**School of Information Technologies**

Faculty of Engineering & IT

**ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT**

**Unit of Study:** SOFT2412

**Assignment name:** Group Project 1 – Tool for Agile Software

**Tutorial time:** Tuesday 4pm **Tutor name:** Afiq Asyraf Shahabudin

**DECLARATION**

We the undersigned declare that we have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project team members** | | | | |
| **Student name** | **Student ID** | **Participated** | **Agree to share** | **Signature** |
| **1. Beckham Smith** | 500544244 | **Yes/No** | **Yes/No** | B.Smith |
| **2. Ryan Zangari** | 500500158 | **Yes/No** | **Yes/No** | R.Zangari |
| **3. Ricardo Akkari** |  | **Yes/No** | **Yes/No** |  |
| **4. Jacqui O’Sullivan** | 490402111 | **Yes/No** | **Yes/No** | J. O’Sullivan |
| **5. Adrian Lin** | 49050082 | **Yes/No** | **Yes/No** | A.Lin |

|  |  |  |
| --- | --- | --- |
| Level 2, SIT Building, J12  The University of Sydney  NSW 2006 Australia | **T** +61 2 9351 3423  **F** +61 2 9351 3838  **E** sit.info@sydney.edu.au **sydney.edu.au/it** | ABN 15 211 513 464 CRICOS 00026A |

**Technical Report SOFT2412 Assignment 1**

**2021, Semester 2**

*By Jacqui O’Sullivan, Beckham Smith, Ricardo Akkari, Adrian Lin, & Ryan Zangari*

*Group/Individual Collaboration*

For the main group collaboration, we used Messenger as a primary form of communication. This proved effective as it allowed the team to easily communicate ideas with each other and discuss different implementations for the assignment. Hence, by all members being active within the chat everyone was constantly updated on any decisions made by the team as well the history of messages allowed us to go back to check previous choices which were made. Another reason for its effectiveness was it ensured that merge conflicts would be minimised since a member could type in chat when they were working on an aspect of the assignment.

*Individual contributions:*

Jacqui O’Sullivan:

I wrote the Card class with the necessary attributes like the cardNumber, pinNumber, fundsAvailable and isLostOrStolen. Each Card instance has methods for validity checks on the card number and PIN, reported cards and checking the dates. The setFunds() method was written to take in an amount specified by the user to the ATM, which will change that card’s current available amount by a positive or negative value depending on whether a withdrawal or deposit is being processed. Wrote basic withdraw() and deposit() methods to change the card balance as necessary by calling setFunds(), and error checks for the amount were added by other group members later on. Also wrote a couple of the CardTest tests, and made changes/additions to other tests when needed. In the WelcomeController component of the UI, I added the checks for 3 attempts at a correct card number + pin combination before the card could no longer be used.

Beckham Smith:

I help implement the ATM class inside the ATM.java file. This involved initialling writing the withdraw function as well as aiding in the deposit function in the class. Additionally, I helped set up the class in general including the variables, setter, getter and constructor methods for the ATM. This worked also involved the writing of the constructCards and moneyAmountsSetUp functions. Once this was setup, I aided in error checking for edge cases and helped implement simple functions for checking if card numbers were valid. While conducting this work I also made minor changes to the Card class inside the Card.java file to ensure the logic run without errors.

Ricardo Akkari:

Adrian Lin:

I contributed to the project with the testing and fixing of the Card and ATM classes in addition to the development of reading and writing data from JSON files to the ATM class and vice versa. I revised the moneyAmountsSetUp() from a default ATM set-up, to extract the data from a JSON file and set up the ATM with the corresponding data. I also constructed both the saveCardInfo() and saveATMInfo(). Additionally, I wrote test cases for both Card and ATM classes, that predicted both normal behaviour and abnormal behaviour from both the user and application. While testing bugs would occur with errors such as precision, class cast exceptions, etc. that I solved either by refactoring a section of code or reformatting variables to the desired output.

Ryan Zangari:

I helped construct the ATM class in the ATM.java file. As well as the overall setup of the class including getter and setter methods and creating the attributes for the ATM I also helped write many of the methods in this class. Some of these included the setMoneyAmounts, deposit, withdraw, maintenance. Along with these methods, I also added to methods that needed edge cases to be considered such as the withdraw function. I also made some patches to the code when errors or bugs were found.

*Group contributions:*

Apart from completing larger components of the application like individual classes, JSON files and tests, all group members made small changes when necessary to fix any bugs and often would reflect on how our approach to the problem should be implemented with plenty of discussions leading to changes throughout the assignment. These discussions were crucial in determining the structure of our project.

