Discovery Project

N van der Linde

Project submitted for the degree *Baccalaureus Scientiae* in Information Technology at the North-West University

Supervisor: Dr. JT Janse van Rensburg
Co-supervisor: Mr. Z. Boonzaaier

Student number: 25909932

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION AND OVERVIEW

1.1 Introduction

This project is about designing an application for Discovery Vitality. The application will be the Account system that manages the Active Rewards currency, Discovery Miles.

Discovery rewards its members for making healthy choices and living a healthy lifestyle through Vitality. Vitality's Active Rewards programme plays a key part in this. Active Rewards looks at members' Health and Fitness, Driving and Spending behaviour to track towards each members' weekly goals. Members who complete their weekly goals are awarded plays on the weekly gameboard.

Every week there is a new gameboard and members use their plays earned in the previous week to play. The gameboard is filled with hidden tiles which contain Discovery Miles. The member uses the plays to pick tiles and earn the miles on the tiles. Once all the plays are used up, the entire game board is revealed.

Members accumulate Miles earned on the gameboard and can exchange their Miles for a reward voucher. Active Rewards has many rewards partners and offer many different reward categories. The categories can be anything, for example, shopping, healthcare, or entertainment just to name a few. Each category then lists many rewards that the member can choose from when they have enough Miles, for example from Spotify, DStv, LEGO, Exclusive Books or even Cape Union Mart.

1.2 Highlights

The highlights for this project was the ERD diagram as well as the Use Case diagram. Drawing these diagrams was valuable. It helps to stay updated on what Use Case as well as ERD diagrams need to look like or contain.

The weekly Discovery Live Sessions was another highlight. To be able to interact with someone working for Discovery was beneficial. It was a very good learning opportunity. She provided a lot of solutions and help with this project.

1.3 Challenges

The challenges for this project was the different technology stacks that was expected to be used to build this project. It was a bit difficult to implement all the different technologies in the project.

A recommendation would be to expose the different technologies in the first semester. Give a description or tutorial of each stack every week. Then when the second semester arrives, the building of the application begins. This way, the students will have a much better idea of what is expected of them and how to use the different technologies.

Another recommendation or challenge was to start the project. The motivation to start this project was lacking. Maybe if once a week, there was something that was needed to be submitted then it would be better. This is also an advantage because then all students will be at the same place at the same time. So, none of them would fall behind or begin to stress when the submission date neared.

1.4 System Overview

There are three user story requirements, namely:

- Adding Miles: The Discovery member must add Miles to their Miles account. They can earn and accumulate Miles when they reveal tiles on the gameboard.
- View Miles: The Discovery member must view Miles in their Miles account. This is for the member to know whether they have accumulated enough Miles for the reward they want.
- Subtract Miles: The Discovery member must be able to subtract their Miles. This is necessary for the member to exchange their Miles for a reward voucher.

There are different technology stacks that need to be implemented for this project to run successfully. The following need to be used:

- GIT: this is used for the source control. GitHub was used for this project's source control.
- Java: version 1.7 was used.
- IDE: IntelliJ IDEA was used to build the project.
- Build tool: Maven or Gradle can be used. For this project, Maven was applied.
- Spring framework: Spring is a widely used framework that does a lot of boiler plate code in the background. It provides a comprehensive programming and configuration model for modern Java-based enterprise applications. It is not necessary for Spring to be installed as it will be pulled in as a dependency via the build tool.
- Swagger: this is a quick way to generate web service documentation and provide an easy-to-use interface to call the services.
- Docker: this is optional. The application will need to run in some java container. Docker provides an environment to run the container in.

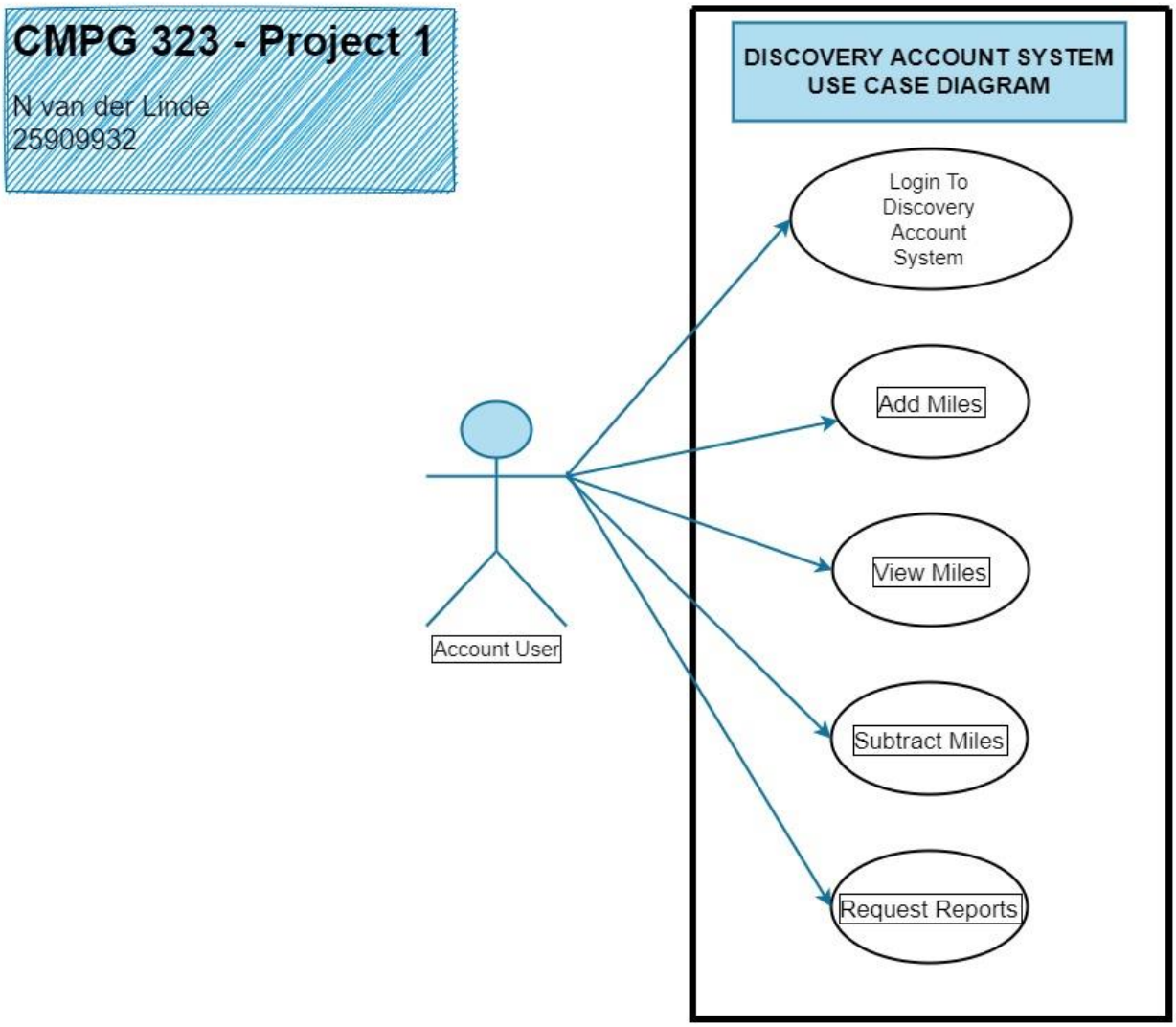
- Diagrams: it is required to draw diagrams (ERD, Flow and Use Case diagrams) for this project. Any tool of could have been used. For this project, draw.io was used to design the diagrams.
- Logging: the application will need logging. A logging framework was implemented.
- Code coverage: this is a type of tool, library, plugin, or framework that needs to be used to check the project's code coverage. For this project, JaCoCo was used.

