

Shape

Description automatically generated with medium confidence

Coursework – Family Tree Report

COMP08034 – Structures & Algorithms

B00418245 - Brandon Robson

2022

Contents

[Note 2](#_Toc99399177)

[Introduction 2](#_Toc99399178)

[Step 1 3](#_Toc99399179)

[Brief 3](#_Toc99399180)

[Implementation 3](#_Toc99399181)

[Marking Scheme 4](#_Toc99399182)

[Classes and Methods 5](#_Toc99399183)

[Code 6](#_Toc99399184)

[Difficulties 10](#_Toc99399185)

[Step 2 11](#_Toc99399186)

[Brief 11](#_Toc99399187)

[Implementation 11](#_Toc99399188)

[Marking Scheme 12](#_Toc99399189)

[Classes and Methods 13](#_Toc99399190)

[Code 13](#_Toc99399191)

[Difficulties 18](#_Toc99399192)

[Testing 19](#_Toc99399193)

[Github 21](#_Toc99399194)

# Note

This project is intended for the IntelliJ IDE. To run on IntelliJ: Add Configuation -> Add New Configuration (Alt + Ins) -> Application -> Change main class to FamilyTreeTest -> OK.

How it works:

For any given command my tree acts based on the ‘current’ family member that is set. To change the ‘current’ family member simply use the ‘Search By Name’ or ‘Search By Identifier’.

# Introduction

For my coursework I have been assigned the task of creating a family tree application that will allow for the addition of a prime ancestor, a partner, children and their partners etc. To do this I will implement a binary search tree data structure.

I chose to develop this coursework on my own, so everything created was made by myself. I also made use of a GitHub repository to store the work each step of the way.

This repository can be found at <https://github.com/brandonrobson/Coursework>

Throughout my report I will include the basic requirements for each of the 2 steps, a walkthrough for each part of the marking scheme complete with screenshots, as well as my testing and GitHub evidence at the end.

# Step 1

## Brief

This step involves the initial creation of the tree structure. I was given a rough idea of what my tree should look like.

Diagram

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

The basic functions include a constructor, the ability to add a child, and the ability to add a partner.

## Implementation

In my implementation of the coursework, I made a few adaptations. For example, each node has an ancestor, and when displayed, each child will include their parent and their partner.

The first step in my implementation was to create the interface. This was made by simply having a switch statement enclosed within a do-while loop.

Next I made the tree itself, I had already created a tree structure multiple times previously in the course, so this wasn’t difficult.

Finally I added the family tree related names, exceptions, and the rest of the functionality.

### Marking Scheme

|  |  |  |
| --- | --- | --- |
| Interface |  | The simple interface I created using a switch and a do-while |
| Setup partner and ancestor |  | When the program first starts, it takes in the name of the ancestor and their partner. Handled by the constructor. |
| Add each child  Unique names, not case sensitive, and uses exception handling. |  | Each time a child is added, the name is added to a list which is then checked, regardless of case, to make sure each name’s unique. If not, an exception is thrown. |
| Display via Ancestor  With partner, children, no children message. |  | Depending on whether the couple have children or not, a different output is displayed. |
| Display via Partner  With partner, children, no children message. |  | Similar to above but from the point of view of the partner. |
| Quit message |  | When quitting, a message is displayed. |
| Invalid Option |  | When an invalid numeral command is detected, a message is displayed. |

