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().

**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**

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;

}

}

**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");

}

}

}

**HeatIndexDisplay.java**

public class HeatIndexDisplay implements Observer, DisplayElement {

float heatIndex = 0.0f;

private WeatherData weatherData;

public HeatIndexDisplay(WeatherData weatherData) {

this.weatherData = weatherData;

weatherData.registerObserver(this);

}

public void update(float t, float rh, float pressure) {

heatIndex = computeHeatIndex(t, rh);

display();

}

private float computeHeatIndex(float t, float rh) {

float index = (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)));

return index;

}

public void display() {

System.out.println("Heat index is " + heatIndex);

}

}

**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);

}

}

**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");

}

}

**WeatherStation.java**

public class WeatherStation {

public static void main(String[] args) {

WeatherData weatherData = new WeatherData();

CurrentConditionsDisplay currentDisplay =

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);

}

}

**OUTPUT:**

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

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

**LowerCaseInputStream.java**

import java.io.\*;

import java.util.\*;

//extend FilterInputStream which is abstract decorator for all InputStreams

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;

}

}

**InputTest.java**

class InputTest

{

public static void main(String[] args) throws IOException

{

int c;

try

{

//set up the FileInputStream and decorate it,first with a BufferedInputStream and then our

//new LowerCaseInputStream filter

InputStream in =new LowerCaseInputStream(new BufferedInputStream(new FileInputStream("a.txt")));

// Read the charater until end of file and print it

while((c = in.read()) >= 0)

{

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

}

in.close();

}

catch(IOException e)

{

e.printStackTrace();

}

}

}

**OUTPUT:**

hi this is uppercase to lowercase program.

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

**PizzaStore.java**

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;

}

}

**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;

}

}

**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;

NYStyleCheesePizza();

}else if(item.equals("clam")){

return new;

NYStyleCheesePizza();

}else if(item.equals("pepperoni")){

return new;

NYStyleCheesePizza();

}else return null;

}

}

**Pizza.java**

import java.util.ArrayList;

public abstract class Pizza {

String name;

String dough;

String sauce;

ArrayList<String> toppings = new ArrayList<String>();

void prepare() {

System.out.println("Prepare " + name);

System.out.println("Tossing dough...");

System.out.println("Adding sauce...");

System.out.println("Adding toppings: ");

for (String topping : toppings) {

System.out.println(" " + topping);

}

}

void bake() {

System.out.println("Bake for 25 minutes at 350");

}

void cut() {

System.out.println("Cut 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 (String topping : toppings) {

display.append(topping + "\n");

}

return display.toString();

}

}

**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");

}

}

**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");

}

}

**ChicagoStyleClamPizza.java**

public class ChicagoStyleClamPizza extends Pizza {

public ChicagoStyleClamPizza() {

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("Clams");

toppings.add("Jalapeons");

}

void cut() {

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

}

}

**ChicagoStyleCheesePizza.java**

public class ChicagoStyleCheesePizza extends Pizza {

public ChicagoStyleCheesePizza() {

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("Mayo");

toppings.add("Cheddar");

}

void cut() {

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

}

}

**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("First order was a " + pizza.getName() + "\n");

pizza = nyStore.orderPizza("cheese");

System.out.println("Second order was a " + pizza.getName() + "\n");

}

}

**OUTPUT:**

---Making a NY Style Sauce and Cheese Pizza---

Prepare NY Style Sauce and Cheese Pizza

Tossing dough...

Adding sauce...

Adding toppings:

Grated Reggiano Cheese

Bake for 25 minutes at 350

Cut the pizza into diagonal slices

Place pizza in official PizzaStore box

First order was a NY Style Sauce and Cheese Pizza

---Making a NY Style Sauce and Cheese Pizza---

Prepare NY Style Sauce and Cheese Pizza

Tossing dough...

Adding sauce...

Adding toppings:

Grated Reggiano Cheese

Bake for 25 minutes at 350

Cut the pizza into diagonal slices

Place pizza in official PizzaStore box

Second order was a NY Style Sauce and Cheese Pizza

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

**Singleton.java**

public class Singleton{

private static Singleton uniqueInstance;

private Singleton() {

System.out.println("Instance has been Created");

}

public static Singleton getInstance() {

if(uniqueInstance== null) {

synchronized (Singleton.class) {

if(uniqueInstance == null) {

uniqueInstance=new Singleton();

}

}

}

return uniqueInstance;

}

public static void main(String[] args) {

Thread t1= new Thread(new Runnable() {

public void run() {

Singleton obj=Singleton.getInstance();

}

});

Thread t2=new Thread(new Runnable() {

public void run() {

Singleton obj=Singleton.getInstance();

}

});

t1.start();

t2.start();

}

}

**OUTPUT:**

Instance has been Created

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

**Command.java**

interface Command{

public void execute();

