ҚАЗАҚСТАН РЕСПУБЛИКАСЫНЫҢ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

СӘТБАЕВ УНИВЕРСИТЕТІ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ институты

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_кафедрасы



|  |
| --- |
|  |

ЗЕРТХАНАЛЫҚ ЖҰМЫС

Тақырыбы **Паттерны поведения. Команда. Шаблонный метод. Посредник**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Орындалған жұмыстың сапасы** | **Баға диапазоны** | **Алған %** |
| 1 | Орындалмаған  Себепсіз сабақтан қалуы | 0% |  |
| 2 | Орындалуы және білім алушының белсенділігі | 0-50% |  |
| 3 | Жұмыстың рәсімделуі | 0-20% |  |
| 4 | Анықтамаларды, техникалық әдебиеттерді, дәріс конспектілерін, пәннің оқу-әдістемелік кешенін пайдалана білуі | 0-5% |  |
| 5 | Техникалық құралдарды пайдалана білуі | 0-5% |  |
| 6 | Жұмысты қорғауы | 0-20% |  |
|  | Қорытынды | 0-100% |  |

Білім алушының аты-жөні

Шахмуханбетов Ханкелді

Мамандық шифрі

6B06102 Computer Science

Оқытушының аты-жөні

Иманбекова У.Н.

Алматы 2024 ж

Ф ҚазҰТЗУ 706-04. Зертханалық жұмыс

**Команда**

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

public class Light {  
 public void on() {  
 System.*out*.println("Свет включен.");  
 }  
  
 public void off() {  
 System.*out*.println("Свет выключен.");  
 }  
}

public class LightOffCommand implements Command {  
 private Light light;  
  
 public LightOffCommand(Light light) {  
 this.light = light;  
 }  
  
 @Override  
 public void execute() {  
 light.off();  
 }  
  
 @Override  
 public void undo() {  
 light.on();  
 }  
}

public class LightOnCommand implements Command {  
 private Light light;  
  
 public LightOnCommand(Light light) {  
 this.light = light;  
 }  
  
 @Override  
 public void execute() {  
 light.on();  
 }  
  
 @Override  
 public void undo() {  
 light.off();  
 }  
}

public class Television {  
 public void on() {  
 System.*out*.println("Телевизор включен.");  
 }  
  
 public void off() {  
 System.*out*.println("Телевизор выключен.");  
 }  
}

public class TelevisionOffCommand implements Command {  
 private Television tv;  
  
 public TelevisionOffCommand(Television tv) {  
 this.tv = tv;  
 }  
  
 @Override  
 public void execute() {  
 tv.off();  
 }  
  
 @Override  
 public void undo() {  
 tv.on();  
 }  
}

public class TelevisionOnCommand implements Command {  
 private Television tv;  
  
 public TelevisionOnCommand(Television tv) {  
 this.tv = tv;  
 }  
  
 @Override  
 public void execute() {  
 tv.on();  
 }  
  
 @Override  
 public void undo() {  
 tv.off();  
 }  
}

import java.util.ArrayList;  
import java.util.List;  
  
public class RemoteControl {  
 private Command onCommand;  
 private Command offCommand;  
 private List<String> commandLog; *// Лог для отслеживания выполненных команд* public RemoteControl() {  
 this.commandLog = new ArrayList<>();  
 }  
  
 public void setCommands(Command onCommand, Command offCommand) {  
 this.onCommand = onCommand;  
 this.offCommand = offCommand;  
 }  
  
 public void pressOnButton() {  
 if (onCommand != null) {  
 onCommand.execute();  
 logCommand("On");  
 } else {  
 System.*out*.println("Ошибка: Команда для включения не назначена.");  
 }  
 }  
  
 public void pressOffButton() {  
 if (offCommand != null) {  
 offCommand.execute();  
 logCommand("Off");  
 } else {  
 System.*out*.println("Ошибка: Команда для выключения не назначена.");  
 }  
 }  
  
 public void pressUndoButton() {  
 if (onCommand != null) {  
 onCommand.undo();  
 logCommand("Undo");  
 } else {  
 System.*out*.println("Ошибка: Команда для отмены не назначена.");  
 }  
 }  
  
 private void logCommand(String commandType) {  
 commandLog.add(commandType);  
 System.*out*.println("Команда '" + commandType + "' выполнена.");  
 }  
  
 public void showCommandLog() {  
 System.*out*.println("\nИстория команд:");  
 for (String log : commandLog) {  
 System.*out*.println(log);  
 }  
 }  
}

public class MacroCommand implements Command {  
 private Command[] commands;  
  
 public MacroCommand(Command[] commands) {  
 this.commands = commands;  
 }  
  
 @Override  
 public void execute() {  
 for (Command command : commands) {  
 command.execute();  
 }  
 }  
  