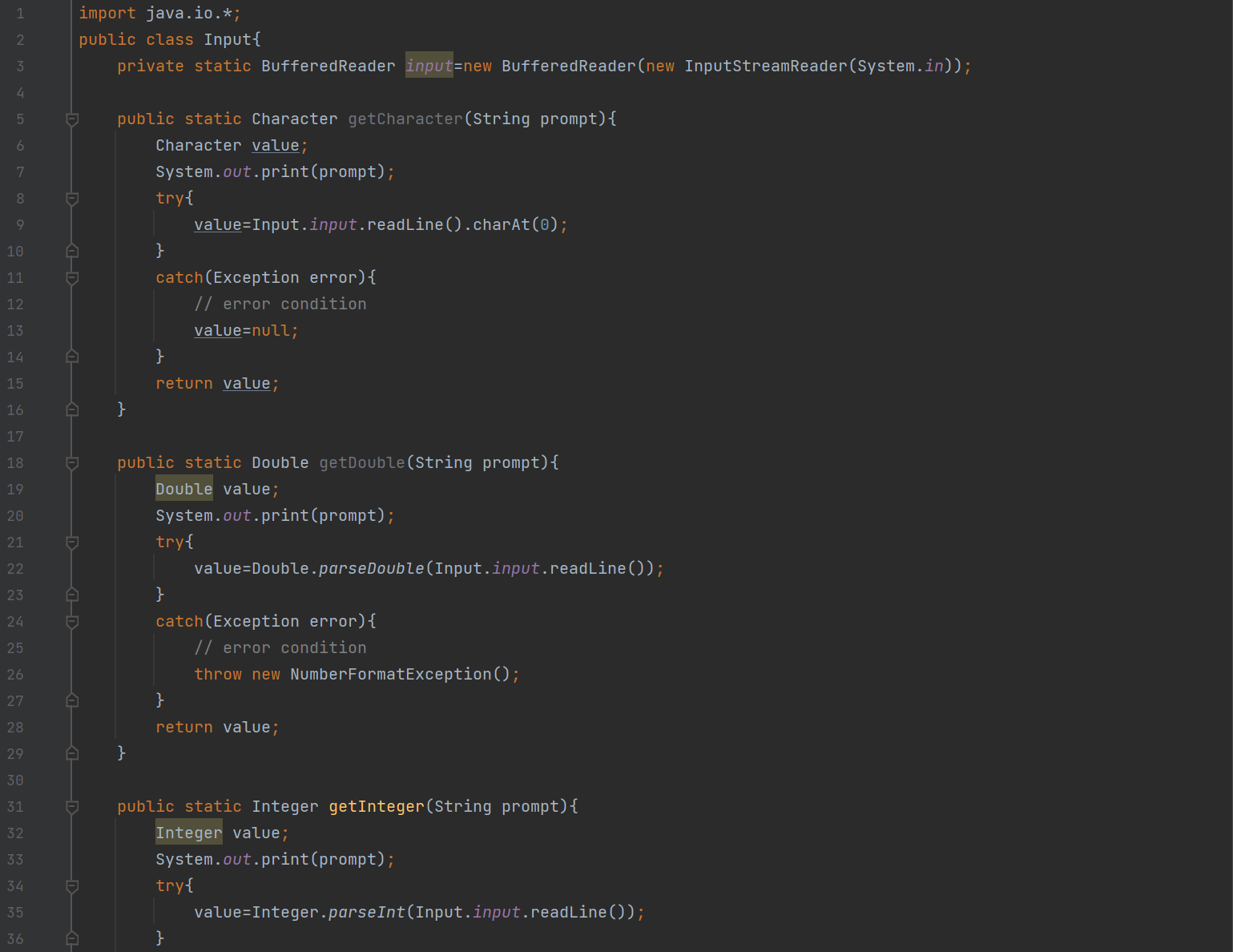
### Classes and Methods

The Classes and Methods created for Step 1 include:

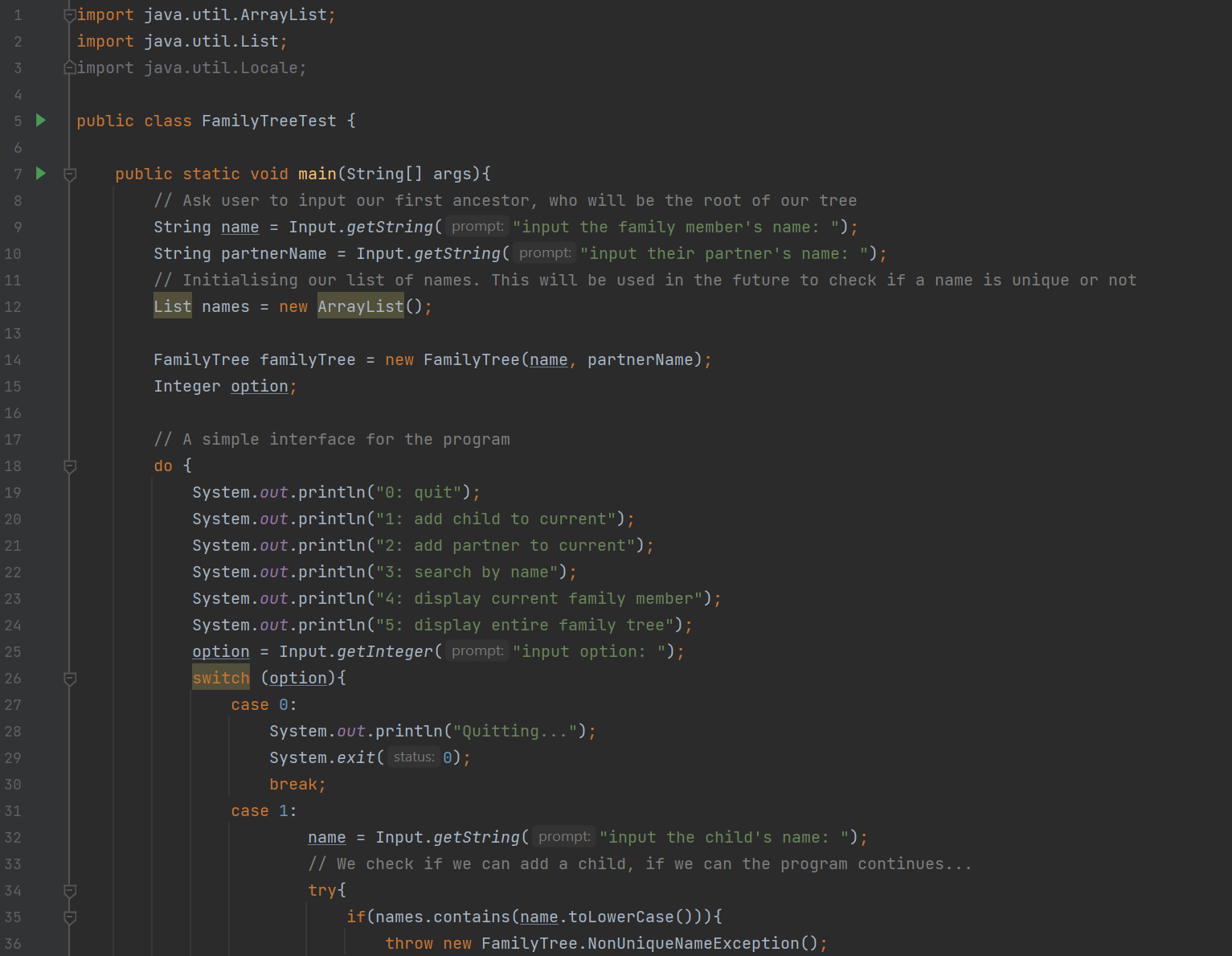
|  |  |  |
| --- | --- | --- |
| FamilyTreeTest | Class | This class houses the interface and the main method. |
| FamilyTree | Class | The data structure that contains the inner workings of the family tree functionality. |
| Input | Class | The class responsible for taking in input for each data type. |
| getCharacter | Method – Input | Takes in input and converts it to the Character data type |
| getDouble | Method - Input | Takes in input and converts it to the Double data type |
| getInteger | Method – Input | Takes in input and converts it to the Integer data type |
| getString | Method – Input | Takes in input and converts it to the String data type |
| main | Method - FamilyTreeTest | The main method, compilation starts here. |
| FamilyTreeNode | Structure – FamilyTree | The structure of each node of the tree. Includes the data for the name, ancestor, partner, sibling, and child. |
| FamilyTree | Contructor - FamilyTree | Initialises each instance of the FamilyTree class. Sets up an ancestor and their partner. |
| addChild | Method – FamilyTree | Takes in a name, creates a new node in the tree and points to the ancestor’s child slot. Automatically checks for siblings. |
| addPartner | Method – FamilyTree | Takes in a name, creates a new node in the tree and points to the ancestor’s partner slot. |
| toString | Method – FamilyTree | Configures and styles the entire tree into string form. |
| getChildren | Method - FamilyTree | Obtains information for each child. |
| getPartner | Method – FamilyTree | Obtains information for each partner |
| getCurrent | Method – FamliyTree | Returns information for the current family member in focus. |
| findFamilyMember | Method – Family Tree | Searches the tree for particular node with a specified name. |
| checkChildren | Method – Family Tree | A submethod for the findFamilyMember method. Checks each child’s name. |
| checkPartner | Method – Family Tree | A submethod for the findFamilyMember method. Checks each partner’s name. |

