

Software Architecture & Design Pattern Practical

M.Sc II (Computer Science)

2023-24

Software Architecture & Design Pattern List of Assignments

Q.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.

DisplayElement.java

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

Observer.java

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

Subject.java

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

WeatherData.java

```
import java.util.*;

public class WeatherData implements Subject
{
    private ArrayList<Observer> 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 removeObserver(Observer o)  
{  
    int i = observers.indexOf(o);  
    if (i >= 0)  
    {  
        observers.remove(i);  
    }  
}  
  
public void notifyObservers()  
{  
    for (int i = 0; i < observers.size(); i++)  
    {  
        Observer observer = (Observer)observers.get(i);  
        observer.update(temperature, humidity, pressure);  
    }  
}
```

```
}
```

```
public void measurementsChanged()
```

```
{
```

```
    notifyObservers();
```

```
}
```

```
public void setMeasurements(float temperature, float humidity,  
float pressure)
```

```
{
```

```
    this.temperature = temperature;
```

```
    this.humidity = humidity;
```

```
    this.pressure = pressure;
```

```
    measurementsChanged();
```

```
}
```

```
public float getTemperature()
```

```
{
```

```
    return temperature;
}
```

```
public float getHumidity()
{
    return humidity;
}
```

```
public float getPressure()
{
    return pressure;
}
}
```

CurrentConditionsDisplay.java

```
public class CurrentConditionsDisplay implements Observer,
DisplayElement
{
    private float temperature;
```

```
private float humidity;
```

```
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;
```

```
    display();
```

```
}
```

```
public void display()
```

```
{
```

```
    System.out.println("Current conditions: " + temperature + "F  
degrees and " + humidity + "% humidity");
```

```
}  
}
```

StatisticsDisplay.java

```
public class StatisticsDisplay implements Observer, DisplayElement  
{  
    private float maxTemp = 0.0f;  
    private float minTemp = 200;  
    private float tempSum= 0.0f;  
    private int numReadings;  
    private WeatherData weatherData;  
  
    public StatisticsDisplay(WeatherData weatherData)  
    {  
        this.weatherData = weatherData;  
        weatherData.registerObserver(this);  
    }  
  
    public void update(float temp, float humidity, float pressure)
```

```
{  
    tempSum += temp;  
    numReadings++;  
  
    if (temp > maxTemp)  
    {  
        maxTemp = temp;  
    }  
  
    if (temp < minTemp)  
    {  
        minTemp = temp;  
    }  
  
    display();  
}
```

```
public void display()  
{
```



```
        System.out.println("Avg/Max/Min temperature = " + (tempSum /  
numReadings)+ "/" + maxTemp + "/" + minTemp);  
    }  
}
```

ForecastDisplay.java

```
public class ForecastDisplay implements Observer, DisplayElement  
{  
    private float currentPressure = 29.92f;  
    private float lastPressure;  
    private WeatherData weatherData;  
  
    public ForecastDisplay(WeatherData weatherData)  
    {  
        this.weatherData = weatherData;  
        weatherData.registerObserver(this);  
    }  
  
    public void update(float temp, float humidity, float pressure)  
    {
```

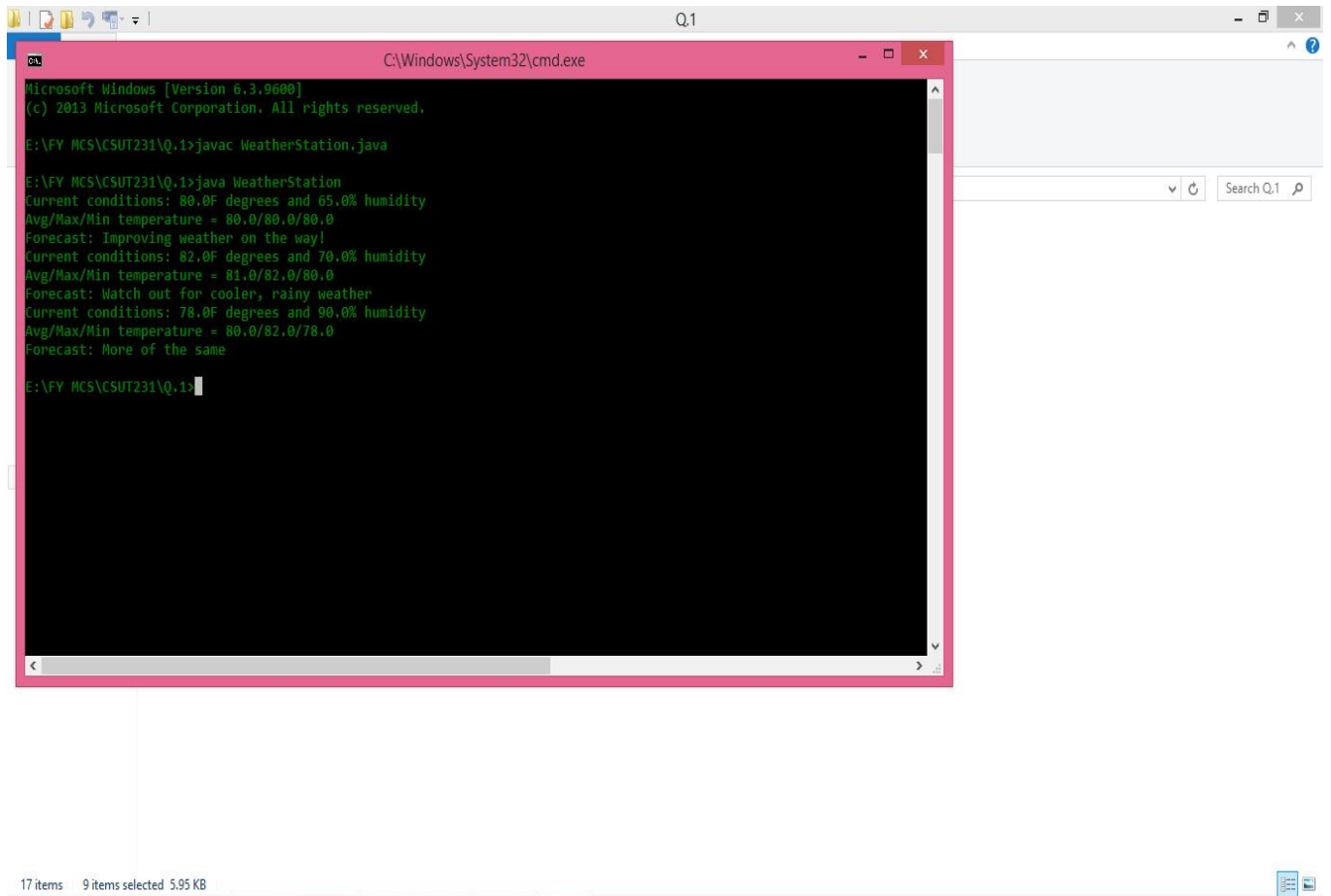
```
lastPressure = currentPressure;

currentPressure = pressure;


display();
}


public void display()
{
    System.out.print("Forecast: ");
    if (currentPressure > lastPressure)
    {
        System.out.println("Improving weather on the way!");
    }
    else if (currentPressure == lastPressure)
    {
        System.out.println("More of the same");
    }
    else if (currentPressure < lastPressure)
    {
        System.out.println("Watch out for cooler, rainy weather");
    }
}
```

```
}  
  
}  
  
}
```



