ҚАЗАҚСТАН РЕСПУБЛИКАСЫНЫҢ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

СӘТБАЕВ УНИВЕРСИТЕТІ

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

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



ЛАБОРАТОРИЯЛЫҚ ЖҰМЫС №7

Тақырыбы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| № | Жұмысты орындау сапасы | Баға диапазоны | Орындалған  % |
| 1 | Орындалған жоқ | 0% |  |
| 2 | Орындалды | 0-50% |  |
| 3 | Материялдық өзіндік жүйелендіру | 0-10% |  |
| 4 | Талап етілген көлемде және көрсетілген мерзімде орындау | 0-5% |  |
| 5 | Қосымша ғылыми әдебиеттерді пайдалану | 0-5% |  |
| 6 | Орындаған тапсырманың ерекшелігі | 0-10% |  |
| 7 | СӨЖ-ді қорғау | 0-20% |  |
|  | Қорытынды: | 0-100% |  |

Оқытушы: \_Иманбекова У.Н.\_\_

Студент: \_Шаймерден Ақбота\_\_

Алматы 2024 ж

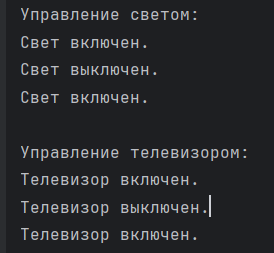
**Курс:** Шаблоны проектирования приложений

**Тема:** Модуль 08 Паттерны поведения. Команда. Шаблонный метод. Посредник

**Цель:**

Изучить и реализовать паттерн **Команда (Command)** на языке C#. Ваша задача — создать систему управления умным домом, где различные устройства могут управляться через команды, отправляемые с пульта дистанционного управления. Каждый тип команды должен быть инкапсулирован в отдельный класс.

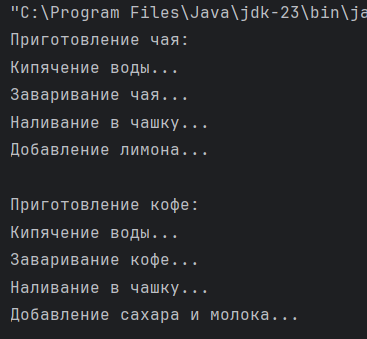
import java.util.ArrayList;  
import java.util.List;  
  
  
interface ICommand {  
 void execute();  
 void undo();  
}  
  
  
class Light {  
 public void on() {  
 System.*out*.println("Свет включен.");  
 }  
  
 public void off() {  
 System.*out*.println("Свет выключен.");  
 }  
}  
  
  
class LightOnCommand implements ICommand {  
 private Light light;  
  
 public LightOnCommand(Light light) {  
 this.light = light;  
 }  
  
 public void execute() {  
 light.on();  
 }  
  
 public void undo() {  
 light.off();  
 }  
}  
  
  
class LightOffCommand implements ICommand {  
 private Light light;  
  
 public LightOffCommand(Light light) {  
 this.light = light;  
 }  
  
 public void execute() {  
 light.off();  
 }  
  
 public void undo() {  
 light.on();  
 }  
}  
  
  
class Television {  
 public void on() {  
 System.*out*.println("Телевизор включен.");  
 }  
  
 public void off() {  
 System.*out*.println("Телевизор выключен.");  
 }  
}  
  
  
class TelevisionOnCommand implements ICommand {  
 private Television television;  
  
 public TelevisionOnCommand(Television television) {  
 this.television = television;  
 }  
  
 public void execute() {  
 television.on();  
 }  
  
 public void undo() {  
 television.off();  
 }  
}  
  
  
class TelevisionOffCommand implements ICommand {  
 private Television television;  
  
 public TelevisionOffCommand(Television television) {  
 this.television = television;  
 }  
  
 public void execute() {  
 television.off();  
 }  
  
 public void undo() {  
 television.on();  
 }  
}  
  
  
class RemoteControl {  
 private ICommand onCommand;  
 private ICommand offCommand;  
 private List<ICommand> history = new ArrayList<>();  
  
 public void setCommands(ICommand onCommand, ICommand offCommand) {  
 this.onCommand = onCommand;  
 this.offCommand = offCommand;  
 }  
  
 public void pressOnButton() {  
 if (onCommand != null) {  
 onCommand.execute();  
 history.add(onCommand);  
 } else {  
 System.*out*.println("Команда не назначена на кнопку включения.");  
 }  
 }  
  
 public void pressOffButton() {  
 if (offCommand != null) {  
 offCommand.execute();  
 history.add(offCommand);  
 } else {  
 System.*out*.println("Команда не назначена на кнопку выключения.");  
 }  
 }  
  
 public void pressUndoButton() {  
 if (!history.isEmpty()) {  
 ICommand lastCommand = history.remove(history.size() - 1);  
 lastCommand.undo();  
 } else {  
 System.*out*.println("Нет действий для отмены.");  
 }  
 }  
}  
  
  
public class SmartHomeControl {  
 public static void main(String[] args) {  
   
 Light livingRoomLight = new Light();  
 Television tv = new Television();  
  
 ICommand lightOn = new LightOnCommand(livingRoomLight);  
 ICommand lightOff = new LightOffCommand(livingRoomLight);  
  
 ICommand tvOn = new TelevisionOnCommand(tv);  
 ICommand tvOff = new TelevisionOffCommand(tv);  
  
