**Software Design Patterns Refactoring Exercise Report**

**B.Sc. in Business Computing**

**Year 4**

As a student I was required to refactor the source code for an application simulating a personnel application. Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure (What is Refactoring, 2019)

The refactoring exercise on was done and the following refactoring were applied to the source code:

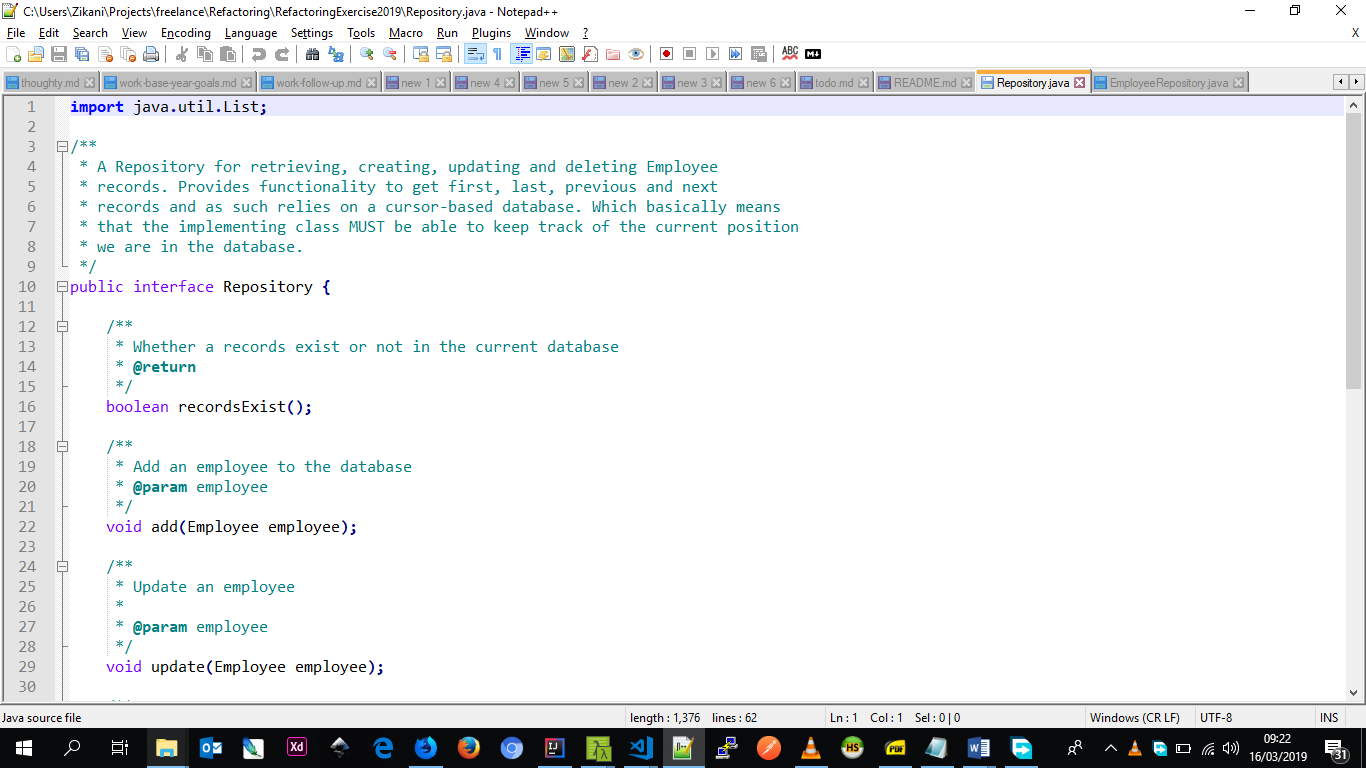
## Introduction of the Repository pattern