The screenshot shows a Windows File Explorer window with the address bar displaying 'Q.1'. The main pane shows a folder named 'Q.1' with a search bar and a 'Search Q.1' button. Overlaid on the File Explorer is a Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The Command Prompt displays the following text:

```
Microsoft Windows [Version 6.3.9600]  
(c) 2013 Microsoft Corporation. All rights reserved.  
  
E:\FY MCS\CSUT231\Q.1>javac WeatherStation.java  
  
E:\FY MCS\CSUT231\Q.1>java WeatherStation  
Current conditions: 80.0F degrees and 65.0% humidity  
Avg/Max/Min temperature = 80.0/80.0/80.0  
Forecast: Improving weather on the way!  
Current conditions: 82.0F degrees and 70.0% humidity  
Avg/Max/Min temperature = 81.0/82.0/80.0  
Forecast: Watch out for cooler, rainy weather  
Current conditions: 78.0F degrees and 90.0% humidity  
Avg/Max/Min temperature = 80.0/82.0/78.0  
Forecast: More of the same  
  
E:\FY MCS\CSUT231\Q.1>
```

At the bottom of the screen, a status bar indicates '17 items', '9 items selected', and '5.95 KB'.

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

Ans:-

changeCase.java

```
public class changeCase
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        String str1="Software Architecture & Design Pattern List of  
Assignments";
```

```
        StringBuffer newStr=new StringBuffer(str1);
```

```
        System.out.println("\nString before case conversion : ");
```

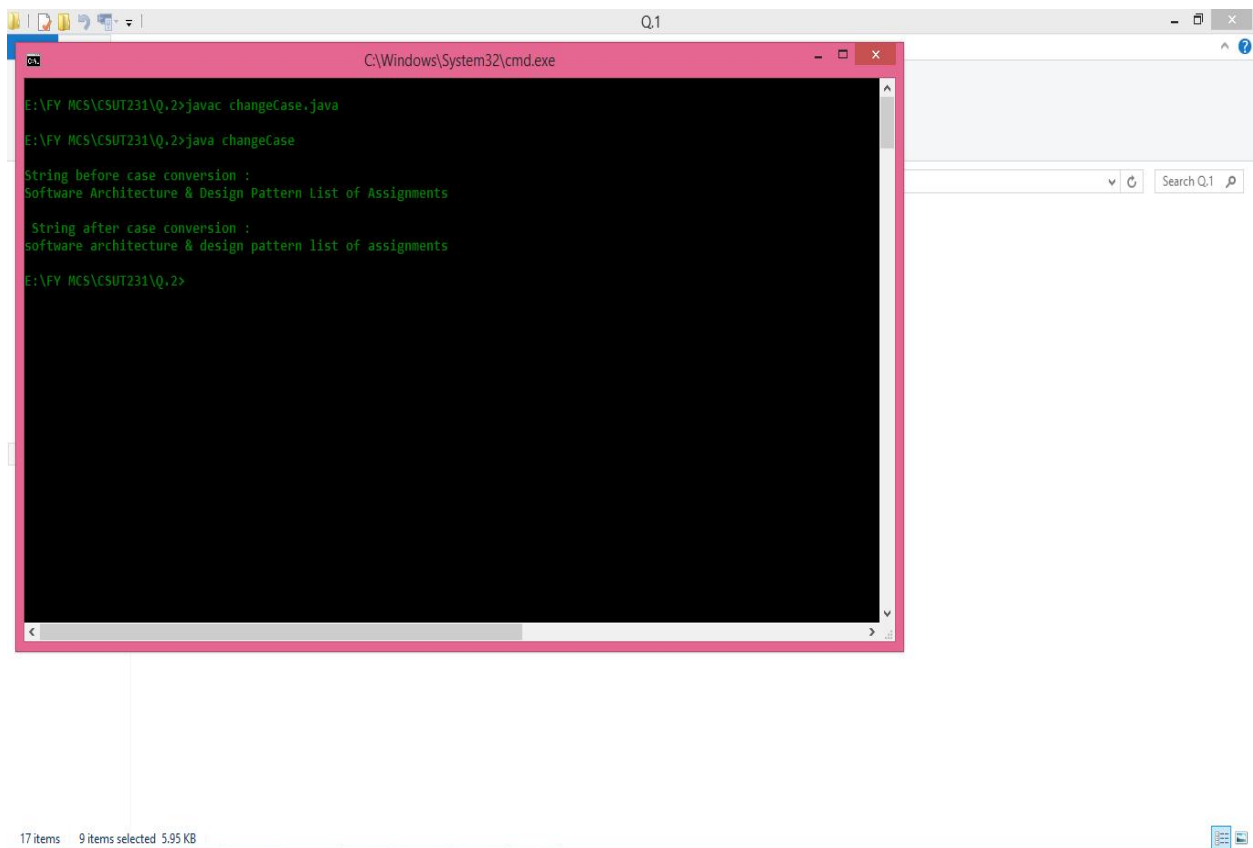
```
        System.out.println(newStr);
```

```
for(int i = 0; i < str1.length(); i++)
{
    //Checks for lower case character
    //if(Character.isLowerCase(str1.charAt(i)))
    //{
        //Convert it into upper case using toUpperCase() function
        //newStr.setCharAt(i, Character.toUpperCase(str1.charAt(i)));
    //}
    //Checks for upper case character
    //else
    if(Character.isUpperCase(str1.charAt(i)))
    {
        //Convert it into upper case using toLowerCase() function
        newStr.setCharAt(i, Character.toLowerCase(str1.charAt(i)));
    }
}

System.out.println("\n String after case conversion : ");

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

}



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

Ans:-

Main.java

