ABEDA INAMDAR SENIOR COLLEGE OF ARTS, SCIENCE & COMMERCE (AUTONOMOUS)
COMPUTER SCIENCE DEPARTMENT
MSC (COMPUTER SCIENCE-II) SEM-III

INDEX

Sr.no	Title	Remark	Signature
1	Write a JAVA Program to implement observer design pattern consider Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(), getPressure()		
2	Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.		
3	Write a Java Program to implement Factory method for Pizza Store with createPizza(), orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.		
4	Write a Java Program to implement Singleton pattern for multithreading.		
5	Write a Java Program to implement command pattern to test Remote Control		
6	Write a Java Program to implement undo command to test Ceiling fan		
7	Write a Java Program to implement Adapter pattern for Enumeration iterator.		
8	Write a Java Program to implement Strategy Pattern to perform arithmetic operation		
9	Write a Java Program to implement State Pattern for providing connection to controller class		
10	Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviours and state transition that can happen. For actions we need to implement methods to insert a quarter, remove a quarter, turning the crank and display gumball		

1. Write a JAVA Program to implement observer design pattern consider Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(), getPressure().

Weather_station.java

```
package Observer;

public class Weather_station {
	int temparature,humidity,pressure;

	public Weather_station(int temparature, int humidity, int pressure) {
		this.temparature = temparature;
		this.humidity = humidity;
		this.pressure = pressure;
	}

	public void measurementsChanged()
	{
		System.out.println("Temprature"+temparature+"\nHumidity:"+humidity+"\n
Measure:"+pressure);
	}

	public void getTemperature()
	{
		System.out.println("Temparature:"+temparature);
	}

	public void getHumidity()
	{
		System.out.println("Humidity:"+humidity);
	}

	public void getPressure()
	{
		System.out.println("Measurement:"+pressure);
	}

	public void setMeaturement(int t, int h, int p) {
		// TODO Auto-generated method stub
		temparature=t;
		humidity=h;
		pressure=p;
	}
}
```

Main.java

```
package Observer;

public class Main {
	public static void main(String[] args) {
		Weather_station w1=new Weather_station(20, 7, 5);
		w1.getTemperature();
		w1.getHumidity();
		w1.getPressure();
		w1.setMeaturement(5, 7, 9);
	}
}
```

OUTPUT:

Temparature:20
Humidity:7
Measurement:5

2. Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

LowerCaseInputStream.java

```
package Decorator;
import java.io.*;
public class LowerCaseInputStream extends FilterInputStream
{
	protected LowerCaseInputStream(InputStream in)
	{
		super(in);
	}

	public int read() throws IOException
	{
		int c=super.read();
		return (c==-1 ? c:Character.toLowerCase((char)c));
	}

	public int read(byte[] b,int offset,int len) throws IOException
	{
		int result=super.read(b,offset,len);
		for(int i=offset;i<offset+result;i++)
		{
			b[i]=(byte)Character.toLowerCase((char)b[i]);
		}
		return result;
	}
}
```

Inputtest.Java

```
package Decorator;

import java.io.*;

public class Inputtest {

	public static void main(String[] args)
	{
		int c;
		try
		{
			InputStream is = new LowerCaseInputStream(new BufferedInputStream
(new FileInputStream("C:\\Users\\Arbaaz\\eclipse
workspace\\Programs\\Decorator\\src\\Deco\\file1.txt")));
			while((c=is.read())>=0)
			{
				System.out.print((char)c);

			}
			is.close();
		}
		catch(IOException e)
		{
			e.printStackTrace();
		}
	}
}
```

file1.txt

```
HELLO WORLD
```

OUTPUT:

```
hello world
```

3. Write a Java Program to implement Factory method for Pizza Store with createPizza(), orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

PizzaStore.java

```
package pizza;

public abstract class PizzaStore
{
	abstract Pizza createPizza(String item);
	public Pizza orderPizza(String type)
	{
		Pizza pizza = createPizza(type);
		System.out.println("--- Making a " + pizza.getName() + " ---");
		pizza.prepare();
		pizza.bake();
		pizza.cut();
		pizza.box();
		return pizza;
	}
}
```

Pizza.java

```
package pizza;

import java.util.ArrayList;
public abstract class Pizza
{
	String name;
	String dough;
	String sauce;
	ArrayList<String> toppings = new ArrayList<String>();
	void prepare()
	{
		System.out.println("Preparing " + name);
		System.out.println("Tossing dough...");
		System.out.println("Adding sauce...");
		System.out.println("Adding toppings: ");
		for (int i = 0; i < toppings.size(); i++)
		{
			System.out.println(" " + toppings.get(i));
		}
	}

	void bake()
	{
		System.out.println("Bake for 25 minutes at 350");
	}

	void cut()
	{

		System.out.println("Cutting the pizza into diagonal slices");
	}

	void box()
	{
		System.out.println("Place pizza in official PizzaStore box");
	}
	public String getName()
	{
		return name;
	}
	public String toString()
	{
```

```
                StringBuffer display = new StringBuffer();
                display.append("---- " + name + " ----\n");
                display.append(dough + "\n");
                display.append(sauce + "\n");
                for (int i = 0; i < toppings.size(); i++)
                {
                        display.append((String )toppings.get(i) + "\n");
                }
                return display.toString();
        }
}
```

