EBU6304 Software Engineering

Case Study

An example of applying Agile methods to the software development

A Simple Banking System

A simple banking system is to be developed with the intention of providing a generic, reusable system from which to develop more realistic systems. The requirements of the system are to offer a number of different accounts, each of which provides specific services to the customer. The following are all types of accounts that must be supported by the system:

- Saver account
- Junior account
- Current account

When a customer joins the bank, they are required choose an account type to open, and must credit it with a minimum figure. A customer may open more than one type of account.

A Simple Banking System

The following core functions are to be supported by the system:

1. **Open Account**: In order to open an account, the customer must provide the following information:
   - name
   - address
   - date of birth
   - type of account to be opened

   Only customers under the age of 16 may open a Junior account. To determine the credit status of a customer, the bank sends the customer's details to a Credit Agency, who then carries out a credit search. Provided that the customer has a satisfactory credit history, a new account is opened. Each account has a unique account number. A customer is also issued a separate personal identification number (PIN) for that account.

A Simple Banking System

2. **Deposit Funds**: Funds may be deposited in an account provided that the depositor provides the appropriate account number. When funds are deposited, they are either cleared (the funds have been fully credited, e.g. cash), or un-cleared (transfer of funds is pending, e.g. Cheque). Cleared funds are immediately credited to the account.
3. **Clear Funds**: An external bank clearing system periodically clears un-cleared funds. Once cleared, they are immediately credited to the account.

A Simple Banking System

4. **Withdraw Funds**: Customers may withdraw funds from an account by supplying their account number, an appropriate identification (in this case, their PIN), and the amount to be withdrawn. A customer cannot withdraw more funds than their limit permits. The type of account the funds are being drawn from determines a customer's limit. In the case of Junior and Saver accounts, no withdrawal should result in a negative balance. In the case of a Current account, a customer may withdraw additional funds, up to, but not exceeding, their overdraft limit. For a withdrawal from a Saver account, a minimum period of notice (in days) must be given before any withdrawal can be made. On leaving the bank, all customers must have re-paid their debts.

A Simple Banking System

5. **Suspend Account**: In some situations, accounts may be suspended and no further transactions may occur until the account is re-instated.
6. **Close Account**: A customer can choose to close their account provided that the balance has been cleared.

```
The full case study PDF file can be
downloaded from QMPLUS
```

The activities and outcomes outlined in this case study are guidelines only. They are not complete. You should try to complete it by yourself.

Bear in mind that there is no absolute right answer – your solution may be perfectly appropriate.

Requirements:

- Epics
- User Stories: prioritisation and estimation
- Product backlog
- Prototype

Activities: story writing workshop

- Find the users
- Find the stories

$\Big\rangle$ **`Using fact finding techniques`**

- Write stories using story cards
  - Epics and break down stories
- Prioritise the stories
- Create product backlog
- Paper prototype
- Estimation and Iteration plan

Find the users

- Key concept:
  - Find users according to the **ROLES**
  - May not be a person
  - A person may have several roles
- Ask the following questions:
  - Who will interact with the system? (Input information to the system and/or Receive information from the system.)
  - Who will use the system?
  - Who will supply information?
  - Any external resources?
  - Interact with other systems?
  - …

Glossary

- Customer:
  - Any person for whom the bank agrees to operate an account.
- Depositor:
  - An individual or company that puts money in a bank account.
- Bank Employee:
  - A worker who is hired to perform a job in the bank.
- Credit Agency:
  - An external system that carries out a credit check of the customers.
- Clearing System:
  - An external system that periodically clears un-cleared funds.

Find the stories

- Key concept:
  - What do you want the system to do for each user?
  - Epics: Top level user stories.
    - Open account
    - Deposit Funds
    - Clear Funds
    - Withdraw Funds
    - Suspend Account
    - Close Account
  - Stories: break down stories

Write epics

Story name	**Open Account**	**Story ID**	**E01**

As a **Bank Employee**

I want to enter the Customer's details and account type in the system along with the opening deposit.

So that I can open an account for the customer.

Priority	**very high, high, medium, low, very low**	**Iteration number**	
Date Started		**Date Finished**	

Write epics

Story name	Deposit Funds	**Story ID**	E02

As a Depositor (who may not be the account holder)

I want to enter the details of the account to be credited and the amount of funds to be deposited.

So that I can deposit the funds to the account.

Priority	very high, high, medium, low, very low	**Iteration number**	
Date Started		**Date Finished**	

Write epics

Story name	Clear Funds	Story ID	E03
As an external bank clearing system I want periodically clears un-cleared funds. So that the funds are credited to the account.			
Priority	very high, high, medium, low, very low	Iteration number	
Date Started		Date Finished	

Write epics

Story name	**Withdraw Funds**	**Story ID**	**E04**

As a **Customer**

I want to supply my account number, the correct PIN, and the amount to be withdrawn.

So that I can withdraw funds from my account.

Priority	**very high, high, medium, low, very low**	**Iteration number**	
Date Started		**Date Finished**	

Write epics

Story name	**Suspend Account**	**Story ID**	**E05**

As a **Bank Employee**

I want suspended an account.

So that no further transactions may occur until the account is re- instated.

Priority	**very high, high, medium, low, very low**	**Iteration number**	
Date Started		**Date Finished**	

Write epics

Story name	**Close Account**	**Story ID**	**E06**

As a **Bank Employee**

I want to check if the account balance is zero when I receive the instruction from the customer to close the account.

So that I can close the account.

Priority	**very high, high, medium, low, very low**	**Iteration number**	
Date Started		**Date Finished**	

breakdown stories

Story name	**add new customer**	**Story ID**	**E01a**

As a **Bank Employee**

I want to obtain a customer’s name, address and date of birth and then verify the details provided. Once verified, I can enter the details to the system.

So that I can add new customer to the system and request credit check.

Priority	**very high, high, medium, low, very low**	**Iteration number**
Date Started		**Date Finished**

Acceptance Criteria

- **verify that the customer does not exist in system**
- **make sure the customer credit status is recorded as unknown**

Notes

breakdown stories

Story name	confirm credit status	Story ID	E01b

As a Bank Employee

I want update customer credit status as satisfactory once I receive the conformation from credit agency following a credit check.

So that I can carry on to open the account.

Priority	very high, high, medium, low, very low	Iteration number	
Date Started		Date Finished	

Acceptance Criteria

- verify that customer exists in system matching details given

Notes

breakdown stories

Story name	**create account**	**Story ID** **E01c**

As a **Bank Employee**

I want to create an account of the type chosen for an existing customer, generate a unique account number and PIN.

So that I can open the account and notify customer of account number and PIN.

Priority	**very high, high, medium, low, very low**	**Iteration number**
Date Started		**Date Finished**

Acceptance Criteria

- **Verify that customer exists in system matching details given**
- **Make sure that customer's credit status has been recorded as satisfactory**
- **Verify that account of required type is created**
- **Verify that account number is unique**
- **Verify that PIN is 6 digits**

Notes

breakdown stories

Story name	deposit cash	**Story ID**	E02a

As a Depositor

I want to supply the account number and the amount of cash to be deposited

So that I can deposit funds to the account.