```
import java.util.ArrayList;

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;
}
}
```

```
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");  
    }  
}
```

```
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");
}
}
```

```
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");  
    }  
}
```

```
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");
    }
}
```

```
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;
}
}

```

```

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;  
    }  
}
```

```
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");
    }
}
```

```
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");
    }
}
```

```
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");
    }
}
```

```
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");
    }
}
```

```
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();
}
}

```

```

abstract class PizzaStore
{
    abstract Pizza createPizza(String item);
}

```

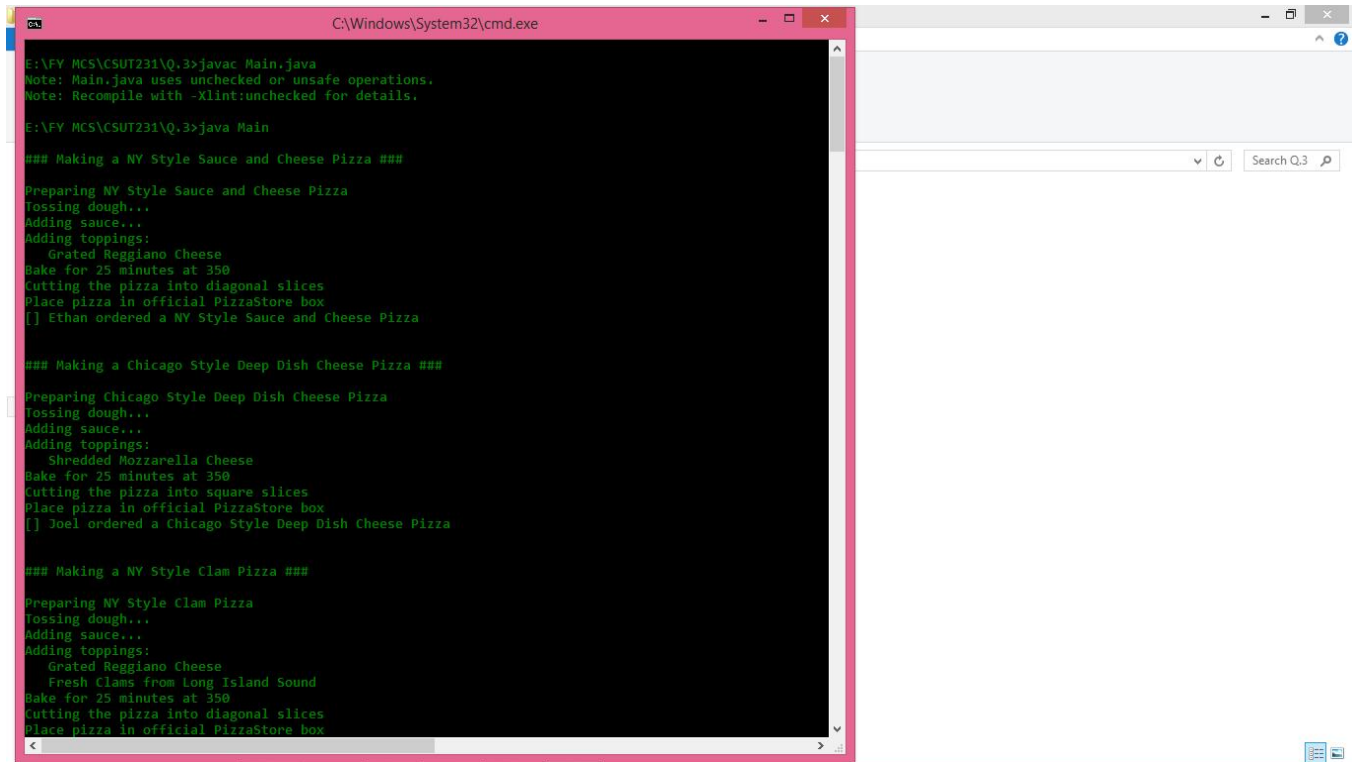
```
public Pizza orderPizza(String type)
{
    Pizza pizza = createPizza(type);
    System.out.println("\n### Making a " + pizza.getName() + " ### \n");
    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
}
```

```
public class Main
{
    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");  
  
}
```

}



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The window contains the following text:

```
E:\FY MCS\CSUT231\Q.3>javac Main.java
Note: Main.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\FY MCS\CSUT231\Q.3>java Main

### 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

### 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
```

The command prompt window is open on a desktop background. To the right of the command prompt, there is a search bar with the text "Search Q.3" and a magnifying glass icon. The desktop background is a light blue color with a subtle pattern.

```
C:\Windows\System32\cmd.exe

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
    Frozen Clams from Chesapeake Bay
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...
```



```
C:\Windows\System32\cmd.exe

### 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 Deep Dish Veggie Pizza ###

Preparing Chicago Deep Dish Veggie Pizza
Tossing dough...
Adding sauce...
Adding toppings:
    Shredded Mozzarella Cheese
    Black Olives
    Spinach
```

```
C:\Windows\System32\cmd.exe

### 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 Deep Dish Veggie Pizza ###

Preparing Chicago Deep Dish Veggie Pizza
Tossing dough...
Adding sauce...
Adding toppings:
    Shredded Mozzarella Cheese
    Black Olives
    Spinach
    Eggplant
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
[] Joel ordered a Chicago Deep Dish Veggie Pizza

E:\FY MCS\CSUT231\Q.3>
```

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

Ans:-

Test.java

```
public class Test {  
  
    public static void main(String ar[]) {  
        Test1 t = new Test1();  
        Test1 t2 = new Test1();  
        Test1 t3 = new Test1();  
        Thread tt = new Thread(t);  
        Thread tt2 = new Thread(t2);  
        Thread tt3 = new Thread(t3);  
        Thread tt4 = new Thread(t);  
        Thread tt5 = new Thread(t);  
        tt.start();  
        tt2.start();  
        tt3.start();  
        tt4.start();  
        tt5.start();  
  
    }
```

```
}
```

```
final class Test1 implements Runnable {
```

```
    @Override
```

```
    public void run() {
```

```
        for (int i = 0; i < 5; i++) {
```

```
            System.out.println(Thread.currentThread().getName() + " : " +  
Single.getInstance().hashCode());
```

