Assignment of Software Architecture and Design

Patterns Class: M.Sc. (Computer Science) Semester-III

Name: Mohammed Salif Shaikh

Roll Number: 9253

Q1. Write a Java Program to implement Singleton pattern for multithreading.

Singleton.java

```
Package singletonpattern;
// NOTE: This is not thread safe!

Public class Singleton {
        Private static Singleton uniqueInstance;

        Private Singleton() {}
        Public static Singleton getInstance() {
                If (uniqueInstance == null) {
                        uniqueInstance = new Singleton();
                }
                Return uniqueInstance;
        }
        Public static void main(String args[])
        {
                System.out.println(getInstance());
        }
}
```

Output:

```
Singleton@2b2fa4f7
```

Q2. Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

InputTest.java

```
Package decorator;

Import java.io.*;


Public class InputTest {

        Public static void main(String[] args) throws IOException {

                Int c;

                Try {

                        InputStream in =

                                New LowerCaseInputStream(

                                        New BufferedInputStream(

                                                New FileInputStream("C:\\Users\\eclipse-
workspace\\DesignPatterns\\src\\decorator\\test.txt")));


                        While((c = in.read()) >= 0) {

                                System.out.print((char)c);

                        }


                        In.close();

                } catch (IOException e) {

                        e.printStackTrace();

                }

        }

}
```

LowerCaseInputStream.java

```
 package decorator;
 import java.io.*;

 public class LowerCaseInputStream extends FilterInputStream {

        public 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;
		}
	}
```

test.txt

```
I Know the Decorator Pattern and I RULE!
```

Output:

```
I know the decorator pattern and I rule!
```

Q3. 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().

WeatherStation.java

```
Package observable;


Public class WeatherStation {


        Public static void main(String[] args) {

                WeatherData weatherData = new WeatherData();CurrentConditionsDisplay currentConditions =
                new CurrentConditionsDisplay(weatherData); StatisticsDisplay statisticsDisplay = new
                StatisticsDisplay(weatherData);
                ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);


                weatherData.setMeasurements(80, 65, 30.4f);

                weatherData.setMeasurements(82, 70, 29.2f);

                weatherData.setMeasurements(78, 90, 29.2f);

        }

}
```

WeatherStationHeatIndex.java

```
 package observable;


 public class WeatherStationHeatIndex {


          public static void main(String[] args) {
                    WeatherData weatherData = new WeatherData();
                    CurrentConditionsDisplay currentConditions = new
 CurrentConditionsDisplay(weatherData);

                    StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);

                    ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

                    HeatIndexDisplay heatIndexDisplay = new HeatIndexDisplay(weatherData);


                    weatherData.setMeasurements(80, 65, 30.4f);

                    weatherData.setMeasurements(82, 70, 29.2f);

                    weatherData.setMeasurements(78, 90, 29.2f);

          }
```

```
}
```

WeatherData.java

```
package observable;

import java.util.Observable;
import java.util.Observer;
public class WeatherData
extends Observable {
private float temperature;
        private float
        humidity; private
        float pressure;

        public WeatherData() { }

        public void measurementsChanged() {
                setChanged();
                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;
        }
}
```

StatisticsDisplay. java

```
package observable;

import java.util.Observable;
import java.util.Observer;

public class StatisticsDisplay implements Observer, DisplayElement {
        private float maxTemp = 0.0f;
        private float minTemp = 200;
        private float tempSum= 0.0f;
        private int numReadings;

        public StatisticsDisplay(Observable observable) {
                observable.addObserver(this);
        }

        public void update(Observable observable, Object arg) {
                if (observable instanceof WeatherData) {
                        WeatherData weatherData = (WeatherData)observable;
                        float temp = weatherData.getTemperature();
                        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);
        }
}
```