Priority	very high, high, medium, low, very low	**Iteration number**	
Date Started		**Date Finished**	

Acceptance Criteria

- Verify that account exists matching account number
- Verify that the account is not inactive or suspended, otherwise return funds to depositor
- Make sure that an receipt is issued to depositor
- Make sure that deposit is marked as cleared

Notes

Prioritise the user stories

- Prioritise the stories
  - Which are the most important stories?
  - Which should be released early?
  - MoSCoW rule
  - Dependency
  - …

Prioritise the user stories

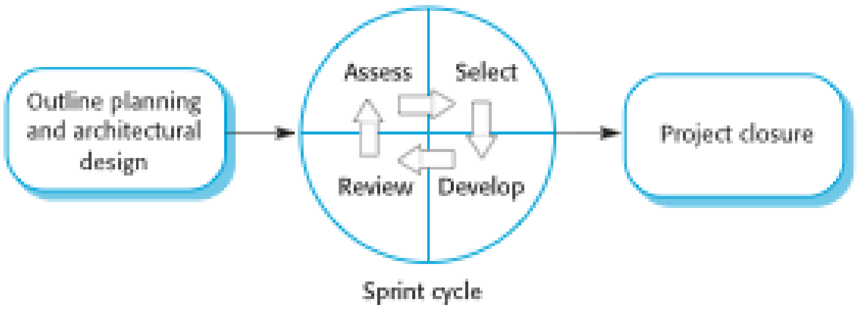
Story id	Story	Priority
E01a	add new customer	Very high
E01b	confirm credit status	Very high
E01c	create account	Very high
E02a	deposit cash	High
E03	clear Funds	Very low
E04	withdraw Funds	High
E05	suspend Account	Low
E06	close account	Medium
…	…	…

Product backlog (excel template)

Story ID	Story Name	Description	Priority	Iteration number	Acceptance Criteria	Notes	Date started	Date finished

The template is available to download from QMPLUS

Project planning



constraint:

- Project duration – 8 weeks
- Resource – number of people in the group
- Hardware and software required
- Budget

Each Iteration: 2 weeks

Scrum master: (project manager)

- Organise daily stand up meeting
- Track backlog
- Record decisions
- Communication

Iteration planning

Story id	Story	Iteration
E01a	add new customer	1
E01b	confirm credit status	1
E01c	create account	1
E02a	deposit cash	2
E03	clear Funds	5
E04	withdraw Funds	3
E05	suspend Account	6
E06	close account	5
…	…	…

Estimating story points

- Story points: using Fibonacci Sequence: 1,2,3,5,8,13,21…

Story id	Story	Story Points
E01a	add new customer	2
E01b	confirm credit status	1
E01c	create account	3
E02a	deposit cash	5
E03	clear Funds	3
E04	withdraw Funds	5
E05	suspend Account	8
E06	close account	13 – may need to break down
…	…	…

Prototype user interface

- Logical user-interface design (information)
  - Which user-interface elements are needed?
  - How are these elements related to each other?
  - What should they look like?
  - How should they be manipulated?

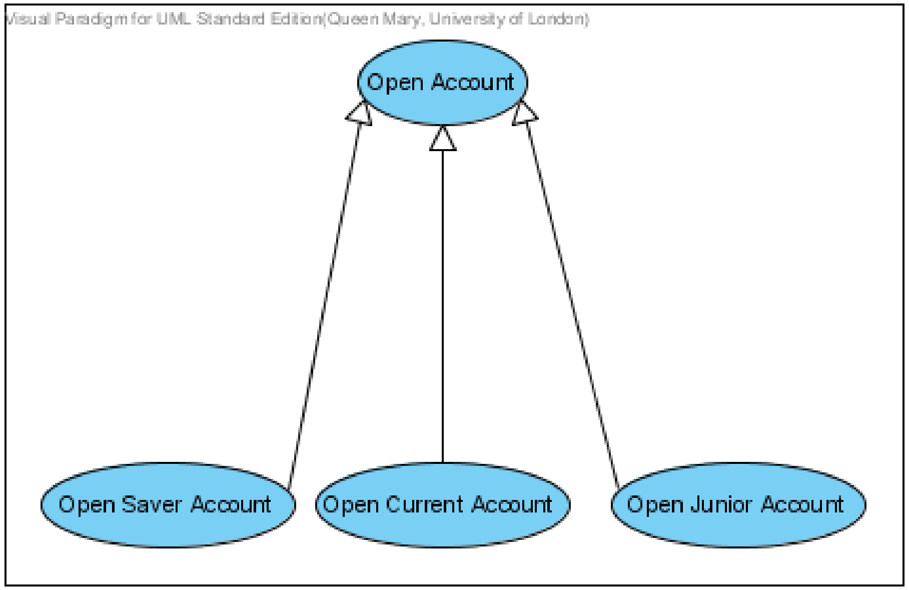
- Low-fidelity Prototyping
  - Paper and sketch
  - Get rapid feedback

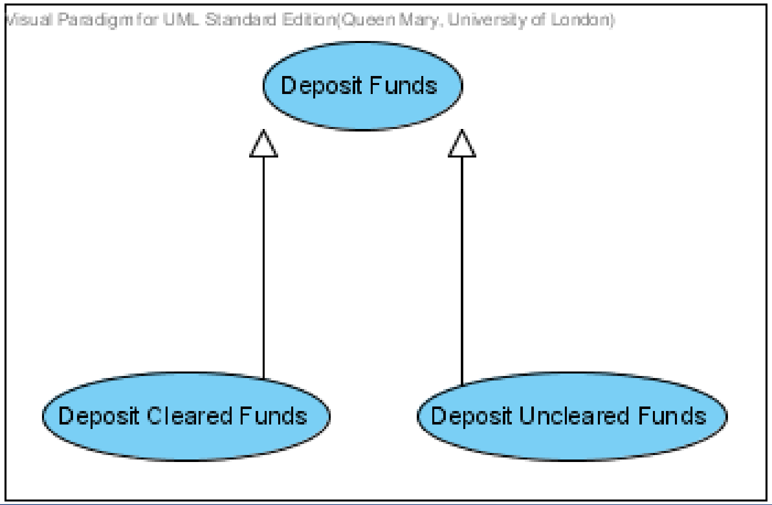
Review product backlog

- There is no right or wrong
- The first product backlog is rarely the final
- Need to be reviewed frequently
  - look for **relationships** between stories.
    - Generalisation
    - Inclusion
    - Extension
  - Look for similarity and difference of users
    - Ask: will they use the system differently?
    - Define actor according to the **Role**!