```
        }
```

```
    }
```

```
}
```

```
class Single {
```

```
    private final static Single sing = new Single();
```

```
    private Single() {
```

```
    }
```

```
    public static Single getInstance() {
```

```
        return sing;
```

```
    }
```

```
}
```

```
C:\Windows\System32\cmd.exe

E:\FY MCS\CSUT231\Q.4>javac Test.java

E:\FY MCS\CSUT231\Q.4>java Test
Thread-4 : 666428379
Thread-3 : 666428379
Thread-4 : 666428379
Thread-1 : 666428379
Thread-0 : 666428379
Thread-1 : 666428379
Thread-2 : 666428379
Thread-4 : 666428379
Thread-3 : 666428379
Thread-4 : 666428379
Thread-2 : 666428379
Thread-1 : 666428379
Thread-0 : 666428379
Thread-1 : 666428379
Thread-2 : 666428379
Thread-4 : 666428379
Thread-3 : 666428379
Thread-2 : 666428379
Thread-1 : 666428379
Thread-0 : 666428379
Thread-2 : 666428379
Thread-3 : 666428379
Thread-3 : 666428379
Thread-0 : 666428379
Thread-0 : 666428379

E:\FY MCS\CSUT231\Q.4>
```

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

Ans:-

RemoteControlTest

// An interface for command

interface Command

{

 public void execute();

}

// Light class and its corresponding command

// classes

class Light

{

 public void on()

 {

 System.out.println("Light is on");

 }

 public void off()

 {

 System.out.println("Light is off");

```
    }  
}  
  
class LightOnCommand implements Command  
{  
    Light light;  
  
    // The constructor is passed the light it  
    // is going to control.  
    public LightOnCommand(Light light)  
    {  
        this.light = light;  
    }  
    public void execute()  
    {  
        light.on();  
    }  
}  
  
class LightOffCommand implements Command  
{  
    Light light;  
    public LightOffCommand(Light light)  
    {
```

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

// Stereo and its command classes

```
class Stereo
{
    public void on()
    {
        System.out.println("Stereo is on");
    }
    public void off()
    {
        System.out.println("Stereo is off");
    }
    public void setCD()
    {
        System.out.println("Stereo is set " +
```

```

        "for CD input");
    }

    public void setDVD()
    {
        System.out.println("Stereo is set"+
                            " for DVD input");
    }

    public void setRadio()
    {
        System.out.println("Stereo is set" +
                            " for Radio");
    }

    public void setVolume(int volume)
    {
        // code to set the volume

        System.out.println("Stereo volume set"
                            + " to " + volume);
    }
}

class StereoOffCommand implements Command
{
    Stereo stereo;

```



```
public StereoOffCommand(Stereo stereo)
{
    this.stereo = stereo;
}

public void execute()
{
    stereo.off();
}
}

class StereoOnWithCDCommand implements Command
{
    Stereo stereo;

    public StereoOnWithCDCommand(Stereo stereo)
    {
        this.stereo = stereo;
    }

    public void execute()
    {
        stereo.on();
        stereo.setCD();
        stereo.setVolume(11);
    }
}
```

```
}  
  
// A Simple remote control with one button  
  
class SimpleRemoteControl  
{  
    Command slot; // only one button  
  
    public SimpleRemoteControl()  
    {  
    }  
  
    public void setCommand(Command command)  
    {  
        // set the command the remote will  
        // execute  
        slot = command;  
    }  
  
    public void buttonWasPressed()  
    {  
        slot.execute();  
    }  
}  
  
// Driver class
```

```
class RemoteControlTest
{
    public static void main(String[] args)
    {
        SimpleRemoteControl remote =
            new SimpleRemoteControl();

        Light light = new Light();

        Stereo stereo = new Stereo();

        // we can change command dynamically
        remote.setCommand(new
            LightOnCommand(light));

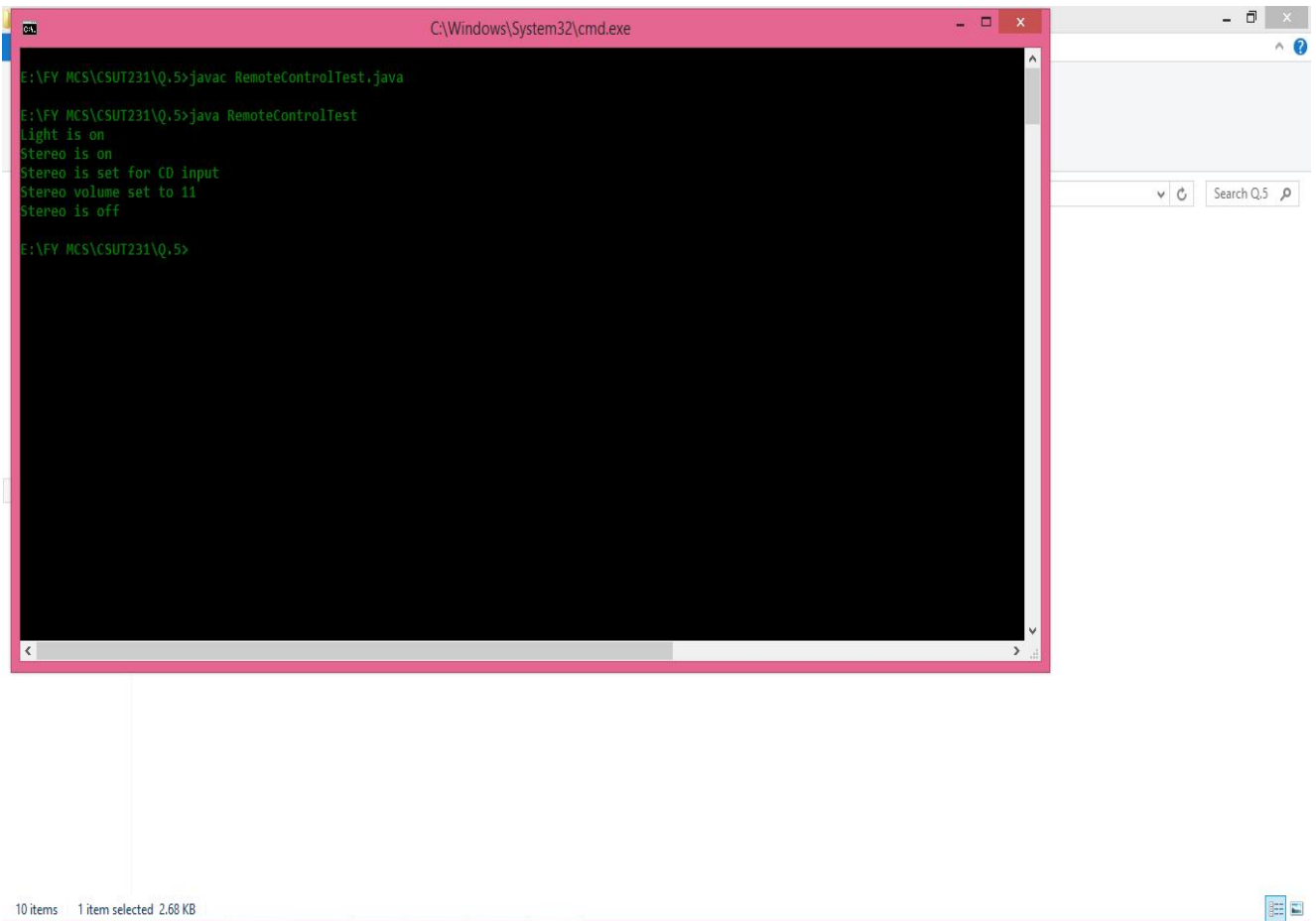
        remote.buttonWasPressed();

