Blue Minev

22808922

CI401 ASSESMENT REPORT

Contents

[INTRODUCTION 1](#_Toc135299819)

[DESIGN AND DEVELOPMENT 1](#_Toc135299820)

[CRITICAL REVIEW 1](#_Toc135299821)

[CONCLUSION 1](#_Toc135299822)

[ESTIMATED GRADE 1](#_Toc135299823)

[REFRENCES 1](#_Toc135299824)

# INTRODUCTION

For my Programming Project I chose to expand on the ATM code. After reviewing this code in tutorials and fixing the issue. I began to consider what changes I wanted to make.

The first adjustment I chose to do was make changes to the CSS of the ATM. I decided to do this to enhance the user experience and make the UI appear more modern and sleeker. This would be done by changing aspects of the buttons and adjusting colours and fonts. I also wanted to ensure that my UI was as accessible to those with disabilities as I could make it.

An addition I made was to add a button to change the pin. Being able to change the pin is a common feature of ATMs so I chose to include this.

A feature I added was a statement button. The goal with this button is to print the last 5 transactions completed. Most ATMs and banking software have this feature as it is good practice to include it. Due to these reasons, I chose to add this change.

The final major change I decided to include was the ability to have an overdraft on an account. This was to add extra functionality to the app and widen the scope of options available to the user.

# DESIGN AND DEVELOPMENT

## design

One major point of the design for this project is the look of the ATM. Before deciding anything, I decided to interview some hypothetical end users to get some requirements for my end software. During this interview they stated that they would prefer the software to have the look of a regular ATM. Because of this I chose to keep the basic layout of the ATM. This was a decision was also made due to the fact that the IDE I chose to use did not have an inbuilt scene builder, meaning that making major changes to the layout would be incredibly difficult.

After this interview I decided on a colour scheme for the ATM. I decided to go for muted greys and a muted blue with large white writing for the button. During one interview with an end user, I discovered that the colour scheme I had proposed was good for those with a high sensitivity to colour and contrast. Due to this information I decided to research more into inclusive design and making design choices to help people with disabilities use my software.

During my research I discovered that my original plan for white text on a light blue background may be difficult for dyslexic people to read. Therefore, I decided to change my button text to a dark grey, increasing the contrast making it easier to read. I also found out that many people with dyslexia fin did easier to read text on a different coloured background, this solidified my choice to have my buttons be a different colour than white.

I also considered those with physical disabilities when choosing the design of my ATM. I chose to make the buttons bigger than in the original code, this means that people who have limited mobility can press these buttons easier, for example those who use a joystick or eye trackers, as there is more space that they could click.

### Final ATM design

A screenshot of a computer

Description automatically generated with medium confidence

## development

### IDE Issues

When I began the development of my software, I chose to use my preferred choice of Visual Studio Code for my integrated development environment (IDE). I struggled to install the JavaFx plugin into VSCode and could not find many tutorials. Once I had found an Up-to-date tutorial, I discovered I would have to install multiple plugins for VSCode to understand my java code. Finally, I got the IDE to compile and interpret my code and output the GUI. However, the IDE was very volatile and would often stop working for no apparent reason. This was causing a great impact to my development timeline, so I decided to switch to BlueJ as it was much easier to use JavaFX and did not require any setup.

### JavaFX CSS Issues

Once I had my code running consistently, I began to make aesthetic changes to the ATM. The first change I tried to make was to the buttons.

A screenshot of a computer

Description automatically generated with medium confidence

Above shows how the buttons originally looked. The goal was to make the button the width of the columns and make them appear as CSS buttons do not default. I tried many different lines of code, and none would work. After finding some documentation for CSS with javaFx and brushing up on certain CSS functions, I was able to change the colour of the buttons.

A screenshot of a computer

Description automatically generated

This was using -fx-background-color.

The next step was wo remove the rounded corners. I tried many iterations of code trying to get sharp corners, but I could not work out how to do it. I decided to try and get a border on the buttons to show up. After much trial and error, I discovered if I set the border colour to the same as the background, it would sharpen the corners.

A screenshot of a computer

Description automatically generated

At this point I had a much stronger grasp of using CSS in this context and was able to make other changes such as changing the font, background colours and adding border colours to the text boxes to make them fit in better.

