****

**实验报告**

**实验三**

**课程名称： 计算机系统安全**

**实验名称： 完整性访问控制系统设计与实现**

**实验日期： 2018.12.27**

**班 级： 1603107**

**学 号： 1160300426**

**姓 名： 李国建**

**指导教师： 张玥**

1. 给出应用系统的安全策略文档。

用户拥有申请存取款功能和查询功能。用户在申请存取款后，会生成一个账单，提交到管理员处。待管理员同意后，才能执行操作。

在登录界面可以进行两种操作——登录和注册。

取款和存款需要首先输入金额，然后生成一个账单。管理员登录后可以看见用户申请的账单信息，里面有用户的用户名和更改金额。管理员认证通过后，该账单被删除，该操作执行成功。如果管理员没有同意，则直接取消账单。注意：取款金额不能超过当前拥有的金额。

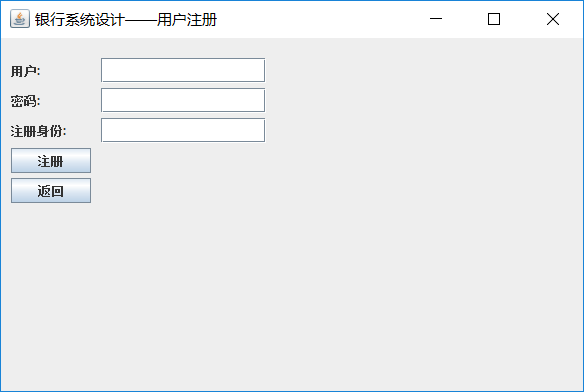
查询是直接调用当前数据库中的数据，没有更改数据库中的信息，不需要管理员进行同意。