        remote.setCommand(new
            StereoOnWithCDCommand(stereo));

        remote.buttonWasPressed();

        remote.setCommand(new
            StereoOffCommand(stereo));

        remote.buttonWasPressed();
    }
}
```



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

Ans:-

Duck.java

```
public interface Duck {  
    public void quack();  
    public void fly();  
}
```

Turkey.java

```
public interface Turkey { //Turkeys don't quack  
    public void gobble();  
    public void fly();  
}
```

MallardDuck.java

```
public class MallardDuck implements Duck {  
    public void quack() {  
        System.out.println("Quack");  
    }  
  
    public void fly() {  
        System.out.println("I'm flying");  
    }  
}
```

WildTurkey.java

```
public class WildTurkey implements Turkey {  
    public void gobble() {  
        System.out.println("Gobble gobble");  
    }  
}
```

```
public void fly() {  
    System.out.println("I'm flying a short distance");  
}  
}
```

TurkeyAdapter.java

```
public class TurkeyAdapter implements Duck {  
    Turkey turkey;  
  
    public TurkeyAdapter(Turkey turkey) {  
        this.turkey = turkey;  
    }  
  
    public void quack() {  
        turkey.gobble();  
    }  
  
    public void fly() {
```

```
        for(int i=0; i < 5; i++) {  
            turkey.fly();  
        }  
    }  
}
```

TurkeyTestDrive.java

```
public class TurkeyTestDrive {  
    public static void main(String[] args) {  
        MallardDuck duck = new MallardDuck();  
        WildTurkey turkey = new WildTurkey();  
        Duck turkeyAdapter = new TurkeyAdapter(turkey);  
  
        System.out.println("\nThe Turkey says ...");  
        turkey.gobble();  
        turkey.fly();  
  
        System.out.println("\nThe Duck says ...");  
    }  
}
```



```
testDuck(duck);
```

```
System.out.println("\nThe TurkeyAdapter says ...");
```

```
testDuck(turkeyAdapter);
```

```
}
```

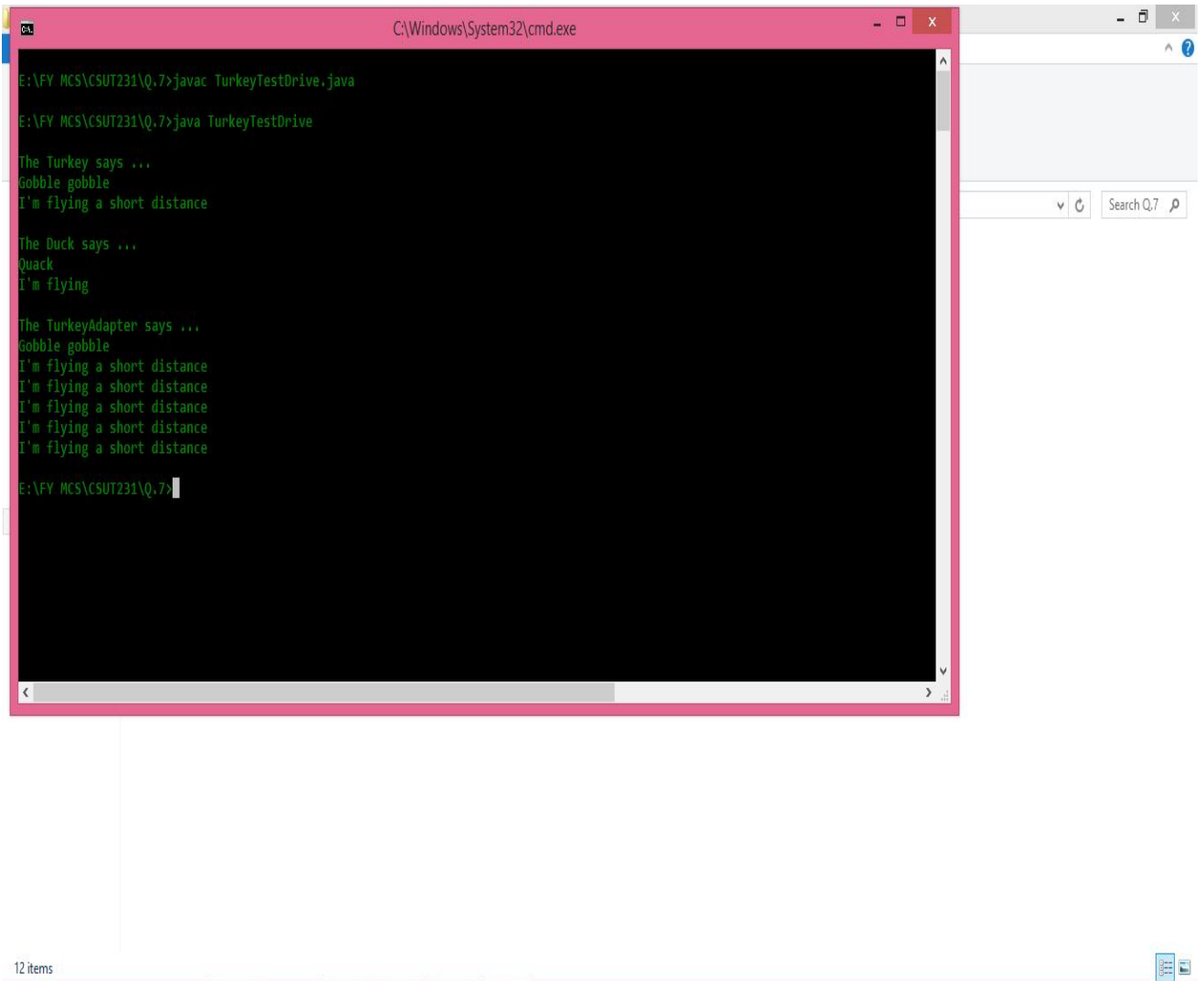
```
static void testDuck(Duck duck) {
```