HeatIndexDisplay.java

```
package observable;

 import java.util.Observable;
 import java.util.Observer;

 public class HeatIndexDisplay implements Observer, DisplayElement {
        float heatIndex = 0.0f;

        public HeatIndexDisplay(Observable observable) {
                observable.addObserver(this);
        }

        public void update(Observable observable, Object arg) {
                if (observable instanceof WeatherData) {
                        WeatherData weatherData = (WeatherData)observable;
                        float t = weatherData.getTemperature();
                        float rh = weatherData.getHumidity();
                        heatIndex = (float)
                                (
                                (16.923 + (0.185212 * t))
                                + (5.37941 * rh) -
                                (0.100254 * t * rh) +
                                (0.00941695 * (t * t)) +
                                (0.00728898 * (rh * rh)) +
                                (0.000345372 * (t * t * rh)) -
                                (0.000814971 * (t * rh * rh)) +
                                (0.0000102102 * (t * t * rh * rh)) -
                                (0.000038646 * (t * t * t)) +
                                (0.0000291583 * (rh * rh * rh)) +
                                (0.00000142721 * (t * t * t * rh)) +
                                    (0.000000197483 * (t * rh * rh * rh)) -
                                    (0.0000000218429 * (t * t * t * rh * rh)) +
                                    (0.000000000843296 * (t * t * rh * rh * rh)) -
                                    (0.0000000000481975 * (t * t * t * rh * rh * rh)));
                              display();
                        }
                  }
                  public void display() {
                        System.out.println("Heat index is " + heatIndex);
                  }
      }
```

ForecastDisplay.java

```
Package observable;

Import java.util.Observable;
Import java.util.Observer;

Public class ForecastDisplay implements Observer, DisplayElement
      { Private float currentPressure = 29.92f;
      Private float lastPressure;

      Public ForecastDisplay(Observable observable) {
            Observable.addObserver(this);
      }

      Public void update(Observable observable, Object arg)
            { If (observable instanceof WeatherData) {
                  WeatherData weatherData = (WeatherData)observable;
                  lastPressure = currentPressure;
                  currentPressure = weatherData.getPressure();
                  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");
                }
        }
}
```

DisplayElement.java

```
 package observable;

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

CurrentConditionsDisplay.java

```
 package observable;

 import java.util.Observable;
 import java.util.Observer;

 public class CurrentConditionsDisplay implements Observer, DisplayElement {
          Observable observable;
          private float
          temperature; private
          float humidity;

          public CurrentConditionsDisplay(Observable observable) {
                    this.observable = observable;
                    observable.addObserver(this);
          }
```

```
        public void update(Observable obs, Object arg) {
                if (obs instanceof WeatherData) {
                        WeatherData weatherData = (WeatherData)obs;this.temperature =
                        weatherData.getTemperature(); this.humidity = weatherData.getHumidity();
                        display();
                }
        }

        public void display() {
                System.out.println("Current conditions: " + temperature
                        + "F degrees and " + humidity + "% humidity");
        }
}
```

Q4. 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.

ChicagoPizzaStore.java

```
package factorypattern;

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 factorypattern;
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

```
package factorypattern;

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

```
package factorypattern;
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 factorypattern;

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");
		}
		}
```

ChicagoStyleVeggiePizza. Java

```
package factorypattern;

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");
        }
}
```

DependentPizzaStore. Java

```
package factorypattern;

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("E
                                      rror: invalid type of
                                      pizza"); return null;
                }
```

```
                        pizza.prepare();
                        pizza.bake();
                        pizza.cut();
                        pizza.box();
                        return pizza;
                }
  }
```

NYPizzaStore. Java

```
  package factorypattern;

  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 factorypattern;

  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 factorypattern;

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 factorypattern;

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 factorypattern;

  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");
                }
  }
```

Pizza. Java

```
Package factorypattern;

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

```
package factorypattern;

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;
        }
}
```

PizzaTestDrive. Java

```
package factorypattern;

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");
        }
}
```

Q5. Write a Java Program to implement command pattern to test Remote Control.

CeilingFan. Java

```
 package commandpattern;

 public class CeilingFan {
        String location =
        ""; int level;
        public static final int HIGH = 2;
        public static final int MEDIUM = 1;
        public static final int LOW = 0;

        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 medium() {
                // turns the ceiling fan on to medium
                level = MEDIUM;
                System.out.println(location + " ceiling fan is on medium");
        }

        public void low() {
                // turns the ceiling fan on to
                low level = LOW;
                System.out.println(location + " ceiling fan is on low");
        }
```

```
            public void off() {
                        // turns the ceiling fan
                        off level = 0;
                        System.out.println(location + " ceiling fan is off");
            }

            public int getSpeed() {
                        return level;
            }
  }
```

CeilingFanOffCommand. Java

```
  package commandpattern;

  public class CeilingFanOffCommand implements Command {
            CeilingFan ceilingFan;

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

Command. Java

```
  package commandpattern;