1. 提供交互界面，能够完成录入、查询等功能。

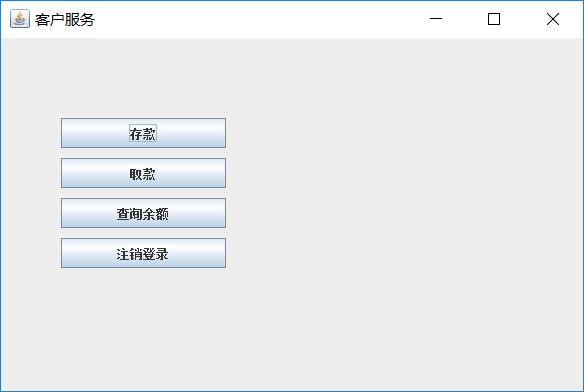
登陆界面：



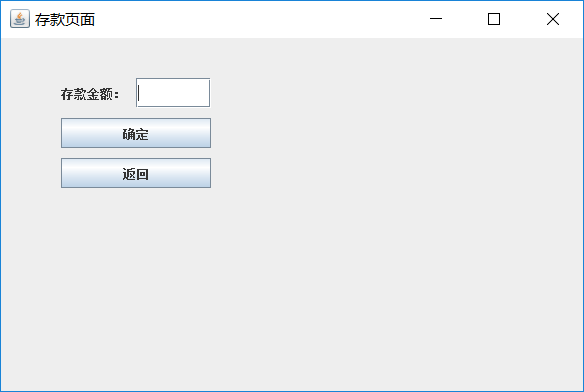
注册界面



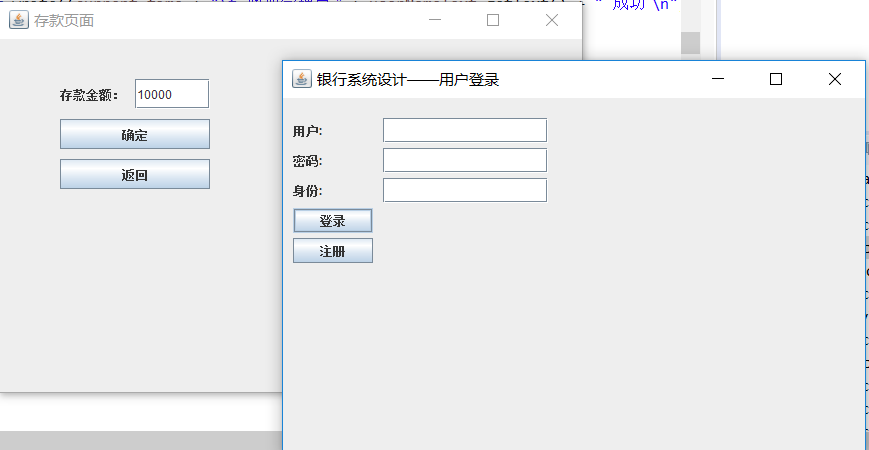
用户登录后的客户服务界面



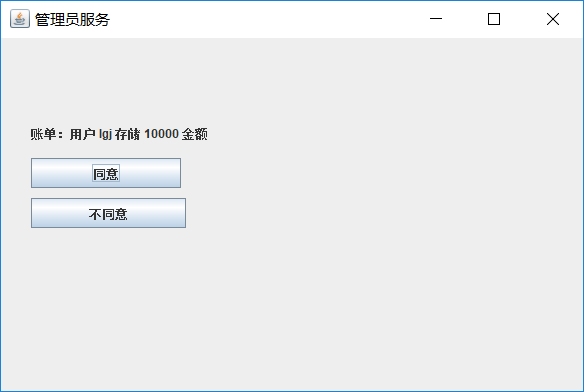
存款界面



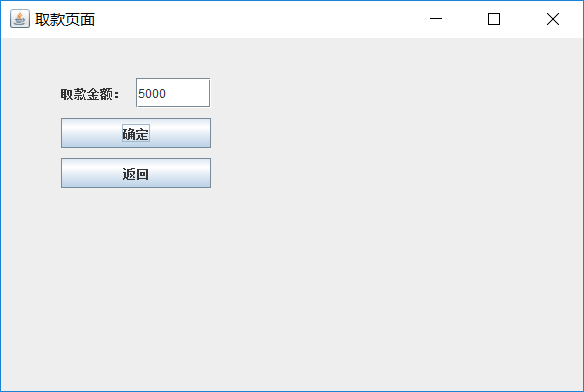
输入金额后弹出的登录界面



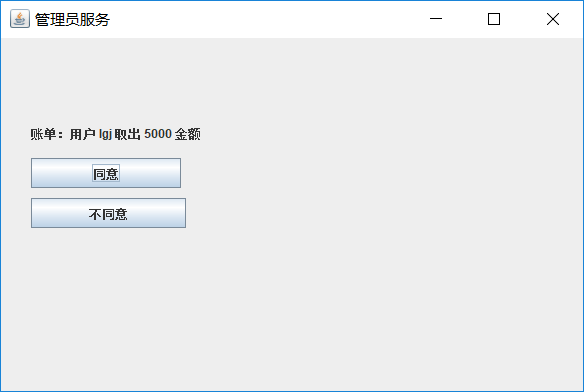
管理员进行确认或取消确认（同意或不同意）：



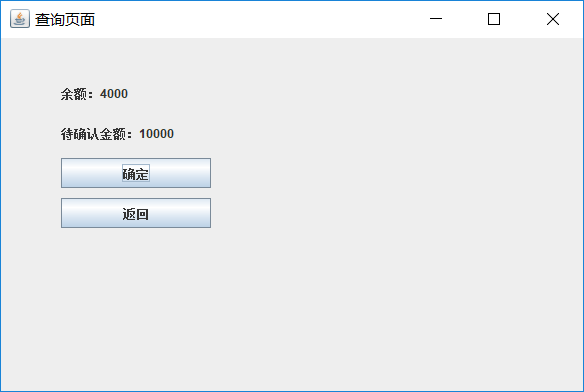
取款界面



取款后，登录管理员账户，进行确认或取消确认（同意或不同意）



查询界面



1. 满足责任分离原则。

在完整性策略定义的授权方式里面，责任分离原则禁止一个实体完成单独完成一个操作。我们的用户相当于一个实体，对于查询和退出登陆两种操作是用户自身的权力，而且不影响数据的完整性。

对于取款和存款两种操作，会对数据库中的内容进行修改，所以此时必须采用责任分离的原则。

为此，增加一个管理员身份的用户。单独有管理员时他不能做任何事情，单独有用户他只能查询余额，只有在用户申请这两种然后管理员同意时操作才能完成，符合责任分离的原则，同时也能保证数据的完整性。