ChicagoPizzaStore.java

```
package pizza;

public class ChicagoPizzaStore extends PizzaStore
{
        Pizza createPizza(String item)
        {
                if (item.equals("cheese"))
                {
                        return new ChicagoStyleCheesePizza();
                }
                else if (item.equals("veggie"))
                {
                        return new ChicagoStyleVeggiePizza();
                }
                else if (item.equals("clam"))
                {
                        return new ChicagoStyleClamPizza();
                }
                else if (item.equals("pepperoni"))
                {
                        return new ChicagoStylePepperoniPizza();
                }
                else return null;
        }
}
```

ChicagoStyleCheesePizza.java

```
package pizza;

public class ChicagoStyleCheesePizza extends Pizza {

        public ChicagoStyleCheesePizza()
        {
                name = "Chicago Style Cheese Pizza";
                dough = "Extra Thick Crust Dough";
                sauce = "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella Cheese");
                toppings.add("Black Olives");
                toppings.add("Spinach");
                toppings.add("Eggplant");
                toppings.add("Sliced Pepperoni");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}
```

ChicagoStyleClamPizza.java

```
package pizza;

public class ChicagoStyleClamPizza extends Pizza {

        public ChicagoStyleClamPizza()
```

```
        {
                name = "Chicago Style Clam Pizza";
                dough = "Extra Thick Crust Dough";
                sauce = "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella Cheese");
                toppings.add("Black Olives");
                toppings.add("Spinach");
                toppings.add("Eggplant");
                toppings.add("Sliced Pepperoni");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}
```

ChicagoStylePepperoniPizza.java

```
package pizza;

public class ChicagoStylePepperoniPizza extends Pizza
{
        public ChicagoStylePepperoniPizza()
        {
                name = "Chicago Style Pepperoni Pizza";
                dough = "Extra Thick Crust Dough";
                sauce = "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella Cheese");
                toppings.add("Black Olives");
                toppings.add("Spinach");
                toppings.add("Eggplant");
                toppings.add("Sliced Pepperoni");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}
```

ChicagoStyleVeggiePizza.java

```
package pizza;

public class ChicagoStyleVeggiePizza extends Pizza {
        public ChicagoStyleVeggiePizza()
        {
                name = "Chicago Style Veggie Pizza";
                dough = "Extra Thick Crust Dough";
                sauce = "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella Cheese");
                toppings.add("Black Olives");
                toppings.add("Spinach");
                toppings.add("Eggplant");
                toppings.add("Sliced Pepperoni");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}
```

NYPizzaStore.java

```
package pizza;

public class NYPizzaStore extends PizzaStore
{
        Pizza createPizza(String item)
        {
                if (item.equals("cheese"))
```

```
                {
                        return new NYStyleCheesePizza();
                } else if (item.equals("veggie"))
                {
                        return new NyStyleVeggiePizza();
                }
                else if (item.equals("clam"))
                {
                        return new NYStyleClamPizza();
                } else if (item.equals("pepperoni"))
                {
                        return new NYstylePepperOniPizza();
                }
                else return null;
        }
}
```

NYStyleCheesePizza.java

```
package pizza;

public class NYStyleCheesePizza extends Pizza
{
        public NYStyleCheesePizza()
        {
                name = "NY Style Sauce and Cheese Pizza";
                dough = "Thin Crust Dough";
                sauce = "Marinara Sauce";
                toppings.add("Grated Reggiano Cheese");
        }
}
```

NYStyleClamPizza.java

```
package pizza;

public class NYStyleClamPizza extends Pizza
{
        public NYStyleClamPizza()
        {
                name = "NY Style Clam Pizza";
                dough = "Thin Crust Dough";
                sauce = "Marinara Sauce";
                toppings.add("Grated Reggiano Cheese");
                toppings.add("Fresh Clams from Long Island Sound");
        }
}
```

NYstylePepperOniPizza.java

```
package pizza;

public class NYstylePepperOniPizza extends Pizza
{
        public NYstylePepperOniPizza()
        {
                name = "NY Style Pepperoni Pizza";
                dough = "Thin Crust Dough";
                sauce = "Marinara Sauce";
                toppings.add("Grated Reggiano Cheese");
                toppings.add("Sliced Pepperoni");
                toppings.add("Garlic");
                toppings.add("Onion");
                toppings.add("Mushrooms");
                toppings.add("Red Pepper");
        }
}
```