```
    duck.quack();
```

```
    duck.fly();
```

```
}
```

```
}
```



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

Ans.

MenuTest.java

```
public class MenuTest {  
  
    public static void main(String[] args) {  
  
        PancakeHouseMenu pancakeHouseMenu = new  
PancakeHouseMenu();  
  
        DinnerMenu dinnerMenu = new DinnerMenu();  
  
        CoffeeMenu coffeeMenu = new CoffeeMenu();  
  
  
        Waitress waitress = new  
Waitress(pancakeHouseMenu,dinnerMenu,coffeeMenu);  
  
        waitress.printMenu();  
  
    }  
}
```

Menu.java

```
import java.util.*;

public interface Menu {

    public Iterator createIterator();

}
```

PancakeHouseMenu.java

```
import java.util.*;

public class PancakeHouseMenu implements Menu {

    ArrayList menuItems;

    public PancakeHouseMenu() {

        menuItems = new ArrayList();

        addItem("kobe's pancake breakfast", "pancakes with
eggs", false, 2.99);

        addItem("lilei's pancake breakfast", "pancakes with toast", false,
3.59);

    }

    public void addItem(String s, String s1, boolean b, double v) {

        MenuItem menuItem = new MenuItem(s, s1, b, v);

        menuItems.add(menuItem);

    }

}
```

```
public Iterator createIterator(){  
    return menuItems.iterator();  
}  
}
```

MenuItem.java

```
public class MenuItem {  
    private String name;  
    private String desc;  
    private boolean vegetarian;  
    private double price;  
  
    public MenuItem(String name, String desc, boolean vegetarian,  
double price) {  
        this.name = name;  
        this.desc = desc;  
        this.vegetarian = vegetarian;  
        this.price = price;  
    }  
}
```

```
public String getName() {  
    return name;  
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```
public String getDesc() {  
    return desc;  
}
```

```
public void setDesc(String desc) {  
    this.desc = desc;  
}
```

```
public boolean isVegetarian() {  
    return vegetarian;  
}
```

```
public void setVegetarian(boolean vegetarian) {  
    this.vegetarian = vegetarian;  
}
```

```
public double getPrice() {  
    return price;  
}
```

```
public void setPrice(double price) {  
    this.price = price;  
}  
}
```

DinnerMenu.java

```
import java.util.*;  
  
public class DinnerMenu implements Menu {  
    private static final int MAX_SIZE = 6;  
    int numOfItems = 0;  
    MenuItem[] menuItems;
```

```
public DinnerMenu() {  
    menuItems = new MenuItem[MAX_SIZE];  
    addItem("Vegetarian BLT","Bacon with tomato",true,2.99);  
    addItem("Hot dog","With onions and cheese",false,3.05);  
}
```

```
private void addItem(String s, String s1, boolean b, double v) {  
    MenuItem menuItem = new MenuItem(s,s1,b,v);  
    if(numOfItems >= MAX_SIZE){  
        System.err.println("sorry,menu is full!");  
    }else{  
        menuItems[numOfItems]=menuItem;  
        numOfItems = numOfItems + 1;  
    }  
}
```

@Override

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



```
}
```

DinerMenuIterator.java

```
import java.util.*;
```

```
public class DinerMenuIterator implements Iterator {
```

```
    MenuItem[] list;
```

```
    int position = 0;
```

```
    public DinerMenuIterator(MenuItem[] list) {
```

```
        this.list = list;
```

```
    }
```

```
    @Override
```

```
    public boolean hasNext() {
```

```
        if(position >= list.length || list[position] == null){
```

```
            return false;
```

```
        }else{
```

```
            return true;
```

```
        }
```

```
}
```

```
@Override
```

```
public Object next() {
```

```
    MenuItem menuItem = list[position];
```

```
    position = position + 1;
```

```
    return menuItem;
```

```
}
```

```
@Override
```

```
public void remove() {
```

```
    if(position <=0){
```

```
        throw new IllegalStateException("now you can not remove an  
item");
```

```
    }
```

```
    if(list[position] != null){
```

```
        for(int i=position-1;i<(list.length-1);i++){
```

```
            list[i] = list[i+1];
```

```
        }
```

```
        list[list.length-1]=null;
```

```
    }  
    }  
}
```

CoffeeMenu.java

```
import java.util.*;
```

```
public class CoffeeMenu implements Menu {
```

```
    Hashtable menuItems = new Hashtable();
```

```
    public CoffeeMenu() {
```

```
        addItem("Mocha","Han Meimei order on couple of  
Mocha",false,3.01);
```

```
    }
```

```
    private void addItem(String s, String s1, boolean b, double v) {
```

```
        MenuItem menuItem = new MenuItem(s,s1,b,v);
```

```
        menuItems.put(menuItem.getName(),menuItem);
```

```
    }
```

```
@Override
```

```
public Iterator createIterator() {  
    return menuItems.values().iterator();  
}  
}
```

Waitress.java

```
import java.util.*;  
  
public class Waitress {  
    Menu pancakeHouseMenu;  
    Menu dinnerMenu;  
    Menu coffeeMenu;  
  
    public Waitress(Menu pancakeHouseMenu, Menu dinnerMenu,  
Menu coffeeMenu) {  
        this.pancakeHouseMenu = pancakeHouseMenu;  
        this.dinnerMenu = dinnerMenu;  
        this.coffeeMenu = coffeeMenu;  
    }  
  
    public void printMenu(){
```

```

    Iterator pancakeHouseIterator =
pancakeHouseMenu.createIterator();