The final change I made was making the buttons wider. I struggled with this as I could not work out what the current function to call would be. However, when going through the CSS provided, I saw the line -fx-pref-tile-width so I decided to try -fx-pref-width to see if that would work, which it did. After I got this change to work, the CSS development was complete. Shown below is the final output and the CSS code.

A screenshot of a computer

Description automatically generated with medium confidence

#Layout {

-fx-grid-lines-visible: false;

-fx-hgap: 5;

-fx-vgap: 5;

-fx-padding: 10;

}

#Buttons {

-fx-pref-columns: 6;

-fx-pref-tile-width: 75;

-fx-hgap: 5;

-fx-vgap: 5;

}

.button {

-fx-font: bold 15pt "Roboto";

-fx-background-color: #778DA9;

-fx-border-color: #778DA9;

-fx-pref-width: 200;

-fx-text-fill: #2C3539;

-fx-cursor: pointer;

}

#message{

-fx-border-color: #778DA9;

-fx-font: 10pt "Roboto";

}

#reply{

-fx-border-color: #778DA9;

-fx-font: 10pt "Roboto";

}

#title{

-fx-font: bold 15pt "Roboto";

}

.root{

-fx-font: 10pt "Roboto";

-fx-background-color: #E0E1DD;

}

## Adding the new pin button

When adding the new pin button, I started by adding the physical button to the labels [][] array in the View file. I then went to the Controller file and added a case to the switch case statement, which would call the model.processNewPin() function. I then went to the Model file and created the processNewPin() function, this function would check whether the input was valid, if so, it would call the bank.changePin(int number) function and change display2 to let the user know the pin was successfully changed. If the input was not valid, it would change the value display2 to let the user know and call the display to update. The next step was to go to the Bank file and crate changePin(int number). This function calls account.newPin (int number). The final step was to create the function newPin() in BankAccount. This function takes the number inputted by the user and sets accPasswd to it and returns the accPasswd.

A picture containing text, font, screenshot

Description automatically generatedView.java

A screenshot of a computer

Description automatically generated with medium confidence

Controller.java

A screenshot of a computer program

Description automatically generated with low confidence

Model.java

A picture containing text, screenshot, font, line

Description automatically generated

Bank.java

A picture containing text, screenshot, font, line

Description automatically generated

BankAccount.java

## adding the statement button

When adding the statement button, I started by adding the physical button to the labels [][] array in the View file. I then went to the Controller file and added a case to the switch case statement, which would call the model.processStatement() function. I then went to the Model file and created the processStatement() function, this function would assign the return it would get from bank.getStatement() to the String Statement. It would then assign this string to display 2 and call the display to update. The next step was to go to the Bank file and crate getStatement(int number). This function calls account.statement (int number). The final step was to add information to the statement and create the function statement() in BankAccount. In the original design this was done by appending items to a string. This used StringBuilder and the .append function. I chose to append information when a deposit or withdrawal took place. This design worked, however with my interviews with end users, the majority of them stated that they would expect only the 5 most recent transactions to appear.

My first attempt to limit the statement to the last 5 transactions was using a Stack. The idea was to pop of the last item in the stack and push on the newest one. However, during testing, I realised that I was only popping the most recent transaction and not popping the oldest transaction.

The next and final idea was to use a linked list. This was due to the function removeFirst(). This also meant I could easily append and introduction to the statement and return the value finalStatement to be displayed.



Creating the required variables

A screenshot of a computer

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with low confidence

The logic in the withdraw and deposit functions to check how long the linked list is, remove a node of the list if needed, adjust the count variable and then add to the statement.

A picture containing text, screenshot, font, line

Description automatically generated

The statement function which returns the statement

A screenshot of a computer code

Description automatically generated with low confidence

Bank.getStatement ()

A screenshot of a computer

Description automatically generated with medium confidence

Model.proccessStatement()

A screenshot of a computer

Description automatically generated with medium confidence

Controller.java

## ADDING AN OVERDRAFT

When adding an overdraft, I started by adding a Boolean value overdraft to the BankAccount file. I set it to set automatically to false. I then added the overdraft variable to all the functions in BankAccount that would need it, such as BankAccount(). I then went to the Bank file and added the overdraft variable to all the functions there that required it such as makeBankAccount() and both versions of addBankAccount()The next step was to go to back to the BankAccount file and add the logic to allow users to go into negative values. This was just simply adding to the logic in the if statement. The final step was to add the value to the test classes in Main. I set one to true and one to false so I could test both user cases.