NyStyleVeggiePizza.java

```
package pizza;

public class NyStyleVeggiePizza extends Pizza
{
	public NyStyleVeggiePizza()
	{
		name = "NY Style Veggie Pizza";
		dough = "Thin Crust Dough";
		sauce = "Marinara Sauce";
		toppings.add("Grated Reggiano Cheese");
		toppings.add("Garlic");
		toppings.add("Onion");
		toppings.add("Mushrooms");
		toppings.add("Red Pepper");
	}
}
```

DependentPizzaStore.java

```
package pizza;

public class DependentPizzaStore
{
	public Pizza createPizza(String style, String type)
	{
		Pizza pizza = null;
		if (style.equals("NY"))
		{
			if (type.equals("cheese"))
			{
				pizza = new NYStyleCheesePizza();

			}
			else if (type.equals("veggie"))
			{
				pizza = new NyStyleVeggiePizza();
			}
			else if (type.equals("clam"))
			{
				pizza = new NYStyleClamPizza();
			}
			else if (type.equals("pepperoni"))
			{
				pizza = new NYstylePepperOniPizza();
			}
		}
		else if (style.equals("Chicago"))
		{
			if (type.equals("cheese"))
			{
				pizza = new ChicagoStyleCheesePizza();
			}
			else if (type.equals("veggie"))
			{
				pizza = new ChicagoStyleVeggiePizza();
			}
			else if (type.equals("clam"))
			{
				pizza = new ChicagoStyleClamPizza();
			}
			else if (type.equals("pepperoni"))
			{
				pizza = new ChicagoStylePepperoniPizza();
			}
		}
		else
```

```
                {
                        System.out.println("Error: invalid type of pizza");
                        return null;
                }
                pizza.prepare();
                pizza.bake();
                pizza.cut();
                pizza.box();
                return pizza;
        }
}
```

PizzaTestDrive.java

```
package pizza;

public class PizzaTestDrive
{
        public static void main(String[] args)
        {
                PizzaStore nyStore = new NYPizzaStore();
                PizzaStore chicagoStore = new ChicagoPizzaStore();
                Pizza pizza = nyStore.orderPizza("cheese");
                System.out.println("Ethan ordered a " + pizza.getName() + "\n");
                pizza = chicagoStore.orderPizza("cheese");
                System.out.println("Joel ordered a " + pizza.getName() + "\n");
                pizza = nyStore.orderPizza("clam");
                System.out.println("Ethan ordered a " + pizza.getName() + "\n");
                pizza = chicagoStore.orderPizza("clam");
                System.out.println("Joel ordered a " + pizza.getName() + "\n");
                pizza = nyStore.orderPizza("pepperoni");
                System.out.println("Ethan ordered a " + pizza.getName() + "\n");
                pizza = chicagoStore.orderPizza("pepperoni");
                System.out.println("Joel ordered a " + pizza.getName() + "\n");
                pizza = nyStore.orderPizza("veggie");
                System.out.println("Ethan ordered a " + pizza.getName() + "\n");
                pizza = chicagoStore.orderPizza("veggie");
                System.out.println("Joel ordered a " + pizza.getName() + "\n");
        }
}
```

OUTPUT:

```
--- Making a NY Style Sauce and Cheese Pizza ---
Preparing NY Style Sauce and Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Grated Reggiano Cheese
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Sauce and Cheese Pizza

--- Making a Chicago Style Cheese Pizza ---
Preparing Chicago Style Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Shredded Mozzarella Cheese
 Black Olives
 Spinach
 Eggplant
 Sliced Pepperoni
```

```
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Cheese Pizza

--- Making a NY Style Clam Pizza ---
Preparing NY Style Clam Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Grated Reggiano Cheese
 Fresh Clams from Long Island Sound
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Clam Pizza

--- Making a Chicago Style Clam Pizza ---
Preparing Chicago Style Clam Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Shredded Mozzarella Cheese
 Black Olives
 Spinach
 Eggplant
 Sliced Pepperoni
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Clam Pizza

--- Making a NY Style Pepperoni Pizza ---
Preparing NY Style Pepperoni Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Grated Reggiano Cheese
 Sliced Pepperoni
 Garlic
 Onion
 Mushrooms
 Red Pepper
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Pepperoni Pizza

--- Making a Chicago Style Pepperoni Pizza ---
Preparing Chicago Style Pepperoni Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Shredded Mozzarella Cheese
 Black Olives
 Spinach
 Eggplant
 Sliced Pepperoni
Bake for 25 minutes at 350
```

```
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Pepperoni Pizza

--- Making a NY Style Veggie Pizza ---
Preparing NY Style Veggie Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Grated Reggiano Cheese
 Garlic
 Onion
 Mushrooms
 Red Pepper
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Veggie Pizza

--- Making a Chicago Style Veggie Pizza ---
Preparing Chicago Style Veggie Pizza
Tossing dough...
Adding sauce...
Adding toppings:
 Shredded Mozzarella Cheese
 Black Olives
 Spinach
 Eggplant
 Sliced Pepperoni
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Veggie Pizza
```