CHAPTER 2 USE CASE DIAGRAM

2.1 The Use of UML Diagrams

The Use Case Diagram is a diagram that depicts the interactions between the system and external systems and users. It graphically describes who will use the system and in what ways the user expects to interact with the system. It is also a set of modelling conventions that is used to specify or describe a software system in terms of objects. The use-case narrative is used in addition to textually describe the sequence of steps of each interaction.

Figure 2-1: Use Case Diagram



From the abovementioned diagram for the Discovery Rewards System, the following can be derived:

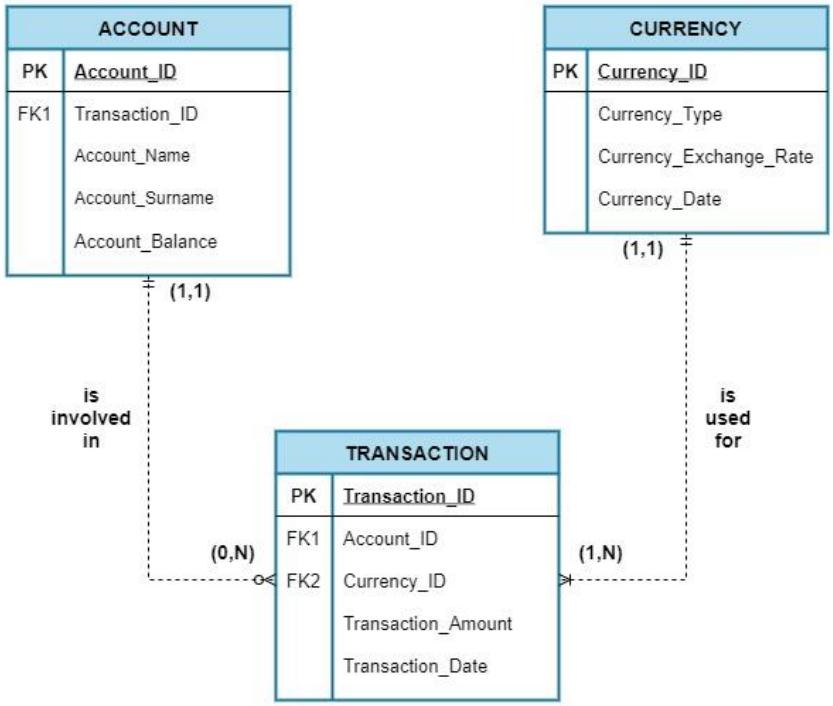
- Login to Discovery Account: the account user will Login to his or her account, using their unique username and password.
- Add Miles: the user can earn and accumulate Miles when they reveal tiles on the gameboard.
- View Miles: the user will know whether they have accumulated enough Miles for the reward they want.
- Subtract Miles: the user will exchange their Miles for a reward voucher.
- Request Reports: an extra Use Case added. This is for the user to see what he or she used their Miles for. For example, were their Miles used or redeemed for shopping, entertainment or even healthcare.

CHAPTER 3 ERD DIAGRAM

3.1 The Use of ERD Diagrams

The ERD diagram is a model or blueprint representing the technical implementation of the database. Therefore, the technical implementation for the Discovery Rewards System is shown below:

Figure 3-1: ERD Diagram



From the abovementioned diagram for the Discovery Rewards System, the following can be derived:

- Account: the account user has a unique Account_ID. The Transaction_ID is stored so that the user can see their purchase history. The Account holder's name and surname is

also kept on record. The Account_Balance is also stored, so that the user can see the amount he or she has left in Discovery Miles.

- Currency: this entity has its own unique ID. The type of currency is important for the purchase, so that the user can choose from the different currencies in the system. The exchange rate is there so that the purchase amount is correctly converted to the currency selected. The date is also stored to keep record of the exchange rates constantly changing.
- Transaction: the user will be involved in a transaction using their Discovery Miles. Once a purchase has been made, it registers a new Transaction _ID. The Account_ID of the user is also stored as well as the currency that was used to make the purchase. The transaction amount is also recorded along with the date that the transaction occurred.

CHAPTER 4 FLOW DIAGRAMS

4.1 The Use of Flow Diagrams

The flow diagram below demonstrates the different application layering for the Discovery Account Rewards System. The Account System consists of five services, namely:

- The Service Layer;
- The Logic Layer; and
- The Persistence Layer.