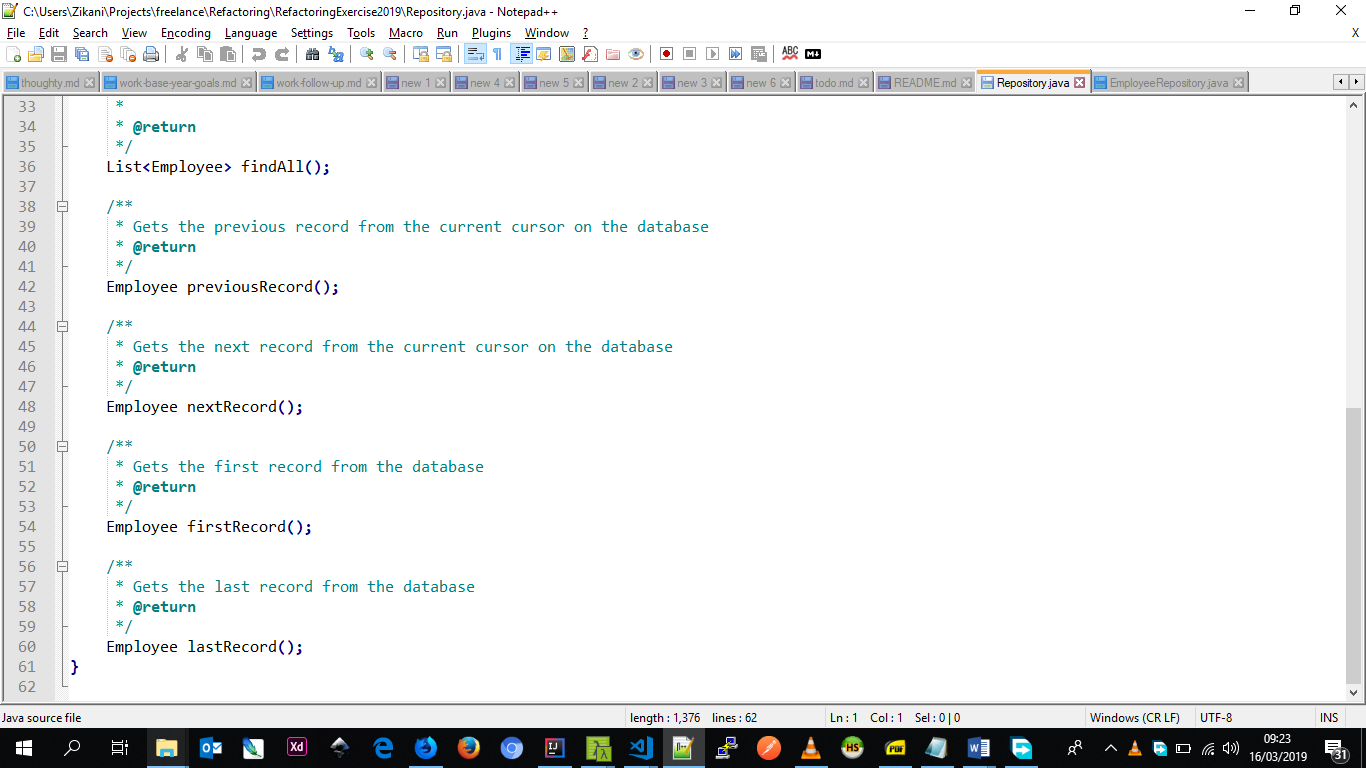
The major refactoring applied on the code was the introduction of the repository pattern. The pattern defines a generic abstract way for an application to work with the data layer without being bothered with if the implementation details. The Repository provides methods to create, retrieve, update, and delete records. (Bergman, 2019)

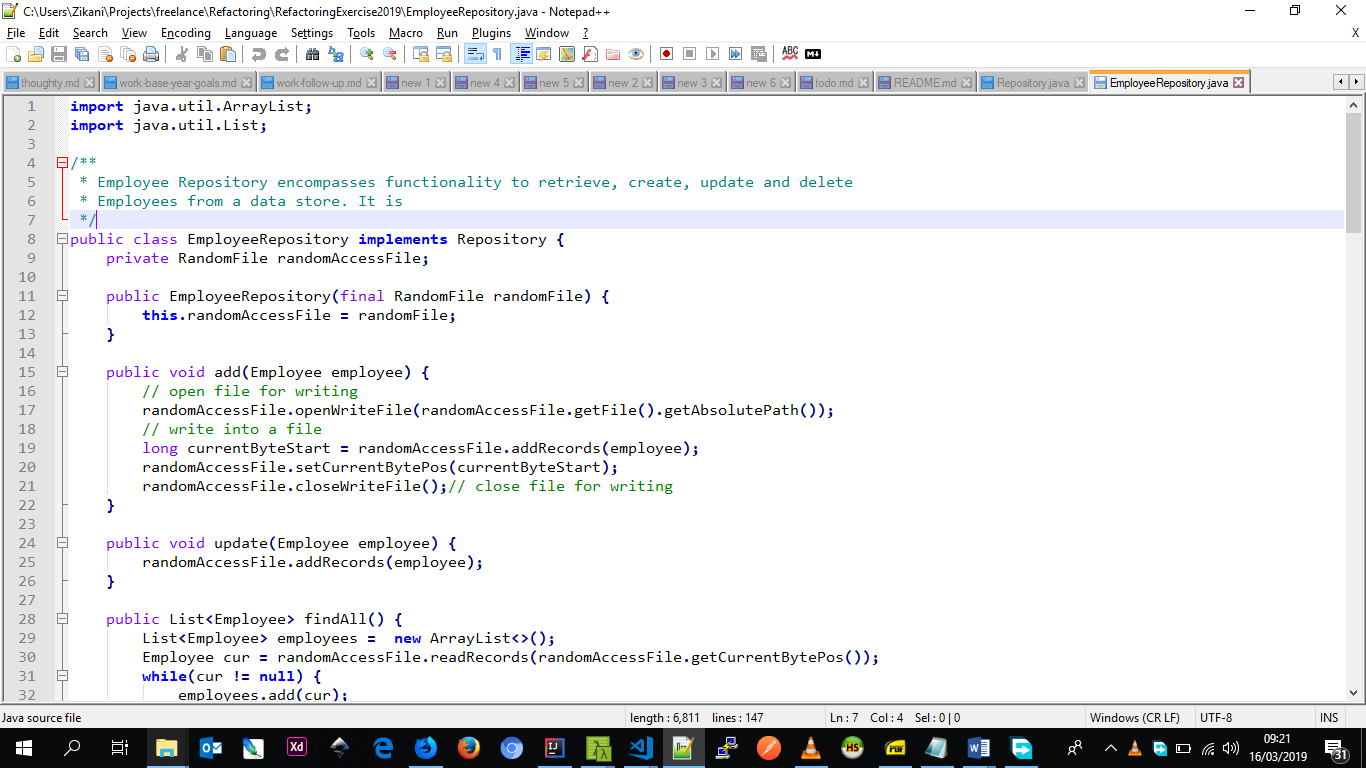
Since the Repository design pattern provides a layer of abstraction from the underlying storage, in this case a random-access file, it allows us to use a different storage mechanism later on for example using an SQL database without changing the client code i.e. the user interface.