4. Write a Java Program to implement Singleton pattern for multithreading.

Singleton.java

```
package singleton;

public class Singleton
{
	private static Singleton instance;
	public String value;
	private Singleton(String value)
	{
		this.value=value;
	}
	public static Singleton getinstance(String value)
	{
		Singleton result= instance;

		if(result !=null)
		{
			return result;
		}
		synchronized(Singleton.class)
		{
			if (instance == null)
			{
				instance = new Singleton(value);
			}
			return instance;
		}
	}
}
```

DemoMultiThreading.java

```
package singleton;

public class DemoMultiThreading
{
	public static void main(String[] args)
	{
		System.out.println("If you see the same value then singleton was reused "
+ "\n" + "If you see two different values then 2 singleton were created"+"\n"+
"RESULT:");

		Thread t1 = new Thread(new t1());
		Thread t2 = new Thread(new t2());

		t1.start();
		t2.start();
	}

	static class t1 implements Runnable
	{
		@Override
		public void run()
		{
			Singleton singleton= Singleton.getinstance("Abeda");
			System.out.println(singleton.value);
		}
	}

	static class t2 implements Runnable
	{
		@Override
		public void run()
		{
			Singleton singleton= Singleton.getinstance("Inamdar");
			System.out.println(singleton.value);
```

```
                }
        }
}
```

OUTPUT:

```
If you see the same value then singleton was reused
If you see two different values then 2 singleton were created
RESULT:
Abeda
Abeda
```

5. Write a Java Program to implement command pattern to test Remote Control

Command.java

```
package remotecontrol;

public interface Command {
        public void execute();
}
```

Ac.java

```
package remotecontrol;

public class Ac {
        String a;
        public Ac(String string)
        {
                this.a= string;
        }

        public void powerOn()
        {
                System.out.println(a+" "+ "AC is ON.");
        }

        public void powerOff()
        {
                System.out.println(a+" "+ "AC is OFF.");
        }

}
```

AcOffCommand.java

```
package remotecontrol;

public class AcOffCommand implements Command{
        Ac ac;
        public AcOffCommand (Ac ac)
        {
                this.ac=ac;
        }
        public void execute()
        {
                ac.powerOff();
        }
}
```

AcOnCommand.java

```
package remotecontrol;

public class AcOnCommand implements Command{
        Ac ac;
        public AcOnCommand (Ac ac)
        {
                this.ac=ac;
```

```
        }
        public void execute()
        {
                ac.powerOn();
        }
}
```

Light.java

```
package remotecontrol;

public class Light
{
        String a;
        public Light(String string)
        {
                this.a=string;
        }

        public void on()
        {
                System.out.print(a+" " + "Light is On.");
        }

        public void off()
        {
                System.out.print(a+" " + "Light is Off.");
        }
}
```

LightOffCommand.java

```
package remotecontrol;

public class LightOffCommand implements Command
{
        Light light;

        public LightOffCommand(Light light) {
                this.light = light;
        }

        public void execute()
        {
                light.off();
        }
}
```

LightOnCommand.java

```
package remotecontrol;

public class LightOnCommand implements Command
{
        Light light;

        public LightOnCommand(Light light) {
                this.light = light;
        }

        public void execute()
        {
                light.on();
        }
}
```

NoCommand.java

```
package remotecontrol;

public class NoCommand implements Command
{
```

```
        public void execute()
        {

        }

}
```

RemoteControl.java

```
package remotecontrol;

public class RemoteControl {
        Command[] onCommands;
        Command[] offCommands;
        public RemoteControl()
        {

                onCommands = new Command[3];
                offCommands = new Command[3];

                Command noCommand = new NoCommand();
                for(int i=0;i<3;i++)
                {
                        onCommands[i]=noCommand;
                        offCommands[i]=noCommand;
                }

        }
        public void setCommand(int slot,Command onCommand, Command offCommand)
        {
                onCommands[slot]=onCommand;
                offCommands[slot]=offCommand;
        }

        public void onButtonPushed(int slot)
        {
                onCommands[slot].execute();
        }
        public void offButtonPushed(int slot)
        {
                offCommands[slot].execute();
        }
        public String toString()
        {
                StringBuffer stringBuff= new StringBuffer();
                stringBuff.append("\n ----------------- Remote Control --------------
\n");
                for(int i=-0; i< onCommands.length; i++)
                {
                        stringBuff.append("[Slot " + i
+"]"+onCommands[i].getClass().getName() + "\n");
                }
                return stringBuff.toString();
        }
}
```