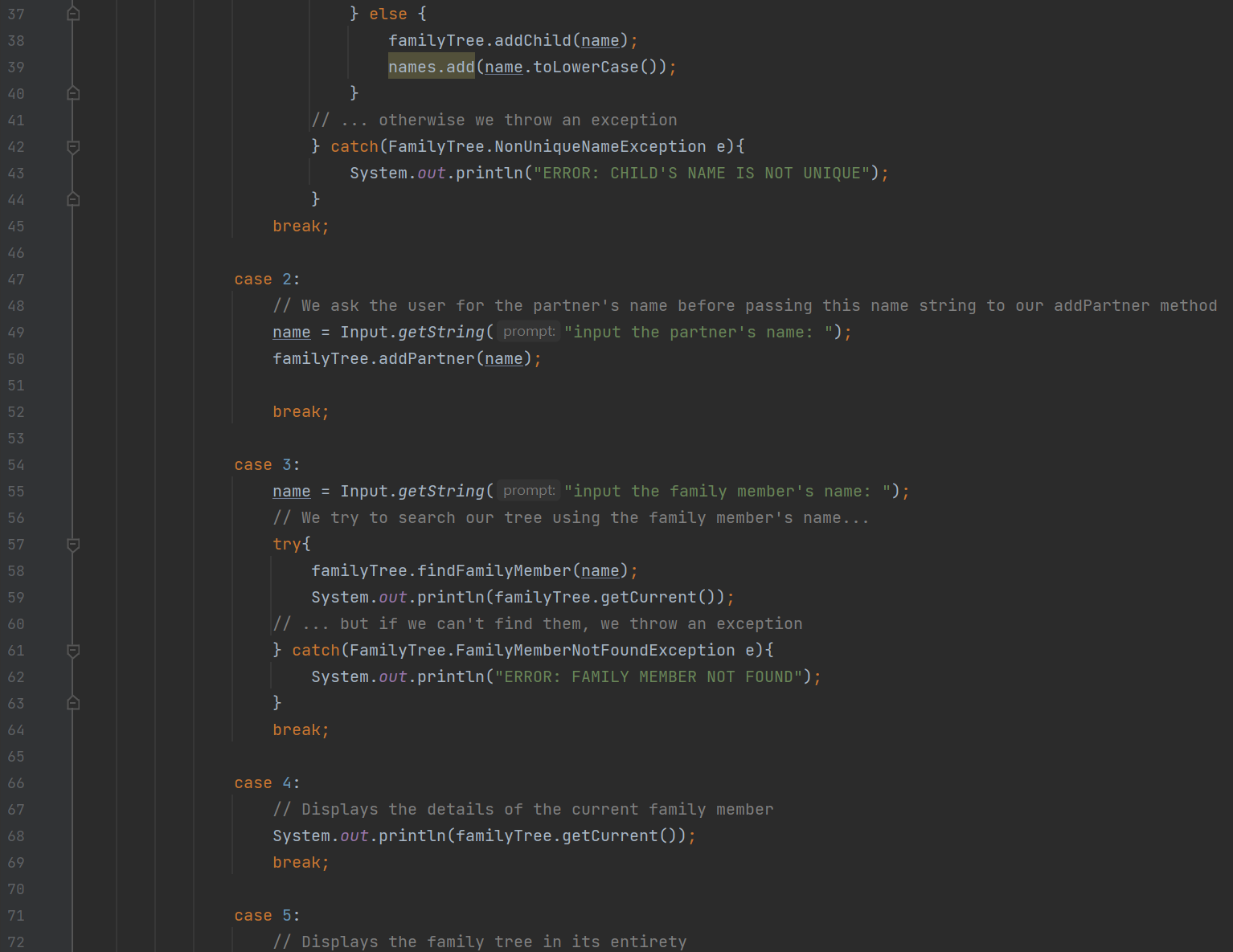
### Code

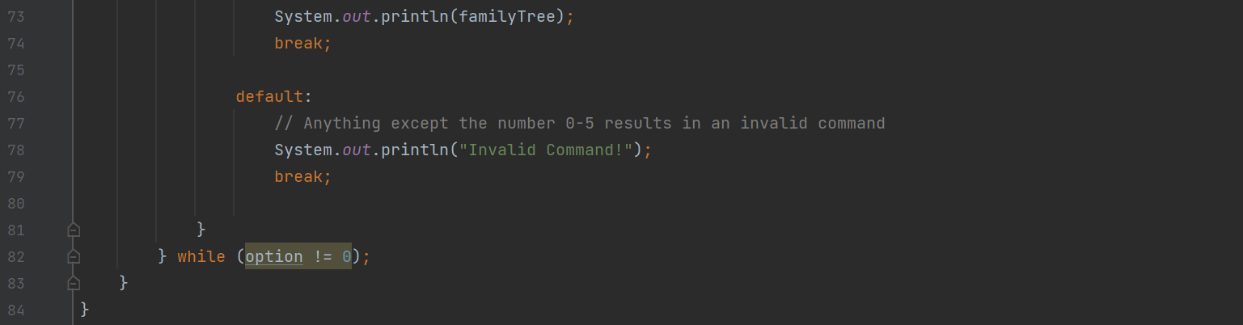
Input Class





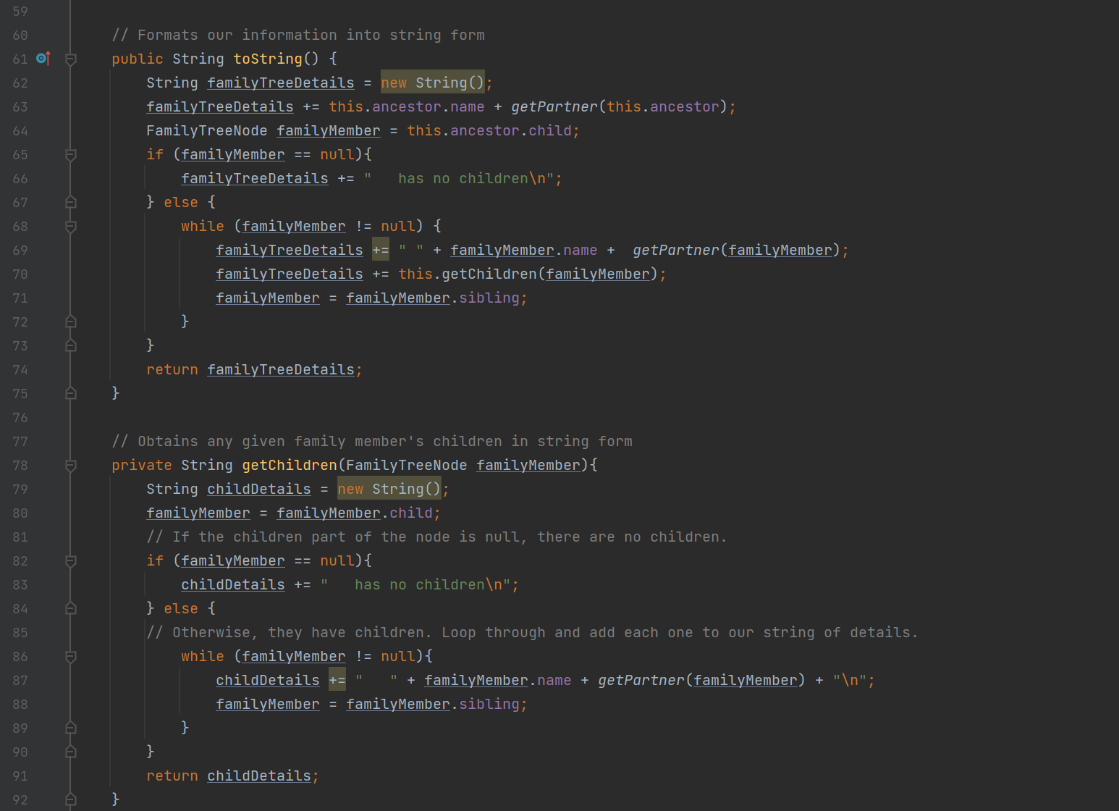
FamilyTreeTest

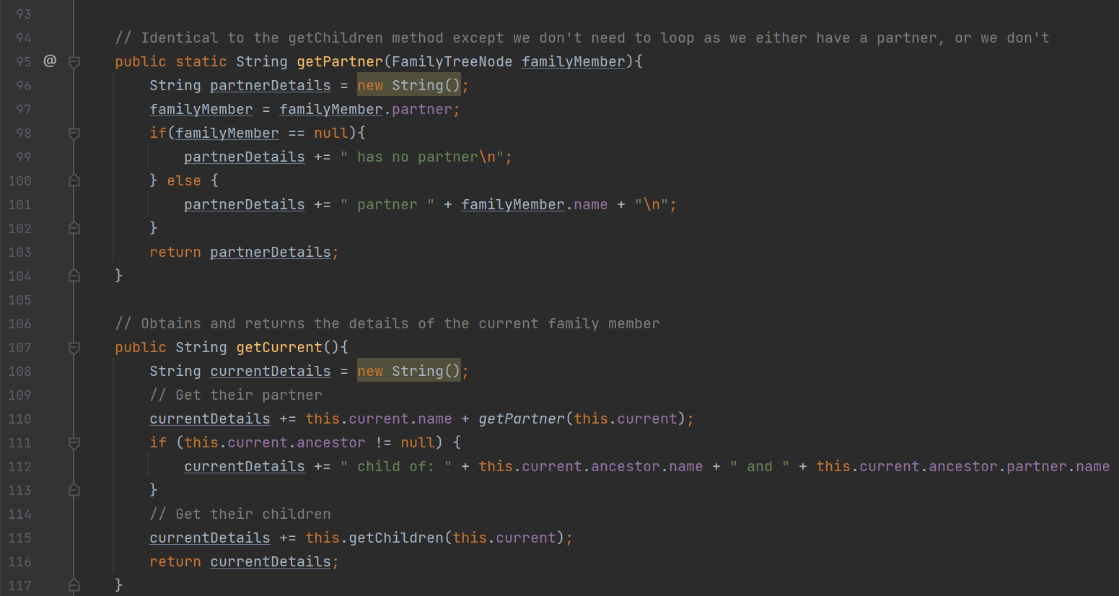


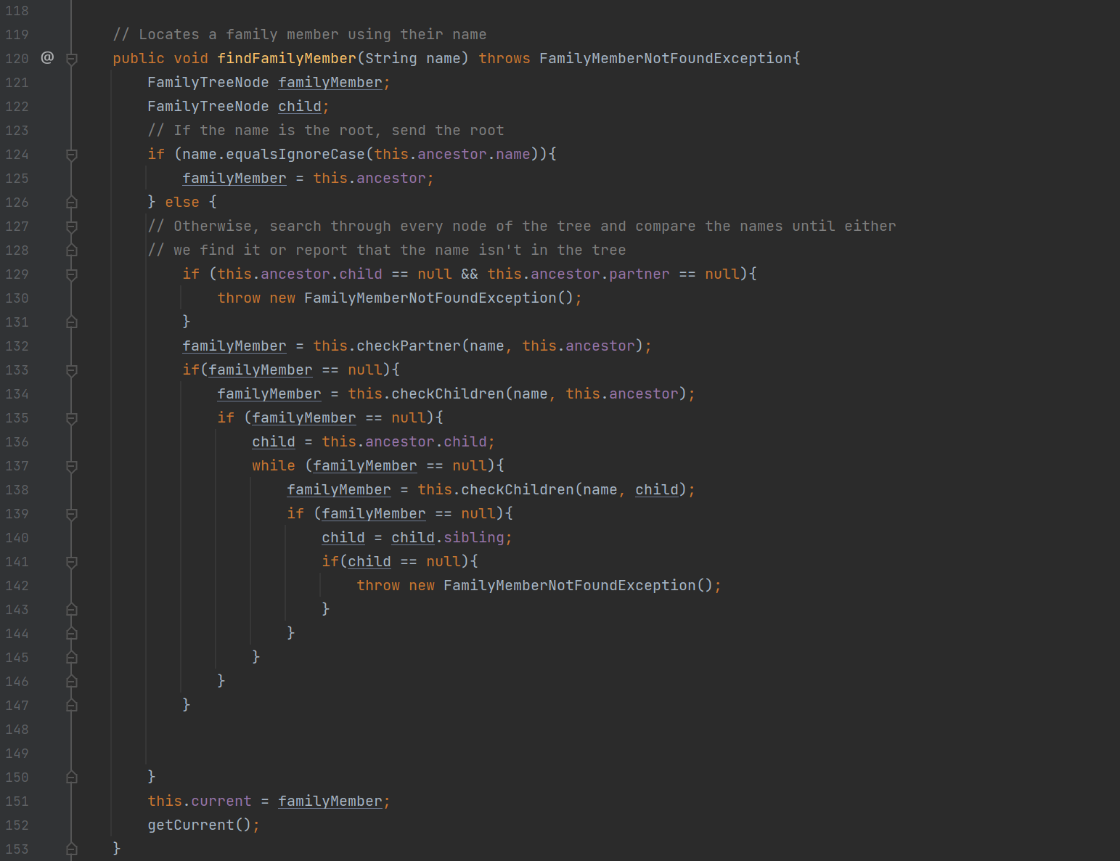


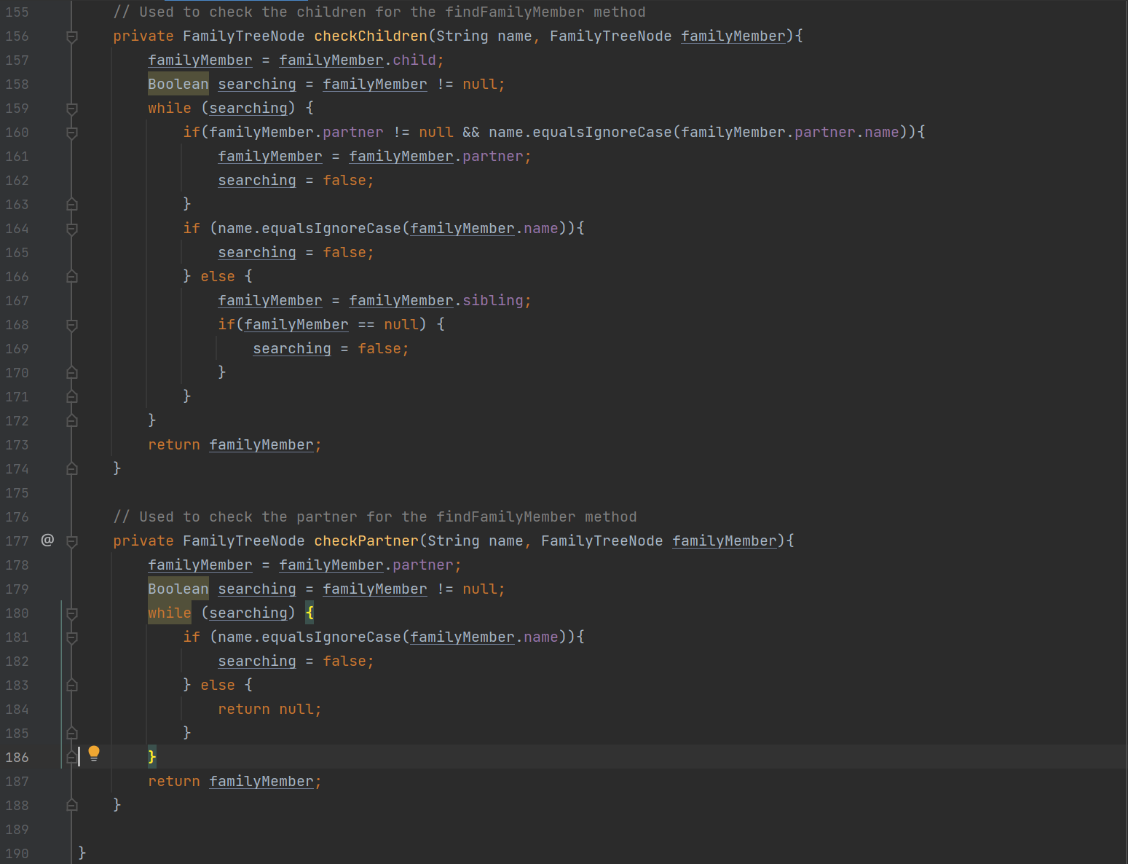
FamilyTree











## Difficulties

Although there were a few difficulties throughout developing this stage, I was pleased that my program seemingly fulfilled every requirement in the marking sheet.

The hardest part was definitely the creation of the search methods. The parts that scan the entire tree, each child, and each partner for a certain name.

# Step 2

## Brief

For this stage at first glance I was just building upon Step 1, but Stage 2 had considerable difficulty and complexity. First I was tasked with revamping my tree with a unique identifier number for each family member node.

Diagram

Description automatically generated

As the tree gets more complicated, there is more functionality. Children and Partners can now only be added under certain circumstances.

## Implementation

Step 2 definitely took more time to develop, to start off I implemented the new identifier system. This proved to not be too difficult, I initialised the integer in the main method and each time a child, a partner, or any node is added, the identifier increments. They are also assigned automatically.