}

**Light.java**

public 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();

}

}

**LightOffCommand.java**

public class LightOffCommand implements Command

{

Light light;

public LightOffCommand(Light light)

{

this.light = light;

}

public void execute()

{

light.off();

}

}

**LightOnCommand.java**

public 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();

}

}

**Stereo.java**

public 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);

}

}

**StereoOffCommand.java**

public class StereoOffCommand implements Command

{

Stereo stereo;

public StereoOffCommand(Stereo stereo)

{

this.stereo = stereo;

}

public void execute()

{

stereo.off();

}

}

**StereoOnWithCDCommand.java**

public class StereoOnWithCDCommand implements Command

{

Stereo stereo;

public StereoOnWithCDCommand(Stereo stereo)

{

this.stereo = stereo;

}

public void execute()

{

stereo.on();

stereo.setCD();

stereo.setVolume(11);

}

}

**SimpleRemoteControl.java**

public 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();

}

}

**RemoteControlTest.java**

public 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();

}

}

**OUTPUT:**

Light is on

Stereo is on

Stereo is set for CD input

Stereo volume set to 11

Stereo is off

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

**Command.java**

interface Command {

public void execute();

public void undo();

}

**NoCommand.java**

class NoCommand implements Command {

public void execute() { }

public void undo() { }

}

**CeilingFan.java**

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;

}

}

**CeilingFanHighCommand.java**

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**

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**

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**

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();

}

}

}

**RemoteControlWithUndo.java**

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**

class RemoteLoader {

public static void main(String[] args) {

RemoteControlWithUndo remoteControl = new RemoteControlWithUndo();

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:**

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.

**EnumerationIterator.java**

import java.util.\*;

// Enumeration interface is used to get elements from vector

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();

}

}

class EnumIterator {

public static void main (String args[]) {

Vector v = new Vector(Arrays.asList("JAVA","CPP","SQL","HTML"));

Iterator iterator = new EnumerationIterator(v.elements());

while (iterator.hasNext()) {

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

}

}

}

**IterEnum.java**

import java.util.\*;

public class IterEnum

{

public static void main(String[] args)

{

String[] wordArr = {"Book", "Number", "Place", "Lemon", "Apple", "Tree"};

Vector<String> wordList = new Vector<>(Arrays.asList (wordArr)); //directly convert array to vector

System.out.println("\nThe word list: \n"+wordList + "\n");

Enumeration<String> vectorEnum = wordList.elements();

//Enumeration iterates through vector show elements one by one

while(vectorEnum.hasMoreElements())

{ //when vector Enum has more element to get System.out.println(vectorEnum.nextElement());

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

}

LinkedList<String> wordLinkedList = new LinkedList<>();

wordLinkedList.addAll(wordList); //add elements from vector to linked list add some additional items

wordLinkedList.add("Ball");

wordLinkedList.add("Mango");

wordLinkedList.remove("Book");

System.out.println("\nThe word list (LinkedList): \n" + wordLinkedList + "\n");

Iterator<String> it =wordLinkedList.iterator(); //the iterator it will point elements of the linked list

while(it.hasNext())

{ //when vector Enum has more element to get

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

}

}

}

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

**Menu.java**

import java.util.Iterator;

public interface Menu {

public Iterator<MenuItem> createIterator();

}

**CafeMenu.java**

import java.util.\*;

public class CafeMenu implements Menu {

HashMap<String, MenuItem> menuItems = new HashMap<String, MenuItem>();

public CafeMenu() {

addItem("Veggie Burger and Air Fries",

"Veggie burger on a whole wheat bun, lettuce, tomato, and fries",

true, 3.99);

addItem("Soup of the day",

"A cup of the soup of the day, with a side salad",

false, 3.69);

addItem("Burrito",

"A large burrito, with whole pinto beans, salsa, guacamole",

true, 4.29);

}

public void addItem(String name, String description,

boolean vegetarian, double price)

{

MenuItem menuItem = new MenuItem(name, description, vegetarian, price);

menuItems.put(name, menuItem);

}

public Map<String, MenuItem> getItems() {

return menuItems;

}

public Iterator<MenuItem> createIterator() {

return menuItems.values().iterator();

}

}

**DinerMenu.java**

import java.util.Iterator;

public class DinerMenu implements Menu {

static final int MAX\_ITEMS = 6;

int numberOfItems = 0;