Relationships between stories

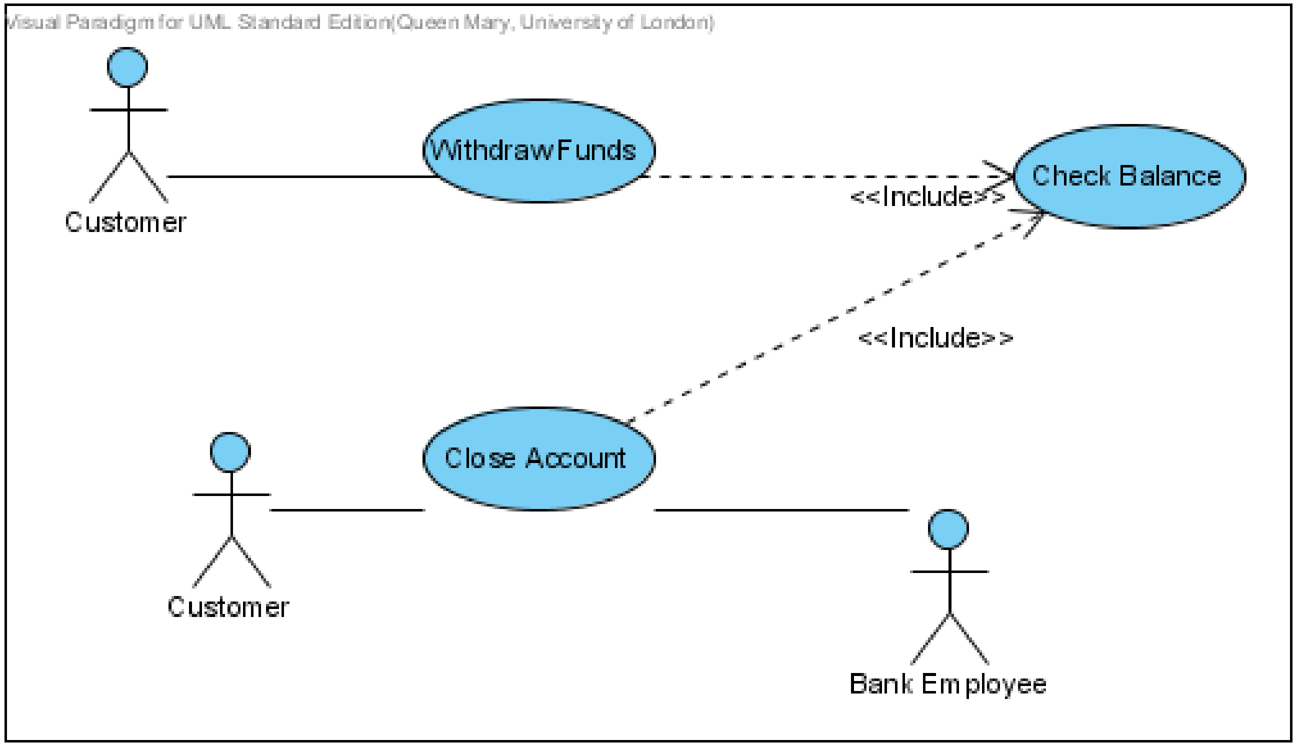
- **Generalisation**
  - Inheritance relationship between stories
    - One story is more general and the other is a specialisation of the first one
    - They are similar but one does a little bit more
  - The general story is more generic and can be applied to different situations


Visual Paradigm for UML Standard Edition(Queen Mary, University of London)
Open Account
Open Saver Account
Open Current Account
Open Junior Account


Visual Paradigm for UML Standard Edition(Queen Mary, University of London)
Deposit Funds
Deposit Cleared Funds
Deposit Uncleared Funds

Relationships between stories

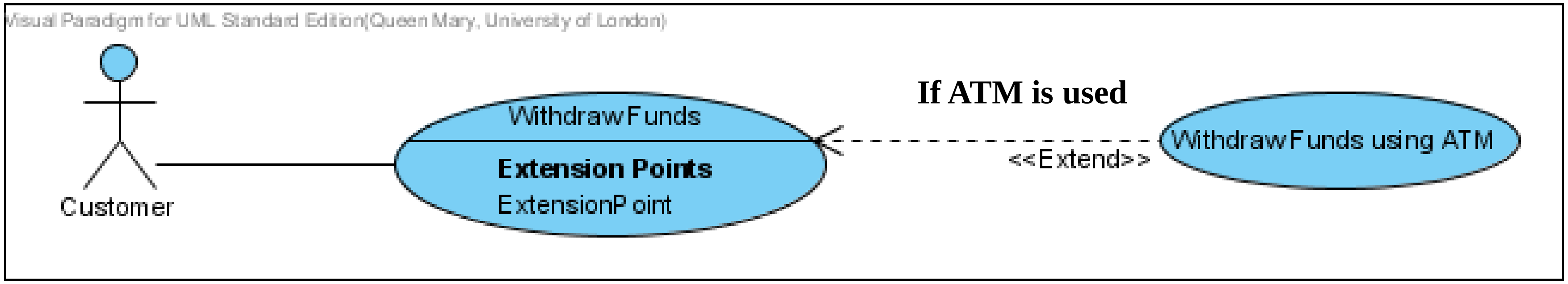
- Inclusion
  - There is a **shared behaviour** in stories



When a part of the behaviour (actions) is similar in more than one story and you do not want to keep copying the same actions again and again in different stories.

Relationships between stories

- Extension
  - Additional or optional stories that can be extended from a more general story
    - The story is extended if some condition is true



Analysis and Design:

Classes

Class relationships

Attributes and operations

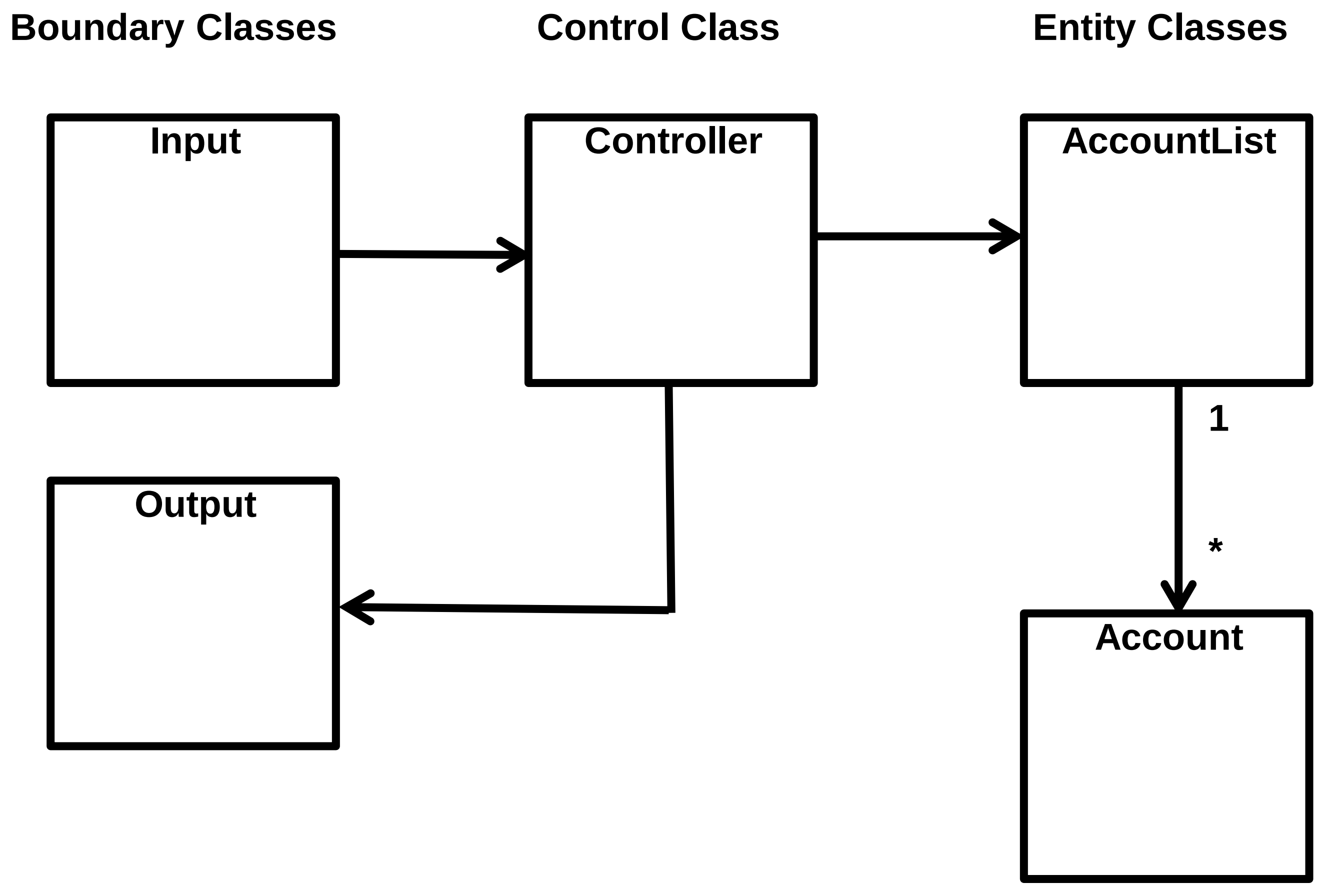
Activities

- Identify Entity, Boundary and Control classes
- Identify class relationships
- A conceptual class diagram
- Identify attributes for each class
- Add constraints
- Operations

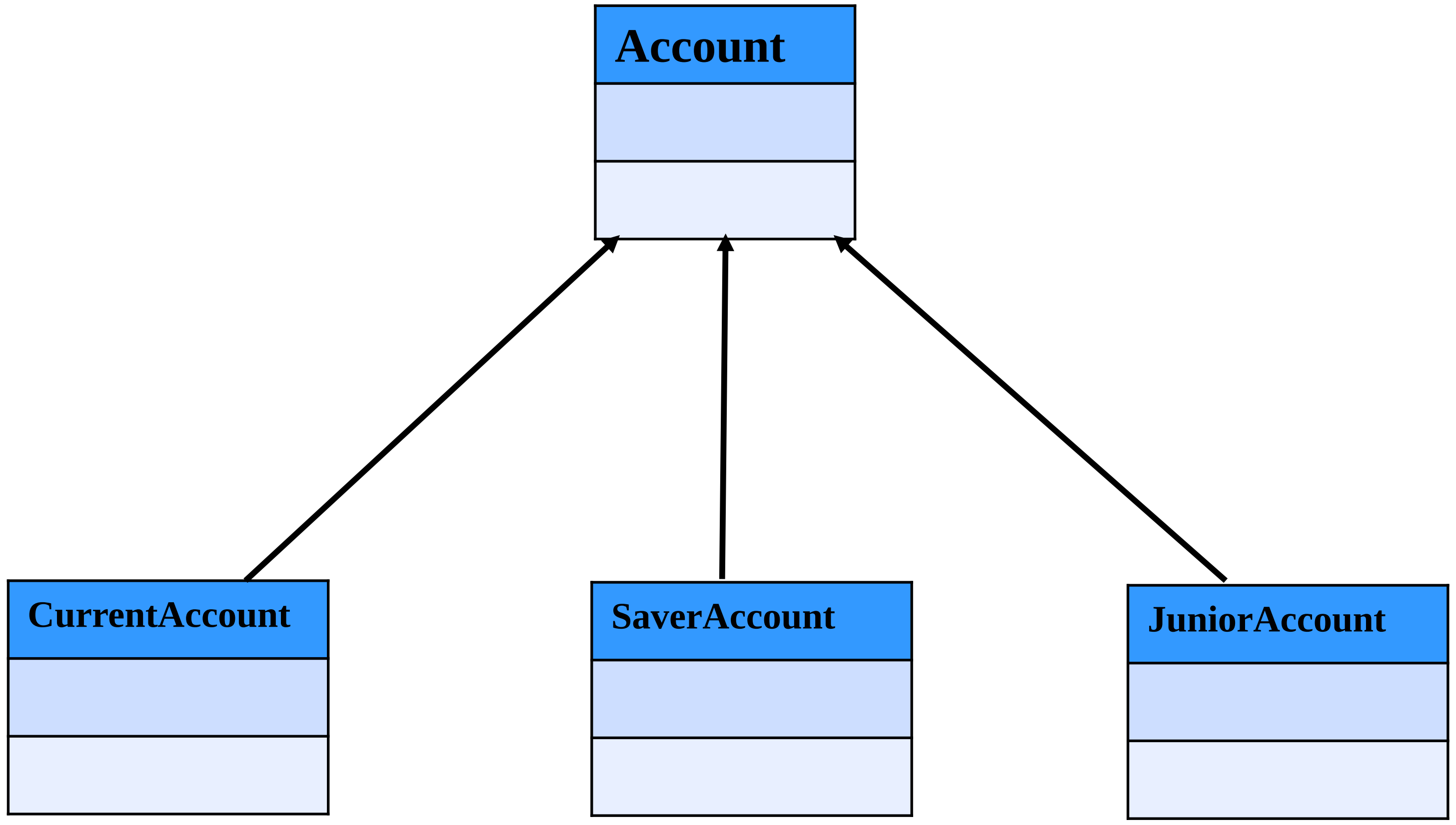
Identify Entity, Boundary and Control classes

- Boundary class – is dependent on the specific interface technologies in use
- Entity classes – which represent persistent information are usually created using databases
  - Ex. Creating a design classes that map to tables in a relational data model
- Control classes – Distribution issues, performance issues, transaction issues

