重 庆 大 学

学 生 实 验 报 告

实验课程名称 软件架构与设计模式

开课实验室 DS1501

学 院 大数据与软件学院 年级 2021 级 专业班 软件工程 X 班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2023 至 2024 学年第 2 学期

总 成 绩	
教师签名	

大数据与软件学院制

《软件架构与设计模式》实验报告

开课实验室：大数据与软件学院　　2024 年　　6 月　　13 日

学院	大数据与软件学院	年级、专业、班	2021 级软件工程 X 班	姓名	XXX	成绩	

课程名称	软件架构与设计模式	实验项目名　称	“行为型模式”的应用	指导教师	XXX

教师评语	教师签名： 年　月　日

一、实验目的

以设计命令模式(Command)为实验实例，掌握“行为型模式”的工作原理、应用环境和应用方法。

二、实验条件

计算机上安装 Word 字处理软件、Rational 软件、Eclipse 或其他编程环境。

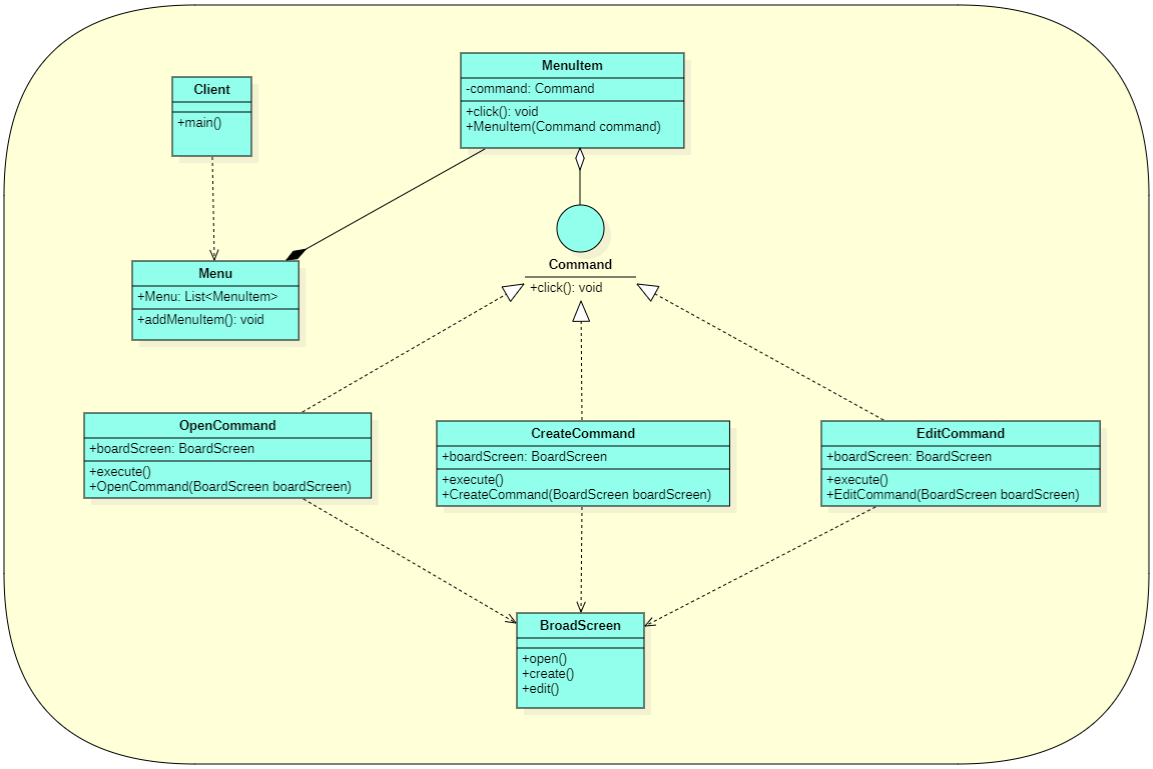
三、实验内容

用命令模式设计一个公告板系统模块：

某软件公司欲开发一个基于 Windows 平台的公告板系统。系统提供一个主菜单(Menu)，在主菜单中包含了一些菜单项(MenuItem)，可以通过 Menu 类的 addMenuItem()方法增加菜单项。菜单项的主要方法是 click()，每一个菜单项包含一个抽象命令类，具体命令类包括 OpenCommand(打开命令)，CreateCommand(新建命令)，EditCommand(编辑命令)等，命令类具有一个 execute()方法，用于调用公告板系统界面类(BoardScreen)的 open()、create()、edit()等方法。现使用命令模式设计该系统，使得 MenuItem 类与 BoardScreen 类的耦合度降低，绘制类图并编程实现。

四、实验步骤

1、用 UML 设计“公告板系统”的类图。



2、根据类图写出“公告板系统”的源代码。

(1)Command.java

```
// Command interface
public interface Command {
    3 个实现
    void execute();
}
```

(2)CreateCommand.java

```java
// CreateCommand class
public class CreateCommand implements Command {
    private BoardScreen boardScreen;

    public CreateCommand(BoardScreen boardScreen) { this.boardScreen = boardScreen; }

    @Override
    public void execute() { boardScreen.create(); }
}
```

(3)OpenCommand.java

```java
// OpenCommand class
public class OpenCommand implements Command {
    private BoardScreen boardScreen;

    public OpenCommand(BoardScreen boardScreen) { this.boardScreen = boardScreen; }

    @Override
    public void execute() { boardScreen.open(); }
}
```

(4)EditCommand.java

```java
// EditCommand class
public class EditCommand implements Command {
    private BoardScreen boardScreen;

    public EditCommand(BoardScreen boardScreen) { this.boardScreen = boardScreen; }

    @Override
    public void execute() { boardScreen.edit(); }
}
```

(5)BoardScreen.java

