0.1 the Design of Login and Signup

0.1.1 Idea and Method

Note: we only consider the situations of login/signup failure mentioned in the requirement document, and other operation failures such as database/network errors are not included in the error handling.

A. Login

As the document says, login can be divided into two situations: login successfully (both the name and password are matched) and login failed (throw the runtime exception).

Therefore, in the whole login method, we use "loginStatus", the private variable to record the status, and apply the if-else branches to these two situations: if the return value of the "getUser" method in "UserRepository" class is not null and the return value of "getPassword" method and the parameter user's password are the same, it goes to login successfully, and the logger record information.

Otherwise, it goes to login failed. After logger, it will directly throw a runtime exception.

B. Signup

Also, the signup part can be divided into two cases: signup successfully (the name can be used) and signup failed (the name already exists in the file).

Therefore, in the whole signup method: if the name doesn't exist in the file, it means the name is not repetitive. So it goes to the catch branch so as to create the new user, and write the logger.

Otherwise, it throws the runtime exception and write the logger.

C.Cost

The price of a cup of coffee consists of two parts: coffee cost and cup-type cost. Therefore, the *cost* method in class *Coffee* only needs to return the sum of the two costs. In order to get the cost of different cup types, I design a method called *size2price*. And it works out the price with switch-case statement. In terms of the *cost* method in class *PriceService*, I only need to calculate the sum of all orders and output each order's information to the console.

What's more, I create a class called *Size*–which includes only three meaningful constant values–to express the cup types clearly. I think it maybe a good idea and it is very useful when to read codes.

0.1.2 Encoding specification

Detailed Descriptions of Thrown Exceptions

```
throw new RuntimeException(InfoConstant.
    USERNAME_OR_PASS_ERROR);
throw new RuntimeException(userAlreadyExistStr);
```

Catch the Particular Exceptions

```
{
...
} catch (RuntimeException e){
        .....
}
```

Readability of Variables

e.g.userSignupOkStr/userLoginStr

Proper comments and consistent comment style

```
//user exist and the password correct
```

0.2 Problems and Methods

0.2.1 the error caused by nextLine( )

solution: add another line of "in.nextLine( )" to absorb the redundant line feed.

0.2.2 Code Checking: Password or Certificate

solution: rename the password-related constants (since this lab does not relate to encryption)

0.3 testing and optimizing

0.3.1 testing

result:find two problems:

1. the PriceService.cost module output:

" name: null, size: 1, number: 2, price:4$
20$ "

solution:use the new CappuccinoRepositoryImpl() and new EspressoRepositoryImpl()

```
Coffee coffee = coffeeType == 1 ? new
    CappuccinoRepositoryImpl()
                .getCappuccino("cappuccino") : new
                    EspressoRepositoryImpl()
                .getEspresso("espresso");
```

2. the logic of login after signing up

our program let the user to login automatically after signing up before. When the TA said that user should login in by themself after signing up, I changed the logic of Lab2Application.main function.

0.3.2 optimizing

the name and password's validity verification:

at first we write two while loop for the null value and mismatching value condition, we merged them to one while loop.

```
while ((nameStr.equals("")) || (!nameStr.matches(
    InfoConstant.USER_REGEX))) {
        ...
    }
```