Technical Report for SOFT2412 Assignment 1 - XYZ Bank ATM

Development Team:

- **William Gan (wgan5014)**
- **Melissa Nguyen (mngu6648)**
- **Ian Chen (iche7333)**
- **Nafi Robayat (nrob6757)**
- **Ishan Zuam (izua3346)**

THE UNIVERSITY OF SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: SOFT2412

Assignment name: Tools for Agile Software Development

Tutorial time: W14_C3_Wed - 2pm **Tutor name:** Sudeshna

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. Ishan Zuaim	490625640	[✓ Yes] / No	[✓ Yes] / No	[illegible]
2. William Gan	500492989	[✓ Yes] / No	[✓ Yes] / No	[illegible]
3. Ian Chen	500508659	[✓ Yes] / No	[✓ Yes] / No	Ian Chen
4. Nafi Robayat	510549279	[✓ Yes] / No	[✓ Yes] / No	[illegible]
5. Melissa Nguyen	480224135	[✓ Yes] / No	[✓ Yes] / No	MNguyen
6.		Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

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

Contents Page

I - Individual, Group Contribution and Communication
Note: All members working in common areas of functionality reviewed each other's work via a pull request, and then merging.

E.g. person A and B work in area Z. B makes pull requests for changes to Z, to which A reviews and merges into the master branch

Ian	Ishan	Melissa	Nafi	William
Admin.java class AdminLoginScreen.java GUI design, logic AdminScreen.java GUI design, logic Tests for guiLogic, adminLogin Debug and optimize some of the interface	DepositScreen.java view, logic and design WithdrawScreen.java view Homescreen date checker logic Invalid Date Screen Navigational components of the GUI screens Tests for guiLogic/deposit Logic	Overall GUI design (creating and connecting nodes) Linking GUI to functionality + logic associated app, homeScreen, nodeCreator, optionScreen, receiptScreen, apologyScreen Tests for guiLogic	Overall project class design (object-oriented design) Implementing data loading and unloading in IO.java and IO testing Screens for the admin, cardsScreen, accountsScreen ,transactionScreen	Overall project workflow and logic design ATM.java, Card.java Screen for ATM having insufficient funds Tests for Card, Admin, Account, ATM, and Transaction classes Jenkins

Tools for team communication

- **Github**
  - Main platform for the project
  - Pull request & merging, for members who are working in common class/functionality.
  - Check history and changes across all members.
  - Report issues
- **Facebook messenger**
  - Organize meetings
  - Discuss issues and concerns
  - Clarify doubts
  - Updates on work progress
- **Zoom**
  - Group meetings

Group meetings

Meeting 1:

3/9/21 7:30pm-8:45pm

- Design the ATM with a UML diagram
- Decided to make the project with a GUI

Meeting 2:

8/9/21 3:20pm-4:00pm

- Divide work for everyone

Meeting 3:

9/9/21 7:30pm-8:00pm

- Further talk about the functionality in each classes
- How each class will interact with each other

Meeting 4:

15/9/21 4:00pm-5:00pm

- Basic design of the GUI
- Format and framework for GUI classes

Meeting 5:

16/9/21 7:40pm-8:20pm

- How the app is going to work
- Set deadline on Saturday (18/9/21) for everyone

Meeting 6:

18/9/21 7:40pm-9:15pm

- Resolved merge conflicts
- Finalized some of the GUI components

Meeting 7:

20/9/21 8:00pm-8:50pm

- Solving GUI issues, layout, and bugs (handle user input)
- Split up work for testing
- Split up work for report

Meeting 8:

22/9/21 2:00pm-4:00pm

- Made methods cleaner (better logic)
- Moved all methods in the GUI into a class "guiLogic.java" for testing

Meeting 9:

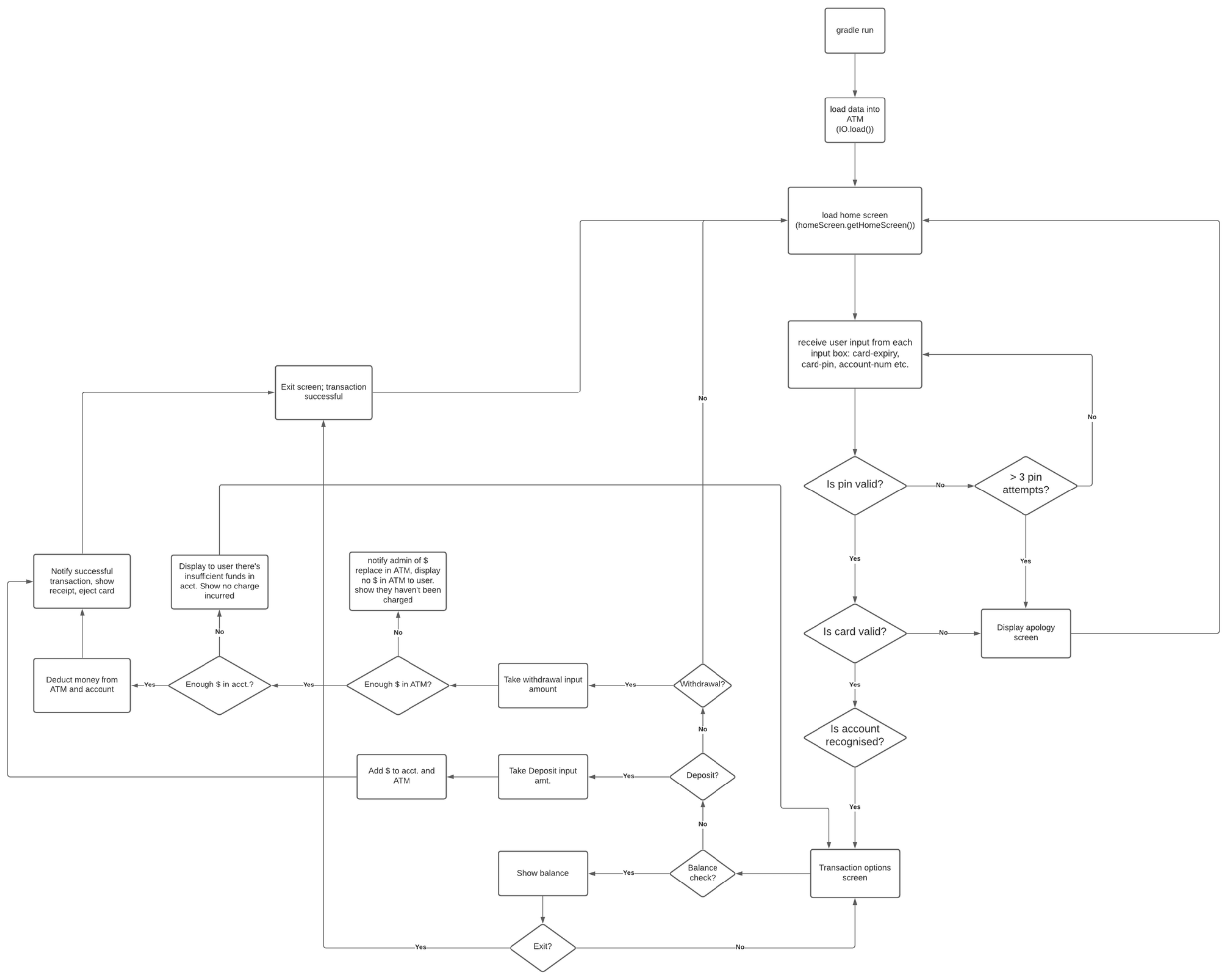
23/9/21 8:00pm-9:25pm

- Finalized the project

II - Workflow of the software application

About

- This program simulates the functionalities of an ATM software application. The overall functionality of the program is displayed below.