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

NoCommand. Java

```
  package commandpattern;

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

RemoteControl. Java

```
package commandpattern;

import java.util.*;

//
// This is the invoker
//
public class RemoteControl {
        Command[] onCommands;
        Command[]
        offCommands;

        public RemoteControl() {
                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;
                }
        }

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

        public void onButtonWasPushed(int slot) {
                onCommands[slot].execute();
        }

        public void offButtonWasPushed(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()
                                + " " + offCommands[i].getClass().getName() + "\n");
                }
                return stringBuff.toString();
        }
}
```

RemoteLoader. Java

```
package commandpattern;

public class RemoteLoader {

        public static void main(String[] args) {
                RemoteControl remoteControl = new RemoteControl();

                //Light livingRoomLight = new Light("Living Room");
                //Light kitchenLight = new Light("Kitchen");
                CeilingFan ceilingFan= new CeilingFan("Living Room");
                //GarageDoor garageDoor = new GarageDoor("");
                //Stereo stereo = new Stereo("Living Room");

                /*LightOnCommand livingRoomLightOn =
                                new LightOnCommand(livingRoomLight);
                LightOffCommand livingRoomLightOff =
                                new LightOffCommand(livingRoomLight);
                LightOnCommand kitchenLightOn =
                                new LightOnCommand(kitchenLight);
                LightOffCommand kitchenLightOff =
                                new LightOffCommand(kitchenLight);*/

                CeilingFanOnCommand ceilingFanOn =
                                    new CeilingFanOnCommand(ceilingFan);
                    CeilingFanOffCommand ceilingFanOff =
                                    new CeilingFanOffCommand(ceilingFan);

                    /*GarageDoorUpCommand garageDoorUp =
                                    new GarageDoorUpCommand(garageDoor);
                    GarageDoorDownCommand garageDoorDown =
                                    new GarageDoorDownCommand(garageDoor);

                    StereoOnWithCDCommand stereoOnWithCD =
                                    new StereoOnWithCDCommand(stereo);
                    StereoOffCommand stereoOff =
                                    new StereoOffCommand(stereo);*/

                    //remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);
                    //remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);
                    remoteControl.setCommand(2, ceilingFanOn, ceilingFanOff);
                    //remoteControl.setCommand(3, stereoOnWithCD, stereoOff);*/

                    System.out.println(remoteControl);

                    remoteControl.onButtonWasPushed(0);
                    remoteControl.offButtonWasPushed(0);
                    remoteControl.onButtonWasPushed(1);
                    remoteControl.offButtonWasPushed(1);
                    remoteControl.onButtonWasPushed(2);
                    remoteControl.offButtonWasPushed(2);
                    remoteControl.onButtonWasPushed(3);
                    remoteControl.offButtonWasPushed(3);
          }
}
```

Q6. Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu.

El. Java

```
Package iteratorpattern;


Import java.util.*;


Public class EI {

        Public static void main (String args[]) {

                Vector v = new Vector(Arrays.asList(args));

                Enumeration enumeration = v.elements();

                While (enumeration.hasMoreElements()) {

                        System.out.println(enumeration.nextElement());

                }

                Iterator iterator =

                v.iterator(); While

                (iterator.hasNext()) {

                        System.out.println(iterator.next());

                }

        }

}
```

EnumerationIterator. Java

```
 package iteratorpattern;


 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. Java

```
package iteratorpattern;

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());
                    }
            }
}
```

IteratorEnumeration. Java

```
package iteratorpattern;

import java.util.*;

public class IteratorEnumeration implements Enumeration
            { Iterator iterator;

            public IteratorEnumeration(Iterator iterator)
                    { this.iterator = iterator;
```

}

```
            public boolean hasMoreElements() {
                    return iterator.hasNext();
            }

            public Object nextElement() {
                    return iterator.next();
            }
    }
```

IteratorEnumerationTestDrive. Java

```
Package iteratorpattern;

Import java.util.*;

Public class IteratorEnumerationTestDrive {
        Public static void main (String args[]) {

                String[] str= {“Apple” ,”Tomato”, “Banana”, “Orange”};

                ArrayList l = new ArrayList(Arrays.asList(str));
                Enumeration enumeration = new
                IteratorEnumeration(l.iterator()); While
                (enumeration.hasMoreElements()) {
                        System.out.println(enumeration.nextElement());
                }
        }
}
```