NAME ASHAR ALI

REGNO FA20-BSE-158

DATE 16/11/2023

lab mid term

**OBSERVER AND DECORATOR DESIGN PATTERN:**

**OBSERVER CLASS:**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\*/

package com.mycompany.labmidobserver;

/\*\*

\*

\* @author Ashar's Book

\*/

import java.util.ArrayList;

import java.util.List;

// Observer interface

interface MatchObserver {

void update(Match match);

}

// Subject class

class MatchList {

private List<Match> matches;

private List<MatchObserver> observers;

public MatchList() {

matches = new ArrayList<>();

observers = new ArrayList<>();

}

public void addMatchObserver(MatchObserver observer) {

observers.add(observer);

}

public void removeMatchObserver(MatchObserver observer) {

observers.remove(observer);

}

public void notifyMatchObservers(Match match) {

for (MatchObserver observer : observers) {

observer.update(match);

}

}

public List<Match> getMatches() {

return matches;

}

public void addMatch(Match match) {

matches.add(match);

notifyMatchObservers(match);

}

}

// Concrete Subject class

class Match {

private int matchId;

private String[] teams;

private String status;

public Match(int matchId, String[] teams) {

this.matchId = matchId;

this.teams = teams;

this.status = "Not started";

}

public int getMatchId() {

return matchId;

}

public String[] getTeams() {

return teams;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

// Concrete Observer class

class MatchViewer implements MatchObserver {

@Override

public void update(Match match) {

System.out.println("Match updated: " + match.getTeams()[0] + " vs " + match.getTeams()[1] +

", Status: " + match.getStatus());

}

}

// Client code

public class Main {

public static void main(String[] args) {

MatchList matchList = new MatchList();

MatchObserver viewer1 = new MatchViewer();

MatchObserver viewer2 = new MatchViewer();

matchList.addMatchObserver(viewer1);

matchList.addMatchObserver(viewer2);

Match match1 = new Match(1, new String[]{"TeamA", "TeamB"});

Match match2 = new Match(2, new String[]{"TeamC", "TeamD"});

matchList.addMatch(match1);

matchList.addMatch(match2);

}

}

**DECORATOR CLASS:**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template

\*/

package com.mycompany.labmidobserver;

/\*\*

\*

\* @author Ashar's Book

\*/

import java.util.ArrayList;

import java.util.List;

// Component interface

interface MealItem {

float cost();

String description();

}

// Concrete Component class

class BasicMealItem implements MealItem {

@Override

public float cost() {

return 10; // Basic cost

}

@Override

public String description() {

return "Basic Meal Item";

}

}

// Decorator class

abstract class MealDecorator implements MealItem {

protected MealItem mealItem;

public MealDecorator(MealItem mealItem) {

this.mealItem = mealItem;

}

@Override

public float cost() {

return mealItem.cost();

}

@Override

public String description() {

return mealItem.description();

}

}

// Concrete Decorator class

class DecorationDecorator extends MealDecorator {

public DecorationDecorator(MealItem mealItem) {

super(mealItem);

}

@Override

public float cost() {

return mealItem.cost() + 5; // Additional cost for decoration

}

@Override

public String description() {

return mealItem.description() + " with Decoration";

}

}

// Client code

public class MealOrder {

public static void main(String[] args) {

MealItem basicItem = new BasicMealItem();

MealItem decoratedItem = new DecorationDecorator(basicItem);

System.out.println("Meal Order:");

displayItem(basicItem);

displayItem(decoratedItem);

float totalCost = calculateCost(basicItem, decoratedItem);

System.out.println("\nTotal Cost: $" + totalCost);

}

private static void displayItem(MealItem item) {

System.out.println(item.description() + ", Cost: $" + item.cost());

}

private static float calculateCost(MealItem... items) {

float totalCost = 0;

for (MealItem item : items) {

totalCost += item.cost();

}

return totalCost;

}

}

**GITHUB LINK:**