 @Override  
 public void undo() {  
 for (Command command : commands) {  
 command.undo();  
 }  
 }  
}

public class AirConditioner {  
 public void on() {  
 System.*out*.println("Кондиционер включен.");  
 }  
  
 public void off() {  
 System.*out*.println("Кондиционер выключен.");  
 }  
  
 public void setEcoMode() {  
 System.*out*.println("Кондиционер переведен в режим экономии энергии.");  
 }  
}

public class AirConditionerEcoModeCommand implements Command {  
 private AirConditioner ac;  
  
 public AirConditionerEcoModeCommand(AirConditioner ac) {  
 this.ac = ac;  
 }  
  
 @Override  
 public void execute() {  
 ac.setEcoMode();  
 }  
  
 @Override  
 public void undo() {  
 System.*out*.println("Отмена режима экономии энергии.");  
 }  
}

public class AirConditionerOffCommand implements Command {  
 private AirConditioner ac;  
  
 public AirConditionerOffCommand(AirConditioner ac) {  
 this.ac = ac;  
 }  
  
 @Override  
 public void execute() {  
 ac.off();  
 }  
  
 @Override  
 public void undo() {  
 ac.on();  
 }  
}

public class AirConditionerOnCommand implements Command {  
 private AirConditioner ac;  
  
 public AirConditionerOnCommand(AirConditioner ac) {  
 this.ac = ac;  
 }  
  
 @Override  
 public void execute() {  
 ac.on();  
 }  
  
 @Override  
 public void undo() {  
 ac.off();  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 Light livingRoomLight = new Light();  
 Television tv = new Television();  
 AirConditioner ac = new AirConditioner();  
  
 Command lightOn = new LightOnCommand(livingRoomLight);  
 Command lightOff = new LightOffCommand(livingRoomLight);  
  
 Command tvOn = new TelevisionOnCommand(tv);  
 Command tvOff = new TelevisionOffCommand(tv);  
  
 Command acOn = new AirConditionerOnCommand(ac);  
 Command acOff = new AirConditionerOffCommand(ac);  
 Command acEcoMode = new AirConditionerEcoModeCommand(ac);  
  
 RemoteControl remote = new RemoteControl();  
  
 remote.setCommands(lightOn, lightOff);  
 System.*out*.println("Управление светом:");  
 remote.pressOnButton();  
 remote.pressOffButton();  
 remote.pressUndoButton();  
  
 remote.setCommands(tvOn, tvOff);  
 System.*out*.println("\nУправление телевизором:");  
 remote.pressOnButton();  
 remote.pressOffButton();  
  
 remote.setCommands(acOn, acOff);  
 System.*out*.println("\nУправление кондиционером:");  
 remote.pressOnButton();  
 remote.setCommands(acEcoMode, null);  
 remote.pressOnButton();  
  
 Command[] partyModeCommands = { lightOn, tvOn, acOn };  
 MacroCommand partyMode = new MacroCommand(partyModeCommands);  
  
 System.*out*.println("\nАктивируем режим вечеринки:");  
 partyMode.execute();  
  
 System.*out*.println("\nОтмена режима вечеринки:");  
 partyMode.undo();  
 }  
}

**Өзін-өзі тексеру сұрақтары:**

**1. "Команда" паттернін қолданудың құрылғыларды басқарудағы артықшылығы қандай?**

**"Команда" паттерні** басқару әрекеттерін бөлек сыныптарға инкапсуляциялауға мүмкіндік береді. Бұл келесі артықшылықтарды береді:

* **Икемділік**: Құрылғыларды қосу, өшіру, режимдерді өзгерту сияқты әрекеттер нақты сыныптар арқылы орындалады, сондықтан жаңа құрылғылар мен әрекеттерді қосу оңай болады.
* **Кодты өзгеріссіз кеңейту**: Әрбір жаңа құрылғы үшін жаңа сыныптар қосылып, негізгі кодқа өзгеріс енгізбестен жүйе кеңейтіле береді.
* **Әрекеттерді болдырмау мүмкіндігі**: Әр команда өзіне тән "Execute" және "Undo" әдістерін іске асырып, әрекеттерді кері қайтаруға мүмкіндік береді.