MenuItem[] menuItems;

public DinerMenu() {

menuItems = new MenuItem[MAX\_ITEMS];

addItem("Vegetarian BLT",

"(Fakin') Bacon with lettuce & tomato on whole wheat", true, 2.99);

addItem("BLT",

"Bacon with lettuce & tomato on whole wheat", false, 2.99);

addItem("Soup of the day",

"Soup of the day, with a side of potato salad", false, 3.29);

addItem("Hotdog",

"A hot dog, with sauerkraut, relish, onions, topped with cheese",

false, 3.05);

addItem("Steamed Veggies and Brown Rice",

"A medly of steamed vegetables over brown rice", true, 3.99);

addItem("Pasta",

"Spaghetti with Marinara Sauce, and a slice of sourdough bread",

true, 3.89);

}

public void addItem(String name, String description,

boolean vegetarian, double price)

{

MenuItem menuItem = new MenuItem(name, description, vegetarian, price);

if (numberOfItems >= MAX\_ITEMS) {

System.err.println("Sorry, menu is full! Can't add item to menu");

} else {

menuItems[numberOfItems] = menuItem;

numberOfItems = numberOfItems + 1;

}

}

public MenuItem[] getMenuItems() {

return menuItems;

}

public Iterator<MenuItem> createIterator() {

return new DinerMenuIterator(menuItems);

//return new AlternatingDinerMenuIterator(menuItems);

}

// other menu methods here

}

**DinerMenuIterator.java**

import java.util.Iterator;

public class DinerMenuIterator implements Iterator<MenuItem> {

MenuItem[] list;

int position = 0;

public DinerMenuIterator(MenuItem[] list) {

this.list = list;

}

public MenuItem next() {

MenuItem menuItem = list[position];

position = position + 1;

return menuItem;

}

public boolean hasNext() {

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

return false;

} else {

return true;

}

}

public void remove() {

if (position <= 0) {

throw new IllegalStateException

("You can't remove an item until you've done at least one next()");

}

if (list[position-1] != null) {

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

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

}

list[list.length-1] = null;

}

}

}

**MenuItem.java**

public class MenuItem {

String name;

String description;

boolean vegetarian;

double price;

public MenuItem(String name,

String description,

boolean vegetarian,

double price)

{

this.name = name;

this.description = description;

this.vegetarian = vegetarian;

this.price = price;

}

public String getName() {

return name;

}

public String getDescription() {

return description;

}

public double getPrice() {

return price;

}

public boolean isVegetarian() {

return vegetarian;

}

}

**PancakeHouseMenu.java**

import java.util.ArrayList;

import java.util.Iterator;

public class PancakeHouseMenu implements Menu {

ArrayList<MenuItem> menuItems;

public PancakeHouseMenu() {

menuItems = new ArrayList<MenuItem>();

addItem("K&B's Pancake Breakfast",

"Pancakes with scrambled eggs and toast",

true,

2.99);

addItem("Regular Pancake Breakfast",

"Pancakes with fried eggs, sausage",

false,

2.99);

addItem("Blueberry Pancakes",

"Pancakes made with fresh blueberries and blueberry syrup",

true,

3.49);

addItem("Waffles",

"Waffles with your choice of blueberries or strawberries",

true,

3.59);

}

public void addItem(String name, String description,

boolean vegetarian, double price)

{

MenuItem menuItem = new MenuItem(name, description, vegetarian, price);

menuItems.add(menuItem);

}

public ArrayList<MenuItem> getMenuItems() {

return menuItems;

}

public Iterator<MenuItem> createIterator() {

return menuItems.iterator();

}

// other menu methods here

}

**Waitress.java**

import java.util.Iterator;

public class Waitress {

Menu pancakeHouseMenu;

Menu dinerMenu;

Menu cafeMenu;

public Waitress(Menu pancakeHouseMenu, Menu dinerMenu, Menu cafeMenu) {

this.pancakeHouseMenu = pancakeHouseMenu;

this.dinerMenu = dinerMenu;

this.cafeMenu = cafeMenu;

}

public void printMenu() {

Iterator<MenuItem> pancakeIterator = pancakeHouseMenu.createIterator();

Iterator<MenuItem> dinerIterator = dinerMenu.createIterator();

Iterator<MenuItem> cafeIterator = cafeMenu.createIterator();

System.out.println("MENU\n----\nBREAKFAST");

printMenu(pancakeIterator);

System.out.println("\nLUNCH");

printMenu(dinerIterator);

System.out.println("\nDINNER");

printMenu(cafeIterator);

}

private void printMenu(Iterator<MenuItem> iterator) {

while (iterator.hasNext()) {

MenuItem menuItem = iterator.next();

System.out.print(menuItem.getName() + ", ");

System.out.print(menuItem.getPrice() + " -- ");

System.out.println(menuItem.getDescription());

}

}

public void printVegetarianMenu() {

System.out.println("\nVEGETARIAN MENU\n---------------");

printVegetarianMenu(pancakeHouseMenu.createIterator());

printVegetarianMenu(dinerMenu.createIterator());

printVegetarianMenu(cafeMenu.createIterator());