RemoteLoader.java

```
package remotecontrol;

public class RemoteLoader {
        public static void main(String args[])
        {
                RemoteControl remoteControl=new RemoteControl();
                Light KitchenRoomLight= new Light("Living Room");
                Ac BedRoomAc = new Ac("Bed Room");

                //Kitchen Room light on and off
```

```
                LightOnCommand kitchenRoomLightOn=new LightOnCommand(KitchenRoomLight);
                LightOffCommand kitchenRoomLightOff=new LightOffCommand(KitchenRoomLight);

                // Bedroom AC On and Off
                AcOnCommand BedRoomAcOn=new AcOnCommand(BedRoomAc);
                AcOffCommand BedRoomAcOff=new AcOffCommand(BedRoomAc);

                remoteControl.setCommand(0, kitchenRoomLightOn, kitchenRoomLightOff);
                remoteControl.setCommand(1, BedRoomAcOn, BedRoomAcOff);

                System.out.println(remoteControl);

                remoteControl.onButtonPushed(0);
                remoteControl.offButtonPushed(0);
                remoteControl.onButtonPushed(1);
                remoteControl.offButtonPushed(1);

        }

}
```

OUTPUT:

```
----------------- Remote Control ---------------
[Slot 0]remotecontrol.LightOnCommand
[Slot 1]remotecontrol.AcOnCommand
[Slot 2]remotecontrol.NoCommand

Living Room Light is On.
Living Room Light is Off.
Bed Room AC is ON.
Bed Room AC is OFF.
```

6. Write a Java Program to implement undo command to test Ceiling fan.

Command.java

```
package Command;

public interface Command {
        public void execute();
        public void undo();
}
```

NoCommand.java

```
package Command;

public class NoCommand implements Command
{
        public void execute()
        {}
        public void undo()
        {}
}
```

CeilingFan.java

```
package Command;

public class CeilingFan {
    public static final int OFF = 0;
    public static final int LOW = 1;
    public static final int MEDIUM = 2;
    public static final int HIGH = 3;
    String location;
    int speed;
    public CeilingFan(String location) {
        this.location = location;
        speed = OFF;
    }
    public void off() {
        speed = OFF;
        System.out.println(location + "Fan is off.");
    }
    public void low() {
        speed = LOW;
        System.out.println(location + "Fan is low.");
    }
    public void medium() {
        speed = MEDIUM;
        System.out.println(location + "Fan is medium.");
    }
    public void high() {
        speed = HIGH;
        System.out.println(location + "Fan is high.");
    }
    public int getSpeed() {
        return speed;
    }
}
```

CeilingFanHigh.java

```
package Command;

public class CeilingFanHigh implements Command {
    CeilingFan ceilingFan;
    int prevSpeed;
    public CeilingFanHigh(CeilingFan ceilingFan) {
        this.ceilingFan = ceilingFan;
    }
    public void execute() {
```

```
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.medium();
    }
    public void undo() {
        if (prevSpeed == CeilingFan.OFF)
            ceilingFan.off();
        else if (prevSpeed == CeilingFan.LOW)
            ceilingFan.low();
        else if (prevSpeed == CeilingFan.MEDIUM)
            ceilingFan.medium();
        else if (prevSpeed == CeilingFan.HIGH)
            ceilingFan.high();
    }
}
```

CeilingFanLow.java

```
package Command;

public class CeilingFanLow implements Command {
    CeilingFan ceilingFan;
    int prevSpeed;
    public CeilingFanLow(CeilingFan ceilingFan) {
        this.ceilingFan = ceilingFan;
    }
    public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.low();
    }
    public void undo() {
        if (prevSpeed == CeilingFan.OFF)
            ceilingFan.off();
        else if (prevSpeed == CeilingFan.LOW) ceilingFan.low();
        else if (prevSpeed == CeilingFan.MEDIUM)
            ceilingFan.medium();
        else if (prevSpeed == CeilingFan.HIGH)
            ceilingFan.high();
    }
}
```

CeilingFanMedium.java

```
package Command;

public class CeilingFanMedium implements Command {
    CeilingFan ceilingFan;
    int prevSpeed;
    public CeilingFanMedium(CeilingFan ceilingFan) {
        this.ceilingFan = ceilingFan;
    }
    public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.medium();
    }
    public void undo() {
        if (prevSpeed == CeilingFan.OFF)
            ceilingFan.off();
        else if (prevSpeed == CeilingFan.LOW)
            ceilingFan.low();
        else if (prevSpeed == CeilingFan.MEDIUM)
            ceilingFan.medium();
        else if (prevSpeed == CeilingFan.HIGH)
            ceilingFan.high();
    }
}
```