**2. Пульттің функционалын өзгеріссіз кеңейту үшін қалай жаңа құрылғыларды қосуға болады?**

**Пульттің функционалын** өзгеріссіз кеңейту үшін, жаңа құрылғыларға арналған жаңа командалық сыныптарды қосып, пульттің қолданыстағы механизмін қолдану арқылы оларға жаңа батырмалар тағайындауға болады. Әр құрылғы өзіне тән командаларға ие болады (мысалы, "Телевизорды қосу", "Кондиционерді өшіру"). Бұл кезде негізгі пульт кодына өзгеріс енгізудің қажеті жоқ, тек жаңа командалар пультке тағайындалады.

**3. Қарапайым командалар мен макрокомандалардың айырмашылығы неде және олар жүйенің икемділігін қалай арттырады?**

**Қарапайым командалар** тек бір ғана әрекетті орындаса, **макрокомандалар** бірнеше команданы біріктіріп, оларды бір уақытта орындауға мүмкіндік береді. Бұл жүйенің икемділігін арттырады, себебі бір батырманы басу арқылы бірнеше құрылғыны басқаруға болады (мысалы, "Вечерний режим" — жарықты өшіріп, кондиционерді қосу). Макрокомандалар арқылы күрделі әрекеттер жиынтығын оңай басқаруға болады, бұл жүйені әлдеқайда қуатты және қолжетімді етеді.

**Шаблондық метод**

public abstract class Beverage {  
 *// Шаблондық метод* public void prepareRecipe() {  
 try { boilWater();  
 brew();  
 pourInCup();  
 if (customerWantsCondiments()) {  
 addCondiments();  
 }  
 } catch (Exception e) {  
 System.*out*.println("Ошибка при приготовлении напитка: " + e.getMessage());  
 }  
}  
  
 private void boilWater() {  
 System.*out*.println("Кипячение воды...");  
 }  
  
 private void pourInCup() {  
 System.*out*.println("Наливание в чашку...");  
 }  
  
 *// Абстрактты әдістер, оларды туынды сыныптарда жүзеге асырылады* protected abstract void brew();  
 protected abstract void addCondiments() throws Exception;  
  
 *// Хук әдісі, қосымшаларды өшіру үшін қайта анықталуы мүмкін* protected boolean customerWantsCondiments() {  
 return true; *// По умолчанию қосымшаларды қосамыз* }  
}

public class Coffee extends Beverage {  
 private String milkType;  
 private boolean ingredientsAvailable;  
  
 public Coffee(String milkType, boolean ingredientsAvailable) {  
 this.milkType = milkType;  
 this.ingredientsAvailable = ingredientsAvailable;  
 }  
  
 @Override  
 protected void brew() {  
 System.*out*.println("Заваривание кофе...");  
 }  
  
 @Override  
 protected void addCondiments() throws Exception {  
 if (!ingredientsAvailable) {  
 throw new Exception("Отсутствуют необходимые ингредиенты (сахар или молоко).");  
 }  
 System.*out*.println("Добавление сахара и " + milkType + " молока...");  
 }  
  
 @Override  
 protected boolean customerWantsCondiments() {  
 *//Пайдаланушыдан қосымшаларды қалауын сұраймыз (кіріс оқуына өзгертуге болады)* return true;*// Пайдаланушының қосымшаларды қалайтынын есептейік* }  
}

public class Tea extends Beverage {  
 private boolean ingredientsAvailable;  
  
 public Tea(boolean ingredientsAvailable) {  
 this.ingredientsAvailable = ingredientsAvailable;  
 }  
  
 @Override  
 protected void brew() {  
 System.*out*.println("Заваривание чая...");  
 }  
  
 @Override  
 protected void addCondiments() throws Exception {  
 if (!ingredientsAvailable) {  
 throw new Exception("Ингредиенты для чая отсутствуют.");  
 }  
 System.*out*.println("Добавление лимона...");  
 }  
  