I then decided to add a limit to the overdraft. I started the same as adding an overdraft, creating the integer variable and adding it to all the functions that would require it. When it came to adding the logic, it too me a bit longer to do as I had to mess around with the direction of the greater than/less than sign before I got it to work successfully.

A screenshot of a computer code

Description automatically generated with low confidence

The withdraw() function in BankAccount.

A screenshot of a computer code

Description automatically generated with medium confidence

Adding the variables and putting them in the required function in BankAccount

A screenshot of a computer program

Description automatically generated with low confidence

Adding the variables to the functions required in Bank.

## testing

The majority of the testing I did was white box testing, where I tested every function at the end of my development phase, along with as I went along, testing the current function I was working on, including as many edge cases as I could.

During this testing I discovered that during login if the user inputted the wrong account number or password, they were not being sent back to the initial state as they should, they were stuck on the ‘enter your password’ screen and no button input was changing anything, except for the log out button.

I started by adding more debug statements to the parts of the code that I thought were the issue. Through this, I found that the account number was staying when I believed it should have been reset. Due to this, I tried many ways to set this value to 0 but none were working. I think checked the command line and found that there was an error being sent.

This error was a ‘NullPointerException’ after reading the error and trying to understand what was causing this, I realised that I was looking in the wrong file. Originally, I was looking in the Model file at the processEnter() function. In this function, it calls bank.login() and this was where the issue was.

I returned to my starting strategy of adding debug statements to the function to try and understand what was going wrong. I decided to put a debug statement after the for loop to see if this loop was executing correctly. During this I realised that the for loop was executing 3 times when there were only 2 bank accounts. This was due to the fact that it was looping through the value of maxAccounts. However, once it reached the third iteration, the values in b.accNumber and b.accPassword were null, this caused the code to throw the ‘NullPointerException’ error.

I chose to solve this issue through exception handling. Using a try{} catch{} statement I was able to allow the code to run how it is expected to.

A screenshot of a computer

Description automatically generated with medium confidence

After white box testing, I did black box testing with the end user I interviewed. I gave him a list of instructions to complete and monitored his interaction. During this, we discovered a bug with the new pin button where if you pressed it without inputting a number first, it would set the pin to 0.

My first approach to fixing this was to add extra logic to the if statement in changePin(), located in the bank file. A screenshot of a computer code

Description automatically generated with low confidence

This, however, caused the function to return -1, though the pin is not actually changed to -1 I chose to move the logic to the proccessNewPin() function in the Model file.

A screenshot of a computer

Description automatically generated with medium confidence

I did this by adding an if else statement that checks that the number is not 0. If the number is 0, it changes the display to remind the user to enter a valid pin before pressing the button.

I then conducted white box testing on this specific button again to ensure that there were no more issues.

# CRITICAL REVIEW

When reviewing my code, I chose to review it in the style that would be typical in a peer code review. To do this I gave myself a checklist as to what ‘good code’ would be. That checklist included:

* Is the code formatted correctly?
  + Is the code properly aligned?
  + Are there any unneeded whitespaces.
  + Is it easy to see where functions and loops start and end?
* Is the code maintainable?
  + Is the code readable and easily understood?
  + If the code is difficult to understand are there comments to help others understand the code
* Is the code usable?
  + Is the output of the code easy to understand for the general population?

In review of my code, I would say that it passes the first check point, BlueJ has an inbuilt auto-layout that I used as well as the colour system, meant that I ensured my code was formatted correctly.

I would say my code also passes the second checkpoint. All variable names I chose make sense and the code is easy to read. I also ensured that all of my function had a corresponding Javadoc comment. And for variables I struggled to understand firstly I also added Javadoc comments for those.

For the final checkpoint, I personally could not decide this one, so I had one of my end users partake in black box testing to see if the code was easy to understand. He was very able to complete the required tasks proving that the code is useable.

To conclude this section of the critical review, I would argue that my code could be seen as ‘good code’.