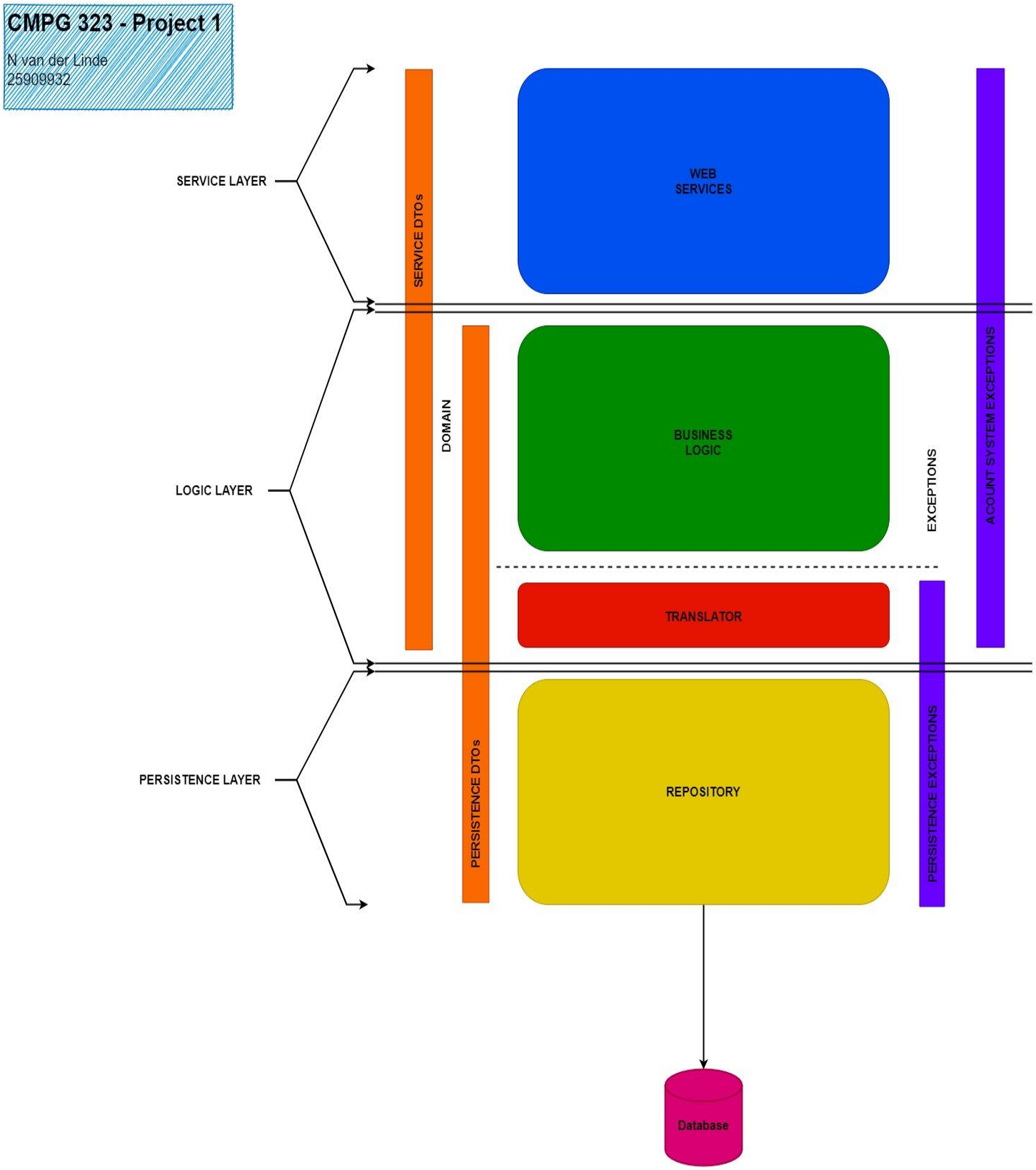
The Account System also has five services, specifically:

- Domain;
- Business Logic;
- Repository;
- Translator; and
- Web Services.

The Domain service has Persistence DTOs as well as Service DTOs. Then there are two Exceptions, which is the Account System Exceptions and the Persistence Exceptions.

Lastly, all of the services, layers, DTOs and the Exceptions connect to a Database.

Figure 4-1: Flow Diagram



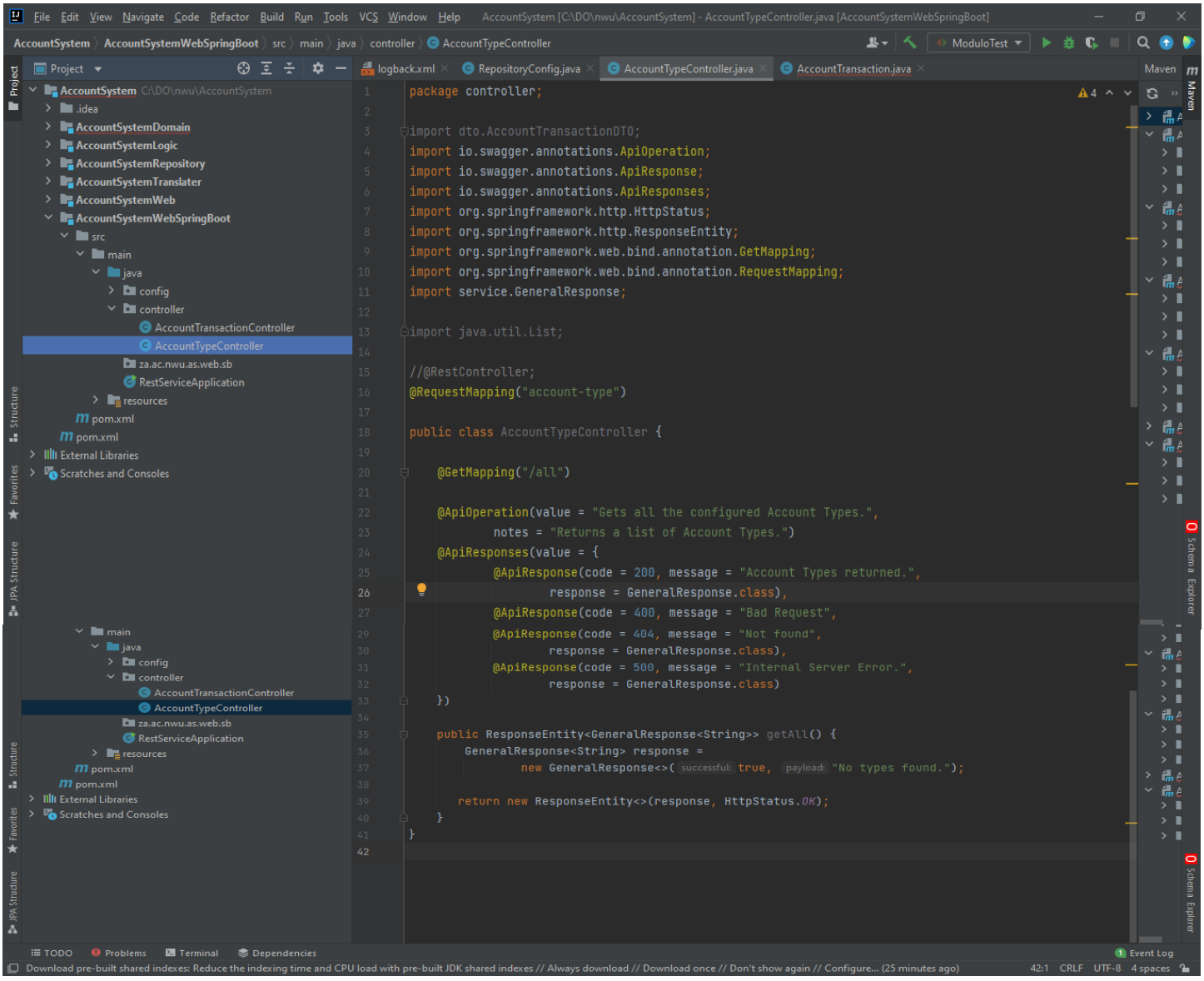
CHAPTER 5 THE USER GUIDE

5.1 The Account Type Controller Class

The AccountTypeController Class:

- This Java Class makes use of the GeneralResponse Class created in the Domain service. It returns the appropriate response when the program is installed. It can either return a positive or a negative result, as specified below:

Figure 5-1: The Account Type Controller Class

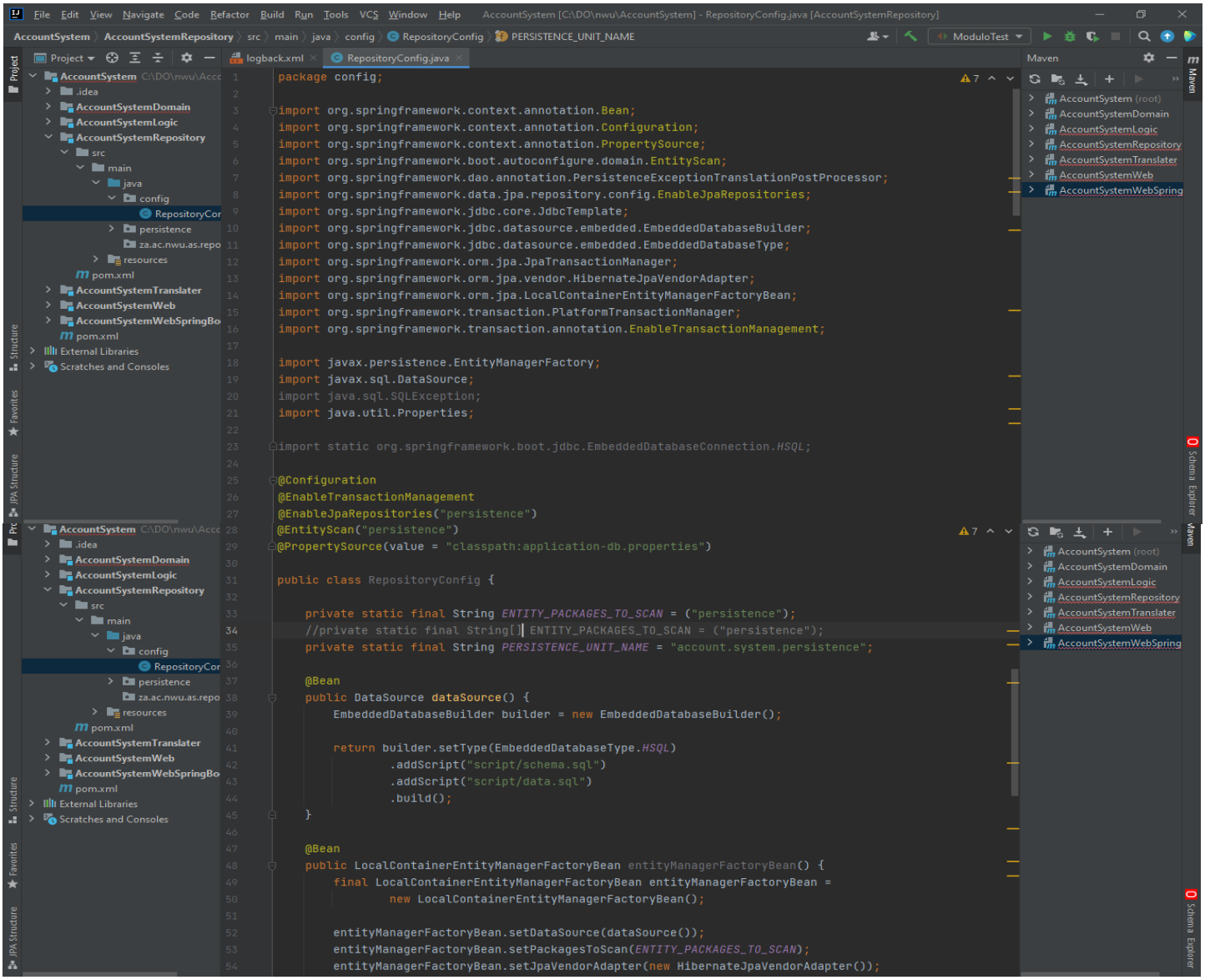


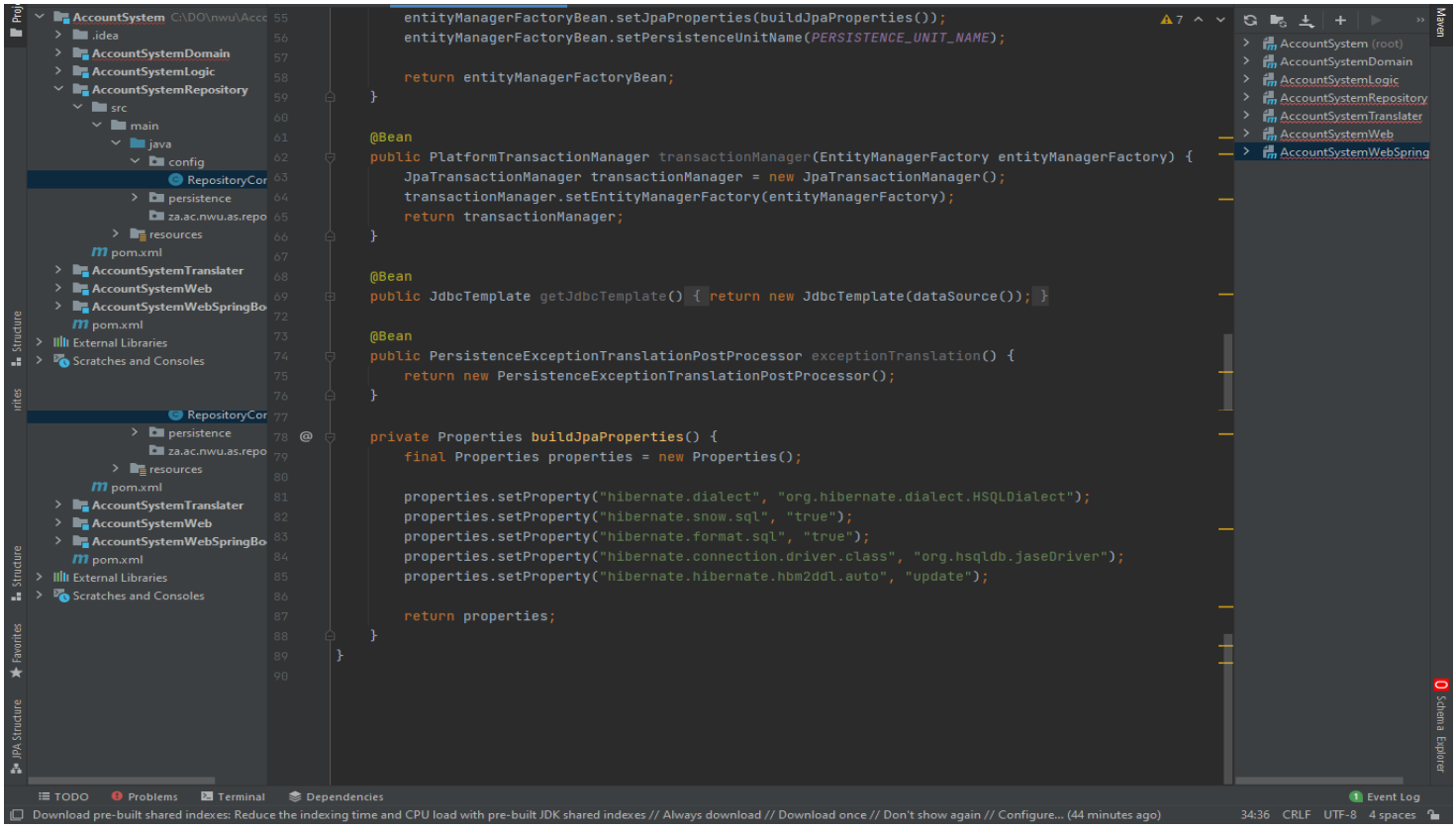
5.2 The Repository Configuration Class

The RepositoryConfig Class:

- This Class shows the configuration for the Repository service.

Figure 5-2: The Repository Configuration Class



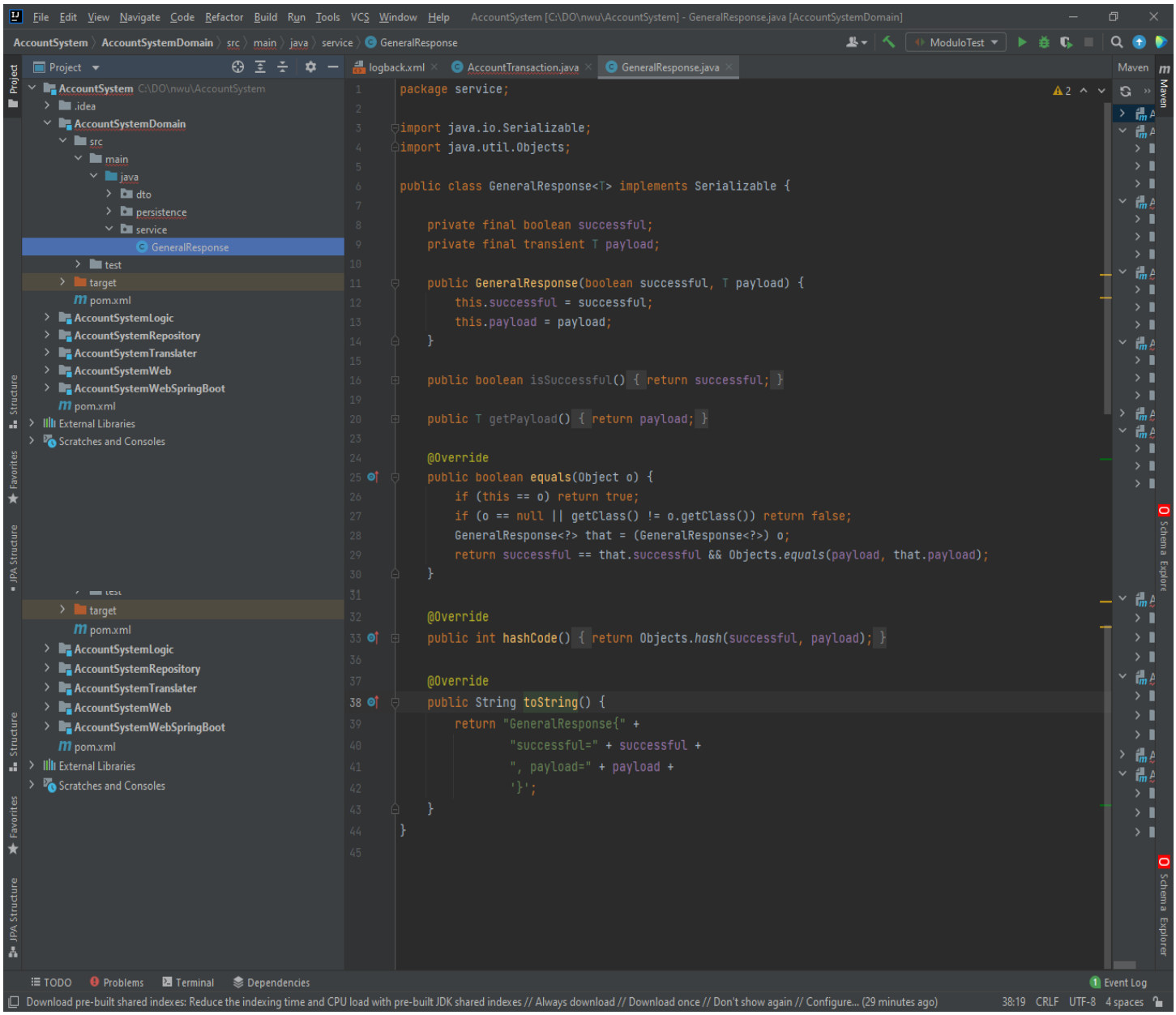

AccountSystem C:\DO\nwu\Acc
.idea
AccountSystemDomain
AccountSystemLogic
AccountSystemRepository
src
main
java
config
RepositoryCor
persistence
za.ac.nwu.as.repo
resources
pom.xml
AccountSystemTranslater
AccountSystemWeb
AccountSystemWebSpringBo
pom.xml
External Libraries
Scratches and Consoles
RepositoryCor
persistence
za.ac.nwu.as.repo
resources
pom.xml
AccountSystemTranslater
AccountSystemWeb
AccountSystemWebSpringBo
pom.xml
External Libraries
Scratches and Consoles
entityManagerFactoryBean.setJpaProperties(buildJpaProperties());
entityManagerFactoryBean.setPersistenceUnitName(PERSISTENCE_UNIT_NAME);
return entityManagerFactoryBean;
}
@Bean
public PlatformTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
JpaTransactionManager transactionManager = new JpaTransactionManager();