*GitHub Collaboration*

We created a shared repository to which we all contributed using git commands on each of our local machines. Branches were created for some individual tasks, such as a Card\_branch and ATM\_branch which were used to keep the versions of different sections of the application separate from others. This allowed work on the ATM class to be done on the ATM\_branch and work on the Card class to be done on the Card\_branch hence, meaning ensuring that the rest of the code was untouched while implementing these changes. Thus, at the end of working on a branch it would then be merged back with the master branch.

A project board was used for individual tasks and to track the progress of the entire project. For example, separate tasks for classes, tests of individual classes, and the application interface were made so we could easily see what needed to be done.

There were minimal breaks or problems that got pushed to the repository, but we recorded any issues on Github that needed to be resolved and communicated whenever the build was broken to identify the cause of the issue as soon as possible.

From IntelliJ: Evidence of all group members’ contributions on GitHub, including the branches merged and integration of changes as the application progressed.

*Gradle Use*

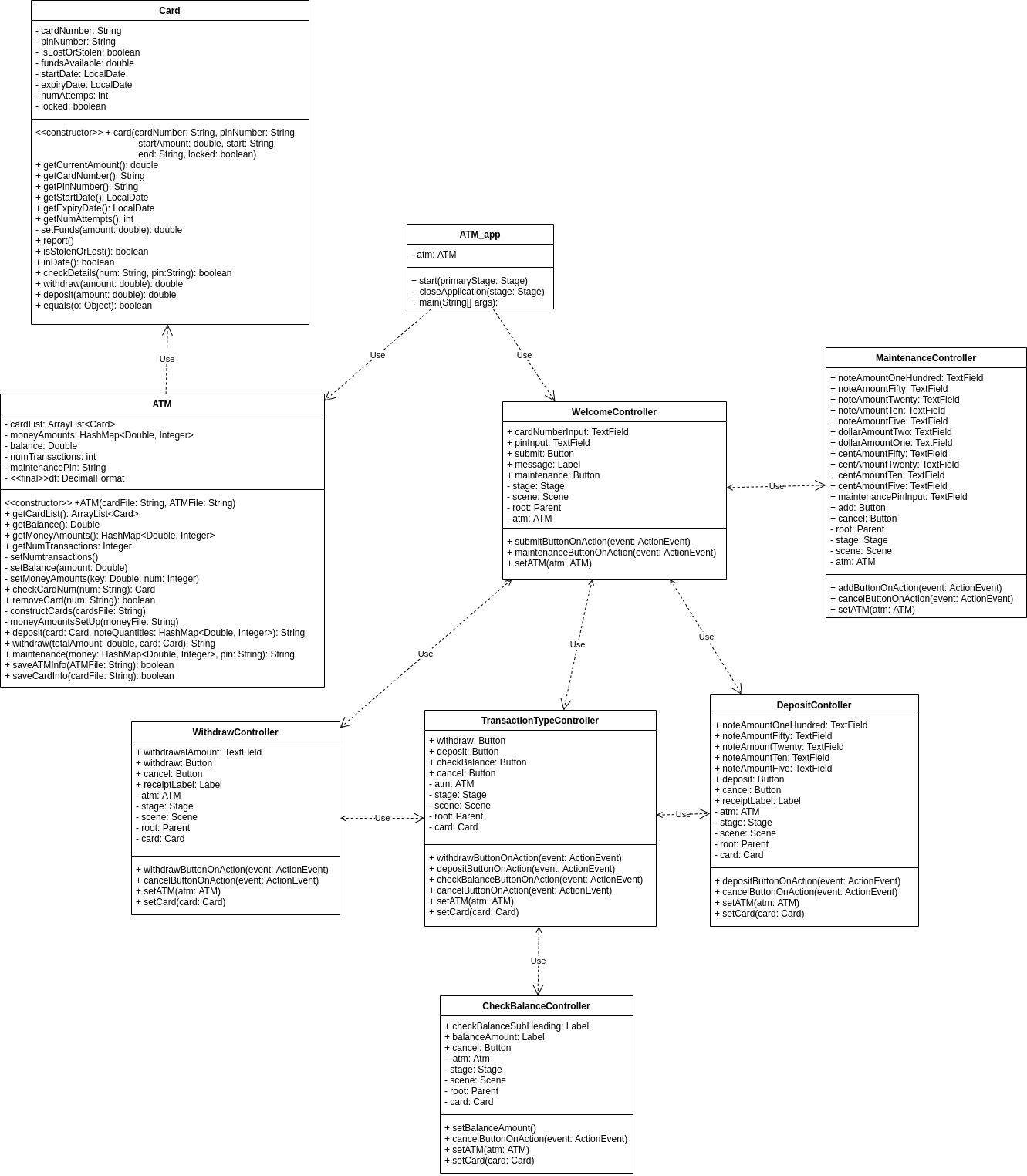
Gradle was used with the application and jacoco plugins in the build.gradle file, as well as the javafx plugin as that was the user interface used for the ATM application. Dependencies were specified for JUnit testing, and JSON which was used to create input ATM cards that could be tested.