1. 保存审计日志。

所有的日志，都保存在 log.txt 内，部分信息内容如下：



1. 遵循Clark-Wilson模型，定义应用系统的完整性限制条件。

Clark-Wilson模型考虑如下几点：

1）主体必须被识别和认证

2）客体只能通过规定的程序进行操作

3）主体只能执行规定的程序

4）必须维护正确的审计日志

5）系统必须被证明能够正确工作

如果把管理员看作主体，用户为客体，则管理员登陆界面即为认证过程，管理员的用户和密码只有自己才能知道的，满足1）。

作为客体的用户只能进行取款、存款、查询、返回四种操作，满足2）

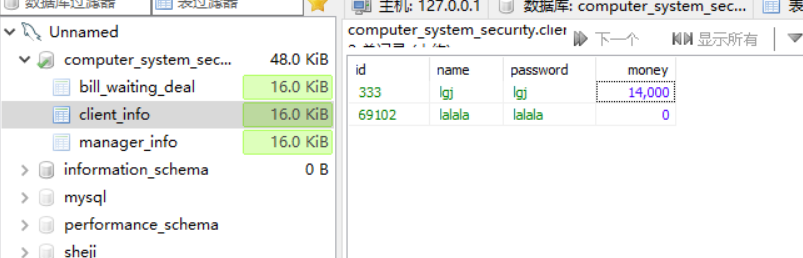
主体管理员只能进行认证，不能单独进行存款取款等操作，满足3）

证明4）、5）

下面以几个操作为例：1、登陆2、查询3、取款1000 4、存款10000 5、返回

首先登陆查询，可以发现与当前存款完全相同

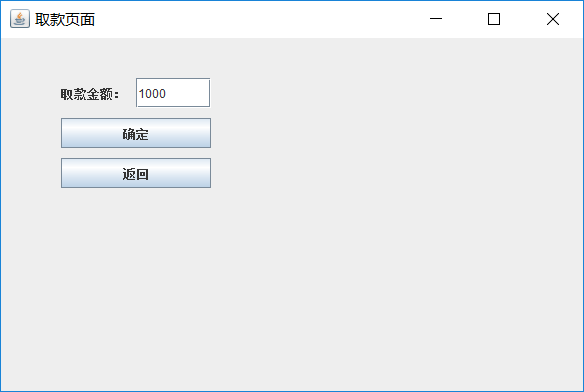
数据库内的信息：



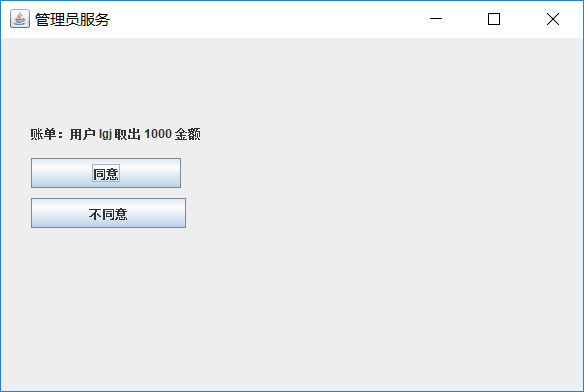
查询的信息：



然后取款1000

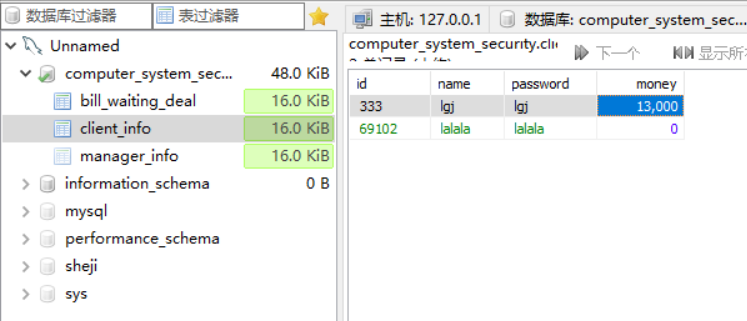


管理员同意：



取款结果：



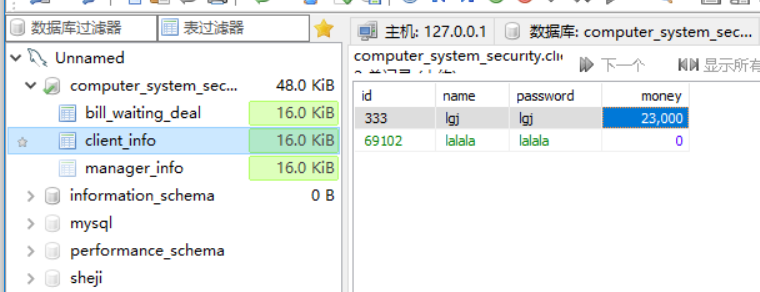


最后存款 10000



存款结果：





（6）遵循Clark-Wilson模型的证明规则和实施规则，并在设计报告中有所体现。

证明规则1：

当任意 IVP 运行时, 它必须保证所有的CDI处于有效状态

当用户登录时，在没有管理员同意的情况下，不能对数据库中的数据进行操作。只能提交账单申请，待管理员同意后，才能更改数据库。

证明规则2：

对相关联的CDI, 一个TP必须将这些CDI从一个有效状态转到另一个有效状态

在管理员同意后，账单就会被删除。这个交易过程，会将用户的申请状态，变为完成状态或错误状态（被管理员拒绝或读取/存入数据库失败）。

证明规则3：

系统执行操作时，符合责任分离原则。

模型需要保证用户身份和执行代码身份一致。所以需要验证身份。

这里设计的验证身份就是“登录”。

实施规则1：

系统要维护关联关系，保证经过验证的TP操作相应的CDI

在用户提出存取款申请后，管理员同意，就代表该账单已经被验证。被验证的这个账单可以对数据库中，相应的存款金额进行更改。

实施规则2：

TP操作CDI时，保证操作用户有权对相应CDI做操作，TP所代表的用户是CDI的真实用户

经过验证的账单，即管理员同意后，可以对数据库中的 CDI（即用户的存款金额）进行更改。

实施规则3：

系统执行操作时，符合责任分离原则

模型需要保证用户身份和执行代码身份一致

满足责任分离原则，用户和管理员都不能单独对存款金额进行更改，只有用户申请，管理员同意后，才能进行操作。

实施规则4：

只有可以授予TP访问规则的主体才能修改列表中相应的表项，授权主体不能执行TP操作

只有用户提出申请，才能进行 TP 操作。授权的管理员，没有执行 TP 操作的能力。

**主要代码**

代码入口：

**public** **static** **void** main(String[] args) {

// 查询数据库，存储相应的信息

MysqlConnect.*getClientInformation*();

MysqlConnect.*getManagerInformation*();