transactionManager.setEntityManagerFactory(entityManagerFactory);
return transactionManager;
}
@Bean
public JdbcTemplate getJdbcTemplate() { return new JdbcTemplate(dataSource()); }
@Bean
public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {
return new PersistenceExceptionTranslationPostProcessor();
}
private Properties buildJpaProperties() {
final Properties properties = new Properties();
properties.setProperty("hibernate.dialect", "org.hibernate.dialect.HSQLDialect");
properties.setProperty("hibernate.snow.sql", "true");
properties.setProperty("hibernate.format.sql", "true");
properties.setProperty("hibernate.connection.driver.class", "org.hsqldb.jaseDriver");
properties.setProperty("hibernate.hibernate.hbm2ddl.auto", "update");
return properties;
}
}
AccountSystem (root)
AccountSystemDomain
AccountSystemLogic
AccountSystemRepository
AccountSystemTranslater
AccountSystemWeb
AccountSystemWebSpring
TODO
Problems
Terminal
Dependencies
Event Log
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (44 minutes ago)
34:36
CRLF
UTF-8
4 spaces

5.3 The General Response Class

The GeneralResponse Class:

- This Java Class is created under the Domain service. This Class is used by the AccountTypeController Class, as can be seen in *Section 5.1.* above, as well as the AccountTransactionController Class.