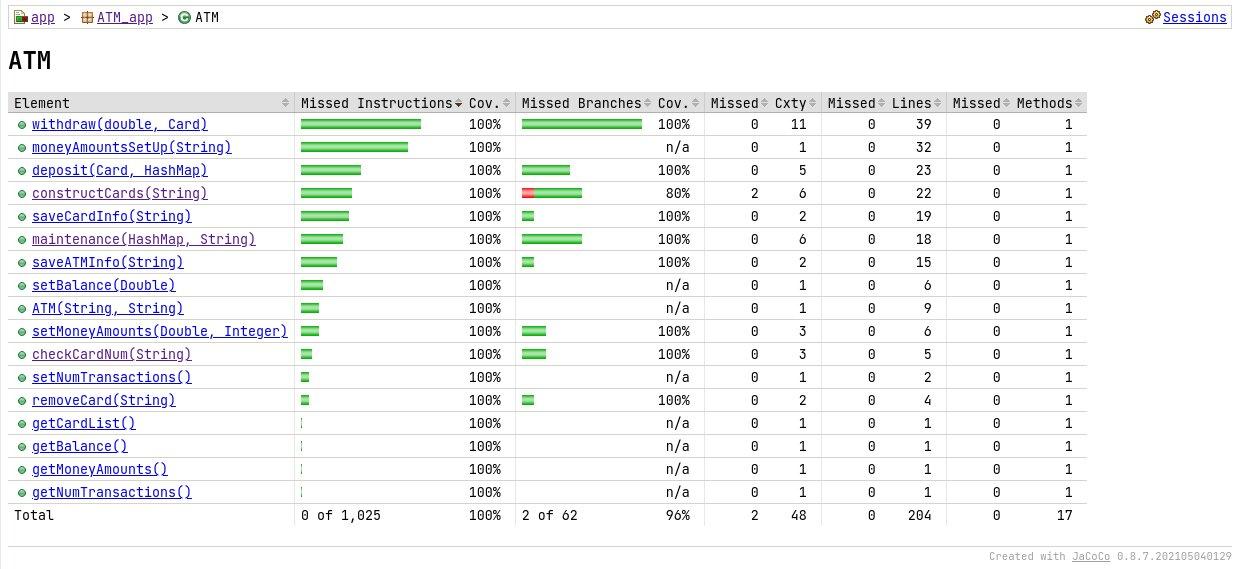
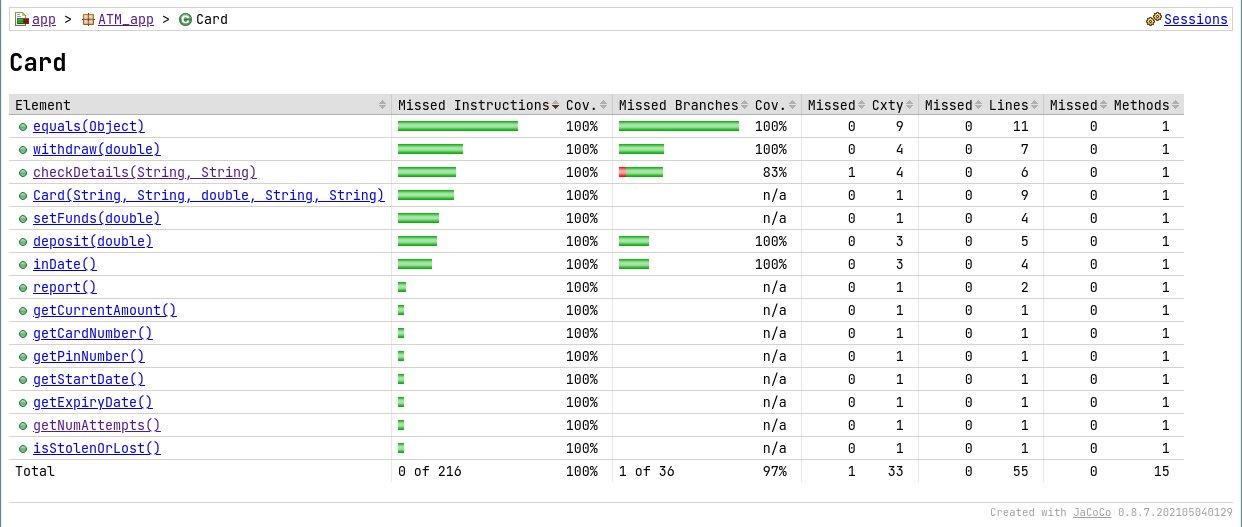
All group members used Gradle ‘build’ and ‘test’ commands regularly and notified one another when Jenkins detected a build failure.

