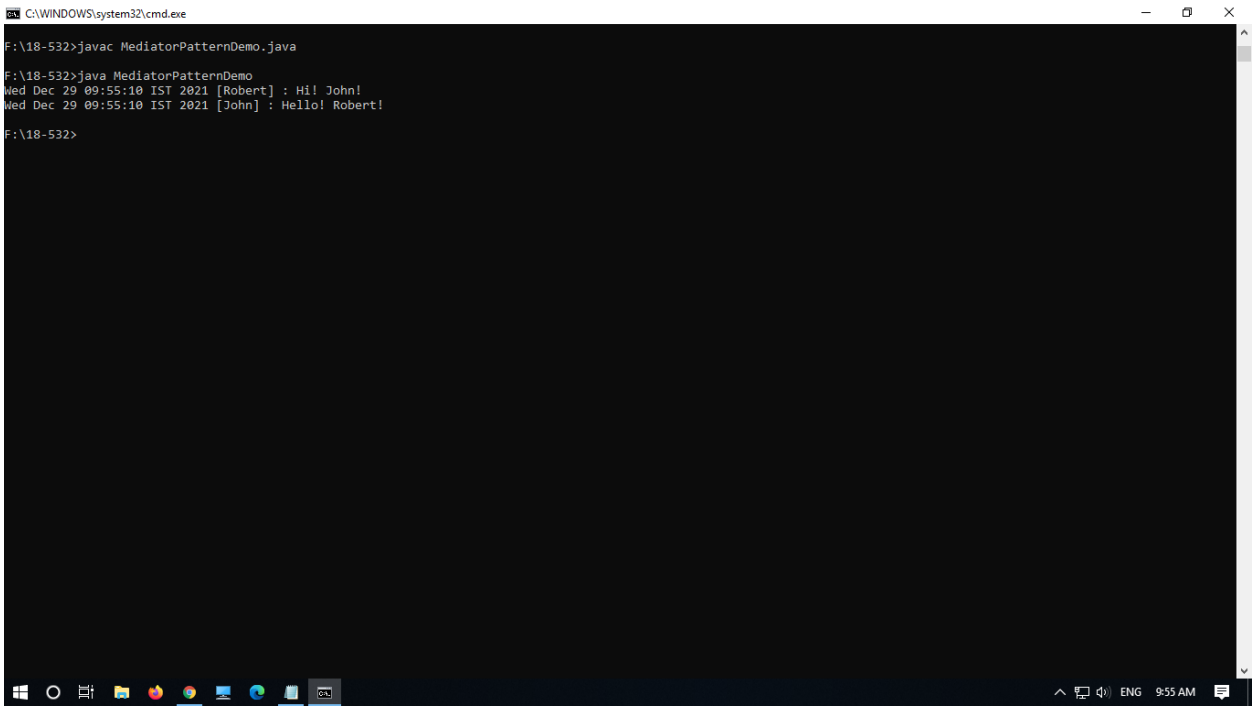



Diagram:


14.Proxy:

Program:

```
interface OfficeInternetAccess {

public void grantInternetAccess();

}
```

```
class RealInternetAccess implements OfficeInternetAccess {
private String employeeName;
public RealInternetAccess(String empName) {
this.employeeName = empName;
}
public void grantInternetAccess() {
System.out.println("Internet Access granted for employee: "+ employeeName);
}
}
class ProxyInternetAccess implements OfficeInternetAccess {
private String employeeName;
private RealInternetAccess realaccess;
public ProxyInternetAccess(String employeeName) {
this.employeeName = employeeName;
}
public void grantInternetAccess()
{
if (getRole(employeeName) > 4)
{
realaccess = new RealInternetAccess(employeeName);
realaccess.grantInternetAccess();
}
else
{
System.out.println("No Internet access granted. Your job level is below 5");
```

```
}

}

public int getRole(String emplName) {

return 9;

}

}

public class ProxyPatternClient{


public static void main(String[] args)

{

OfficeInternetAccess access = new ProxyInternetAccess("Ashwani Rajput");

access.grantInternetAccess();

}

}
```

