CERTIFICATE

This is to certify that Mr. _____ student of M.Sc(C.S.) Semester III at Suryadatta College of Management Information Research & Technology (SCMIRT), Pune, has successfully completed the assigned practical journal in Software Architecture And Design Pattern prescribed by the Savitribai Phule Pune University during the academic year
2022-2023.

Internal Examiner **External Examiner**

HOD **Principal**

Place: Pune

Date:

INDEX

Sr.No.	Name	Page No.	Remark	Sign
1)	Write a JAVA Program to implement built-in support (java.util.Observable) Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(), getPressure()	3 - 5		
2)	Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.	6 - 7		
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.	8 - 15		
4)	Write a Java Program to implement Singleton pattern for multithreading.	16		
5)	Write a Java Program to implement command pattern to test Remote Control.	17 - 19		
6)	Write a Java Program to implement undo command to test Ceiling fan.	20 - 27		
7)	Write a Java Program to implement Adapter pattern for Enumeration iterator.	28		
8)	Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu.	29 - 34		
9)	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, behaviors 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.	35 - 36		
10)	Write a java program to implement Adapter pattern to design Heart Model to Beat Model.	37 - 39		

1) **Write a JAVA Program to implement built-in support (java.util.Observable) Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(), getPressure()**

Ans:

main class:

```
public class WeatherStation {
   public static void main(String[] args) {
      WeatherData weatherData = new WeatherData();
      CurrentConditionsDisplay currentDisplay = new
CurrentConditionsDisplay(weatherData);
      weatherData.setMeasurements(80, 65, 30.4f);
      weatherData.setMeasurements(82, 70, 29.2f);
      weatherData.setMeasurements(78, 90, 29.2f);
   }
}
```

class CurrentConditionsDisplay:

```
public class CurrentConditionsDisplay implements Observer, DisplayElement {
   private float temperature;
   private float humidity;
   private float pressure;
   private Subject weatherData;
   public CurrentConditionsDisplay(Subject weatherData) {
      this.weatherData = weatherData;
      weatherData.registerObserver(this);
   }
   public void update(float temperature, float humidity, float pressure) {
      this.temperature = temperature;
      this.humidity = humidity;
      this.pressure = pressure;
      display();
   }
   public void display() {
      System.out.println("Current conditions: " + temperature + "F degrees and "
+ humidity + "% humidity " + pressure + "pa pressure");
   }
```

```
}
```

WeatherData class:

```
import java.util.*;
public class WeatherData implements Subject {
   private ArrayList observers;
   private float temperature;
   private float humidity;
   private float pressure;
   public WeatherData() {
      observers = new ArrayList();
   }
   public void registerObserver(Observer o) {
      observers.add(o);
   }
   public void notifyObservers() {
      for (int i = 0; i < observers.size(); i++) {
         Observer observer = (Observer)observers.get(i);
         observer.update(temperature, humidity, pressure);
      }
   }
   public void setMeasurements(float temperature, float humidity, float
pressure) {
      this.temperature = temperature;
      this.humidity = humidity;
      this.pressure = pressure;
      notifyObservers();
   }
}
```

Observer interfaces:

```
public interface Observer {
   public void update(float temp, float humidity, float pressure);
}
```

subject interface:

```
public interface Subject {
   public void registerObserver(Observer o);
   public void notifyObservers();
```

```
}
```

DisplayElement interface:

```
public interface DisplayElement {
   public void display();
}
```

Output:

```
Current conditions: 80.0F degrees and 65.0% humidity 30.4pa pressure
Current conditions: 82.0F degrees and 70.0% humidity 29.2pa pressure
Current conditions: 78.0F degrees and 90.0% humidity 29.2pa pressure
```

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

Ans:

Main class

```
// package headfirst.decorator.io;
import java.io.*;
public class InputTest {
   public static void main(String[] args) throws IOException {
      LowerCaseInputStream in = new LowerCaseInputStream("THi s Me");
      UpperCaseInputStream out = new UpperCaseInputStream("THi s Me");
      System.out.println(in.convertlower());
      System.out.println(out.convertupper());
   }
}
```