Next I implemented all the exceptions, this was also fairly easy as I had already created a few exceptions for Step 1 so I just followed on from there.

Finally I studied the marking scheme and implemented features (that I could implement) as I saw them.

### Marking Scheme

|  |  |  |
| --- | --- | --- |
| Constructor only sets up ancestor, assigned identifier, display no partner. | Text  Description automatically generated | Unlike Step 1, for Step 2 the constructor is to only setup the ancestor. This also automatically assigns identifiers and displays no partners. |
| Detects attempt to add child without partner, exception handling, suitable message displayed. | Text  Description automatically generated | When you try to add a child without a partner, an exception is raised. |
| Detects when ancestor already has partner, exception handling, suitable message displayed, displays structure, displays identifiers. | Graphical user interface, text  Description automatically generated  Text  Description automatically generated | Likewise, if you try to add a partner to someone already taken, you also get an exception. When successfully added, the identifier is displayed. |
| Can add partner to ancestor | Graphical user interface, text  Description automatically generated | Partners can bed added without issue |
| Can add partner to child | Text  Description automatically generated | And as long as they have a partner, a child can be added. |
| Graphical user interface, text  Description automatically generated  Displays child’s partner |  | If a child has a partner, their information is displayed. |
| Displays no children for child | Graphical user interface, text  Description automatically generated | If the couple have no children, an appropriate message is displayed. |
| Display family member, correctly identified, or uses exception if ID not found, includes child’s partners. | Text  Description automatically generated  Text  Description automatically generated | When searched, the family members information is displayed for that branch including parents, children, and partners. If not found, an exception is raised. |

### Classes and Methods

Most of the classes and methods remained the same, however there were a few changes.

|  |  |  |
| --- | --- | --- |
| findFamilyMemberID | Method – FamilyTree | Similar to findFamilyMember except it uses Identifiers instead of names to search the tree. |
| checkChildrenID | Method – FamilyTree | The identifier equivalent to checkChildren method |
| checkPartnerID | Method - FamilyTree | The identifier equivalent to checkPartner method |

### Code

The Input class remains the same.