}

public boolean isItemVegetarian(String name) {

Iterator<MenuItem> pancakeIterator = pancakeHouseMenu.createIterator();

if (isVegetarian(name, pancakeIterator)) {

return true;

}

Iterator<MenuItem> dinerIterator = dinerMenu.createIterator();

if (isVegetarian(name, dinerIterator)) {

return true;

}

Iterator<MenuItem> cafeIterator = cafeMenu.createIterator();

if (isVegetarian(name, cafeIterator)) {

return true;

}

return false;

}

private void printVegetarianMenu(Iterator<MenuItem> iterator) {

while (iterator.hasNext()) {

MenuItem menuItem = iterator.next();

if (menuItem.isVegetarian()) {

System.out.print(menuItem.getName() + ", ");

System.out.print(menuItem.getPrice() + " -- ");

System.out.println(menuItem.getDescription());

}

}

}

private boolean isVegetarian(String name, Iterator<MenuItem> iterator) {

while (iterator.hasNext()) {

MenuItem menuItem = iterator.next();

if (menuItem.getName().equals(name)) {

if (menuItem.isVegetarian()) {

return true;

}

}

}

return false;

}

}

**MenuTestDrive.java**

public class MenuTestDrive {

public static void main(String args[]) {

PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();

DinerMenu dinerMenu = new DinerMenu();

CafeMenu cafeMenu = new CafeMenu();

Waitress waitress = new Waitress(pancakeHouseMenu, dinerMenu, cafeMenu);

waitress.printMenu();

waitress.printVegetarianMenu();

System.out.println("\nCustomer asks, is the Hotdog vegetarian?");

System.out.print("Waitress says: ");

if (waitress.isItemVegetarian("Hotdog")) {

System.out.println("Yes");

} else {

System.out.println("No");

}

System.out.println("\nCustomer asks, are the Waffles vegetarian?");

System.out.print("Waitress says: ");

if (waitress.isItemVegetarian("Waffles")) {

System.out.println("Yes");

} else {

System.out.println("No");

}

}

}

**OUTPUT:**

MENU

----

BREAKFAST

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

Regular Pancake Breakfast, 2.99 -- Pancakes with fried eggs, sausage

Blueberry Pancakes, 3.49 -- Pancakes made with fresh blueberries and blueberry syrup

Waffles, 3.59 -- Waffles with your choice of blueberries or strawberries

LUNCH

Vegetarian BLT, 2.99 -- (Fakin') Bacon with lettuce & tomato on whole wheat

BLT, 2.99 -- Bacon with lettuce & tomato on whole wheat

Soup of the day, 3.29 -- Soup of the day, with a side of potato salad

Hotdog, 3.05 -- A hot dog, with sauerkraut, relish, onions, topped with cheese

Steamed Veggies and Brown Rice, 3.99 -- A medly of steamed vegetables over brown rice

Pasta, 3.89 -- Spaghetti with Marinara Sauce, and a slice of sourdough bread

DINNER

Soup of the day, 3.69 -- A cup of the soup of the day, with a side salad

Veggie Burger and Air Fries, 3.99 -- Veggie burger on a whole wheat bun, lettuce, tomato, and fries

Burrito, 4.29 -- A large burrito, with whole pinto beans, salsa, guacamole

VEGETARIAN MENU

---------------

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

Blueberry Pancakes, 3.49 -- Pancakes made with fresh blueberries and blueberry syrup

Waffles, 3.59 -- Waffles with your choice of blueberries or strawberries

Vegetarian BLT, 2.99 -- (Fakin') Bacon with lettuce & tomato on whole wheat

Steamed Veggies and Brown Rice, 3.99 -- A medly of steamed vegetables over brown rice

Pasta, 3.89 -- Spaghetti with Marinara Sauce, and a slice of sourdough bread

Veggie Burger and Air Fries, 3.99 -- Veggie burger on a whole wheat bun, lettuce, tomato, and fries

Burrito, 4.29 -- A large burrito, with whole pinto beans, salsa, guacamole

Customer asks, is the Hotdog vegetarian?

Waitress says: No

Customer asks, are the Waffles vegetarian?

Waitress says: Yes

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.

**State.java**

public interface State {

public void insertQuarter();

public void ejectQuarter();

public void turnCrank();

public void dispense();

public void refill();

}

**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";

}

}

**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";

}

}

**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";

}

}

**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 #2004");

result.append("\nInventory: " + count + " gumball");

if (count != 1) {

result.append("s");

}

result.append("\n");

result.append("Machine is " + state + "\n");

return result.toString();

}

}

**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);

}

}

**OUTPUT:**

Mighty Gumball, Inc.

Java-enabled Standing Gumball Model #2004

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 #2004

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 #2004

Inventory: 4 gumballs

Machine is waiting for quarter

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

11.Design simple HR Application using Spring Framework