Bhartiya Vidya Bhavan's

Sardar Patel Institute of Technology

(Autonomous Institute Affiliated to University of Mumbai)

[Knowledge is Nectar]

Department of Computer Engineering

ISE – 2 (Open Book) **Max Marks:10**

Class: TE COMP Semester:V
Course Name: System Analysis & Design Course Code: CE52
Date: 28-Oct-2020 Academic Year: 2020-21

Note: Upload assignment with name and UID written on top right corner. Draw diagrams using scale.
Create 2 documents, one for question 1 and second for question 2 .
Upload question 1 and 2 separately on moodle under topic ISE2 Exam.

Name: Vishal Shashikant Salvi
UID: 2019230069
SE COMPS
BATCH C
52

Q2) Implement Strategy design pattern using a simple Shopping Cart that uses two payment strategies – using Credit Card or using Paytm. Create the interface for strategy pattern that has method pay which takes payment amount as argument. Pay method prints the amount and payment method used.

CreditCard payment uses your name, credit card number, cvv and expiry date
Paytm payment uses your name and password.
Shopping cart can add and remove item.

5Marks

Code:

PaymentStrategy.java

```
public interface PaymentStrategy {

        public void pay(int amount);
}
```

CreditCardStrategy.java

```
public class CreditCardStrategy implements PaymentStrategy {

	private String name;
	private String cardNumber;
	private String cvv;
	private String dateOfExpiry;

	public CreditCardStrategy(String nm, String ccNum, String cvv, String expiryDate){
		this.name=nm;
		this.cardNumber=ccNum;
		this.cvv=cvv;
		this.dateOfExpiry=expiryDate;
	}

	public void pay(int amount) {
		System.out.println(amount +" paid with credit card");
	}

}
```

PaytmStrategy.java

```
public class PaytmStrategy implements PaymentStrategy {

	private String name;
	private String password;

	public PaytmStrategy(String nm, String pwd){
		this.name=nm;
		this.password=pwd;
	}


	public void pay(int amount) {
		System.out.println(amount + " paid using Paytm.");
	}

}
```

Item.java

```
public class Item {

	private String upcCode;
	private int price;

	public Item(String upc, int cost){
		this.upcCode=upc;
		this.price=cost;
	}

	public String getUpcCode() {
		return upcCode;
	}

	public int getPrice() {
		return price;
	}

}
```

ShoppingCart.java

```
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {


	List<Item> items;

	public ShoppingCart(){
		this.items=new ArrayList<Item>();
	}

	public void addItem(Item item){
		this.items.add(item);
	}

	public void removeItem(Item item){
		this.items.remove(item);
	}

	public int calculateTotal(){
		int sum = 0;
		for(Item item : items){
			sum += item.getPrice();
		}
		return sum;
```

```
	}

	public void pay(PaymentStrategy paymentMethod){
		int amount = calculateTotal();
		paymentMethod.pay(amount);
	}
}
```

ShoppingCartDetails.java

```
public class ShoppingCartDetails {

	public static void main(String[] args) {
		ShoppingCart cart = new ShoppingCart();

		Item item1 = new Item("1764",50);
		Item item2 = new Item("5898",100);

		cart.addItem(item1);
		cart.addItem(item2);
		cart.removeItem(item1);
		cart.removeItem(item2);


		//pay by paytm
		cart.pay(new PaytmStrategy("Vishal Salvi", "Vishal@pwd"));

		//pay by credit card
		cart.pay(new CreditCardStrategy("Vishal Salvi", "1234567843218765", "172",
"12/24"));
	}

}
```

Output:

```
150 paid using Paytm.
150 paid with credit card
```

Strategy design Pattern

1)A Strategy Pattern says that "defines a family of functionality, encapsulate each one, and make them interchangeable".

2)In Strategy pattern, a class behaviour or its algorithm can be changed at run time. This type of design pattern comes under behaviour pattern.

3)In Strategy pattern, we create objects which represent various strategies and a context object whose behaviour varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

Benefits:

1. It provides a substitute to subclassing.
2. It defines each behavior within its own class, eliminating the need for conditional statements.
3. It makes it easier to extend and incorporate new behavior without changing the application.