FamilyTreeTest

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedFamilyTree

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

## Difficulties

Unlike Step 1, there were a few difficulties I ran into that could not be resolved, although these were minor features that don’t affect the functionality of the program. Most are merely aesthetic omissions. In the end I think I only missed out 4 features as the implementation of these were either too difficult or would require a complete restructure of my entire program.

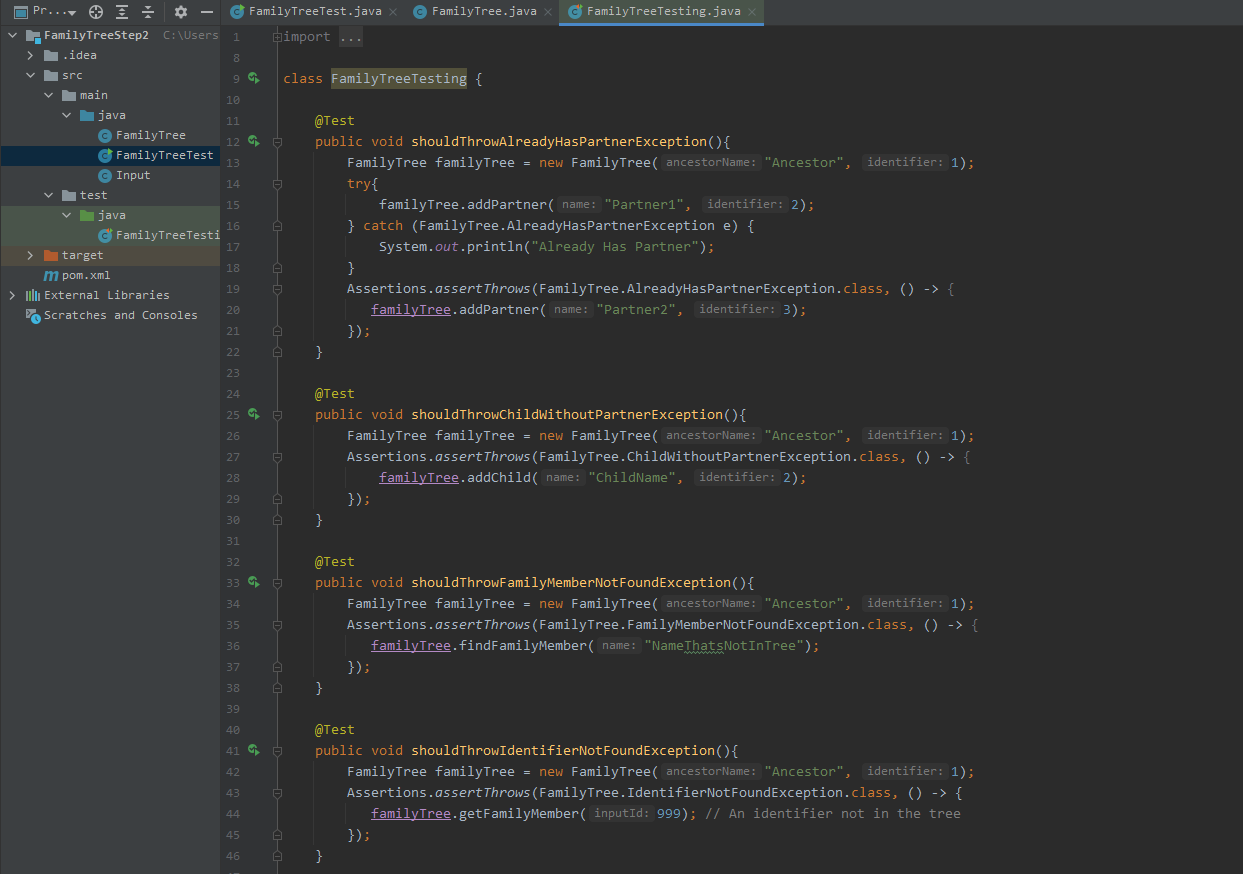
# Testing

As part of my evaluation for the project, I chose to implement two methods of testing. I used a traditional test table and wrote the tests out manually, and a set of 7 Junit tests within the IntelliJ IDE.

Test Table

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Description | Expected | Result |
| Add child (without partner) | The first ancestor has been added and we try to create a child before assigning a partner | A child cannot be added, an exception will be raised and the user will be alerted of this. | The user can enter a name, but the child isn’t actually created. An exception is raised and the user is alerted. |
| Add child (with partner) | Same as before but this time the ancestor has a partner | The user will enter a name. If the name isn’t a duplicate, the child is added to the tree. | The child is added, assuming the name isn’t a duplicate. |
| Add child (duplicate name) | The user attempts to add a child, but the name is a duplicate. | The user will enter a name, but the system will throw an exception and the user is alerted. No child is created. | The user enters the name, and the system correctly raises a duplicate name exception. |
| Add partner (No partner) | The user attempts to add a partner to a single family member. | The user will enter a name for the partner and the partner will be added to the current family member. | A partner is created using the inputted name. |
| Add partner (Already has partner) | The user attempts to add a partner but the family member is already taken. | The user will attempt to create a new partner, but the system detects that the family member is already taken. An exception is raised | The system detects that the family member the partner is to be assigned to is already taken and then raises an exception. |
| Search for family member by name | The user enters a name and the system searches the tree for the family member with that name. | If the name is in the tree, the system will return that family member’s details. Otherwise, an exception will be raised. | Given that the name is actually in the system, that family member is made the current. Otherwise, an exception is raised |
| Search for family member by ID | The user enters an ID and the system searches for the family member with that ID. | If the ID is in the tree, the system will return that family member’s details. Otherwise, an exception will be raised. | Given that the ID is actually in the system, that family member is made the current, and their details are displayed. |
| Display family tree | A method that displays the entire family tree is called. | The system displays the family tree in its entirety. Including partners, and children’s partners. | The system displays the family tree in its entirety. Including partners, and children’s partners. |

Junit Tests



1. **AlreadyHasPartner exception**

This unit test intentionally triggers the AlreadyHasPartner exception by trying to add another partner when the family member already has a partner.