After this review I also looked at the code in the eyes of the developer and considered what I believed was good about my code.

I believe that I have added a good amount of functionality to the app, such as the new pin button and overdrafts. This added functionality means that in a real-world scenario would not have to leave the software to do common things like getting a statement. Thus, increasing the time spent on the application.

I also believe that my code and software has a high awareness of users and accessibility. This is due to my interviews with end users and my research into inclusivity in UI design. By making my software more accessible I have increased the scope of users as well as improved the experience of the original user demographic.

Due to my end user interviews I believe that I also had a good understanding of what features are common in ATMs widely used. I believe that this increased the quality of the end software as I was able to decide which features would be the most used and organise which features, I would work on.

I also considered what I think I could have improved about my code.

This included expanding the overdraft feature, I would have liked to add a button to make changes to the account such as adding an overdraft or editing the limit however, due to my issues in the beginning of the project I had to sacrifice this to make a fully functional application.

Another feature I would have liked to explore more was the CSS and the model view. I would have investigated more changes I could have made with the CSS to make a more enjoyable experience for the user as well as increasing the accessibility, such as adding a large font feature or the ability to change the font to a more dyslexia friendly design.

A feature I would have added if I had time was a receipt function. This function would have printed a log of all transactions to a .txt file for the user to keep.

# CONCLUSION

In conclusion, I have learnt a lot during this project, mainly from my major issues. The main take away message is to familiarise myself with my chosen IDE before taking on a major project and to not have loyalty to one, when others may have features better suited to the criteria.

I have learnt a lot about java development, such as: Javadoc, exception handling and event driven programming. I have also learnt how to find information about plugins that are not widely documented.

Overall, I have greatly improved my problem-solving skills as well as my general understanding of java and javaFx.

# ESTIMATED GRADE

Report: a

Development: b

# REFERENCES

British Dyslexia Association (2018). *Dyslexia friendly style guide*. [online] British Dyslexia Association. Available at: https://www.bdadyslexia.org.uk/advice/employers/creating-a-dyslexia-friendly-workplace/dyslexia-friendly-style-guide [Accessed 19 May 2023].

GeeksforGeeks. (2018a). *Stack capacity() method in Java with Example*. [online] Available at: https://www.geeksforgeeks.org/stack-capacity-method-in-java-with-example/ [Accessed 19 May 2023].

GeeksforGeeks. (2018b). *Stack pop() Method in Java*. [online] Available at: https://www.geeksforgeeks.org/stack-pop-method-in-java/ [Accessed 19 May 2023].

KZedUK (2020). *A PSA on ‘Dyslexia Friendly Fonts’ from the /r/dyslexia mod team*. [online] Reddit. Available at: https://www.reddit.com/r/Dyslexia/comments/jtgovc/a\_psa\_on\_dyslexia\_friendly\_fonts\_from\_the/ [Accessed 19 May 2023].

Payne, S. (2020). *Accessible Design for Users With Disabilities*. [online] Boldist. Available at: https://boldist.co/usability/accessible-design-for-users-with-disabilities/ [Accessed 19 May 2023].

Reddy Gutha, S. (2015). *Code Review Checklist – To Perform Effective Code Reviews*. [online] Evoke Technologies. Available at: https://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-code-reviews/ [Accessed 19 May 2023].

Rello, L. and Baeza-Yates, R. (2013). *Good Fonts for Dyslexia*. [online] Available at: https://dyslexiahelp.umich.edu/sites/default/files/good\_fonts\_for\_dyslexia\_study.pdf [Accessed 19 May 2023].

Schwarz, D. (2016). *5 Ways to Offer a Better UX to Disabled Users*. [online] SitePoint. Available at: https://www.sitepoint.com/5-ways-offer-better-ux-disabled-users/ [Accessed 19 May 2023].

Stack Overflow. (2013). *FIFO class in Java*. [online] Available at: https://stackoverflow.com/questions/9580457/fifo-class-in-java [Accessed 19 May 2023].

Stack Overflow. (2015). *In Java, how to append a string more efficiently?* [online] Available at: https://stackoverflow.com/questions/12899953/in-java-how-to-append-a-string-more-efficiently [Accessed 19 May 2023].