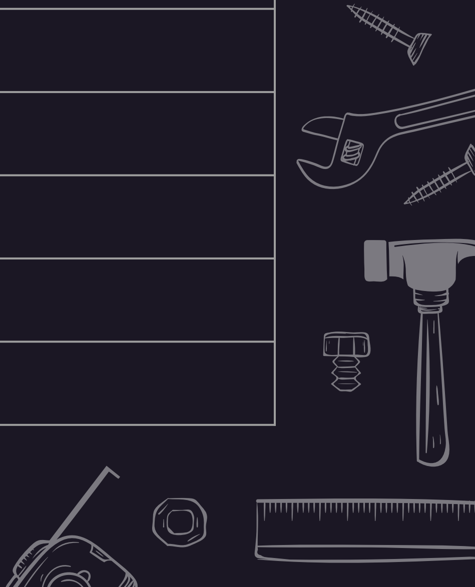
CSC1109 Object-Oriented Programming Team 14

Alex | Kai Yang | Timothy | Zaw

Project Contributions

Features	**Contributor**
Basic Features	Alex, Timothy
SQL Database	Alex, Timothy
Custom Generated Data [100k Transaction]	Alex, Timothy
Command Line Interface	Alex, Timothy
GUI [Thymeleaf, HTML, CSS, Javascript]	Kai Yang
Spring Boot, Unit Testing [JUnit, Mockito]	Zaw Wana



Features

ATM Bank Features

01

Basic Features

Fundamental ATM Features

02

SQL Database

Store all data required for the ATM (e.g. transactions)

03

GUI

Fully functional CLI and HTML to provide users a choice to use our ATM the way they prefer

04

Unit Testing

Improve code quality and maintainability

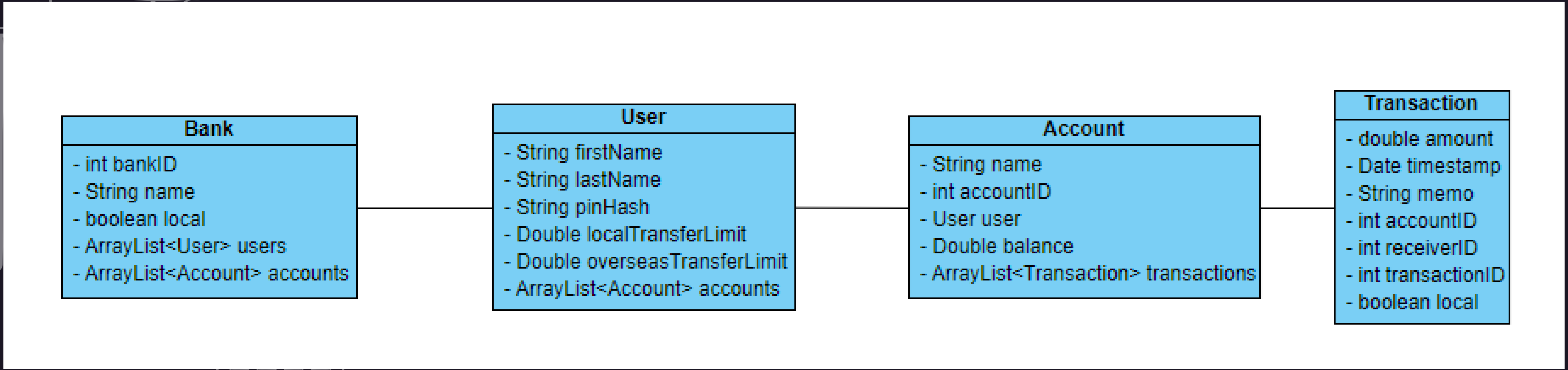
01

Basic Features

Basic Features

1. Account Information (Account Numbers)
2. Balance Check (Available Balance)
3. Authentication (Password Check / Reset)
4. Money Transfer (Inter-Account Transfer / Third Party Transfer)
5. Settings (change password, change account name, etc)