LowerCaseInputStream class:

```
// package headfirst.decorator.io;
public class LowerCaseInputStream{
   String data;
   Decorator dc = new Decorator();
   LowerCaseInputStream(String in)
   {
      this.data = in;
   }
   public String convertlower()
   {
      dc.decoratorLower();
      return data.toLowerCase();
   }
}
```

UpperCaseInputStream class :

```
// package headfirst.decorator.io;
import java.io.*;
public class UpperCaseInputStream{
   String data;
   Decorator dc = new Decorator();
   UpperCaseInputStream(String in)
   {
```

```
        this.data = in;
    }
    public String convertupper()
    {
        dc.decoratorUpper();
        return data.toUpperCase();
    }
}
```

Decorator class:

```
public class Decorator
{
    public void decoratorLower()
    {
        System.out.println("You are converting in lowercase");
    }
    public void decoratorUpper()
    {
        System.out.println("You are converting in Uppercase");
    }
}
```

Output:

```
You are converting in lowercase
thi s me
You are converting in Uppercase
THI S ME
```

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.

Ans:

Pizza.java:

```
import java.util.ArrayList;
public abstract class Pizza {
   String name;
   String dough;
   String sauce;
   ArrayList toppings = new ArrayList();
   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();
    }
}
```

pizzastore.java:

```
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;
    }
}
```

NYPizzaStore.java:

```
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;
    }
}
```

ChicagoPizzaStore.java:

```
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:

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

ChicagoStyleClamPizza.java:

```
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("Frozen Clams from Chesapeake Bay");
    }
    void cut() {
        System.out.println("Cutting the pizza into square slices");
    }
}
```

ChicagoStylePepperoniPizza.java:

```
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:

```
public class ChicagoStyleVeggiePizza extends Pizza {
    public ChicagoStyleVeggiePizza() {
        name = "Chicago Deep Dish 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");
    }
    void cut() {
        System.out.println("Cutting the pizza into square slices");
    }
}
```

NYStyleCheesePizza.java:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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 Deep Dish Cheese Pizza ---
Preparing Chicago Style Deep Dish Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Shredded Mozzarella Cheese
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Deep Dish Cheese Pizza
```

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

Ans:

SingletonClient.java:

```
public class SingletonClient {
   public static void main(String[] args) {
      Singleton singleton = Singleton.getInstance();
      System.out.println("Instance is created");
   }
}
```

Singleton.java:

```
public class Singleton {
   protected static Singleton uniqueInstance;
   // other useful instance variables here
   protected Singleton() {}
   public static synchronized Singleton getInstance() {
      if (uniqueInstance == null) {
         uniqueInstance = new Singleton();
      }
      return uniqueInstance;
   }
   // other useful methods here
}
```

Output:

Instance is created

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

Ans:

Command.java:

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

NoCommand.java:

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

CeilingFanOnCommand.java:

```
public class CeilingFanOnCommand implements Command {
   CeilingFan ceilingFan;
   public CeilingFanOnCommand(CeilingFan ceilingFan) {
      this.ceilingFan = ceilingFan;
   }
   public void execute() {
      ceilingFan.high();
   }
}
```

CeilingFanOffCommand.java:

```
public class CeilingFanOffCommand implements Command {
   CeilingFan ceilingFan;
   public CeilingFanOffCommand(CeilingFan ceilingFan) {
      this.ceilingFan = ceilingFan;
   }
   public void execute() {
      ceilingFan.off();
   }
}
```

CeilingFan.java:

```
public class CeilingFan {
   String location = "";
```

```
    int level;
    public static final int HIGH = 2;
    public CeilingFan(String location) {
        this.location = location;
    }
    public void high() {
// turns the ceiling fan on to high
        level = HIGH;
        System.out.println(location + " ceiling fan is on high");
    }
    public void off() {
// turns the ceiling fan off
        level = 0;
        System.out.println(location + " ceiling fan is off");
    }
    public int getSpeed() {
        return level;
    }
}
```

RemoteControl.java:

```
import java.util.*;
public class RemoteControl {
    Command onCommands;
    Command offCommands;
    public RemoteControl() {
        Command noCommand = new NoCommand();
        onCommands = noCommand;
        offCommands = noCommand;
    }
    public void setCommand(Command onCommand, Command offCommand)
{
        onCommands = onCommand;
        offCommands = offCommand;
    }
    public void onButtonWasPushed() {
        onCommands.execute();
    }
    public void offButtonWasPushed() {
        offCommands.execute();
```

```
    }
    public String toString() {
        StringBuffer stringBuff = new StringBuffer("[slot ] " +
onCommands.getClass().getName()+"  "+ offCommands.getClass().getName()
+ "\n");
        return stringBuff.toString();
    }
}
```

RemoteLoader.java:

```
import java.util.*;
public class RemoteLoader {
    public static void main(String[] args) {
        RemoteControl remoteControl = new RemoteControl();
        CeilingFan ceilingFan= new CeilingFan("Living Room");
        CeilingFanOnCommand ceilingFanOn = new
CeilingFanOnCommand(ceilingFan);
        CeilingFanOffCommand ceilingFanOff = new
CeilingFanOffCommand(ceilingFan);
        remoteControl.setCommand(ceilingFanOn, ceilingFanOff);
        System.out.println(remoteControl);
        remoteControl.onButtonWasPushed();
        remoteControl.offButtonWasPushed();
    }
}
```

Output:

```
[slot ] CeilingFanOnCommand  CeilingFanOffCommand

Living Room ceiling fan is on high
Living Room ceiling fan is off
```

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

Ans:

Command.java:

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

CeilingFanHighCommand.java:

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

CeilingFanLowCommand.java:

```
public class CeilingFanLowCommand implements Command {
	CeilingFan ceilingFan;
```

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

CeilingFanMediumCommand.java:

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

```
                ceilingFan.off();
            }
        }
}
```

CeilingFanOffCommand.java:

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

NoCommand.java:

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

CeilingFan.java:

```
public class CeilingFan {
        public static final int HIGH = 3;
```

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

RemoteControlWithUndo.java:

```
import java.util.*;
// This is the invoker
public class RemoteControlWithUndo {
	Command[] onCommands;
	Command[] offCommands;
	Command undoCommand;
	public RemoteControlWithUndo() {
		onCommands = new Command[7];
		offCommands = new Command[7];
```

```
		Command noCommand = new NoCommand();
		for(int i=0;i<7;i++) {
			onCommands[i] = noCommand;
			offCommands[i] = noCommand;
		}
		undoCommand = noCommand;
	}
	public void setCommand(int slot, Command onCommand, Command
offCommand) {
		onCommands[slot] = onCommand;
		offCommands[slot] = offCommand;
	}
	public void onButtonWasPushed(int slot) {
		onCommands[slot].execute();
		undoCommand = onCommands[slot];
	}
	public void offButtonWasPushed(int slot) {
		offCommands[slot].execute();
		undoCommand = offCommands[slot];
	}
	public void undoButtonWasPushed() {
		undoCommand.undo();
	}
	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()
					+ "    " + offCommands[i].getClass().getName() +
"\n");
		}
		stringBuff.append("[undo] " +
undoCommand.getClass().getName() + "\n");
		return stringBuff.toString();
	}
}
```

RemoteLoader.java:

