**Assignment**

Name:-Suraj Shantaram Darde.

Class: - M.Sc (CS)-II

Div:-A

Subject:-Software Architecture & Design Pattern Practical

Seat No-53

Q1.Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

import java.io.\*;

import java.util.\*;

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;

}

}

class uplo

{

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

{

int c;

try

{

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

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

{

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

}

in.close();

}

catch(IOException e)

{

e.printStackTrace();

}

}

}

Q2. 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().

import java.util.\*;

interface Observer {

public void update(float temp, float humidity, float pressure);

}

interface DisplayElement {

public void display();

}

interface Subject {

public void registerObserver(Observer o);

public void removeObserver(Observer o);

public void notifyObservers();

}

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;

}

}

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 weather, rainy weather");

}

}

}

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);

}

}

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("Avgerage/Maximum/Minimum temperature = " + (tempSum / numReadings)

+ "/" + maxTemp + "/" + minTemp);

}

}

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");

}

}

class weather {

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(70, 55, 40.4f);

weatherData.setMeasurements(72, 60, 39.2f);

weatherData.setMeasurements(68, 80, 39.2f);

}

}

Q3.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.

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("--- Making a " + pizza.getName() + " ---");

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");

}}

Q4.Write a Java Program to implement Adapter pattern for Enumeration iterator

import java.util.\*;

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 EnumerationIteratorTestDrive1 {

public static void main (String args[]) {

Vector v = new Vector(Arrays.asList("apple","mango","grapes"));

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

while (iterator.hasNext()) {

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

}

}

}

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

public interface Command {

public void execute();

public void undo();

}

public class CeilingFanOffCommand implements Command {

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanOffCommand(CeilingFan ceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.off();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

public class CeilingFanMediumCommand implements Command {

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanMediumCommand(CeilingFan ceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.medium();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

public class CeilingFanHighCommand implements Command {

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanHighCommand(CeilingFan ceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.high();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

public class CeilingFanLowCommand implements Command {

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanLowCommand(CeilingFan ceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.low();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

public class CeilingFan {

public static final int HIGH = 3;

public static final int MEDIUM = 2;

public static final int LOW = 1;

public static final int OFF = 0;

String location;

int speed;

public CeilingFan(String location) {

this.location = location;

speed = OFF;

}

public void high() {

speed = HIGH;

System.out.println(location + " ceiling fan is on high");

}

public void medium() {

speed = MEDIUM;

System.out.println(location + " ceiling fan is on medium");

}

public void low() {

speed = LOW;

System.out.println(location + " ceiling fan is on low");

}

public void off() {

speed = OFF;

System.out.println(location + " ceiling fan is off");

}

public int getSpeed() {

return speed;

}

}