Basic UML Diagram



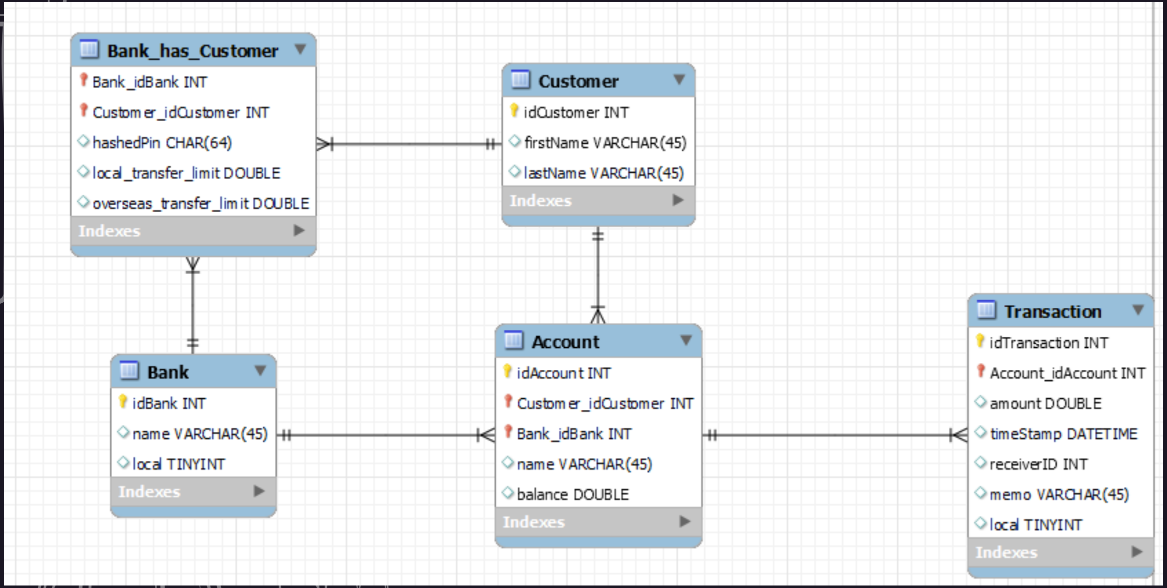
02

SQL Database

MySQL Database

- Store all user account data
- Storing the 100k sample transactions
- Allow us to have centralised place to store and load data from.
- We cannot load 100k transactions on our local memory
- We only retrieve and save objects in Java locally when needed to save memory

Database Schema



- Simple schema
- Schema similar to our classes in Java
- Stores only hashed pins for security
- Enables all user actions in ATM to be stored and updated in real time

03

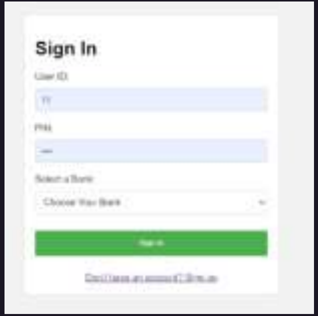
GUI

GUI

Thymeleaf Engine Template

Thymeleaf is a modern server-side Java template engine for both web and standalone environments.

Purpose: To allow cleaner design for users to experience our ATM system.