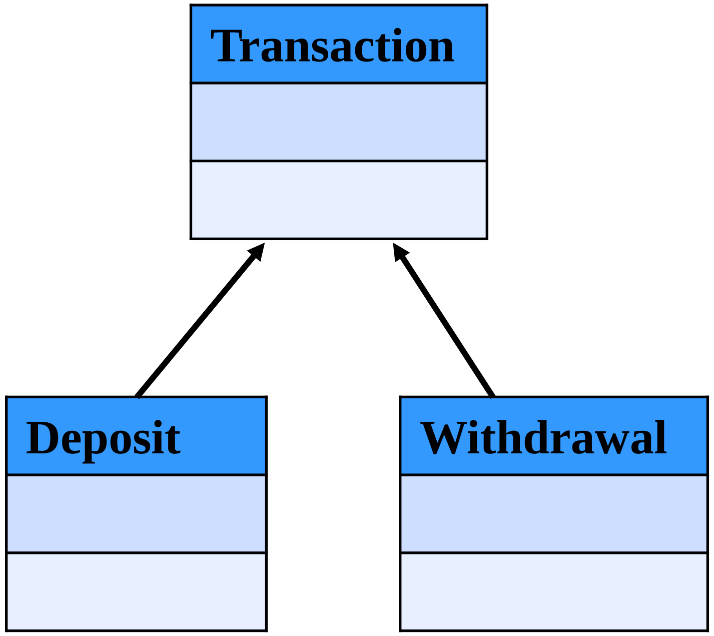
```
Entity classes: Customer, Account, Transaction
Boundary classes: BankUI
Control classes: BankControl
```


Boundary Classes
Control Class
Entity Classes
Input
Controller
AccountList
Output
Account
1
*