Figure 5-3: The General Response Class

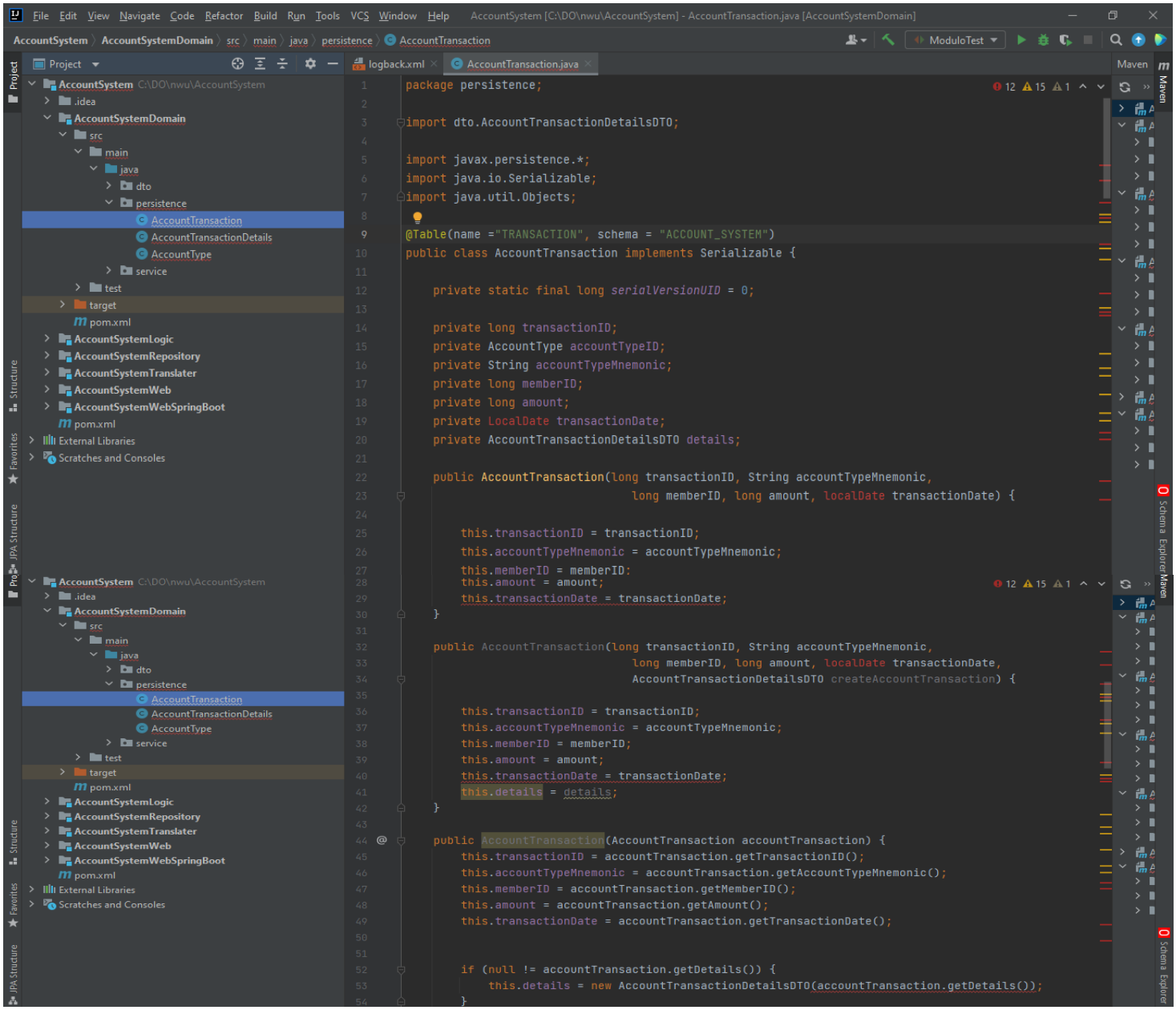


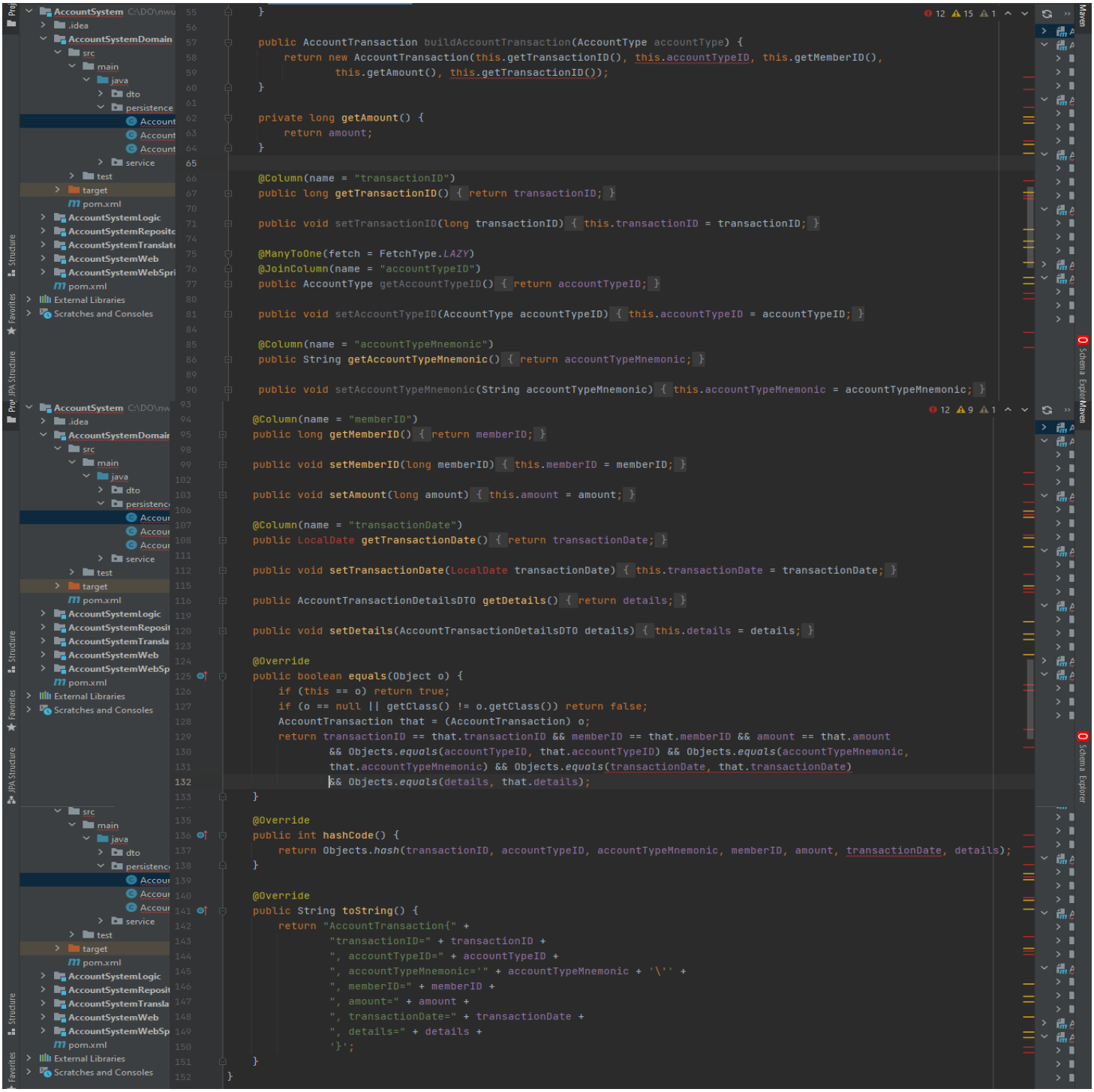
5.4 The Account Transaction Class

The AccountTransaction Class:

- This Class is created to keep track of a member's transactions. All transactions made or completed will use this Class. It links with the AccountType Class.
- It was designed according to the ERD. This program is not entirely the same as the ERD, it is just similar.