 @Override  
 protected boolean customerWantsCondiments() {  
*// Пайдаланушының қосымшалардың қажет екеніне шешім қабылдай алатынын есептейік* return true; *// Егер қосымшалар қажет болса, true қайтарамыз* }  
}

public class HotChocolate extends Beverage {  
 private boolean hasMarshmallows;  
  
 public HotChocolate(boolean hasMarshmallows) {  
 this.hasMarshmallows = hasMarshmallows;  
 }  
  
 @Override  
 protected void brew() {  
 System.*out*.println("Смешивание горячего шоколада...");  
 }  
  
 @Override  
 protected void addCondiments() throws Exception {  
 if (!hasMarshmallows) {  
 throw new Exception("Маршмеллоу отсутствуют!");  
 }  
 System.*out*.println("Добавление маршмеллоу...");  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 Beverage tea = new Tea(true); *// Ингредиенты бар* System.*out*.println("Приготовление чая:");  
 tea.prepareRecipe();  
  
 System.*out*.println();  
  
 Beverage coffee = new Coffee("миндальное", true); *// Ингредиент бар* System.*out*.println("Приготовление кофе с миндальным молоком:");  
 coffee.prepareRecipe();  
  
 System.*out*.println();  
  
 Beverage hotChocolate = new HotChocolate(false); *// Маршмеллоу жоқ* System.*out*.println("Приготовление горячего шоколада:");  
 hotChocolate.prepareRecipe();  
 }  
}

**Өзін-өзі тексеру сұрақтары:**

1. **Шаблондық әдіс қалай жұмыс істейді және оның кәдімгі мұрагерліктен артықшылығы неде?** Шаблондық әдіс — бұл жоғары деңгейдегі алгоритмнің құрылымын анықтайтын паттерн. Негізгі класс шаблондық әдісті анықтайды, ал нақты қадамдарды туынды кластар жүзеге асырады. Артықшылығы — алгоритмнің негізгі құрылымын қайта пайдалануға мүмкіндік береді, ал нақты әрекеттер туынды класстарда оңай өзгертілуі мүмкін. Бұл кодтың түсініктілігін және техникалық қызмет көрсетуін арттырады.
2. **Хук әдістерді туынды кластардың мінез-құлқын баптау үшін қалай қолдануға болады?** Хук әдістер — бұл базалық класта анықталған, бірақ туынды кластарда қайта анықталуы мүмкін әдістер. Оларды туынды класта өзгерту арқылы пайдаланушылардың мінез-құлқын конфигурациялауға мүмкіндік береді. Мысалы, базалық класта "қосымшаларды қосу" хук әдісі бар, оны туынды класс қайта анықтап, пайдаланушының қосымша қосу ниетін сұрау арқылы жүзеге асыра алады.
3. **Нағыз жобаларда "Шаблондық әдіс" паттернін пайдалануды қажет ететін басқа жағдайлар қандай?** Шаблондық әдіс паттернін әртүрлі жағдайларда қолдануға болады, мысалы:

* Әртүрлі форматтағы құжаттарды генерациялау (PDF, DOCX, HTML) кезінде жалпы құрылымды сақтау.
* Ойындарда, кейіпкерлер мен деңгейлердің дамуын басқару үшін, бірдей механизмдерді қайта пайдалану.
* Деректерді өңдеу кезінде, жалпы деректерді жүктеу және трансформациялау алгоритмдерін пайдалану.

**Медиатор**

import java.util.List;  
  
public interface IMediator {  
 void sendMessage(String message, Colleague colleague);  
 void sendPrivateMessage(String message, Colleague sender, Colleague recipient);  
 void registerColleague(Colleague colleague);  
}

public abstract class Colleague {  
 protected IMediator mediator;  
 protected String name;  
  
 public Colleague(IMediator mediator, String name) {  
 this.mediator = mediator;  
 this.name = name;  
 }  
  
 public abstract void receiveMessage(String message);  
  
 public String getName() {  
 return name;  
 }  
}

import java.util.ArrayList;  
import java.util.List;  
  
public class ChatMediator implements IMediator {  
 private List<Colleague> colleagues;  
 private List<String> messageLog;  
  
 public ChatMediator() {  
 colleagues = new ArrayList<>();  
 messageLog = new ArrayList<>();  
 }  
  
 public void registerColleague(Colleague colleague) {  
 colleagues.add(colleague);  
 }  
  