    Iterator dinnerIterator = dinnerMenu.createIterator();

    Iterator coffeeIterator = coffeeMenu.createIterator();

    System.out.println("Menu\n====Breakfast==start====");
    printMenu(pancakeHouseIterator);

    System.out.println("Menu\n====Breakfast===end====");

    System.out.println("Menu\n====Lunch==start====");
    printMenu(dinnerIterator);

    System.out.println("Menu\n====Lunch===end====");

    System.out.println("Menu\n====Coffee==start====");
    printMenu(coffeeIterator);

    System.out.println("Menu\n====Coffee===end====");
}

```

```

private void printMenu(Iterator iterator){
    while (iterator.hasNext()){

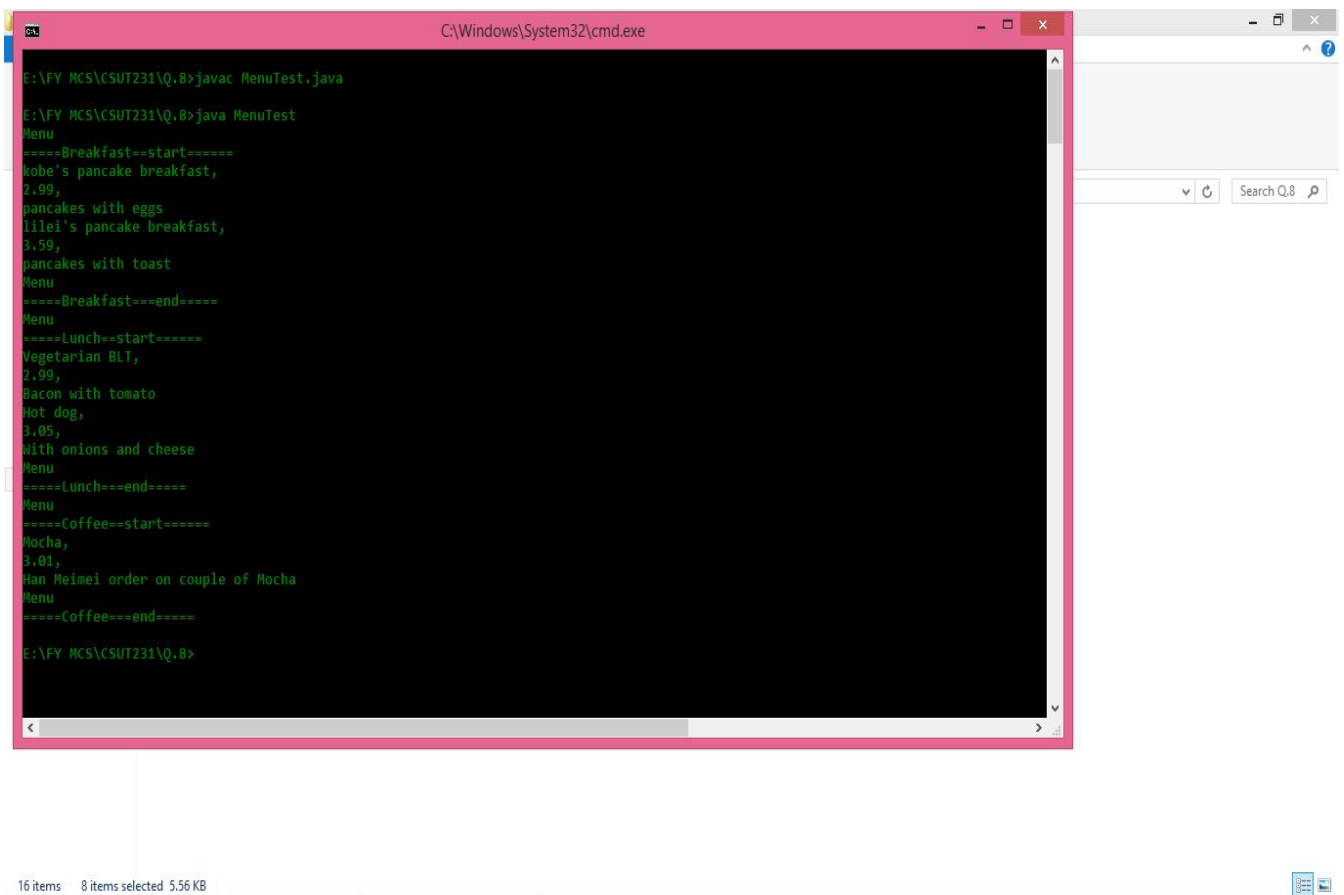
        MenuItem menuItem = (MenuItem)iterator.next();

        System.out.println(menuItem.getName()+" ");

        System.out.println(menuItem.getPrice()+" ");
    }
}

```

```
        System.out.println(menuItem.getDesc());
    }
}
}
```



```
C:\Windows\System32\cmd.exe

E:\FY MCS\CSUT231\Q.8>javac MenuTest.java

E:\FY MCS\CSUT231\Q.8>java MenuTest
Menu
=====Breakfast==start=====
kobe's pancake breakfast,
2.99,
pancakes with eggs
lilei's pancake breakfast,
3.59,
pancakes with toast
Menu
=====Breakfast===end=====
Menu
=====Lunch==start=====
Vegetarian BLT,
2.99,
Bacon with tomato
Hot dog,
3.05,
With onions and cheese
Menu
=====Lunch===end=====
Menu
=====Coffee==start=====
Mocha,
3.01,
Han Meimei order on couple of Mocha
Menu
=====Coffee===end=====

E:\FY MCS\CSUT231\Q.8>
```

16 items 8 items selected 5.56 KB

Q.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.

GumballMachineTestDrive.java

```
public class GumballMachineTestDrive {  
  
    public static void main(String[] args) {  
        GumballMachine gumballMachine = new GumballMachine(2);  
  
        System.out.println(gumballMachine);  
  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
  
        System.out.println(gumballMachine);  
  
        gumballMachine.insertQuarter();
```

```
        gumballMachine.turnCrank();  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
  
        gumballMachine.refill(5);  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
  
        System.out.println(gumballMachine);  
    }  
}
```

State.java

```
public interface State {  
    public void insertQuarter();  
    public void ejectQuarter();  
    public void turnCrank();  
    public void dispense();  
    public void refill();  
}
```


GumballMachine.java