MysqlConnect.*getBillInformation*();

// 初始化 账单信息

**if** (Information.*bills* != **null**) {

**if** (Information.*bills*.size() == 0) {

*bill* = **new** Bill();

} **else** {

*bill* = Information.*bills*.get(0);

}

}

// 进入登录界面

*loginPage*();

**new** Log();

}

登录界面：

/\*\*

\* 用户登录

\* \*/

**private** **static** **void** loginPage() {

JFrame frame = **new** JFrame("银行系统设计——用户登录");

frame.setSize(600, 400);

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

// 创建面板

JPanel panel = **new** JPanel();

panel.setLayout(**null**); // 面板布局

//创建 标签 & 输入框 & 按钮

JLabel userLabel = **new** JLabel("用户:");

JLabel passwordLabel = **new** JLabel("密码:");

JLabel identityLabel = **new** JLabel("身份:");

JTextField userNameText = **new** JTextField(20);

JTextField userPasswordText = **new** JTextField(20);

JTextField identityText = **new** JTextField(20);

JButton loginButton = **new** JButton("登录");

JButton registerButton = **new** JButton("注册");

// 设置标签的大小和位置

userLabel.setBounds(10, 20, 80, 25);

userNameText.setBounds(100, 20, 165, 25);

passwordLabel.setBounds(10, 50, 80, 25);

userPasswordText.setBounds(100, 50, 165, 25);

identityLabel.setBounds(10, 80, 80, 25);

identityText.setBounds(100, 80, 165, 25);

loginButton.setBounds(10, 110, 80, 25);

registerButton.setBounds(10, 140, 80, 25);

// 设置面板内容

panel.add(userLabel);

panel.add(userNameText);

panel.add(passwordLabel);

panel.add(userPasswordText);

panel.add(identityLabel);

panel.add(identityText);

panel.add(loginButton);

panel.add(registerButton);

// 将面板加入到窗口中

frame.add(panel);