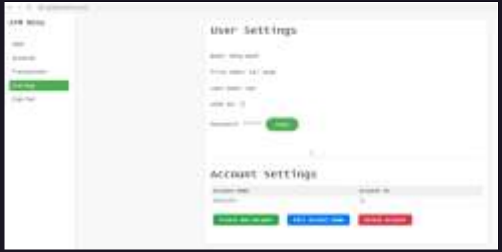

Sign In

User ID

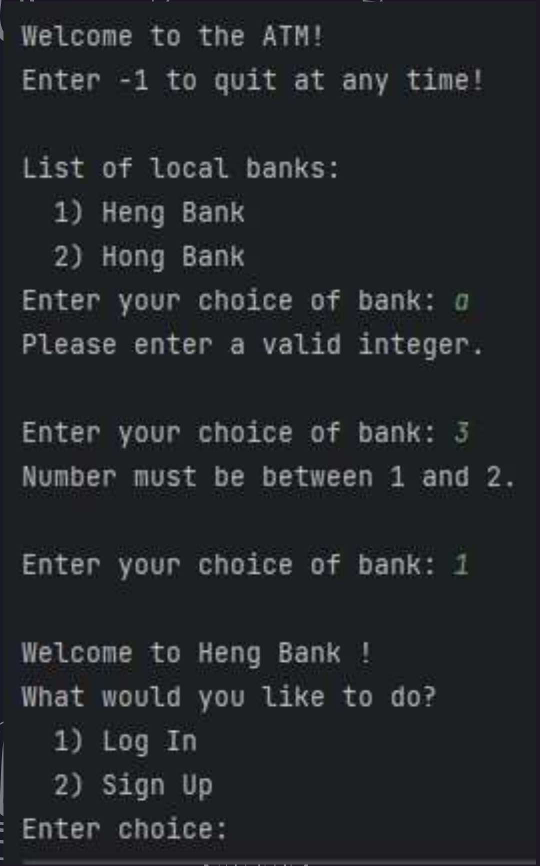
PIN

Select a Bank

Choose Your Bank


User Settings

Account Settings

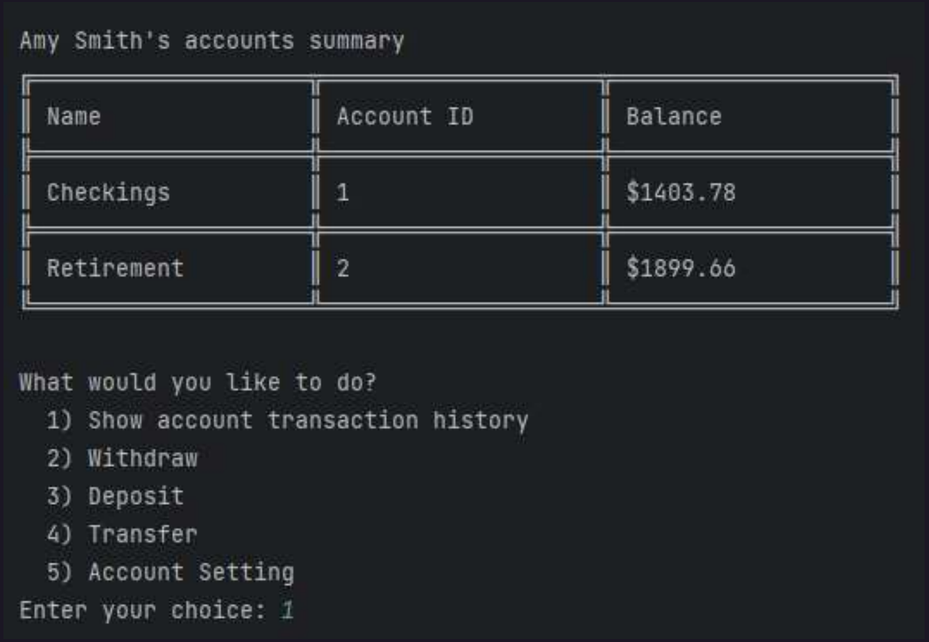


```
Welcome to the ATM!
Enter -1 to quit at any time!

List of local banks:
  1) Heng Bank
  2) Hong Bank
Enter your choice of bank: a
Please enter a valid integer.

Enter your choice of bank: 3
Number must be between 1 and 2.

Enter your choice of bank: 1

Welcome to Heng Bank !
What would you like to do?
  1) Log In
  2) Sign Up
Enter choice:
```

CLI

1. CLI for users who prefer a simpler way to use our ATM
2. Fully functional alternative to GUI
3. All user input is validated so no exceptions will stop our program