```
public class RemoteLoader {
```

```
        public static void main(String[] args) {
                RemoteControlWithUndo remoteControl = new
RemoteControlWithUndo();
                remoteControl.onButtonWasPushed(0);
                remoteControl.offButtonWasPushed(0);
                System.out.println(remoteControl);
                remoteControl.undoButtonWasPushed();
                remoteControl.offButtonWasPushed(0);
                remoteControl.onButtonWasPushed(0);
                System.out.println(remoteControl);
                remoteControl.undoButtonWasPushed();
                CeilingFan ceilingFan = new CeilingFan("Living Room");
                CeilingFanMediumCommand ceilingFanMedium =
                                new CeilingFanMediumCommand(ceilingFan);
                CeilingFanHighCommand ceilingFanHigh =
                                new CeilingFanHighCommand(ceilingFan);
                CeilingFanOffCommand ceilingFanOff =
                                new CeilingFanOffCommand(ceilingFan);
                remoteControl.setCommand(0, ceilingFanMedium, ceilingFanOff);
                remoteControl.setCommand(1, ceilingFanHigh, ceilingFanOff);
                remoteControl.onButtonWasPushed(0);
                remoteControl.offButtonWasPushed(0);
                System.out.println(remoteControl);
                remoteControl.undoButtonWasPushed();
                remoteControl.onButtonWasPushed(1);
                System.out.println(remoteControl);
                remoteControl.undoButtonWasPushed();
        }
}
```

Output:

```
------ Remote Control -------
[slot 0] NoCommand    NoCommand
[slot 1] NoCommand    NoCommand
[slot 2] NoCommand    NoCommand
[slot 3] NoCommand    NoCommand
[slot 4] NoCommand    NoCommand
[slot 5] NoCommand    NoCommand
[slot 6] NoCommand    NoCommand
[undo] NoCommand
```

```
------ Remote Control -------
[slot 0] NoCommand    NoCommand
[slot 1] NoCommand    NoCommand
[slot 2] NoCommand    NoCommand
[slot 3] NoCommand    NoCommand
[slot 4] NoCommand    NoCommand
[slot 5] NoCommand    NoCommand
[slot 6] NoCommand    NoCommand
[undo] NoCommand

Living Room ceiling fan is on medium
Living Room ceiling fan is off

------ Remote Control -------
[slot 0] CeilingFanMediumCommand    CeilingFanOffCommand
[slot 1] CeilingFanHighCommand    CeilingFanOffCommand
[slot 2] NoCommand    NoCommand
[slot 3] NoCommand    NoCommand
[slot 4] NoCommand    NoCommand
[slot 5] NoCommand    NoCommand
[slot 6] NoCommand    NoCommand
[undo] CeilingFanOffCommand

Living Room ceiling fan is on medium
Living Room ceiling fan is on high

------ Remote Control -------
[slot 0] CeilingFanMediumCommand    CeilingFanOffCommand
[slot 1] CeilingFanHighCommand    CeilingFanOffCommand
[slot 2] NoCommand    NoCommand
[slot 3] NoCommand    NoCommand
[slot 4] NoCommand    NoCommand
[slot 5] NoCommand    NoCommand
[slot 6] NoCommand    NoCommand
[undo] CeilingFanHighCommand

Living Room ceiling fan is on medium
```

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

Ans:

Derived class:

```
import java.util.*;
public class EnumerationIterator implements Iterator {
   Enumeration enumeration;
   public EnumerationIterator(Enumeration enumeration) {
      this.enumeration = enumeration;
   }
   public boolean hasNext() {
      return enumeration.hasMoreElements();
   }
   public Object next() {
      return enumeration.nextElement();
   }
   public void remove() {
      throw new UnsupportedOperationException();
   }
}
```

EnumerationIteratorTestDrive:

```
import java.util.*;
public class EnumerationIteratorTestDrive
{
   public static void main (String args[])
   {
      Vector v = new Vector(Arrays.asList(args));
      Iterator iterator = new EnumerationIterator(v.elements());
      while (iterator.hasNext())
      {
         System.out.println(iterator.next());
      }
   }
}
```

8) Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu.

Ans:

main class:

```
public class MenuTestDrive {
   static PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
   static DinerMenu dinerMenu = new DinerMenu();
   static LunchMenu lunchMenu = new LunchMenu();
   public static void main(String args[]) {
      printMenu();
   }
   public static void printMenu() {
      Iterator pancakeIterator = pancakeHouseMenu.createIterator();
      Iterator dinerIterator = dinerMenu.createIterator();
      Iterator LunchIterator = lunchMenu.createIterator();