CeilingFanOff.java

```
package Command;
```

```java
public class CeilingFanOff implements Command {
    CeilingFan ceilingFan;
    int prevSpeed;
    public CeilingFanOff(CeilingFan ceilingFan) {
        this.ceilingFan = ceilingFan;
    }
    public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.off();
    }
    public void undo() {
        if (prevSpeed == CeilingFan.OFF)
            ceilingFan.off();
        else if (prevSpeed == CeilingFan.LOW)
            ceilingFan.low();
        else if (prevSpeed == CeilingFan.MEDIUM)
            ceilingFan.medium();
        else if (prevSpeed == CeilingFan.HIGH)
            ceilingFan.high();
    }
}
```

RemoteControl.java

```java
package Command;

public class RemoteControl {
    Command[] onCommands;
    Command[] offCommands;
    Command undoCommands;
    public RemoteControl() {
        onCommands = new Command[5];
        offCommands = new Command[5];
        Command noCommand = new NoCommand();
        //NoCommand object is for null object
        for (int i = 0; i < 5; i++) {
            onCommands[i] = noCommand;
            offCommands[i] = noCommand;
        }
        undoCommands = noCommand;
    }
    public void setCommand(int slot, Command onCommand, Command offCommand) {
        onCommands[slot] = onCommand;
        offCommands[slot] = offCommand;
    }
    public void onButtonPushed(int slot) {
        onCommands[slot].execute();
        undoCommands = onCommands[slot];
    }
    public void offButtonPushed(int slot) {
        offCommands[slot].execute();
        undoCommands = offCommands[slot];
    }
    public void undoButtonPushed() {
        undoCommands.undo();
    }
    public String toString() {
        StringBuffer strBuff = new StringBuffer();
        strBuff.append("\n----------Remote Control---------\n");
        for (int i = 0; i < onCommands.length; i++) {
            strBuff.append("[slot" + i + ")" + onCommands[i].getClass().getName() + " "
+ offCommands[i].getClass().getName() + "\n");
            return strBuff.toString();
        }
        return null;
    }
}
```

RemoteLoader.java

```
package Command;

public class RemoteLoader {
    public static void main(String[] args) {
        RemoteControl remoteControl = new RemoteControl();
        CeilingFan ceilingFan = new CeilingFan("Living Room.");
        CeilingFanOff ceilignFanOff = new CeilingFanOff(ceilingFan);
        CeilingFanLow ceilignFanLow = new CeilingFanLow(ceilingFan);
        CeilingFanMedium ceilingFanMedium = new CeilingFanMedium(ceilingFan);
        CeilingFanHigh ceilingFanHigh = new CeilingFanHigh(ceilingFan);
        remoteControl.setCommand(0, ceilignFanLow, ceilignFanOff);
        remoteControl.setCommand(1, ceilingFanMedium, ceilignFanOff);
        remoteControl.setCommand(2, ceilingFanHigh, ceilignFanOff);
        remoteControl.onButtonPushed(0);
        remoteControl.offButtonPushed(0);
        System.out.println(remoteControl);
        remoteControl.undoButtonPushed();
        remoteControl.onButtonPushed(1);
        remoteControl.offButtonPushed(1);
        System.out.println(remoteControl);
        remoteControl.undoButtonPushed();
        remoteControl.onButtonPushed(2);
        remoteControl.offButtonPushed(2);
        System.out.println(remoteControl);
        remoteControl.undoButtonPushed();
    }
}
```

OUTPUT:

```
Living Room.Fan is low.
Living Room.Fan is off.

-----------Remote Control---------
[slot0)Command.CeilingFanLow Command.CeilingFanOff

Living Room.Fan is low.
Living Room.Fan is medium.
Living Room.Fan is off.

-----------Remote Control---------
[slot0)Command.CeilingFanLow Command.CeilingFanOff

Living Room.Fan is medium.
Living Room.Fan is medium.
Living Room.Fan is off.

-----------Remote Control---------
[slot0)Command.CeilingFanLow Command.CeilingFanOff

Living Room.Fan is medium.
```

7. Write a Java Program to implement Adapter pattern for Enumeration iterator.

EnumProduct.java

```
package Adaptor;

import java.util.Enumeration;
import java.util.Vector;

public class EnumProduct {

	private Vector product;

	public EnumProduct()
	{
		System.out.println("Implemeting adaptor pattern for enumeratior");

		product= new Vector();

		setProduct("ProductA:Laptop");
		setProduct("ProductB:Moblile");
		setProduct("ProductC:Tablets");
		setProduct("ProductD:Router");
	}

	public void setProduct(String s)
	{
		product.add(s);
	}

	public Enumeration getProduct()
	{
		Enumeration eproduct=product.elements();
		return eproduct;
	}
}
```