 @Override  
 public void sendMessage(String message, Colleague sender) {  
 if (!colleagues.contains(sender)) {  
 System.*out*.println("Ошибка: Участник " + sender.getName() + " не зарегистрирован в чате.");  
 return;  
 }  
 logMessage(message);  
 for (Colleague colleague : colleagues) {  
 if (colleague != sender) {  
 colleague.receiveMessage(message);  
 }  
 }  
 }  
  
 @Override  
 public void sendPrivateMessage(String message, Colleague sender, Colleague recipient) {  
 if (!colleagues.contains(sender) || !colleagues.contains(recipient)) {  
 System.*out*.println("Ошибка: Один из участников не зарегистрирован в чате.");  
 return;  
 }  
 String privateMessage = "Приватное от " + sender.getName() + ": " + message;  
 recipient.receiveMessage(privateMessage);  
 logMessage(privateMessage);  
 }  
  
 public void logMessage(String message) {  
 messageLog.add(message);  
 }  
  
 public List<String> getMessageLog() {  
 return messageLog;  
 }  
}

public class User extends Colleague {  
 public User(IMediator mediator, String name) {  
 super(mediator, name);  
 }  
  
 public void send(String message) {  
 System.*out*.println(name + " отправляет сообщение: " + message);  
 mediator.sendMessage(message, this);  
 }  
  
 public void sendPrivateMessage(String message, User recipient) {  
 System.*out*.println(name + " отправляет приватное сообщение: " + message);  
 mediator.sendPrivateMessage(message, this, recipient);  
 }  
  
 @Override  
 public void receiveMessage(String message) {  
 System.*out*.println(name + " получил сообщение: " + message);  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 ChatMediator chatMediator = new ChatMediator();  
  
 User user1 = new User(chatMediator, "Алиса");  
 User user2 = new User(chatMediator, "Боб");  
 User user3 = new User(chatMediator, "Чарли");  
  
 *//Чатқа қатысушыларды тіркеу* chatMediator.registerColleague(user1);  
 chatMediator.registerColleague(user2);  
 chatMediator.registerColleague(user3);  
  
 *//Қатысушылар хабарлама алмасады* user1.send("Привет всем!");  
 user2.send("Привет, Алиса!");  
 user3.send("Всем привет!");  
  
 *// Жеке хабарлама жіберу* user1.sendPrivateMessage("Как дела, Боб?", user2);  
 user3.sendPrivateMessage("Привет, Алиса!", user1);  
  
 *// Хабар журналдарын көрсету* System.*out*.println("\nИстория сообщений:");  
 for (String log : chatMediator.getMessageLog()) {  
 System.*out*.println(log);  
 }  
 }  
}

**Өзін-өзі тексеру сұрақтары:**

1. **Көптеген қатысушылар бар жүйеде "Посредник" паттернін қолданудың артықшылықтары қандай?** "Посредник" паттерні қатысушылар арасындағы байланысты орталықтандырады, бұл жүйенің күрделілігін төмендетеді. Паттерн арқылы барлық қатысушылар тек ортақ интерфейс арқылы хабарлама алмасады, тікелей байланыссыз. Бұл кодты ұйымдастыруды жеңілдетеді, өзгерістер енгізуді оңайлатады, сонымен қатар жүйенің кеңейту мүмкіндігін арттырады.
2. **"Посредник" паттернін қалай өзгертуге болады, сонда ол белгілі бір қатысушылар тобына хабарламаларды жіберуге мүмкіндік берсін?** "Посредник" паттернінің интерфейсіне жаңа әдіс қосып, қатысушыларды топтарға бөлу керек. Мысалы, топтық хабарламаларды жіберу үшін медиаторда қатысушы топтарын тіркеуге арналған механизм жасау қажет. Бұл механизм арқылы белгілі бір топтағы барлық қатысушыларға хабарламалар жіберіледі, және қатысушылар топтарының идентификаторларын пайдалану арқылы байланыс орнатылады.
3. **Шынайы өмірде "Посредник" паттернін қолдану арқылы шешуге болатын міндеттер қандай?** "Посредник" паттернін шынайы өмірде келесі міндеттерді шешу үшін қолдануға болады:

* Чат жүйелері мен әлеуметтік желілерде қатысушылар арасындағы байланыс орнату.
* Бизнес құрылымдарындағы бөлімдер арасында ақпарат алмасу.
* Ойындар мен интерактивті платформаларда ойыншылар арасындағы коммуникацияны басқару.
* Қызметтер мен клиенттер арасында байланыс орнату, мысалы, интернет-дүкендерде.