BANKING SYSTEM 具体实现

孙硕 82350468

1. 数据库

采用 csv格式存储数据。AccountData.csv

数据保存方式为 `[ string Name, string Account, string Password, float Balance]`

分别对应 姓名、账户号码、账户密码、账户余额，

储存在一行四列中，以逗号隔开，换行符结束，形成 csv格式。

读取通过 MATLAB 的 `import data` 功能生成的函数，读取后自动生成包含不同类型数据的矩阵。

（采用 csv实属无奈之举，由于 mac 上的 office 无法实现 MATLAB 对 .xls文件的写入，要不然会方便很多，但是 csv的读取速度比 xls快很多，也算是各有所长。）

2. 脚本文件

包括 main.m、database 类(BankDB.m)和 ticket 类(Ticket.m)，具体实现过程中还添加一个单独的用于读取数据库数据的函数文件 importfile.m。

A. Main.m
   i. 建立 database 和 ticket 类。
   ii. 为 database 与 ticket 的属性初始化，下文提及。
   iii. 为三个 UI 的属性初始化，下文提及。

B. BankDB.m
   i. 包含属性：`int queue[] , accountdata, addr`
   ii. queue 初始化为空向量， `addr = 数据库文件， accountdata = fopen(addr,"a+")`。以"a+"读取，使文件指针落在结尾处。方便写入数据。

iii. 包含函数：`inqueue, outqueue, generate_account, write_end, rewrite`

1. `inqueue(ticketNO)` 将号码放入队尾

```
DB.queue = [DB.queue,ticketNO]
```

2. `outqueue()` 将队首取出

```
DB.queue(1) = []
```

3. `generate_account()` 生成 6 位随机数，用于开户操作

```
round(899999*rand)+100000;
```

4. `write_end(data)` 将 data 写入数据库新的一行

```
fprintf(.accountdata,'%s,',data{1});
fprintf(.accountdata,'%s,',data{2});
fprintf(.accountdata,'%s,',data{3});
fprintf(.accountdata,'%f\n',data{4});
.accountdata = fopen('AccountData.csv','a+');
```

（不同数据类型使用不同写入方式）

5. `rewrite(data)` 将数据库从头更新为 data（受迫于 csv 无法写入特定位置，将来可能会采用别的方式）

```
[r,~] = size(data);
.accountdata = fopen('AccountData.csv','w+');
for i = 1:r
    fprintf(.accountdata,'%s,',data.Name(i));
    fprintf(.accountdata,'%s,',data.Account(i));
    fprintf(.accountdata,'%s,',data.Password(i));
    fprintf(.accountdata,'%f\n',data.Balance(i));
end
```

C. Ticket.m

仅包含属性：`int order` 保存当前号码，用于与客户 UI 交换信息。

3. UI 文件

采用了较笨的界面切换方法——使某些元素消失（`Visible`设为 0，更改 `Text` 或 `Value`值），某些出现（`Visible`设为 1），这样虽实现起来麻烦，但是相较于新开启一个 UI 速度快很多，而且用户使用起来比较友好。

判断在哪个界面，可以用当前界面上可见的元素判断，比如 `TextAera`上的文字，某些按钮的 `Visible`与否。

某些功能，如检查账号密码匹配，转账等，由于操作较简单，代码量很少，使用次数也不多，没有在 Database 中实现，直接在 UI 代码中实现。

A. Customer UI (Ticket)

i. 包含属性：Ticket 用于记录票号, Database用于与 queue的交互, CashierUI用于 CashierUI上的票号显示。

ii. 包含文本框*1 和按钮*1

iii. 文本框显示当前票号，按钮为取票

iv. 取票的反馈为文本框显示票号 Ticket.order，票号进入 queue，票号++，为方便，将 CashierUI上的票号置为 queue的首个元素

B. Cashier UI

i. 包含属性：Database用于函数的使用等等，Openfirst用于开户时确认密码时储存第一次输入，Cur_ind记录当前账户在数据库中的行数，To_ind转账时用于记录转入账户在数据库中的行数。

ii. 每个界面的最上方都有公用的 TextArea，每个状态显示不同的文字来引导操作，编程时，可以用文字确定界面。除主界面外，每个界面都有返回上一层的按钮，就是通过判断界面元素来确定返回哪一层，其他按钮同理。下文如何切换界面就不多赘述。

iii. 主界面

包括插卡，开户和完成三个按钮。

a. 插卡，进入输入账号密码界面，和我们平时看到的无差。错误处理：若不输入就点击登陆则在 TextArea报错。登陆按钮的反馈为在数据库中通过 importfile()取出 Account的那一列，通过 find()寻找账号是否存在，若存在则匹配密码，若匹配则登陆成功，进入功能界面，否则都不进入。将 find()的返回值存入属性中的 cur_ind，以便后续使用。同时将左上方的 NameLabel设为相应的 Name。

b. 开户，进入另一个输入账号密码和姓名的界面，使用与之前不同的 EditField，进入界面时，将账号的数值用 generate_account()生成，并设为不可编辑。确认按钮的错误处理: 若账户密码不为 6 位数字则报错，通过两个嵌套的 if和函数 length()、isstrprop(,'digit')实现。若无错误，确认按钮的反馈为，使用 write_end()在数据库末尾写入姓名账号密码和余额 **0**。

c. 完成，若 queue为空，退出程序，否则修改左上角的号码为 queue(1)。

iv. 功能界面

包括查询余额、转账、存款、取款、销户（和未完成的交易记录）

a. 这几个功能都大同小异，以最复杂的转账为例。

b. 点击转账进入转账界面，显示相应的转入账号编辑框、转出账号编辑框、转账金额编辑框和确认按钮（每个确认按钮都是不同的），隐藏相应元素。转出账号编辑框显示出当前账户账号，将其值设置为通过 `importfile()`取出的 `matrix.Account(cur_ind)`,并设置为不可编辑。确认的错误处理：若转入账号不存在或转账金额大于 `matrix.Balance(cur_ind)`则报错。确认的反馈为：若不报错，则查找转入账号，并存在 To_ind 中，

```
matrix.Balance(cur_ind) -= 转账金额；
matrix.Balance(To_ind) += 转账金额；
rewrite(matrix);
```

完成后将转入账户和金额置为 0。

c. 剩余几个功能同理，在此不多赘述。销户功能在完成销户后直接退出程序。

C. ATM UI

1. 最简单的实现方式为将 CashierUI 复制过来，然后删掉开户，销户相关回调、按钮和出现的代码。
2. 将左上方的票号文本框删除。

4. 迭代计划

交易记录功能已经完成了 UI，由于找不到合适的储存方式，代码部分尚未实现，会在近期实现。