gradle run
load data into ATM (IO.load())
load home screen (homeScreen.getHomeScreen())
receive user input from each input box: card-expiry, card-pin, account-num etc.
Is pin valid?
No
> 3 pin attempts?
No
Yes
Yes
Display apology screen
Is card valid?
No
Yes
Is account recognised?
Yes
Transaction options screen
Balance check?
Yes
Show balance
No
Deposit?
Yes
Take Deposit input amt.
Add $ to acct. and ATM
No
Withdrawal?
No
Yes
Take withdrawal input amount
Enough $ in ATM?
No
notify admin of $ replace in ATM, display no $ in ATM to user. show they haven't been charged
Yes
Enough $ in acct.?
No
Display to user there's insufficient funds in acct. Show no charge incurred
Yes
Deduct money from ATM and account
Notify successful transaction, show receipt, eject card
Exit screen; transaction successful
Exit?
Yes
No

- The README file can be found in the github repository

III - Github Collaboration
Github was used for everyone to push their local changes on the projecto a central remote repository, with changes merged from each developer's local branch into the master branch. This tracked version history of the canonical version, and each developer's independent versions of the project.

Besides pushing changes, pull-requests were generated via Github to prompt each developer to review the changes which were added to the canonical version. Changes which were reviewed before being put into the master branch for the project ensured deliverables would work efficiently and correctly, in addition to limiting the number of merge conflicts. This prevented any "doubling-back" which would have otherwise slowed the development process.

Additionally, Github was integrated with Jenkins and Gradle (see below) to create a CI/CD pipeline. With every push made to *any* branch, Gradle 'clean build test' tasks were run to ensure all changes still maintained correct functionality.

IV - Use of Gradle
Gradle was used in the CI/CD pipeline, wherein (mentioned above) 'clean build test' tasks were used to check if changes still ensured functionality.

Code-coverage and JUnit testing reports were generated via Gradle to ensure a satisfactory number of use-cases in the code were accounted for. This was achieved using the 'jacocoTestReport' task, with the 'jacoco' dependency (as placed in the build.gradle file).

Furthermore, the javafx plugin enlisted in the 'build.gradle' made the javafx package 'known' to Gradle during the 'gradle build' task, which automated code compilation, logic checks, and execution.

Finally, the 'classDirectories' package was included as part of the jacocoTestReport execution to exclude the 'javaATM.GUI' package from the source code. This was done in the interests of only displaying the back-end functionality logic test coverage, disregarding the front which maintained mostly graphical display logic
.

V - JUnit testing and Code Coverage

Testing the quality of the code was done through JUnit testing. JUnit testing allows us to check that our program is designed correctly according to the specifications provided. This is done through assertion tests which are manually written and check if the output of methods and functions correspond with the expected output.
Our final testing files include:

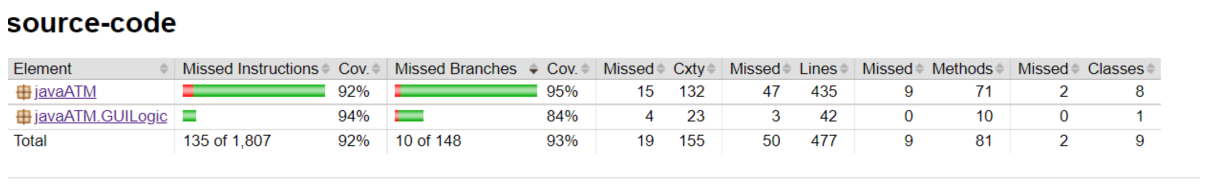
- **ATMTest.java**
  - ATMTest contains 33 test methods which test the following:
    - Withdrawing and depositing to the ATM returns the correct output
    - Adding or removing accounts/cards/transactions returns the correct output
- **AccountTest.java**
  - AccountTest contains 10 test methods which test the following:
    - Adding or removing cards are correctly done
    - Getting account attributes such as accountNumber and associated cards works as intended
    - Depositing or withdrawing money from the account works as intended
- **AdminTest.java**
  - AdminTest contains 8 test methods which test the following:
    - If the getter Methods of AdminTest return the correct values for each of its attributes.
- **AppTest.java**
  - AppTest is a dummy class that does nothing as the App class only exists to initialize the GUI.
- **CardTest.java**
  - CardTest contains 30 test methods which test the following:
    - If the getter methods return the correct values for each of its attributes
    - If that status is correctly updated for expired, not activated and other statuses
    - If withdrawing or depositing to the card systematically alters the balance of the associated account
- **IOTest.java**
  - IOTest contains 2 test methods which test the following:
    - Reading from test accounts.csv, cards.csv, ATMState.csv and transactions.csv loads in correct and valid objects and ignores bad data lines
    - Adding new account/card/transaction objects to the ATM is correctly saved to the corresponding files
- **TransactionTest.java**
  - TransactionTest contains 4 test methods which test the following:
    - If the getter methods return the correct values for each of its attributes