CLI



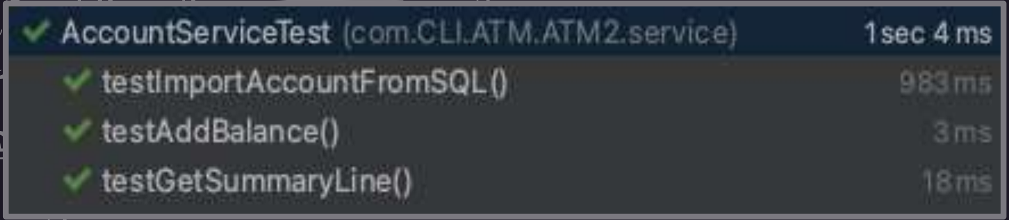
```
Amy Smith's accounts summary
╔══════════════╦══════════════╦══════════════╗
║ Name         ║ Account ID   ║ Balance      ║
╠══════════════╬══════════════╬══════════════╣
║ Checkings    ║ 1            ║ $1403.78     ║
╠══════════════╬══════════════╬══════════════╣
║ Retirement   ║ 2            ║ $1899.66     ║
╚══════════════╩══════════════╩══════════════╝

What would you like to do?
  1) Show account transaction history
  2) Withdraw
  3) Deposit
  4) Transfer
  5) Account Setting
Enter your choice: 1
```

04

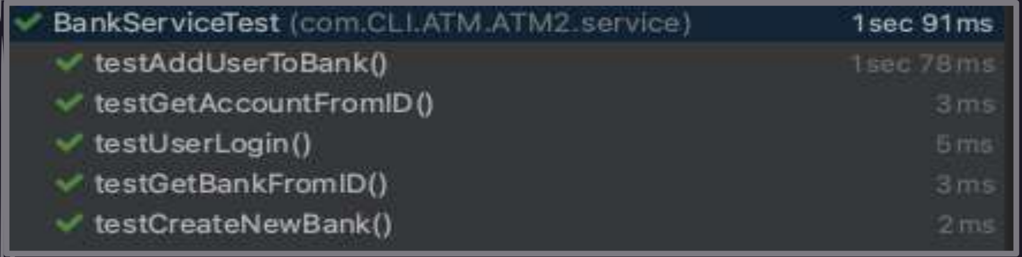
Unit Testing

JUnit Testing

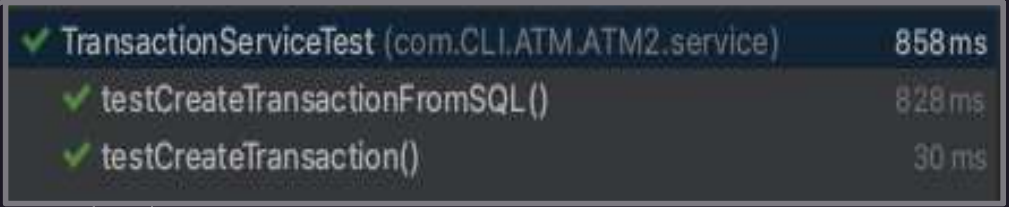
1. Created Unit Tests using Mockito that initializes the required objects and dependencies before each test case is executed.
2. Created Unit Tests to test all basic fundamental features (e.g validating pin, login, creation of new user
3. Created Unit Tests to test the updating and retrieval from SQL Database