Figure 5-4: The Account Transaction Class



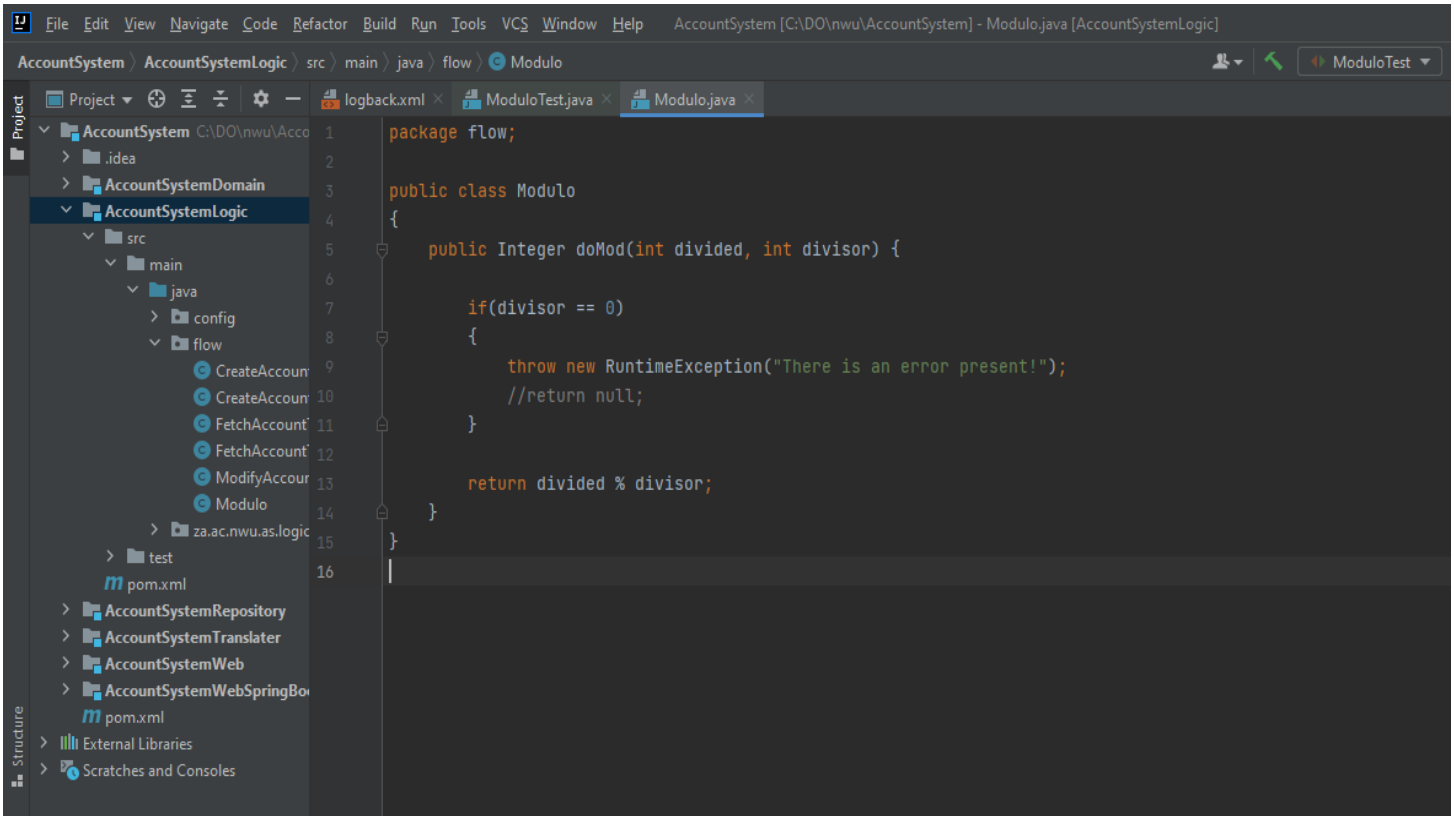

AccountSystem C:\DO\nwu
.idea
AccountSystemDomain
src
main
java
dto
persistence
service
test
target
pom.xml
AccountSystemLogic
External Libraries
Scratches and Consoles
}
public AccountTransaction buildAccountTransaction(AccountType accountType) {
return new AccountTransaction(this.getTransactionID(), this.accountTypeID, this.getMemberID(),
this.getAmount(), this.getTransactionID());
}
private long getAmount() {
return amount;
}
@Column(name = "transactionID")
public long getTransactionID() { return transactionID; }
public void setTransactionID(long transactionID) { this.transactionID = transactionID; }
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "accountTypeID")
public AccountType getAccountTypeID() { return accountTypeID; }
public void setAccountTypeID(AccountType accountTypeID) { this.accountTypeID = accountTypeID; }
@Column(name = "accountTypeMnemonic")
public String getAccountTypeMnemonic() { return accountTypeMnemonic; }
public void setAccountTypeMnemonic(String accountTypeMnemonic) { this.accountTypeMnemonic = accountTypeMnemonic; }
@Column(name = "memberID")
public long getMemberID() { return memberID; }
public void setMemberID(long memberID) { this.memberID = memberID; }
public void setAmount(long amount) { this.amount = amount; }
@Column(name = "transactionDate")
public LocalDate getTransactionDate() { return transactionDate; }
public void setTransactionDate(LocalDate transactionDate) { this.transactionDate = transactionDate; }
public AccountTransactionDetailsDTO getDetails() { return details; }
public void setDetails(AccountTransactionDetailsDTO details) { this.details = details; }
@Override
public boolean equals(Object o) {
if (this == o) return true;
if (o == null || getClass() != o.getClass()) return false;
AccountTransaction that = (AccountTransaction) o;
return transactionID == that.transactionID && memberID == that.memberID && amount == that.amount