- **adminLoginScreenTest.java**

- adminLoginScreenTest contains 8 test methods which test the following:
    - Whether the correct inputs are given for an valid admin to open the adminScreen
- **depositScreenTest.java**
  - depositScreenTest contains 5 test methods which test the following:
    - Depositing money into an account reflects the input being stored correctly
    - Adding money to the deposit correctly increments the final value
    - Deposit size array which holds the money history is appropriately appended
    - Undoing latest cash deposit accurately returns the money which was put in
    - Resetting the deposit, results in the final amount correctly being back to 0, which the history erased
- **homeScreenTest.java**
  - HomeScreenTest contains 2 tests methods which test the following:
    - Whether a card object provided is a valid object i.e is not expired, activated, not reported stolen or lost.
    - Whether the pin number given corresponds with the card given
    - If the date is a valid date

To evaluate the effectiveness of test cases, code coverage is used to determine what percentage of the code is being tested. Missed Instructions indicate the percentage of lines of code never touched by the test code whereas missed branches indicate the percentage of decision trees within the code never reached. In our testing we achieved a coverage of 92% of instructions covered and 93% of branches covered.

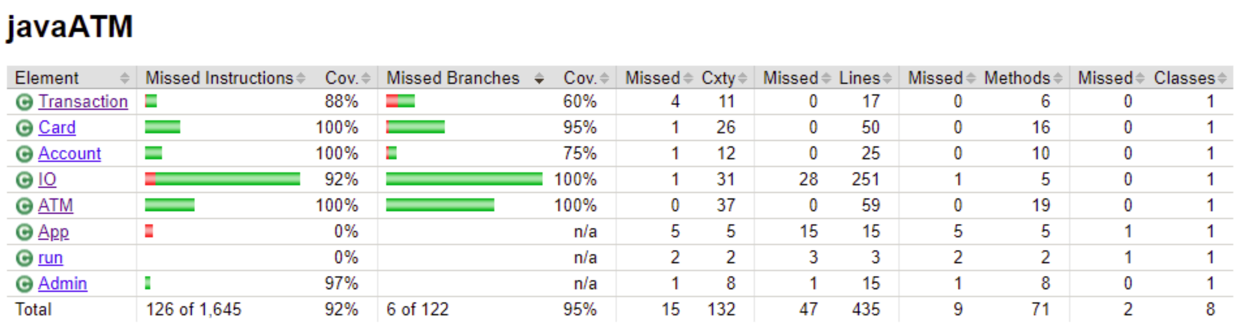
Screenshots



source-code

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
javaATM		92%		95%	15	132	47	435	9	71	2	8
javaATM.GUILogic		94%		84%	4	23	3	42	0	10	0	1
Total	135 of 1,807	92%	10 of 148	93%	19	155	50	477	9	81	2	9