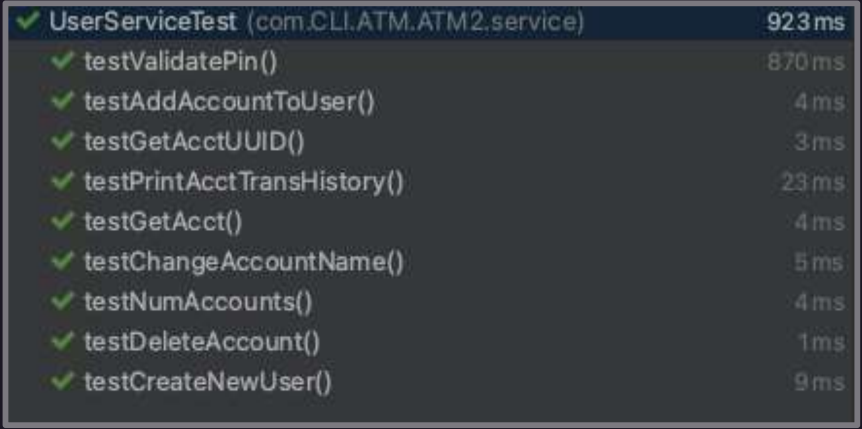
```
✓ AccountServiceTest (com.CLI.ATM.ATM2.service)   1sec 4 ms
    ✓ testImportAccountFromSQL()                      983 ms
    ✓ testAddBalance()                                  3 ms
    ✓ testGetSummaryLine()                             18 ms
```



```
✓ BankServiceTest (com.CLI.ATM.ATM2.service)      1sec 91 ms
    ✓ testAddUserToBank()                         1sec 78 ms
    ✓ testGetAccountFromID()                            3 ms
    ✓ testUserLogin()                                   5 ms
    ✓ testGetBankFromID()                               3 ms
    ✓ testCreateNewBank()                               2 ms
```



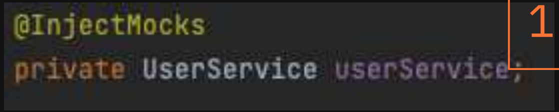
```
✓ TransactionServiceTest (com.CLI.ATM.ATM2.service)   858 ms
    ✓ testCreateTransactionFromSQL()                  828 ms
    ✓ testCreateTransaction()                          30 ms
```



```
✓ UserServiceTest (com.CLI.ATM.ATM2.service)          923 ms
    ✓ testValidatePin()                               870 ms
    ✓ testAddAccountToUser()                            4 ms
    ✓ testGetAcctUUID()                                 3 ms
    ✓ testPrintAcctTransHistory()                      23 ms
    ✓ testGetAcct()                                     4 ms
    ✓ testChangeAccountName()                           5 ms
    ✓ testNumAccounts()                                 4 ms
    ✓ testDeleteAccount()                               1 ms
    ✓ testCreateNewUser()                               9 ms
```

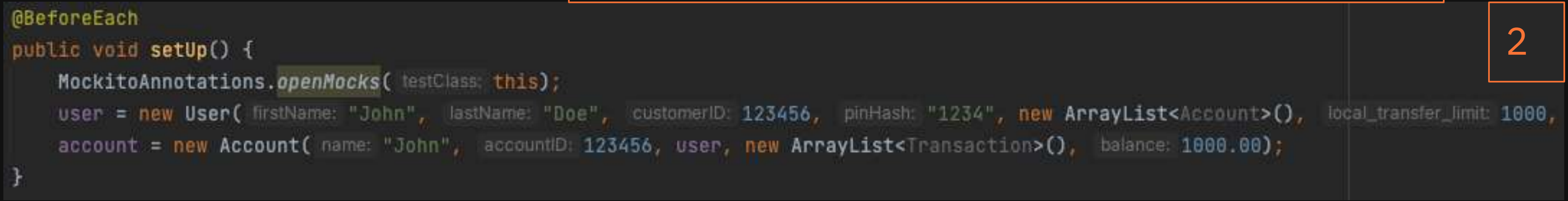
1. Mock userService
2. Create user and account for testing beforeEach test
3. Test each unit