Output:

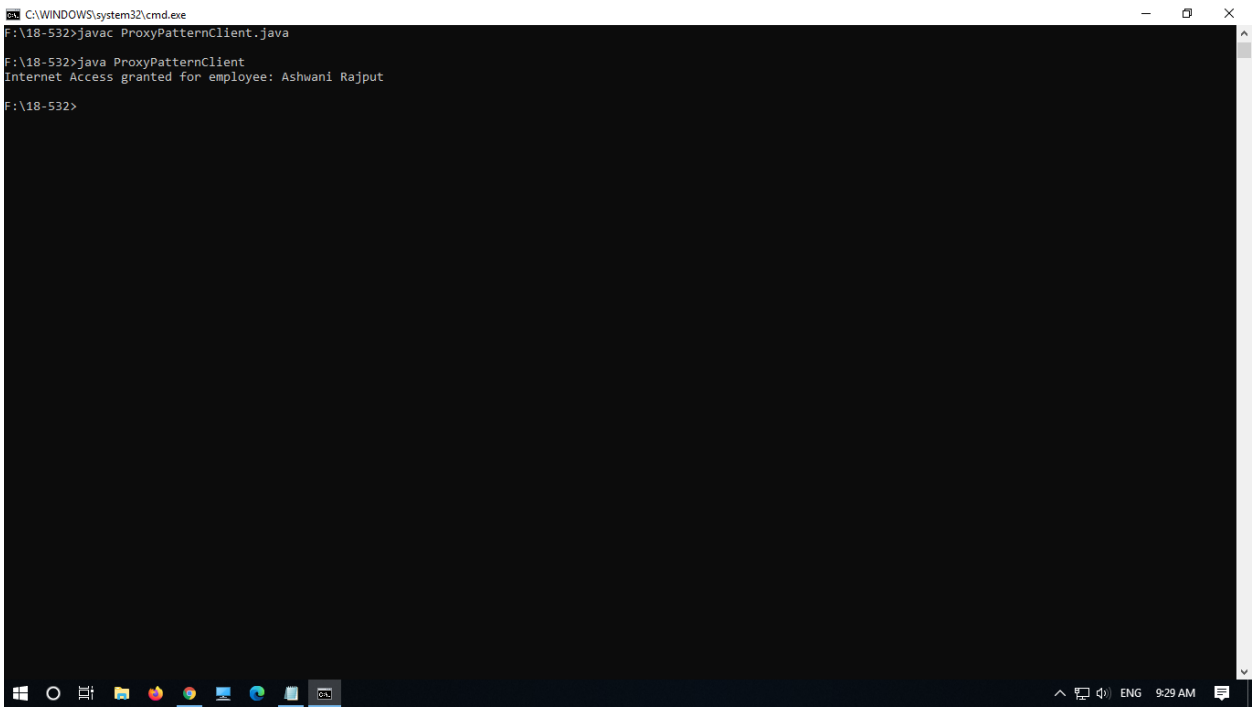
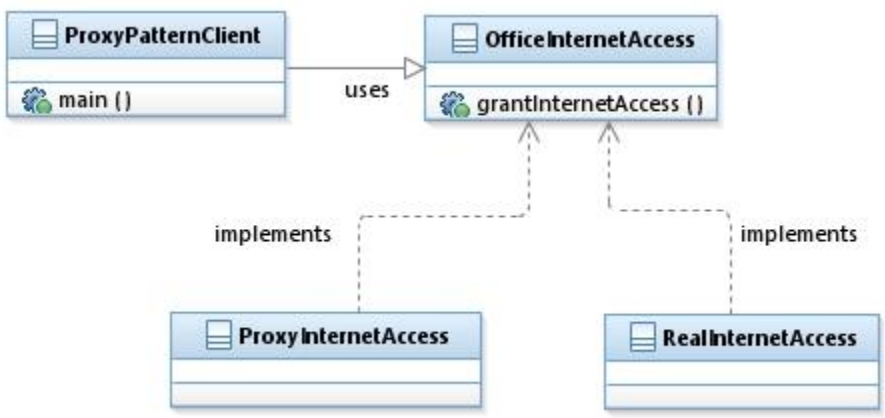