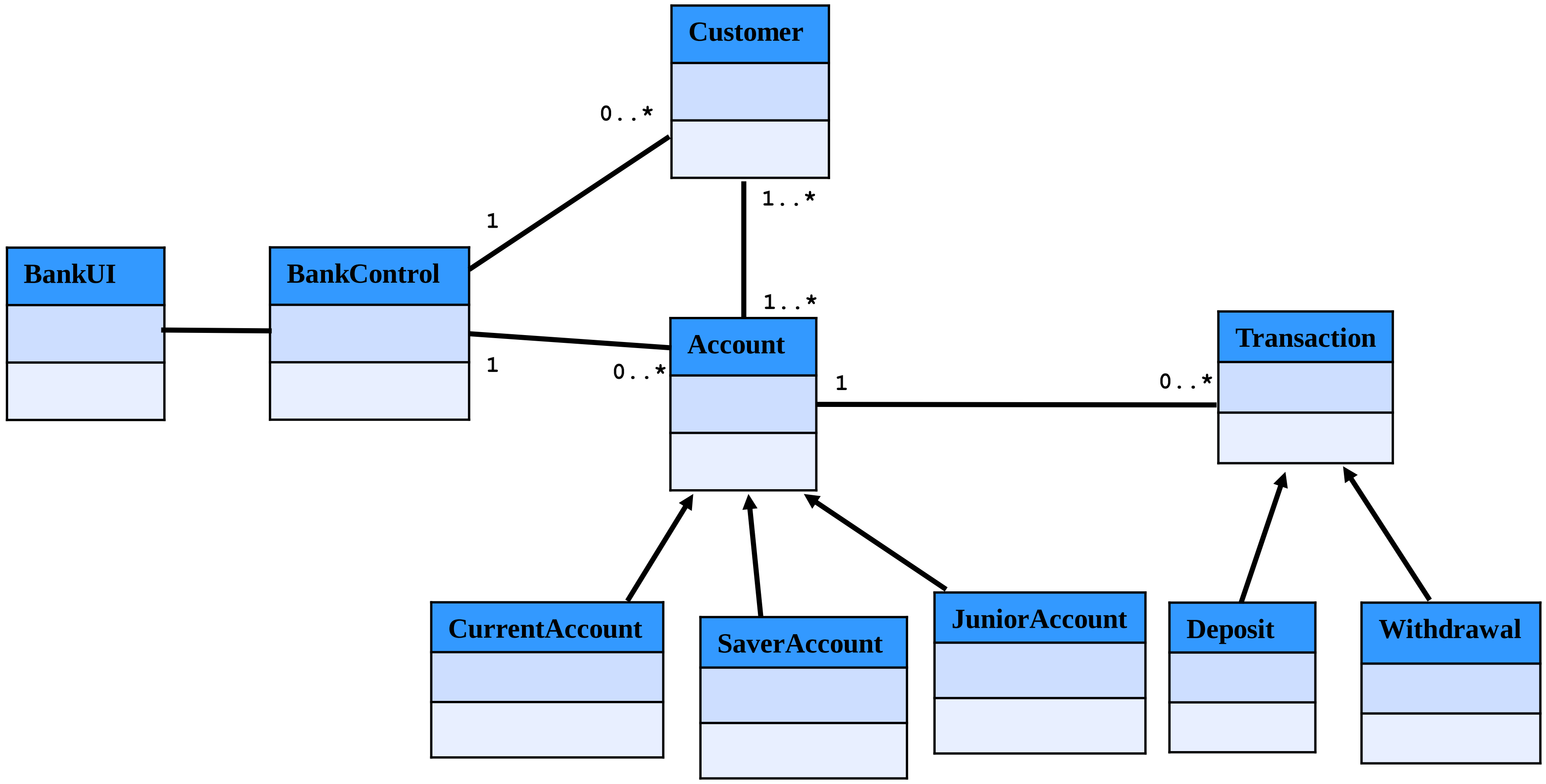
Identify class relationships



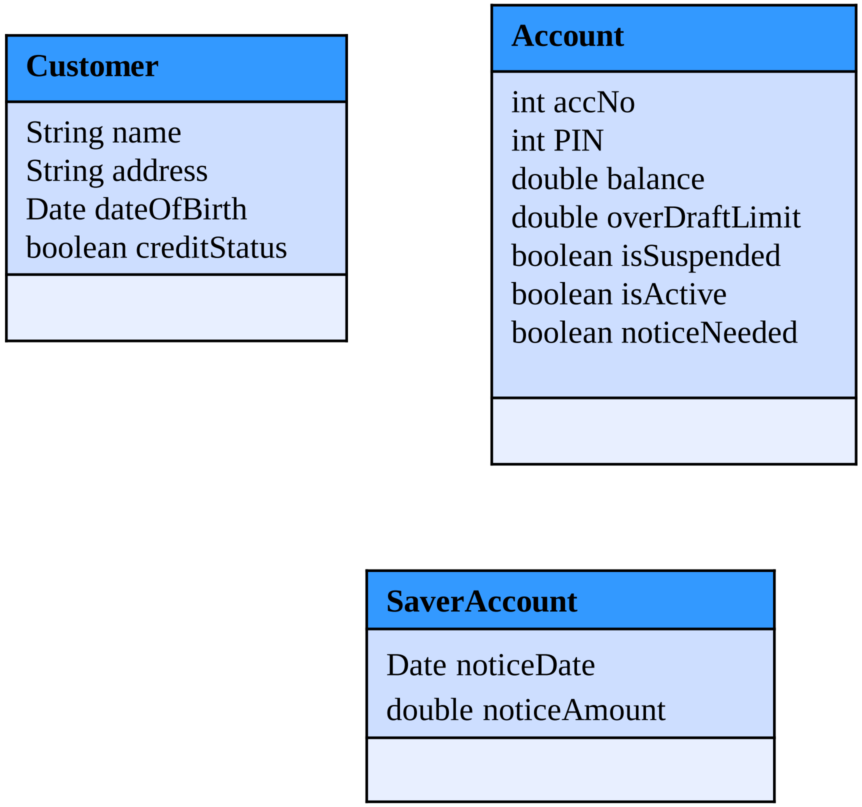
Identify class relationships



A conceptual(initial) class diagram



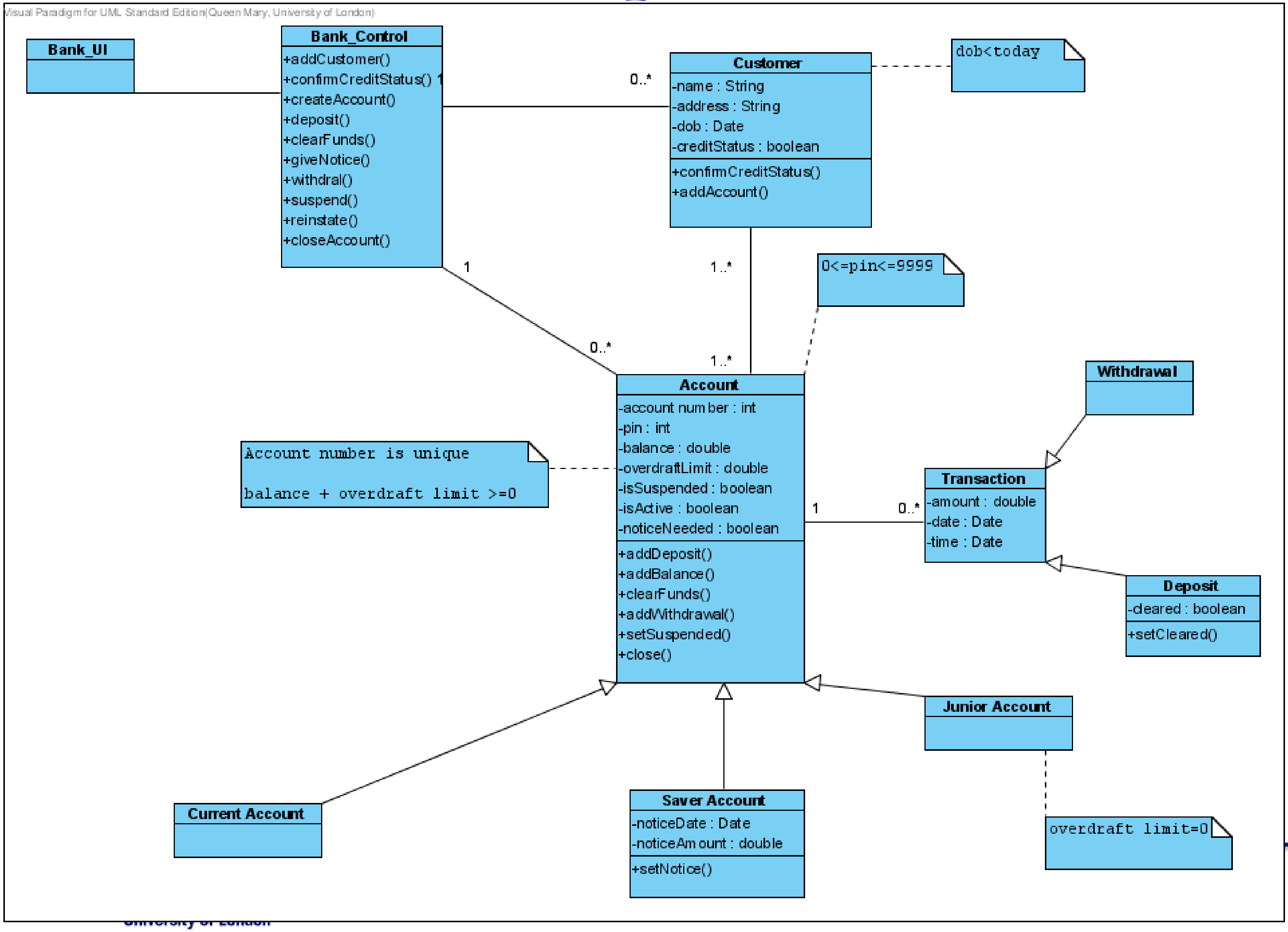
Identify attributes for each class



Add constraints

```
dateOfBirth: valid date, <today
accNo is unique
balance + overDraftLimit >=0
…
…
```

Add operations



Implementation and Testing:

TDD

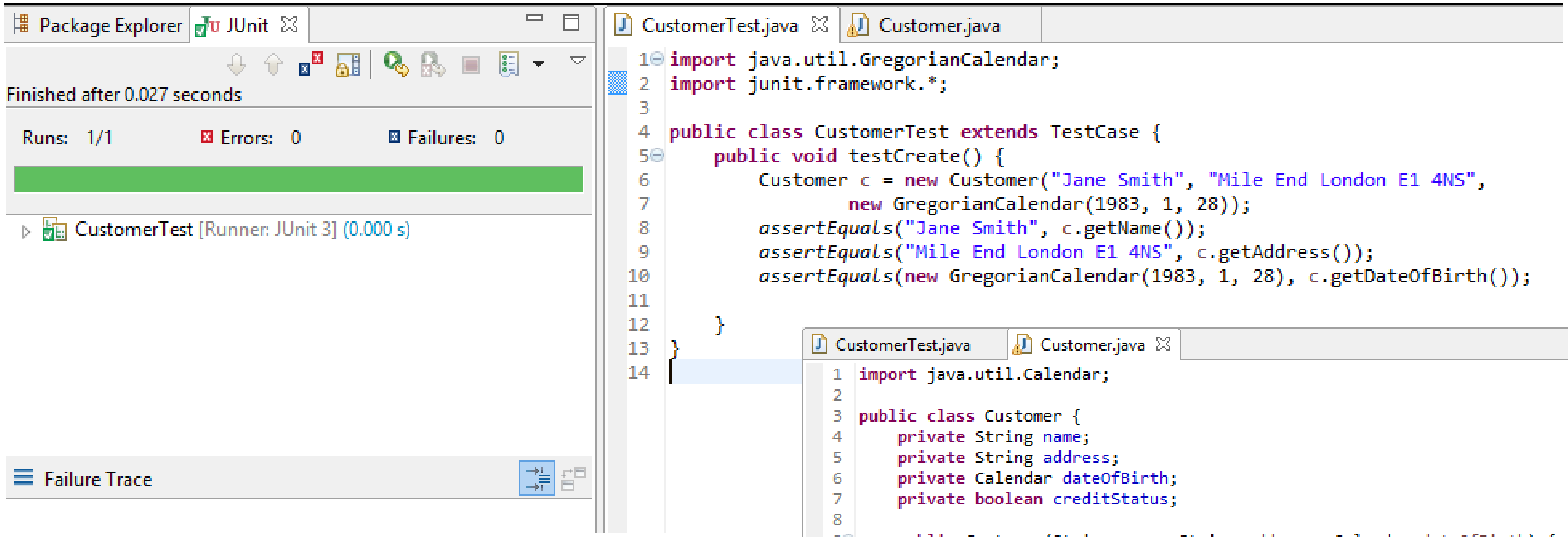
Test classes

Production classes

Refactoring

Activities

- Each iteration:
  - Pick up stories to implement
  - Write test programs
  - Write production programs
  - Run all tests
  - Refactoring
  - Deliverables