Graphical user interface, text

Description automatically generatedText

Description automatically generatedExample (as of 26th of September) of the build output log and the test passed successfully.

Example of the application user interface running successfully upon running the ‘gradle run’ command.

*Code coverage* 

*CardTest.java*:

* setup(): BeforeEach method to create a Card instance ‘card’
* getCurrentAmountTest(): Checking a correct and incorrect value for the card’s balance after being created, for both a positive and negative starting balance on a card.
* getCardNumberTest(): Checking a correct and incorrect value for the card number after it was created – should be the same number provided to the Card constructor.
* getPinNumberTest(): Checking a correct and incorrect value for the pin number after it was created – should be the same number provided to the Card constructor.
* getStartDateTest(): Checking a correct and incorrect start date for the card after it was created – should be the string provided to the Card constructor.
* getExpiryDateTest(): Checking a correct and incorrect expiry date for the card after it was created – should be the string provided to the Card constructor.
* isStolenOrLostAndReportTest(): Tests that a card is valid when created, and invalid once it is reported.
* inDateTest(): Tested some example invalid cards with the inDate() function where the start and expiry dates were the wrong way around, as well as a valid card which had the correct dates.
* checkDetailsTest(): Tests the checkDetails() method with a card with both correct numbers (should be valid), card with only one correct number (should be invalid) and a card with two incorrect numbers (should be invalid). Also tested inputs of the incorrect length for the card number and pin, null inputs and empty string inputs – should be recognised as invalid details.
* loginAttemptsTest(): Tests that entering a card number will give the user 3 tries to enter the correct pin, and an incorrect value entered will decrease the number of tries left. Tests that once a card is blocked, entering information for a separate card will still work and will give the user 3 attempts for a different card number + pin combination.
* depositTest(): Tests that the deposit() method returns the correct value for a valid amount (should return current balance) and an invalid amount (i.e. a negative amount. to deposit - should return -1).
* withdrawTest(): Tests that the withdraw() method updates the card balance correctly for a valid amount (should return current balance) when withdrawing both notes and coins, and an invalid amount (i.e. a negative amount, or withdrawing more than is available - should return a negative number).
* getCurrentAmountTest(): tests if the current balance of the card is equal to correct amount after deposits and withdraws from the card.
* normalCardUsageTest(): tests a number of card functions like multiple withdraws and deposits (current balance should be updated correctly), attempting invalid withdrawals or deposits, and reporting the card (should no longer be usable).

*ATMTest.java*:

* setup(): a BeforeEach method to set up an ATM instance with some cards and money in it.
* getBalanceTest(): Checks that the ATM’s getBalance() is correctly output the expected balance.
* getNumTransTest(): Checks that the ATM’s getNumTransactions() method returns the correct transaction number from initialisation or after valid/invalid withdraws and deposits.
* Deposit(): Tests that the ATMs deposit() method changes both the ATMs and the cards balances to the correct amount, and produces the correct receipt.
* withdrawTest(): Tests that the ATMs withdraw() method works as normal and returns correct receipt and balance for both card and ATM is correct.
* maintenanceTest(): Tests that the ATM’s maintenance() method adds the correct amount of money to the ATM, the ATM’s balance is correctly updated and the money in the ATM is successfully saved in a file.
* compareCardTest(): compares the values of the cards with another card or no card with different or same number, pin, stolenStatus, start date, expiry date and funds to see if the cards are equal
* constructorTest(): test the setup of the ATM and cardList from the json file and test if they equal to the json files content with incorrect cardList and/or ATM information
* getCardListTest(): test an arraylist of cards thats extracted from the json card file and check the cards and cardList change with each valid and invalid interaction
* removeCardTest(): test a card from the arraylist cards with the same card number inputted and with non-existent card number
* saveATMInfoTest(): test the atm with another atm that has read the json file the original atm and another atm with incorrect ATM data
* o  saveCardInfoTest(): test the atm with another atm that has read the json file the original atm and another atm with incorrect Card data
* generalUsageTest(): tests the general functionalities of the ATM with incorporating every function from the ATM class such as multiple valid and invalid withdrawal, deposit, maintenance, saveCardInfo, and saveATMInfo