1



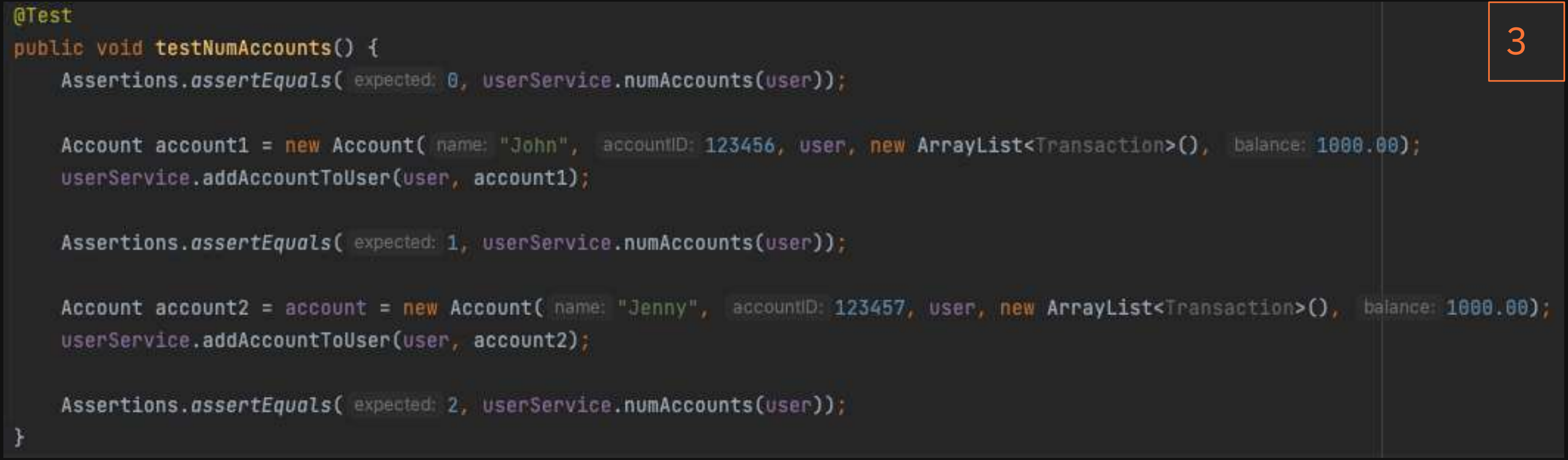
```java
@InjectMocks
private UserService userService;
```

2



```java
@BeforeEach
public void setUp() {
    MockitoAnnotations.openMocks( testClass: this);
    user = new User( firstName: "John",  lastName: "Doe",  customerID: 123456,  pinHash: "1234", new ArrayList<Account>(),  local_transfer_limit: 1000,
    account = new Account( name: "John",  accountID: 123456, user, new ArrayList<Transaction>(),  balance: 1000.00);
}
```

3



```java
@Test
public void testNumAccounts() {
    Assertions.assertEquals( expected: 0, userService.numAccounts(user));

    Account account1 = new Account( name: "John",  accountID: 123456, user, new ArrayList<Transaction>(),  balance: 1000.00);
    userService.addAccountToUser(user, account1);

    Assertions.assertEquals( expected: 1, userService.numAccounts(user));

    Account account2 = account = new Account( name: "Jenny",  accountID: 123457, user, new ArrayList<Transaction>(),  balance: 1000.00);
    userService.addAccountToUser(user, account2);

    Assertions.assertEquals( expected: 2, userService.numAccounts(user));
}
```