Diagram:




15.Visitor:

Program:

```
interface ComputerPart {
  public void accept(ComputerPartVisitor computerPartVisitor);
}
class Keyboard implements ComputerPart {

  @Override
  public void accept(ComputerPartVisitor computerPartVisitor) {
    computerPartVisitor.visit(this);
  }
}
class Monitor implements ComputerPart {

  @Override
  public void accept(ComputerPartVisitor computerPartVisitor) {
    computerPartVisitor.visit(this);
```

```
  }
}
class Mouse implements ComputerPart {

  @Override
  public void accept(ComputerPartVisitor computerPartVisitor) {
    computerPartVisitor.visit(this);
  }
}
class Computer implements ComputerPart {

  ComputerPart[] parts;

  public Computer(){
    parts = new ComputerPart[] {new Mouse(), new Keyboard(), new Monitor()};
  }
 @Override
  public void accept(ComputerPartVisitor computerPartVisitor) {
    for (int i = 0; i < parts.length; i++) {
      parts[i].accept(computerPartVisitor);
    }
    computerPartVisitor.visit(this);
  }
}
interface ComputerPartVisitor {
```

```
        public void visit(Computer computer);
        public void visit(Mouse mouse);
        public void visit(Keyboard keyboard);
        public void visit(Monitor monitor);
}
class ComputerPartDisplayVisitor implements ComputerPartVisitor {

  @Override
  public void visit(Computer computer) {
    System.out.println("Displaying Computer.");
  }

  @Override
  public void visit(Mouse mouse) {
    System.out.println("Displaying Mouse.");
  }

  @Override
  public void visit(Keyboard keyboard) {
    System.out.println("Displaying Keyboard.");
  }

  @Override
  public void visit(Monitor monitor) {
    System.out.println("Displaying Monitor.");
```

```
  }
}
public class VisitorPatternDemo {
  public static void main(String[] args) {

    ComputerPart computer = new Computer();
    computer.accept(new ComputerPartDisplayVisitor());
  }
}
```

Output:

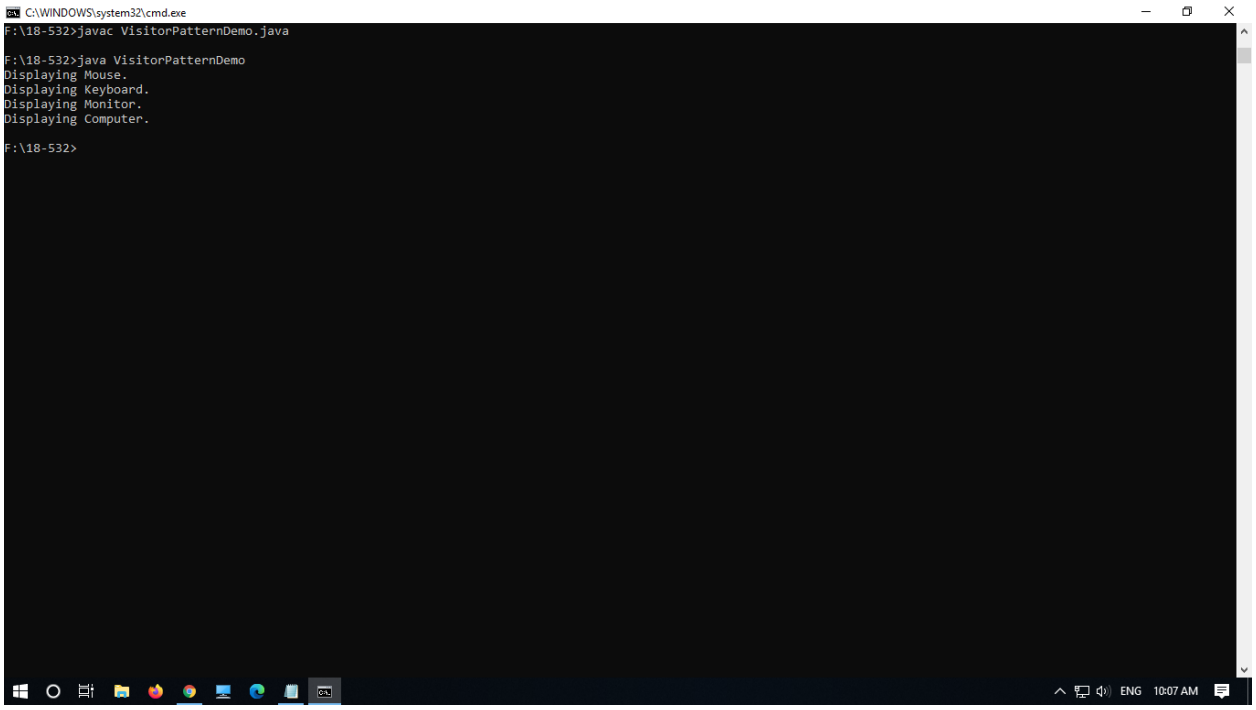
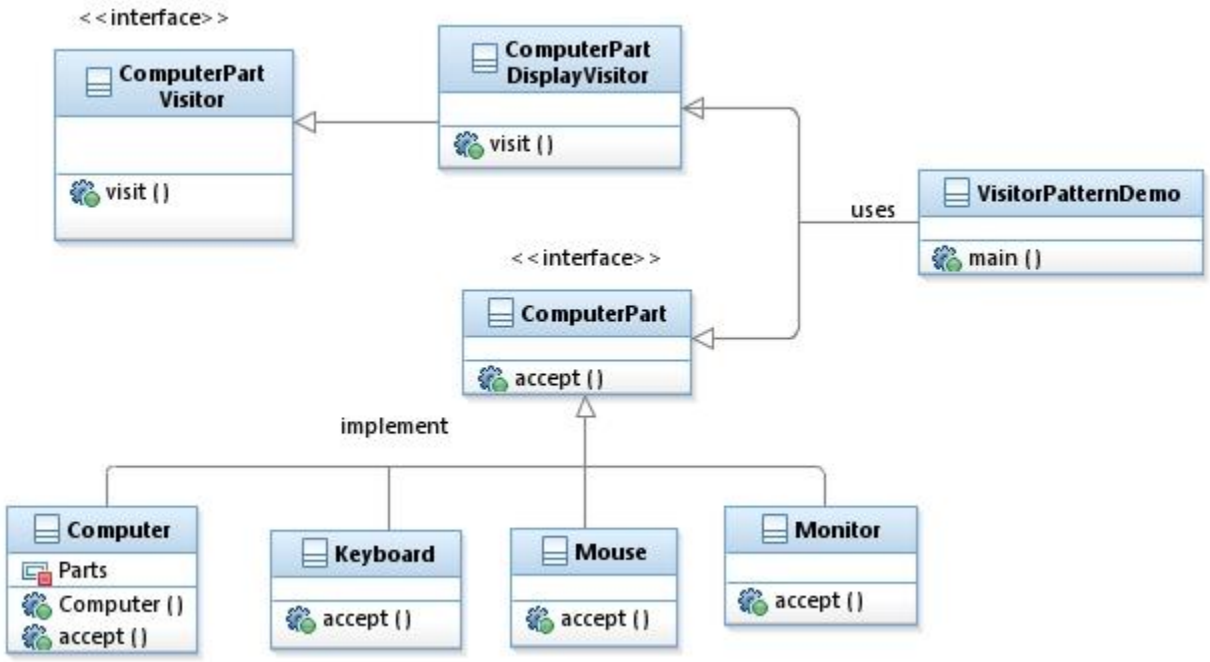


Diagram:


<<interface>>
ComputerPart Visitor
visit ()
ComputerPart DisplayVisitor
visit ()
uses
VisitorPatternDemo
main ()
<<interface>>
ComputerPart
accept ()
implement
Computer
Parts
Computer ()
accept ()
Keyboard
accept ()
Mouse
accept ()
Monitor
accept ()