```
public class GumballMachine {  
    State soldOutState;  
    State noQuarterState;  
    State hasQuarterState;  
    State soldState;  
    State state;  
    int count = 0;  
    public GumballMachine(int numberGumballs) {  
        soldOutState = new SoldOutState(this);  
        noQuarterState = new NoQuarterState(this);  
        hasQuarterState = new HasQuarterState(this);  
        soldState = new SoldState(this);  
  
        this.count = numberGumballs;  
        if (numberGumballs > 0) {  
            state = noQuarterState;  
        } else {  
            state = soldOutState;  
        }  
    }  
}
```

```
}
```

```
public void insertQuarter() {  
    state.insertQuarter();  
}
```

```
public void ejectQuarter() {  
    state.ejectQuarter();  
}
```

```
public void turnCrank() {  
    state.turnCrank();  
    state.dispense();  
}
```

```
void releaseBall() {  
    System.out.println("A gumball comes rolling out the slot...");  
    if (count != 0) {  
        count = count - 1;  
    }  
}
```

```
}
```

```
int getCount() {
```

```
    return count;
```

```
}
```

```
void refill(int count) {
```

```
    this.count += count;
```

```
    System.out.println("The gumball machine was just refilled;  
it's new count is: " + this.count);
```

```
    state.refill();
```

```
}
```

```
void setState(State state) {
```

```
    this.state = state;
```

```
}
```

```
public State getState() {
```

```
    return state;
```

```
}
```

```
public State getSoldOutState() {  
    return soldOutState;  
}
```

```
public State getNoQuarterState() {  
    return noQuarterState;  
}
```

```
public State getHasQuarterState() {  
    return hasQuarterState;  
}
```

```
public State getSoldState() {  
    return soldState;  
}
```

```
public String toString() {  
    StringBuffer result = new StringBuffer();  
    result.append("\nMighty Gumball, Inc.");  
    result.append("\nJava-enabled Standing Gumball Model");  
}
```

```
        result.append("\nInventory: " + count + " gumball");  
        if (count != 1) {  
            result.append("s");  
        }  
        result.append("\n");  
        result.append("Machine is " + state + "\n");  
        return result.toString();  
    }  
}
```

SoldOutState.java

```
public class SoldOutState implements State {  
    GumballMachine gumballMachine;  
  
    public SoldOutState(GumballMachine gumballMachine) {  
        this.gumballMachine = gumballMachine;  
    }  
  
    public void insertQuarter() {
```

```
        System.out.println("You can't insert a quarter, the machine  
is sold out");
```

```
    }
```

```
    public void ejectQuarter() {
```

```
        System.out.println("You can't eject, you haven't inserted a  
quarter yet");
```

```
    }
```

```
    public void turnCrank() {
```

```
        System.out.println("You turned, but there are no gumballs");
```

```
    }
```

```
    public void dispense() {
```

```
        System.out.println("No gumball dispensed");
```

```
    }
```

```
    public void refill() {
```

```
        gumballMachine.setState(gumballMachine.getNoQuarterState());
```

```
    }
```

```
        public String toString() {  
            return "sold out";  
        }  
    }  
}
```

NoQuarterState.java

```
public class NoQuarterState implements State {  
    GumballMachine gumballMachine;  
  
    public NoQuarterState(GumballMachine gumballMachine) {  
        this.gumballMachine = gumballMachine;  
    }  
  
    public void insertQuarter() {  
        System.out.println("You inserted a quarter");  
  
        gumballMachine.setState(gumballMachine.getHasQuarterState());  
    }  
}
```

```
public void ejectQuarter() {
```

```
    System.out.println("You haven't inserted a quarter");
```

```
}
```

```
public void turnCrank() {
```

```
    System.out.println("You turned, but there's no quarter");
```

```
}
```

```
public void dispense() {
```

```
    System.out.println("You need to pay first");
```

```
}
```

```
public void refill() { }
```

```
public String toString() {
```

```
    return "waiting for quarter";
```

```
}
```

```
}
```


HasQuarterState.java

```
public class HasQuarterState implements State {  
    GumballMachine gumballMachine;  
  
    public HasQuarterState(GumballMachine gumballMachine) {  
        this.gumballMachine = gumballMachine;  
    }  
  
    public void insertQuarter() {  
        System.out.println("You can't insert another quarter");  
    }  
  
    public void ejectQuarter() {  
        System.out.println("Quarter returned");  
  
        gumballMachine.setState(gumballMachine.getNoQuarterState());  
    }  
  
    public void turnCrank() {  
        System.out.println("You turned...");
```

```
        gumballMachine.setState(gumballMachine.getSoldState());
    }

    public void dispense() {
        System.out.println("No gumball dispensed");
    }

    public void refill() { }

    public String toString() {
        return "waiting for turn of crank";
    }
}
```

SoldState.java

```
public class SoldState implements State {
```

```
    GumballMachine gumballMachine;
```

```
    public SoldState(GumballMachine gumballMachine) {
```

```
        this.gumballMachine = gumballMachine;
```

```
    }
```

```
    public void insertQuarter() {
```

```
        System.out.println("Please wait, we're already giving you a  
gumball");
```

```
    }
```

```
    public void ejectQuarter() {
```

```
        System.out.println("Sorry, you already turned the crank");
```

```
    }
```

```
    public void turnCrank() {
```

```
        System.out.println("Turning twice doesn't get you another  
gumball!");  
    }  
  
    public void dispense() {  
        gumballMachine.releaseBall();  
        if (gumballMachine.getCount() > 0) {  
  
gumballMachine.setState(gumballMachine.getNoQuarterState());  
        } else {  
            System.out.println("Oops, out of gumballs!");  
  
gumballMachine.setState(gumballMachine.getSoldOutState());  
        }  
    }  
  
    public void refill() { }  
  
    public String toString() {  
        return "dispensing a gumball";  
    }  
}
```

```
C:\Windows\System32\cmd.exe

E:\FY\MCS\CSUT231\Q.9>javac GumballMachineTestDrive.java

E:\FY\MCS\CSUT231\Q.9>java GumballMachineTestDrive

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 2 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 1 gumball
Machine is waiting for quarter

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
No gumball dispensed
The gumball machine was just refilled; it's new count is: 5
You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 4 gumballs
Machine is waiting for quarter

E:\FY\MCS\CSUT231\Q.9>
```