Package Explorer | JUnit

Finished after 0.027 seconds

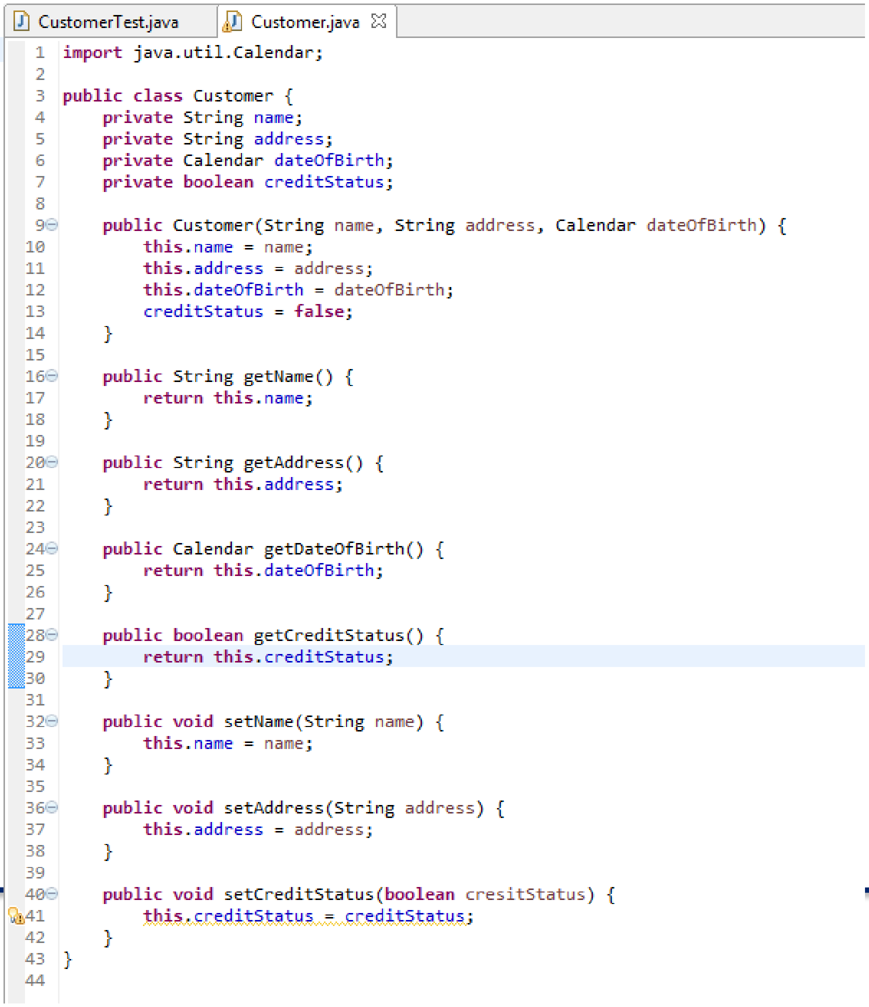
Runs: 1/1 Errors: 0 Failures: 0

CustomerTest [Runner: JUnit 3] (0.000 s)

Failure Trace

CustomerTest.java | Customer.java

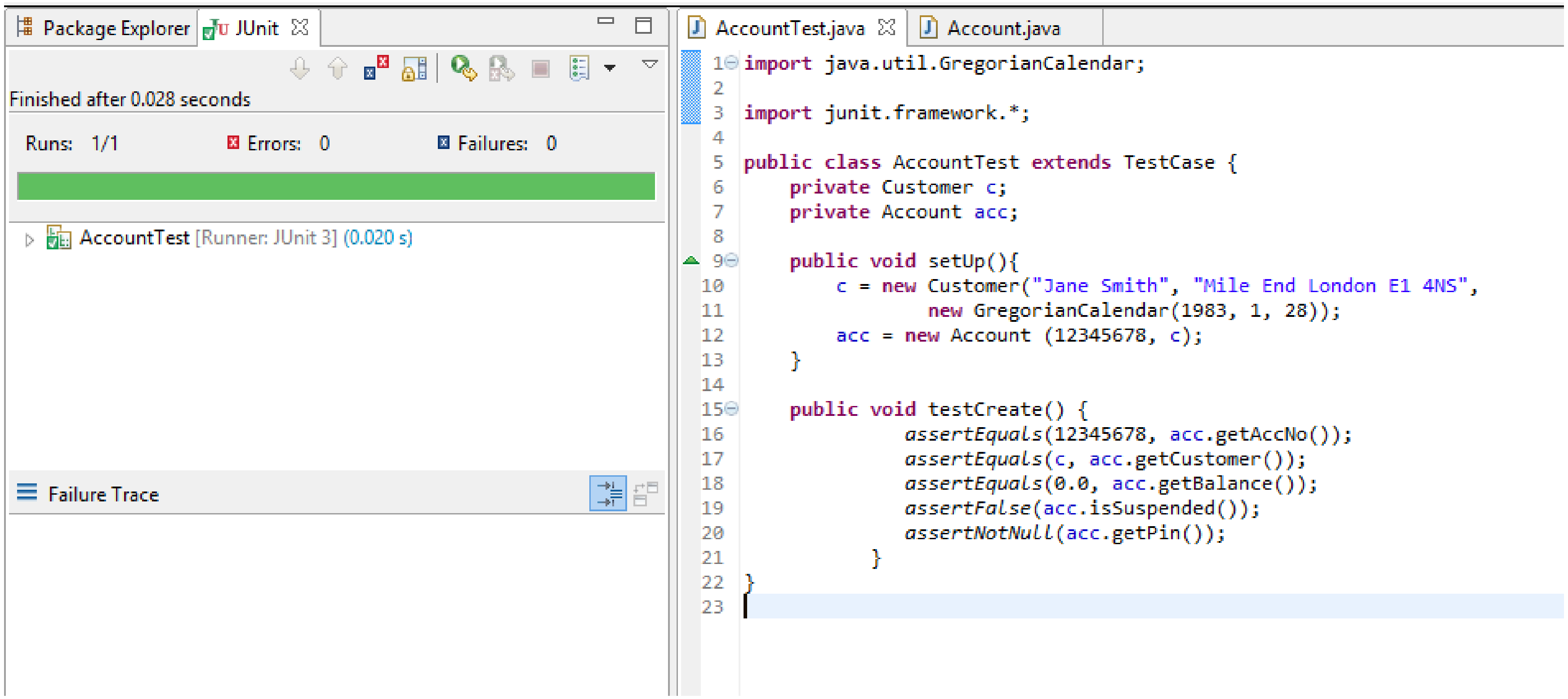
```java
import java.util.GregorianCalendar;
import junit.framework.*;

public class CustomerTest extends TestCase {
    public void testCreate() {
        Customer c = new Customer("Jane Smith", "Mile End London E1 4NS",
                new GregorianCalendar(1983, 1, 28));
        assertEquals("Jane Smith", c.getName());
        assertEquals("Mile End London E1 4NS", c.getAddress());
        assertEquals(new GregorianCalendar(1983, 1, 28), c.getDateOfBirth());

    }
}