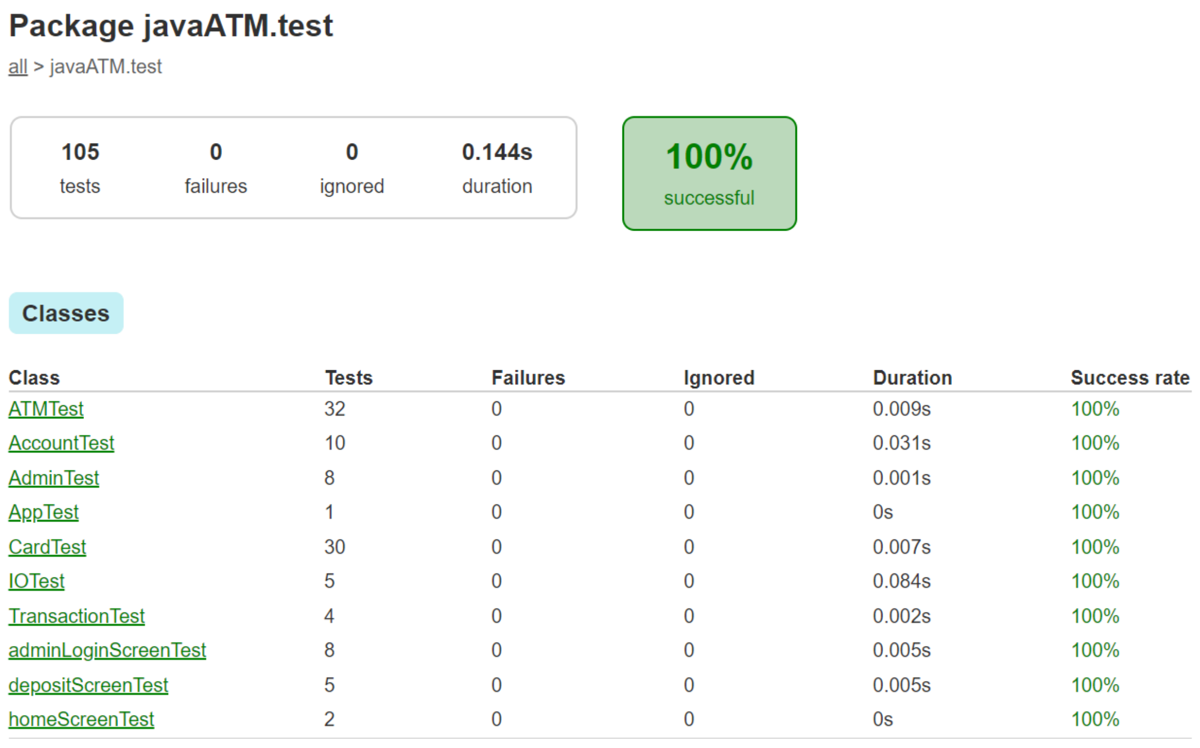
Figure 1. Overall Jacoco Test coverage of project



javaATM

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Transaction		88%		60%	4	11	0	17	0	6	0	1
Card		100%		95%	1	26	0	50	0	16	0	1
Account		100%		75%	1	12	0	25	0	10	0	1
IO		92%		100%	1	31	28	251	1	5	0	1
ATM		100%		100%	0	37	0	59	0	19	0	1
App		0%		n/a	5	5	15	15	5	5	1	1
run		0%		n/a	2	2	3	3	2	2	1	1
Admin		97%		n/a	1	8	1	15	1	8	0	1
Total	126 of 1,645	92%	6 of 122	95%	15	132	47	435	9	71	2	8

