用登录界面来演示MVC架构的逻辑

1. UI设计部分

代码框架：

```
package com.virtualbank.ui;

import com.virtualbank.controller.LoginController;
import com.virtualbank.service.LoginService;

import javax.swing.*;
import java.awt.*;

public class LoginUI extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;
    private JButton registerButton;
    private JLabel usernameLabel;
    private JLabel passwordLabel;
    private JLabel titleLabel;

    public LoginUI() { // 一些组件
        createUI();
        setTitle("Joy Bank");
        final int window_width = 1260;
        final int window_height = 841;
        setSize(window_width, window_height);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setResizable(false);
        setLayout(null);
        getContentPane().setBackground(new Color(255, 255, 230)); // Peach color
background
    }

    private void createUI() { // 构建整个静态界面的

    }


    // 以下四个方法就是UI中需要写的，返回可能有响应的组件的对象本身
    public JTextField getUsernameField() {
        return usernameField;
    }

    public JPasswordField getPasswordField() {
```

```
        return passwordField;
    }

    public JButton getLoginButton() {
        return loginButton;
    }

    public JButton getRegisterButton() {
        return registerButton;
    }

    public static void main(String[] args) { // 用于测试的
        SwingUtilities.invokeLater(() -> {
            LoginUI loginUI = new LoginUI(); // 创建 LoginUI 的实例
            LoginService loginService = new LoginService(); // 创建 LoginService 的实例
            loginUI.setVisible(true); // 显示 LoginUI 界面
            new LoginController(loginService, loginUI); // 创建 LoginController 的实例，并连
接 Service 和 UI
        });
    }


}
```

UI中不需要处理逻辑，而是把有可能发生事件的组件对象写一个返回函数，方便controller调用。（本例中，两个按钮，两个输入框）

2. Controller部分

代码框架：

```
package com.virtualbank.controller;

import com.virtualbank.service.LoginService;
import com.virtualbank.ui.LoginUI;

import javax.swing.*;

public class LoginController {
    private LoginService loginService; // 把Service当做一个属性
    private LoginUI loginUI; // 把UI当做一个属性

    public LoginController(LoginService loginService, LoginUI loginUI) {
        this.loginService = loginService;
        this.loginUI = loginUI;
        initController();
    }
```

```
    private void initController() { // 在这里添加按钮监听器
      // 点击按钮就执行下方的performLogin方法来处理登录事件
      loginUI.getLoginButton().addActionListener(e -> performLogin());
        loginUI.getRegisterButton().addActionListener(e -> navigateToRegister());
    }

    private void performLogin() { // 在这里接收文本框输入
        String username = loginUI.getUsernameField().getText();
        String password = new String(loginUI.getPasswordField().getPassword());

        if (loginService.authenticate(username, password)) { // 在这里调用service层的具体实现登录验证的方法
            JOptionPane.showMessageDialog(loginUI, "Login successful!", "Success", JOptionPane.INFORMATION_MESSAGE);
            // 这里可以跳转到应用程序的其他部分
        } else {
            JOptionPane.showMessageDialog(loginUI, "Invalid username or password", "Login Failed", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void navigateToRegister() {
        // Code to navigate to the Register UI
    }
}
```

在controller中，把LoginUI当做一个属性，在这里处理所有事件。调用UI部分的返回组件的方法，添加监听器或者接收文本。

这里还涉及了service层，下面会介绍。service层是具体实现业务逻辑的。

cotroller是一个中间处理的部分，它把LoginUI和LoginService都当做属性，然后获取UI的组件，调用Service的方法来实现整个响应的业务。

Service部分

代码框架

```
package com.virtualbank.service;

public class LoginService {

    // 定义默认管理员账户的凭据
    private final String adminUsername = "admin";
    private final String adminPassword = "123456";
```

```
    public boolean authenticate(String username, String password) {
        // 检查输入的用户名和密码是否与管理员的凭据匹配
        return adminUsername.equals(username) && adminPassword.equals(password);
    }
}
```

具体的登录验证方法在这里实现，这里会和repository层连接，来处理真正的数据（本例中使用了本地的账号信息来做演示）