```java
// BoardScreen class
public class BoardScreen {
    public void open() { System.out.println("BoardScreen is opened."); }

    public void create() { System.out.println("BoardScreen is created."); }

    public void edit() { System.out.println("BoardScreen is edited."); }
}
```

(6)Menu.java

```java
public class Menu {
    private List<MenuItem> menuItems = new ArrayList<>();

    public void addMenuItem(MenuItem menuItem) { menuItems.add(menuItem); }

    public void clickMenuItem(int index) {
        if (index >= 0 && index < menuItems.size()) {
            menuItems.get(index).click();
        }
    }
}
```

(7)MenuItem.java

```java
// MenuItem class
public class MenuItem {
    private Command command;

    public MenuItem(Command command) { this.command = command; }

    public void click() { command.execute(); }
}
```

(8)Client.java

```java
// Client code
public class Client {
    public static void main(String[] args) {
        BoardScreen boardScreen = new BoardScreen();

        Command openCommand = new OpenCommand(boardScreen);
        Command createCommand = new CreateCommand(boardScreen);
        Command editCommand = new EditCommand(boardScreen);

        MenuItem openMenuItem = new MenuItem(openCommand);
        MenuItem createMenuItem = new MenuItem(createCommand);
        MenuItem editMenuItem = new MenuItem(editCommand);

        Menu menu = new Menu();
        menu.addMenuItem(openMenuItem);
        menu.addMenuItem(createMenuItem);
        menu.addMenuItem(editMenuItem);

        // Simulate clicking menu items
        menu.clickMenuItem( index: 0); // Expected Output: BoardScreen is opened.
        menu.clickMenuItem( index: 1); // Expected Output: BoardScreen is created.
        menu.clickMenuItem( index: 2); // Expected Output: BoardScreen is edited.
    }
}
```

3、上机测试程序，写出运行结果。

结果正常输出且符合预期。

```
BoardScreen is opened.
BoardScreen is created.
BoardScreen is edited.
```