2. **ChildWithoutPartner exception**

This unit test intentionally triggers the ChildWithoutPartner exception by trying to add a child without having a partner.

3. **FamilyMemberNotFound exception**

This unit test intentionally triggers the FamilyMemberNotFound exception by searching for a family member that doesn’t exist using a name string.

4. **IdentifierNotFound exception**

This unit test intentionally triggers the IdentifierNotFound exception by searching for a family member that doesn’t exist by using an identifier integer.

5. **NonUniqueName exception**

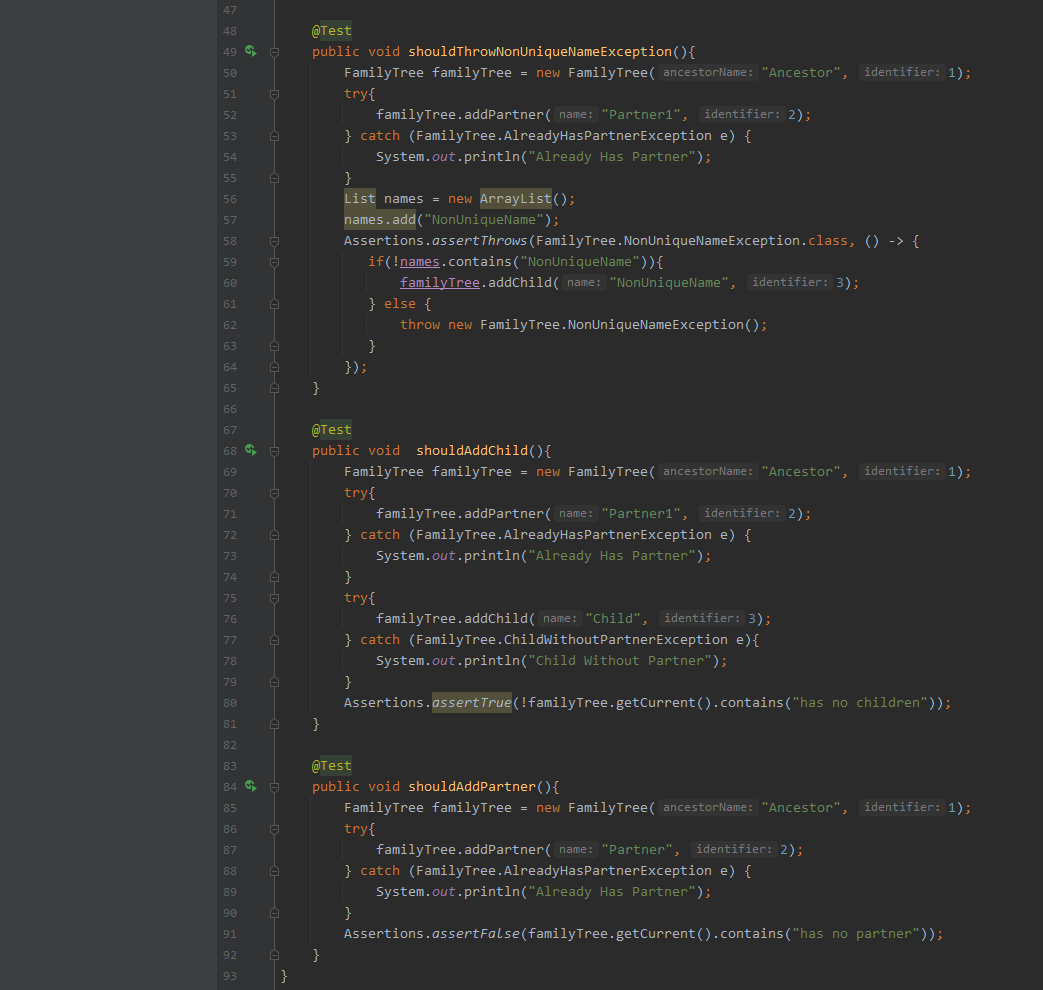
This unit test intentionally triggers a NonUniqueName exception by first adding a child, and then adding another child with the same name. Regardless of case, this exception will be thrown if the names match.

6. **Add child (with partner)**

This unit test adds a child under normal circumstances. The ancestor already has a partner and thus no exceptions are thrown.

7. **Add partner**

This unit test adds a partner under normal circumstances. The ancestor doesn’t yet have a partner and thus no exceptions are thrown.



# Github

Throughout the project I utilised a GitHub repository which can be found at . This repository not only allows for evidence of my project’s development, but also utilises version control and cloud storage which proved handy for each stage when developing my coursework.

Some notable screenshots include:

Main Screen

A screenshot of a computer

Description automatically generated with medium confidenceThis shows the directory of my project

Contributor Timeline

A screenshot of a computer

Description automatically generated with medium confidenceThis shows who contributed to this repository and when. As shown, there is a steady development from when the coursework was distributed and around the time I finished my programming. It also shows that I worked on this alone.