EnumToIteratorAdapter.java

```
package Adaptor;

import java.util.Enumeration;
import java.util.Iterator;
public class EnumToIteratorAdapter implements Iterator {

	Enumeration enumA;
	public EnumToIteratorAdapter(Enumeration e)
	{
		enumA=e;
	}
	public boolean hasNext()
	{
		return enumA.hasMoreElements();
	}
	public Object next()
	{
		return enumA.nextElement();
	}
	public void remove()
	{
		throw new UnsupportedOperationException();
	}
}
```

Product.java

```
package Adaptor;

import java.util.Iterator;
```

```java
public class Product {

	public void displayProduct(Iterator iterator)
	{
		for(;iterator.hasNext();)
			System.out.println(iterator.next());
	}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Product product=new Product();
		EnumProduct enumproduct=new EnumProduct();
		EnumToIteratorAdapter enumToIteratorAdaptor=new
EnumToIteratorAdapter(enumproduct.getProduct());
		product.displayProduct(enumToIteratorAdaptor);
	}

}
```

OUTPUT:

```
Implemeting adaptor pattern for enumeratior
ProductA:Laptop
ProductB:Moblile
ProductC:Tablets
ProductD:Router
```

8. Write a Java Program to implement Strategy Pattern to perform arithmetic operation.

Strategy.java

```
package strategy;

public interface Strategy {
	public int doOperation(int num1, int num2);

}
```

OperationAdd.java

```
package strategy;

public class OperationAdd implements Strategy {
	public int doOperation(int num1, int num2)
	{
		return num1+num2;

	}
}
```

OpeartionMul.java

```
package strategy;

public class OperationMul implements Strategy {
	public int doOperation(int num1, int num2)
	{
		return num1*num2;

	}

}
```

OperationSub.java

```
package strategy;

public class OperationSub implements Strategy {
	public int doOperation(int num1, int num2)
	{
		return num1-num2;

	}
}
```

Context.java

```
package strategy;

public class Context {
	private Strategy strategy;
	public Context (Strategy strategy)
	{
		this.strategy=strategy;

	}
	public int executeStrategy(int num1, int num2)
	{
		return strategy.doOperation(num1, num2);
	}

}
```

StrategyPatternDemo.java

```
package strategy;

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

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

OUTPUT:

```
10+5=15
10-5=5
10*5=50
```

9. Write a Java Program to implement State Pattern for providing connection to controller class.

Connection.java

```
package state;

public interface Connection {
        public void open();
        public void close();
        public void log();
        public void update();
}
```

Accounting.java

```
package state;

public class Accounting implements Connection {

    @Override
    public void open() {
        System.out.println("Open Database for accounting");
    }

    @Override
    public void close() {
        System.out.println("Close Database for accounting");
    }

    @Override
    public void log() {
        System.out.println(" log activities");
    }
    @Override
    public void update() {
        System.out.println("Accountiing has been updated");
    }
}
```

Management.java

```
package state;

public class Management implements Connection {

    @Override
    public void open() {
        System.out.println("Open Database for management");
    }

    @Override
    public void close() {
```

```
        System.out.println("Close Database for management");
    }

    @Override
    public void log() {
        System.out.println(" log activities");
    }
    @Override
    public void update() {
        System.out.println("Management has been updated");
    }
}
```

Sales.java

```
package state;

public class Sales implements Connection {

    @Override
    public void open() {
        System.out.println("Open Database for sales");
    }

    @Override
    public void close() {
        System.out.println("Close Database for sales");
    }

    @Override
    public void log() {
        System.out.println("log activities");
    }
    @Override
    public void update() {
        System.out.println("Sales has been updated");
    }
}
```

Controller.java

```
package state;

public class Controller {
        public static Accounting acc;
        public static Sales sales;
        public static Management mng;

        public static Connection con;

        Controller()
        {
                acc=new Accounting();
                sales=new Sales();
                mng=new Management();
        }

        public void setAccountingConnection()
        {
                con=acc;
        }

        public void setSalesConnection()
        {
                con=sales;
        }

        public void setManagementConnection()
        {
                con=mng;
```

```
        }

        public void open()
        {
                con.open();
        }
        public void close()
        {
                con.close();
        }

        public void log()
        {
                con.log();
        }
        public void update()
        {
                con.update();
        }
}
```

StateDemo.java

```
package state;

public class StateDemo {
    //private static final String management=null;

    Controller controller;
    StateDemo(String con) {
        controller = new Controller();
        if (con.equalsIgnoreCase("Accounting"))
            controller.setAccountingConnection();
        if (con.equalsIgnoreCase("Management"))
            controller.setManagementConnection();
        if (con.equalsIgnoreCase("Sales"))
            controller.setSalesConnection();

        controller.open();
        controller.close();
        controller.log();
        controller.update();
    }
    public static void main(String[] args) {
        String c = "sales";
        new StateDemo(c);
    }
}
```

OUTPUT:

```
Open Database for sales
Close Database for sales
log activities
Sales has been updated
```

10. Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviours and state transition that can happen. For actions we need to implement methods to insert a quarter, remove a quarter, turning the crank and display gumball.

GumballMachine.java

```
package Gumball;

public class GumballMachine {

		final static int SOLD_OUT = 0;
		final static int NO_QUARTER = 1;
		final static int HAS_QUARTER = 2;
		final static int SOLD = 3;

		int state = SOLD_OUT;
		int count = 0;

		public GumballMachine(int count) {
			this.count = count;
			if (count > 0) {
				state = NO_QUARTER;
			}
		}

		public void insertQuarter() {
			if (state == HAS_QUARTER) {
				System.out.println("You can't insert another quarter");
			} else if (state == NO_QUARTER) {
				state = HAS_QUARTER;
				System.out.println("You inserted a quarter");
			} else if (state == SOLD_OUT) {
				System.out.println("You can't insert a quarter, the machine
is sold out");
			} else if (state == SOLD) {
			System.out.println("Please wait, we're already giving you a
gumball");
			}
		}

		public void ejectQuarter() {
			if (state == HAS_QUARTER) {
				System.out.println("Quarter returned");
				state = NO_QUARTER;
			} else if (state == NO_QUARTER) {
				System.out.println("You haven't inserted a quarter");
			} else if (state == SOLD) {
				System.out.println("Sorry, you already turned the crank");
			} else if (state == SOLD_OUT) {
			System.out.println("You can't eject, you haven't inserted a quarter
yet");
			}
		}

		public void turnCrank() {
			if (state == SOLD) {
				System.out.println("Turning twice doesn't get you another
gumball!");
			} else if (state == NO_QUARTER) {
				System.out.println("You turned but there's no quarter");
			} else if (state == SOLD_OUT) {
				System.out.println("You turned, but there are no
gumballs");
			} else if (state == HAS_QUARTER) {
				System.out.println("You turned...");
```

```
                        state = SOLD;
                        dispense();
                }
        }

        private void dispense() {
                if (state == SOLD) {
                        System.out.println("A gumball comes rolling out the slot");
                        count = count - 1;
                        if (count == 0) {
                                System.out.println("Oops, out of gumballs!");
                                state = SOLD_OUT;
                        } else {
                                state = NO_QUARTER;
                        }
                } else if (state == NO_QUARTER) {
                        System.out.println("You need to pay first");
                } else if (state == SOLD_OUT) {
                        System.out.println("No gumball dispensed");
                } else if (state == HAS_QUARTER) {
                        System.out.println("No gumball dispensed");
                }
        }

        public void refill(int numGumBalls) {
                this.count = numGumBalls;
                state = NO_QUARTER;
        }

        public String toString() {
                StringBuffer result = new StringBuffer();
                //result.append("\nMighty Gumball, Inc.");
                //result.append("\nJava-enabled Standing Gumball Model #2004\n");
                result.append("Inventory: " + count + " gumball");
                if (count != 1) {
                        result.append("s");
                }
                result.append("\nMachine is ");
                if (state == SOLD_OUT) {
                        result.append("sold out");
                } else if (state == NO_QUARTER) {
                        result.append("waiting for quarter");
                } else if (state == HAS_QUARTER) {
                        result.append("waiting for turn of crank");
                } else if (state == SOLD) {
                        result.append("delivering a gumball");
                }
                result.append("\n");
                return result.toString();
        }
}
```

Main.java

```
package Gumball;

public class Main {

        public static void main(String[] args) {
                GumballMachine gumballMachine = new GumballMachine(5);

                System.out.println(gumballMachine);

                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();

                System.out.println(gumballMachine);
```

```
                gumballMachine.insertQuarter();
                gumballMachine.ejectQuarter();
                gumballMachine.turnCrank();

                System.out.println(gumballMachine);

                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();
                gumballMachine.ejectQuarter();

                System.out.println(gumballMachine);

                gumballMachine.insertQuarter();
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();

                System.out.println(gumballMachine);
        }
}
```

OUTPUT:

```
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
Quarter returned
You turned but there's no quarter
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot
You inserted a quarter
You turned...
A gumball comes rolling out the slot
You haven't inserted a quarter
Inventory: 2 gumballs
Machine is waiting for quarter

You inserted a quarter
You can't insert another quarter
You turned...
A gumball comes rolling out the slot
You inserted a quarter
You turned...
A gumball comes rolling out the slot
Oops, out of gumballs!
You can't insert a quarter, the machine is sold out
You turned, but there are no gumballs
```

```
Inventory: 0 gumballs
Machine is sold out
```