Figure 2. Test Coverage of JavaATM package



Package javaATM.test

all > javaATM.test

105	0	0	0.144s	100%
tests	failures	ignored	duration	successful

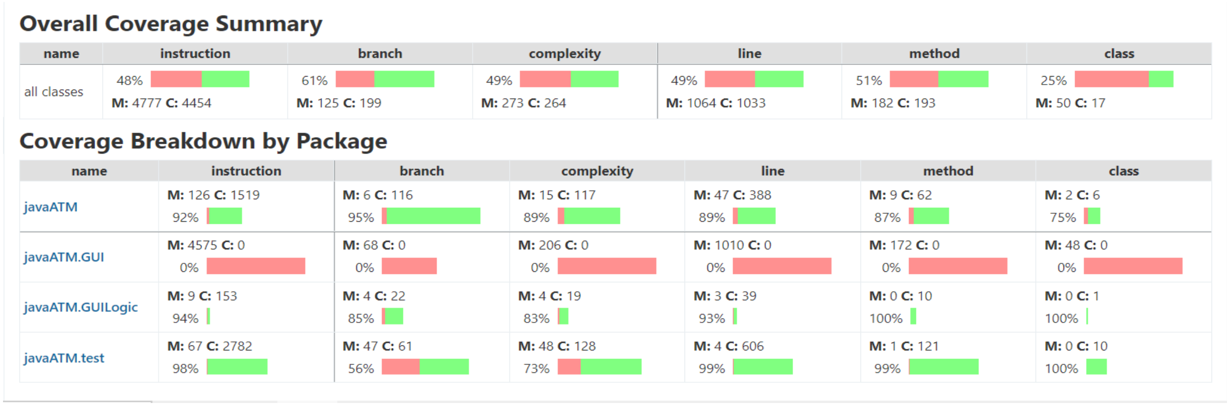
Classes

Class	Tests	Failures	Ignored	Duration	Success rate
ATMTest	32	0	0	0.009s	100%
AccountTest	10	0	0	0.031s	100%
AdminTest	8	0	0	0.001s	100%
AppTest	1	0	0	0s	100%
CardTest	30	0	0	0.007s	100%
IOTest	5	0	0	0.084s	100%
TransactionTest	4	0	0	0.002s	100%
adminLoginScreenTest	8	0	0	0.005s	100%
depositScreenTest	5	0	0	0.005s	100%
homeScreenTest	2	0	0	0s	100%

Figure 3. Test report of all test classes

Jenkins JaCoCo Coverage Report for back-end classes

Overall package coverage



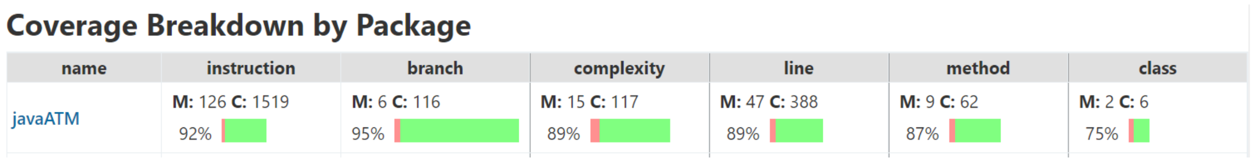
Overall Coverage Summary

name	instruction	branch	complexity	line	method	class
all classes	48% M: 4777 C: 4454	61% M: 125 C: 199	49% M: 273 C: 264	49% M: 1064 C: 1033	51% M: 182 C: 193	25% M: 50 C: 17

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
javaATM	M: 126 C: 1519 92%	M: 6 C: 116 95%	M: 15 C: 117 89%	M: 47 C: 388 89%	M: 9 C: 62 87%	M: 2 C: 6 75%
javaATM.GUI	M: 4575 C: 0 0%	M: 68 C: 0 0%	M: 206 C: 0 0%	M: 1010 C: 0 0%	M: 172 C: 0 0%	M: 48 C: 0 0%
javaATM.GUILogic	M: 9 C: 153 94%	M: 4 C: 22 85%	M: 4 C: 19 83%	M: 3 C: 39 93%	M: 0 C: 10 100%	M: 0 C: 1 100%
javaATM.test	M: 67 C: 2782 98%	M: 47 C: 61 56%	M: 48 C: 128 73%	M: 4 C: 606 99%	M: 1 C: 121 99%	M: 0 C: 10 100%

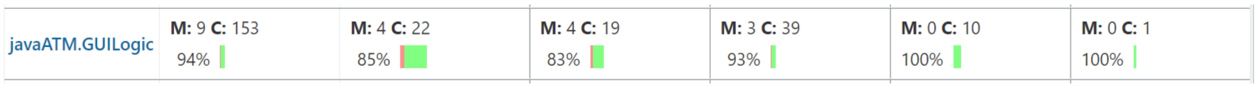
`javaATM` package



Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
javaATM	M: 126 C: 1519 92%	M: 6 C: 116 95%	M: 15 C: 117 89%	M: 47 C: 388 89%	M: 9 C: 62 87%	M: 2 C: 6 75%

`javaATM.GUI` logic



javaATM.GUILogic	M: 9 C: 153 94%	M: 4 C: 22 85%	M: 4 C: 19 83%	M: 3 C: 39 93%	M: 0 C: 10 100%	M: 0 C: 1 100%

UML Class Diagram

https://lucid.app/publicSegments/view/c1207ad7-9aa3-4c0a-83ce-0edaf57b6a5a/image.png