 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.pressUndoButton();  
 }  
}



**Цель:**

Изучить и реализовать паттерн **Шаблонный метод (Template Method)** на языке C#. Ваша задача — создать систему для приготовления различных напитков (например, чай и кофе), где общий алгоритм приготовления будет описан в базовом классе, а конкретные шаги будут определяться в наследниках.

abstract class Beverage {  
 public final void prepareRecipe() {  
 boilWater();  
 brew();  
 pourInCup();  
 addCondiments();  
 }  
  
 private void boilWater() {  
 System.*out*.println("Кипячение воды...");  
 }  
  
 private void pourInCup() {  
 System.*out*.println("Наливание в чашку...");  
 }  
  
 protected abstract void brew();  
 protected abstract void addCondiments();  
}  
  
class Tea extends Beverage {  
 @Override  
 protected void brew() {  
 System.*out*.println("Заваривание чая...");  
 }  
  
 @Override  
 protected void addCondiments() {  
 System.*out*.println("Добавление лимона...");  
 }  
}  
  
class Coffee extends Beverage {  
 @Override  
 protected void brew() {  
 System.*out*.println("Заваривание кофе...");  
 }  
  
 @Override  
 protected void addCondiments() {  
 System.*out*.println("Добавление сахара и молока...");  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
   
 Beverage tea = new Tea();  
 System.*out*.println("Приготовление чая:");  
 tea.prepareRecipe();  
  
 System.*out*.println();  
  
   
 Beverage coffee = new Coffee();  
 System.*out*.println("Приготовление кофе:");  
 coffee.prepareRecipe();  
 }  
}



**Задания:**

1. **Реализуйте код по шагам выше.**
2. **Тестирование**:
   * Проверьте процесс приготовления чая и кофе с выводом на экран.
   * Убедитесь, что каждый напиток выполняет свои специфические шаги (например, добавление лимона в чай и сахара в кофе).
3. **Расширение функционала**:
   * Добавьте новый напиток, например, горячий шоколад, и реализуйте соответствующие шаги.
   * Добавьте в базовый класс возможность задать шаги, которые пользователь может пропускать (например, не добавлять сахар в кофе).
   * Реализуйте возможность кастомизации шагов (например, добавление разных типов молока в кофе).
4. **Обработка ошибок**:
   * Реализуйте обработку ситуации, когда шаг не может быть выполнен (например, отсутствие ингредиентов).

**Цель:**

Изучить и реализовать паттерн **Посредник (Mediator)** на языке C#. Ваша задача — создать систему управления диалогом между несколькими участниками, где взаимодействие между участниками организовано через объект-посредник.

**Описание задачи:**

Необходимо реализовать чат-систему, где несколько участников могут отправлять сообщения друг другу, но вместо прямого взаимодействия между участниками будет использоваться посредник. Посредник отвечает за передачу сообщений между участниками и управляет всей коммуникацией.

**Структура программы:**

1. **Интерфейс Посредник (IMediator)** — определяет контракт для посредника.
2. **Конкретный посредник (ConcreteMediator)** — реализует интерфейс посредника и управляет взаимодействием участников.
3. **Интерфейс участника (IColleague)** — определяет интерфейс для участников, которые будут взаимодействовать через посредника.
4. **Конкретные участники (ConcreteColleague)** — участники чата, которые обмениваются сообщениями через посредника.
5. **Клиентский код** — демонстрирует работу системы.
6. import java.util.ArrayList;  
   import java.util.List;  
     
   interface Mediator {  
    void sendMessage(String message, Colleague colleague);  
    void addColleague(Colleague colleague);  
   }  
     
   abstract class Colleague {  
    protected Mediator mediator;  
     
    public Colleague(Mediator mediator) {  
    this.mediator = mediator;  
    }  
     
    public abstract void receiveMessage(String message);  
   }  
     
   class ChatMediator implements Mediator {  
    private List<Colleague> colleagues;  
     
    public ChatMediator() {  
    this.colleagues = new ArrayList<>();  
    }  
     
    @Override  
    public void addColleague(Colleague colleague) {  
    colleagues.add(colleague);  
    }  
     
    @Override  
    public void sendMessage(String message, Colleague sender) {  
    for (Colleague colleague : colleagues) {  
    if (colleague != sender) {  
    colleague.receiveMessage(message);  
    }  
    }  
    }  
   }  
     
   class User extends Colleague {  
    private String name;  
     
    public User(Mediator mediator, String name) {  
    super(mediator);  
    this.name = name;  
    }  
     
    public void send(String message) {  
    System.*out*.println(name + " отправляет сообщение: " + message);  
    mediator.sendMessage(message, this);  
    }  
     
    @Override  
    public void receiveMessage(String message) {  
    System.*out*.println(name + " получил сообщение: " + message);  
    }  
   }  
     
   public class ChatApp {  
    public static void main(String[] args) {  
    ChatMediator mediator = new ChatMediator();  
    User user1 = new User(mediator, "Алиса");  
    User user2 = new User(mediator, "Боб");  
    User user3 = new User(mediator, "Чарли");  
     
    mediator.addColleague(user1);  
    mediator.addColleague(user2);  
    mediator.addColleague(user3);  
     
      
    user1.send("Привет всем!");  
    user2.send("Привет, Алиса!");  
    user3.send("Всем привет!");  
    }  
   }