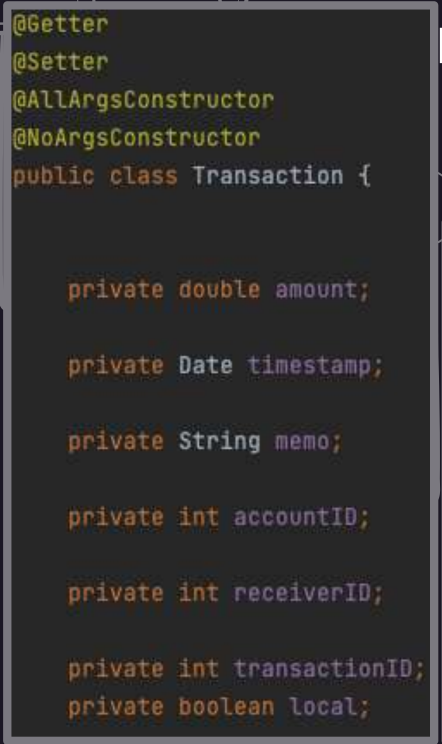
05

Additional Features

Framework

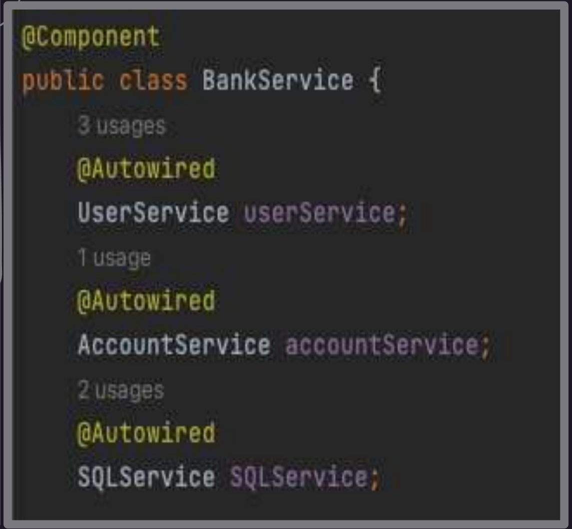
1. Spring Boot
2. Libraries - Lombok, Mockito

Entity



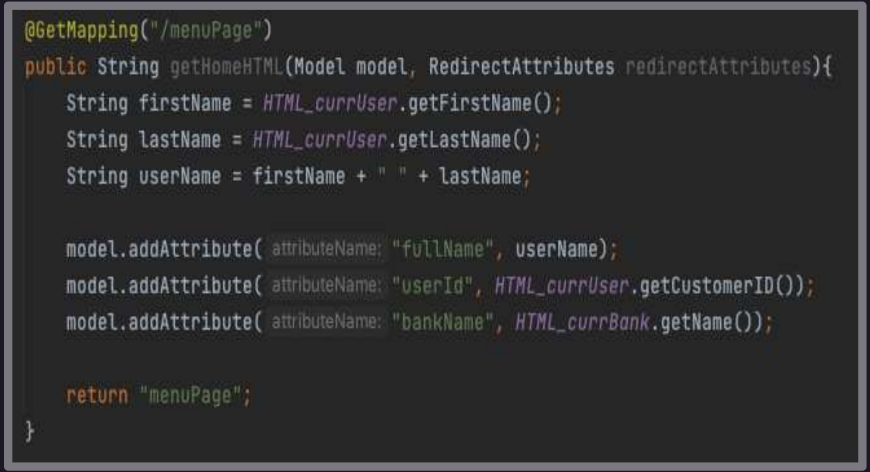
```java
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Transaction {


    private double amount;

    private Date timestamp;

    private String memo;

    private int accountID;

    private int receiverID;

    private int transactionID;
    private boolean local;
```

Loose Coupling via Spring Boot Container



```java
@Component
public class BankService {
    3 usages
    @Autowired
    UserService userService;
    1 usage
    @Autowired
    AccountService accountService;
    2 usages
    @Autowired
    SQLService SQLService;
```

HTTP REQUETS



```java
@GetMapping("/menuPage")
public String getHomeHTML(Model model, RedirectAttributes redirectAttributes){
    String firstName = HTML_currUser.getFirstName();
    String lastName = HTML_currUser.getLastName();
    String userName = firstName + " " + lastName;

    model.addAttribute( attributeName: "fullName", userName);
    model.addAttribute( attributeName: "userId", HTML_currUser.getCustomerID());
    model.addAttribute( attributeName: "bankName", HTML_currBank.getName());

    return "menuPage";
}
```

Thank You!