&& Objects.equals(accountTypeID, that.accountTypeID) && Objects.equals(accountTypeMnemonic,
that.accountTypeMnemonic) && Objects.equals(transactionDate, that.transactionDate)
&& Objects.equals(details, that.details);
}
@Override
public int hashCode() {
return Objects.hash(transactionID, accountTypeID, accountTypeMnemonic, memberID, amount, transactionDate, details);
}
@Override
public String toString() {
return "AccountTransaction{" +
"transactionID=" + transactionID +
", accountTypeID=" + accountTypeID +
", accountTypeMnemonic='" + accountTypeMnemonic + '\'' +
", memberID=" + memberID +
", amount=" + amount +
", transactionDate=" + transactionDate +
", details=" + details +
'}';
}
}

5.5 Modulo and ModuloTest

The Modulo Class:

- This Java Class will calculate what the mod will be of two integers. A Runtime Exception is added to indicate whether there is an error present when the divisor equals null. If not, then the program will return the answer.

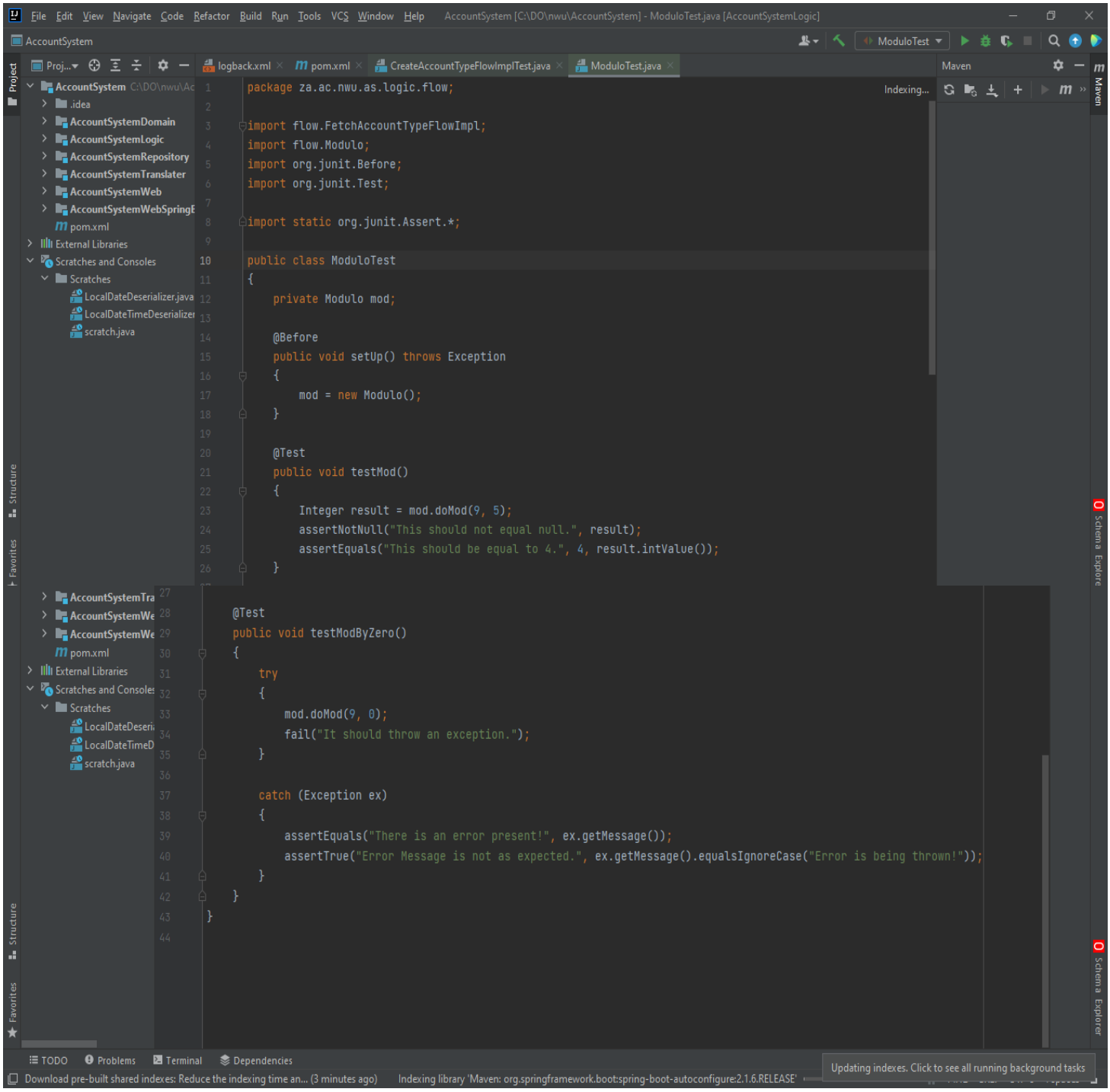
Figure 5-5-5: The Modulo Class



The ModuloTest Class:

- This Java Class tests the Modulo program, as explained above. Two integers are inserted for the dividend and the divisor, respectively. The result is printed out as an integer value. Then, another program tests if the two integers inserted can be divided by zero. This program has an exception error thrown out when the answer is equal to zero.

Figure 5-5-6 The Modulo Test Class



CHAPTER 6 CODE COVERAGE

6.1 Code Coverage Report

The code coverage report does not want to show. I did insert the correct dependency, the JaCoCo dependency, according to the videos provided. I tried to install my project and extract the report from *target, site* and then open the *index.html* file in Browser, but it did not want to work.

You are more than welcome to have a look at the JaCoCo dependency on GitHub, where I have uploaded the entire Discovery Account System's code.