UMAMA NOOR

FA20-BSE-055

import java.util.ArrayList;

import java.util.List;

// Singleton for Settings

class Settings {

private static Settings instance;

private Settings() {

// private constructor to prevent instantiation

}

public static synchronized Settings getInstance() {

if (instance == null) {

instance = new Settings();

}

return instance;

}

// Add settings specific to your application

}

// Mediator to control coupling

class Mediator {

private List<Communication> communications = new ArrayList<>();

public void registerCommunication(Communication communication) {

communications.add(communication);

}

public void notifyCommunications() {

for (Communication communication : communications) {

communication.receiveMessage("Broadcast message");

}

}

}

// Facade for easy interface

class ExamSystemFacade {

private Mediator mediator;

public ExamSystemFacade(Mediator mediator) {

this.mediator = mediator;

}

// Other methods for exam scheduling, progress monitoring, etc.

public void sendNotification(Notification notification) {

mediator.registerCommunication(notification);

}

}

// Strategy for different exam conduction

interface ExamConductionStrategy {

void conductExam();

}

class TimedExamStrategy implements ExamConductionStrategy {

@Override

public void conductExam() {

// Implementation for a timed exam

}

}

class OpenBookExamStrategy implements ExamConductionStrategy {

@Override

public void conductExam() {

// Implementation for an open-book exam

}

}

// Builder for Exam

class ExamBuilder {

private List<Question> questions = new ArrayList<>();

public ExamBuilder addQuestion(Question question) {

questions.add(question);

return this;

}

public Exam build() {

return new Exam(questions);

}

}

class Exam {

private List<Question> questions;

public Exam(List<Question> questions) {

this.questions = questions;

}

// Methods to get/set questions, conduct exam, etc.

}

// Iterator for Stakeholder or Contents Iteration

interface Iterator<T> {

boolean hasNext();

T next();

}

class StakeholderIterator implements Iterator<Stakeholder> {

// Implementation for iterating over stakeholders

}

// Template Method for Taking Exams

abstract class ExamTemplate {

public void takeExam() {

// Common steps for taking an exam

conductInstructions();

conductExam();

submitExam();

}

protected abstract void conductInstructions();

protected abstract void conductExam();

protected abstract void submitExam();

}

// Observer for Real-Time Communication

interface Observer {

void update();

}

class ParentObserver implements Observer {

// Implementation for updating parent on child's progress

@Override

public void update() {

// Update logic

}

}

// Communication Hierarchy

interface Communication {

void receiveMessage(String message);

}

class Message implements Communication {

@Override

public void receiveMessage(String message) {

// Implementation for handling messages

}

}

class Announcement implements Communication {

@Override

public void receiveMessage(String message) {

// Implementation for handling announcements

}

}

class Feedback implements Communication {

@Override

public void receiveMessage(String message) {

// Implementation for handling feedback

}

}

class Survey implements Communication {

@Override

public void receiveMessage(String message) {

// Implementation for handling surveys

}

}

class Polling implements Communication {

@Override

public void receiveMessage(String message) {

// Implementation for handling polling

}

}

// Case Management

class CaseManagement {

private Mediator mediator;

public CaseManagement(Mediator mediator) {

this.mediator = mediator;

}

public void handleCase() {

// Case handling logic

mediator.notifyCommunications();

}

}

// Notification with mandatory feedback

class Notification implements Communication {

private Mediator mediator;

public Notification(Mediator mediator) {

this.mediator = mediator;

}

public void requestFeedback() {

// Request feedback logic

}

public void sendNotification() {

// Send notification logic

mediator.registerCommunication(this);

}

@Override

public void receiveMessage(String message) {

// Implementation for handling notifications

}

}

// Example Question classes

abstract class Question {

// Common properties/methods for all types of questions

}

class MCQQuestion extends Question {

// Implementation for multiple-choice questions

}

class TrueFalseQuestion extends Question {

// Implementation for true/false questions

}

// Other question types...

// Example Stakeholder classes

class Stakeholder {

// Common properties/methods for all stakeholders

}

// Main class

public class Main {

public static void main(String[] args) {

// Initialize components

Settings settings = Settings.getInstance();

Mediator mediator = new Mediator();

ExamSystemFacade examSystemFacade = new ExamSystemFacade(mediator);

CaseManagement caseManagement = new CaseManagement(mediator);

// Create and use objects

Notification notification = new Notification(mediator);

notification.requestFeedback();

notification.sendNotification();

// Handle case triggering notifications

caseManagement.handleCase();

}

}