```



CustomerTest.java | Customer.java

```java
import java.util.Calendar;

public class Customer {
    private String name;
    private String address;
    private Calendar dateOfBirth;
    private boolean creditStatus;

    public Customer(String name, String address, Calendar dateOfBirth) {
        this.name = name;
        this.address = address;
        this.dateOfBirth = dateOfBirth;
        creditStatus = false;
    }

    public String getName() {
        return this.name;
    }

    public String getAddress() {
        return this.address;
    }

    public Calendar getDateOfBirth() {
        return this.dateOfBirth;
    }

    public boolean getCreditStatus() {
        return this.creditStatus;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public void setCreditStatus(boolean cresitStatus) {
        this.creditStatus = creditStatus;
    }
}

```

CustomerTest
and
Customer



Finished after 0.028 seconds

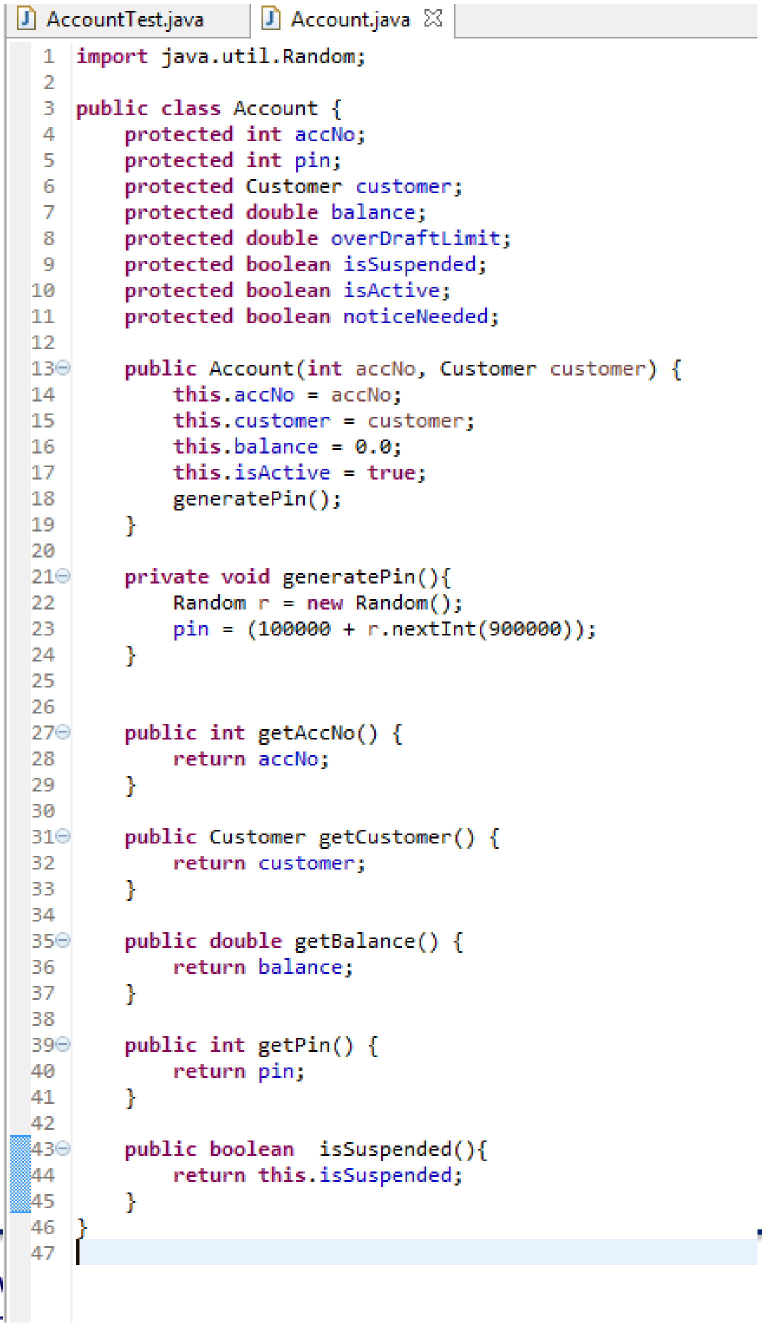
Runs: 1/1 Errors: 0 Failures: 0

AccountTest [Runner: JUnit 3] (0.020 s)

```java
import java.util.GregorianCalendar;

import junit.framework.*;

public class AccountTest extends TestCase {
    private Customer c;
    private Account acc;

    public void setUp(){
        c = new Customer("Jane Smith", "Mile End London E1 4NS",
                new GregorianCalendar(1983, 1, 28));
        acc = new Account (12345678, c);
    }

    public void testCreate() {
            assertEquals(12345678, acc.getAccNo());
            assertEquals(c, acc.getCustomer());
            assertEquals(0.0, acc.getBalance());
            assertFalse(acc.isSuspended());
            assertNotNull(acc.getPin());
        }
}
```

AccountTest

Account



AccountTest.java | Account.java

```java
import java.util.Random;

public class Account {
    protected int accNo;
    protected int pin;
    protected Customer customer;
    protected double balance;
    protected double overDraftLimit;
    protected boolean isSuspended;
    protected boolean isActive;
    protected boolean noticeNeeded;

    public Account(int accNo, Customer customer) {
        this.accNo = accNo;
        this.customer = customer;
        this.balance = 0.0;
        this.isActive = true;
        generatePin();
    }

    private void generatePin(){
        Random r = new Random();
        pin = (100000 + r.nextInt(900000));
    }


    public int getAccNo() {
        return accNo;
    }

    public Customer getCustomer() {
        return customer;
    }

    public double getBalance() {
        return balance;
    }

    public int getPin() {
        return pin;
    }

    public boolean  isSuspended(){
        return this.isSuspended;
    }
}

```

Summary

This case study showed how to solve problems using Agile methods.

You should apply the similar approach to your group project.