      System.out.println("MENU\n \nBREAKFAST");
      printMenu(pancakeIterator);
      System.out.println("\nLUNCH");
      printMenu(LunchIterator);
      System.out.println("\nDinner");
      printMenu(dinerIterator);
   }
   private static void printMenu(Iterator iterator) {
      while (iterator.hasNext()) {
         MenuItem menuItem = (MenuItem)iterator.next();
         System.out.print(menuItem.getName() + ", ");
         System.out.print(menuItem.getPrice() + "  ");
         System.out.println(menuItem.getDescription());
      }
   }
}
```

menu item class:

```
public class MenuItem {
   String name;
   String description;
```

```
    double price;
    public MenuItem(String name,  String description, double price)
    {
        this.name = name;
        this.description = description;

        this.price = price;
    }
    public String getName() {
        return name;
    }
    public String getDescription() {
        return description;
    }
    public double getPrice() {
        return price;
    }
    public String toString() {
        return (name + ", $" + price + "\n   " + description);
    }
}
```

PancakeHouseMenu for Breakfast:

```
import java.util.ArrayList;
public class PancakeHouseMenu implements Menu {
    ArrayList menuItems = new ArrayList();
    public PancakeHouseMenu() {
        addItem("K&B's Pancake Breakfast", "Pancakes with scrambled eggs, and
toast",2.99);
    }
    public void addItem(String name, String description, double price)
    {
        MenuItem menuItem = new MenuItem(name, description, price);
        menuItems.add(menuItem);
    }
    public Iterator createIterator() {
        return new PancakeHouseMenuIterator(menuItems);
    }
}
```

PancakeHouseMenuIterator class:

```
import java.util.ArrayList;
public class PancakeHouseMenuIterator implements Iterator {
   ArrayList items;
   int position = 0;
   public PancakeHouseMenuIterator(ArrayList items) {
      this.items = items;
   }
   public Object next() {
      Object object = items.get(position);
      position = position + 1;
      return object;
   }
   public boolean hasNext() {
      if (position >= items.size()) {
         return false;
      } else {
         return true;
      }
   }
}
```

LunchMenu:

```
public class LunchMenu implements Menu
{
   int numberOfItems = 0;
   MenuItem[] menuItems;
   public LunchMenu() {
      menuItems = new MenuItem[6];
      addItem("pavbhaji","Bacon with lettuce & tomato on whole wheat", 2.99);
   }
   public void addItem(String name, String description, double price)
   {
      MenuItem menuItem = new MenuItem(name, description, price);
      if (numberOfItems >= 6) {
         System.err.println("Sorry, menu is full!  Can't add item to menu");
      } else {
         menuItems[numberOfItems] = menuItem;
         numberOfItems = numberOfItems + 1;
      }
```

```
    }
    public Iterator createIterator() {
        return new LunchMenuIterator(menuItems);
    }

}
```

LunchMenuIterator:

```
public class LunchMenuIterator implements Iterator {
    MenuItem[] items;
    int position = 0;
    public LunchMenuIterator(MenuItem[] items) {
        this.items = items;
    }
    public Object next() {
        MenuItem menuItem = items[position];
        position = position + 1;
        return menuItem;
    }
    public boolean hasNext() {
        if (position >= items.length || items[position] == null) {
            return false;
        } else {
            return true;
        }
    }
}
```

DinerMenu:

```
public class DinerMenu implements Menu
{
    int numberOfItems = 0;
    MenuItem[] menuItems;

    public DinerMenu() {
        menuItems = new MenuItem[6];
        addItem("BLT","Bacon with lettuce & tomato on whole wheat", 2.99);
    }
    public void addItem(String name, String description, double price)
    {
        MenuItem menuItem = new MenuItem(name, description, price);
```

```
        if (numberOfItems >= 6) {
            System.err.println("Sorry, menu is full!  Can't add item to menu");
        } else {
            menuItems[numberOfItems] = menuItem;
            numberOfItems = numberOfItems + 1;
        }
    }
    public Iterator createIterator() {
        return new DinerMenuIterator(menuItems);
    }
}
```

DinerMenuIterator:

```
public class DinerMenuIterator implements Iterator {
    MenuItem[] items;
    int position = 0;
    public DinerMenuIterator(MenuItem[] items) {
        this.items = items;
    }
    public Object next() {
        MenuItem menuItem = items[position];
        position = position + 1;
        return menuItem;
    }
    public boolean hasNext() {
        if (position >= items.length || items[position] == null) {
            return false;
        } else {
            return true;
        }
    }
}
```

INTERFACES Iterator:

```
public interface Iterator {
    boolean hasNext();
    Object next();
}
```

INTERFACES Menu:

```
public interface Menu {
   public Iterator createIterator();
}
```

Output:

MENU

BREAKFAST

K&B's Pancake Breakfast, 2.99 Pancakes with scrambled eggs, and toast

LUNCH

pavbhaji, 2.99 Bacon with lettuce & tomato on whole wheat

Dinner

BLT, 2.99 Bacon with lettuce & tomato on whole wheat

9) 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, behaviors 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.

Ans:

main class :

```
public class StatePatternDemo {
   public static void main(String[] args) {
      Context context = new Context();

      StartState startState = new StartState();
      startState.doAction(context);

      System.out.println(context.getState().toString());

      StopState stopState = new StopState();
      stopState.doAction(context);

      System.out.println(context.getState().toString());
   }
}
```

Context Class :

```
public class Context {
   private State state;
   public Context(){
      state = null;
   }
   public void setState(State state){
      this.state = state;
   }
   public State getState(){
      return state;
   }
}
```

StopState class :

```
public class StopState implements State {
```

```
   public void doAction(Context context) {
      System.out.println("Player is in stop state");
      context.setState(this);
   }

   public String toString(){
      return "Stop State";
   }
}
```

StartState class :

```
public class StartState implements State {

   public void doAction(Context context) {
      System.out.println("Player is in start state");
      context.setState(this);
   }

   public String toString(){
      return "Start State";
   }
}
```

Interface State:

```
public interface State {
   public void doAction(Context context);
}
```

Output:

```
Player is in start state
Start State
Player is in stop state
Stop State
```

10) Write a java program to implement Adapter pattern to design Heart Model to Beat Model.

Ans:

main class :

```
public class AdapterPatternTest {
   public static void main(String[] args) {
      SocketAdapter sockAdapter = new SocketClassAdapterImpl();
      Volt v3 = getVolt(sockAdapter,3);
      Volt v12 = getVolt(sockAdapter,12);
      Volt v120 = getVolt(sockAdapter,120);
      System.out.println("v3 volts using Class Adapter="+v3.getVolts());
      System.out.println("v12 volts using Class Adapter="+v12.getVolts());
      System.out.println("v120 volts using Class Adapter="+v120.getVolts());
   }
   private static Volt getVolt(SocketAdapter sockAdapter, int i) {
      switch (i){
         case 3: return sockAdapter.get3Volt();
         case 12: return sockAdapter.get12Volt();
         case 120: return sockAdapter.get120Volt();
         default: return sockAdapter.get120Volt();
      }
   }
}
```

SocketClassAdapterImpl class :

```
public class SocketClassAdapterImpl extends Socket implements
SocketAdapter{
   public Volt get120Volt() {
      return getVolt();
   }
   public Volt get12Volt() {
      Volt v= getVolt();
      return convertVolt(v,10);
   }
   public Volt get3Volt() {
      Volt v= getVolt();
      return convertVolt(v,40);
   }
   private Volt convertVolt(Volt v, int i) {
```

```
        return new Volt(v.getVolts()/i);
    }
}
```

Volt class :

```
public class Volt {
    private int volts;
    public Volt(int v) {
        this.volts = v;
    }
    public int getVolts() {
        return volts;
    }
    public void setVolts(int volts) {
        this.volts = volts;
    }
}
```

Socket class :

```
public class Socket {
    public Volt getVolt(){
        return new Volt(120);
    }
}
```

interface :

```
public interface SocketAdapter {
    public Volt get120Volt();
    public Volt get12Volt();
    public Volt get3Volt();
}
```

Output:

```
v3 volts using Class Adapter=3
v12 volts using Class Adapter=12
v120 volts using Class Adapter=120
```