// 按钮的监听事件

loginButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

// 检测身份

**int** flag = 0;

**if** (identityText.getText().equals("用户")) {

// 查找该用户

**for** (Client client : Information.*clients*) {

**if** (client.getUserName().equals(userNameText.getText()) &&

client.getUserPassword().equals(userPasswordText.getText())) {

enter.*deposit* = client.getDeposit();

enter.*current\_client\_id* = client.getId();

enter.*current\_user\_name* = client.getUserName();

**try** {

Date now = **new** Date();

SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

String current\_time = dateFormat.format(now);

Log.*output*.write((current\_time + "\t 用户 " + *current\_user\_name* + " 登录成功\n").getBytes());

} **catch** (IOException e2) {

System.***out***.println("写入日志失败!!!");

}

*clientService*();

frame.dispose();

flag = 1;

}

}

} **else** **if** (identityText.getText().equals("管理员")) {

// 查找该用户

**for** (Administrator administrator : Information.*administrators*) {

**if** (administrator.getUserName().equals(userNameText.getText()) &&

administrator.getUserPassword().equals(userPasswordText.getText())) {

enter.*current\_manager\_id* = administrator.getId();

enter.*current\_manager\_name* = administrator.getUserName();

*managerService*();

frame.dispose();

flag = 1;

// 写日志

**try** {

Date now = **new** Date();

SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

String current\_time = dateFormat.format( now );

Log.*output*.write((current\_time + "\t 管理员 " + *current\_manager\_name* + " 登录成功\n").getBytes());

} **catch** (IOException e2) {

System.***out***.println("写入日志失败!!!");

}

}

}

}

// 没有查找到用户信息

**if** (flag == 0) {

frame.dispose();

*loginFailedPage*();

}

}

});

// 注册按钮的监听事件

registerButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

frame.dispose();

*registerPage*();

}

});

// 设置窗口可见

frame.setVisible(**true**);

}

数据创建代码：

create table client\_info (

id VARCHAR(32) NOT NULL comment '用户id',

name VARCHAR(32) NOT NULL comment '用户名',

`password` VARCHAR(32) comment '用户密码',

money NUMERIC(16) comment '账户余额'

);

create table manager\_info (

id VARCHAR(32) NOT NULL comment '用户id',

name VARCHAR(32) NOT NULL comment '用户名',

`password` VARCHAR(32) comment '用户密码'

);

create table bill\_waiting\_deal (

id VARCHAR(32)bill\_waiting\_deal NOT NULL comment '待处理账单id',

client\_id VARCHAR(32) NOT NULL comment '用户id',

name VARCHAR(32) NOT NULL comment '用户名',

money NUMERIC(16) comment '处理的金额'

);

数据库连接代码：

// 查询用户名对应的密码

**public** **static** **void** getClientInformation() {

Information.*clients* = **new** ArrayList<>();

Connection conn = **null**;

Statement stmt = **null**;

**try**{

// 注册 JDBC 驱动

Class.*forName*("com.mysql.jdbc.Driver");

// 打开链接

System.***out***.println("连接数据库...");

conn = DriverManager.*getConnection*(***DB\_URL***,***USER***,***PASS***);

// 执行查询

stmt = conn.createStatement();

String sql;

sql = "SELECT id, name, password, money FROM client\_info";

ResultSet rs = stmt.executeQuery(sql);

// 展开结果集数据库

**while**(rs.next()){

// 通过字段检索

**long** id = rs.getLong("id");

String name = rs.getString("name");

String password = rs.getString("password");

**int** money = rs.getInt("money");

Client client = **new** Client();

client.setDeposit(money);

client.setId(id);

client.setUserName(name);

client.setUserPassword(password);

// 存储数据

Information.*clients*.add(client);

}

// 完成后关闭

rs.close();

stmt.close();

conn.close();

}**catch**(SQLException se){

// 处理 JDBC 错误

se.printStackTrace();

}**catch**(Exception e){

// 处理 Class.forName 错误

e.printStackTrace();

}**finally**{

// 关闭资源

**try**{

**if**(stmt!=**null**) stmt.close();

}**catch**(SQLException se2){

}// 什么都不做

**try**{

**if**(conn!=**null**)

conn.close();

}**catch**(SQLException se){

se.printStackTrace();

}

}

}

介于代码量较大，这里就不全部粘贴了。代码工程见github地址：

<https://github.com/rocketeerli/Computer-System-Security-Lab/tree/master/lab3>