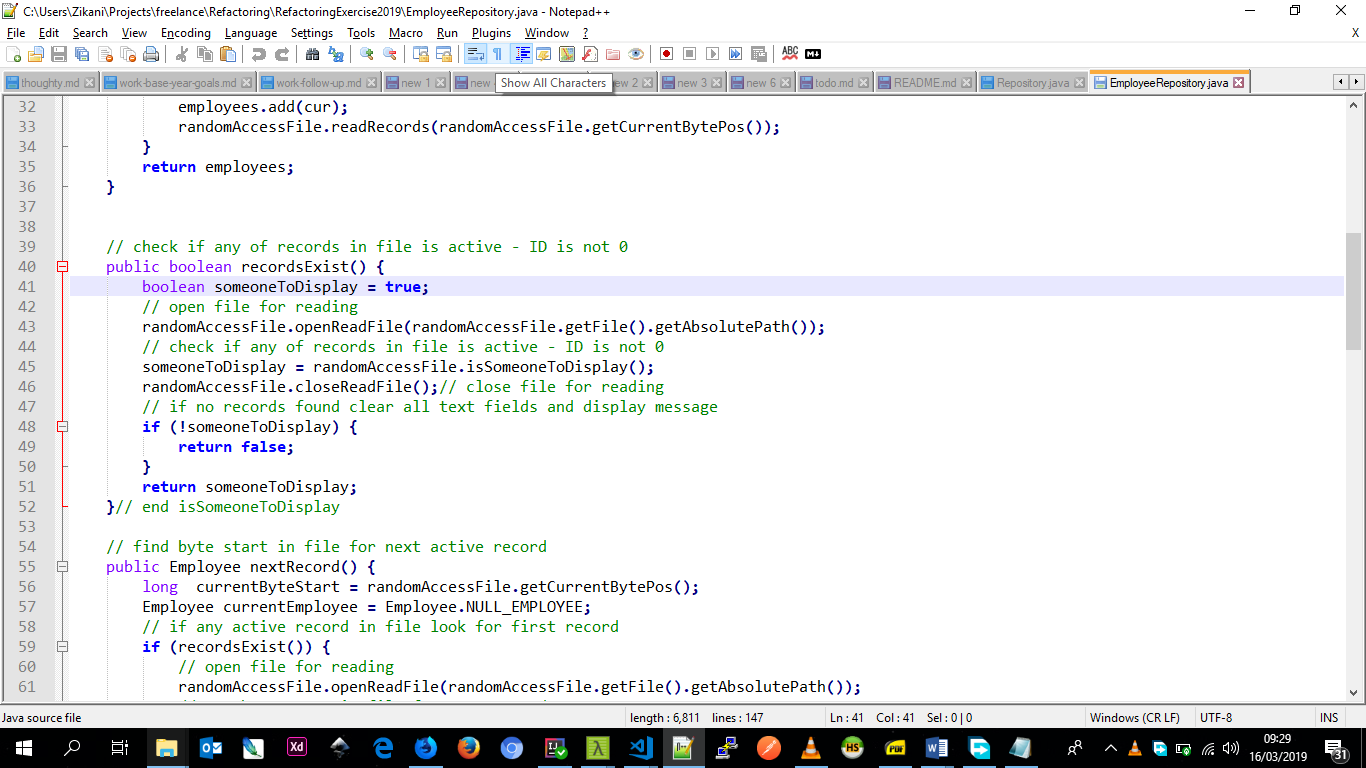
The repository pattern was implemented with an Interface implementation which currently uses the RandomFile class for accessing and manipulating the database.

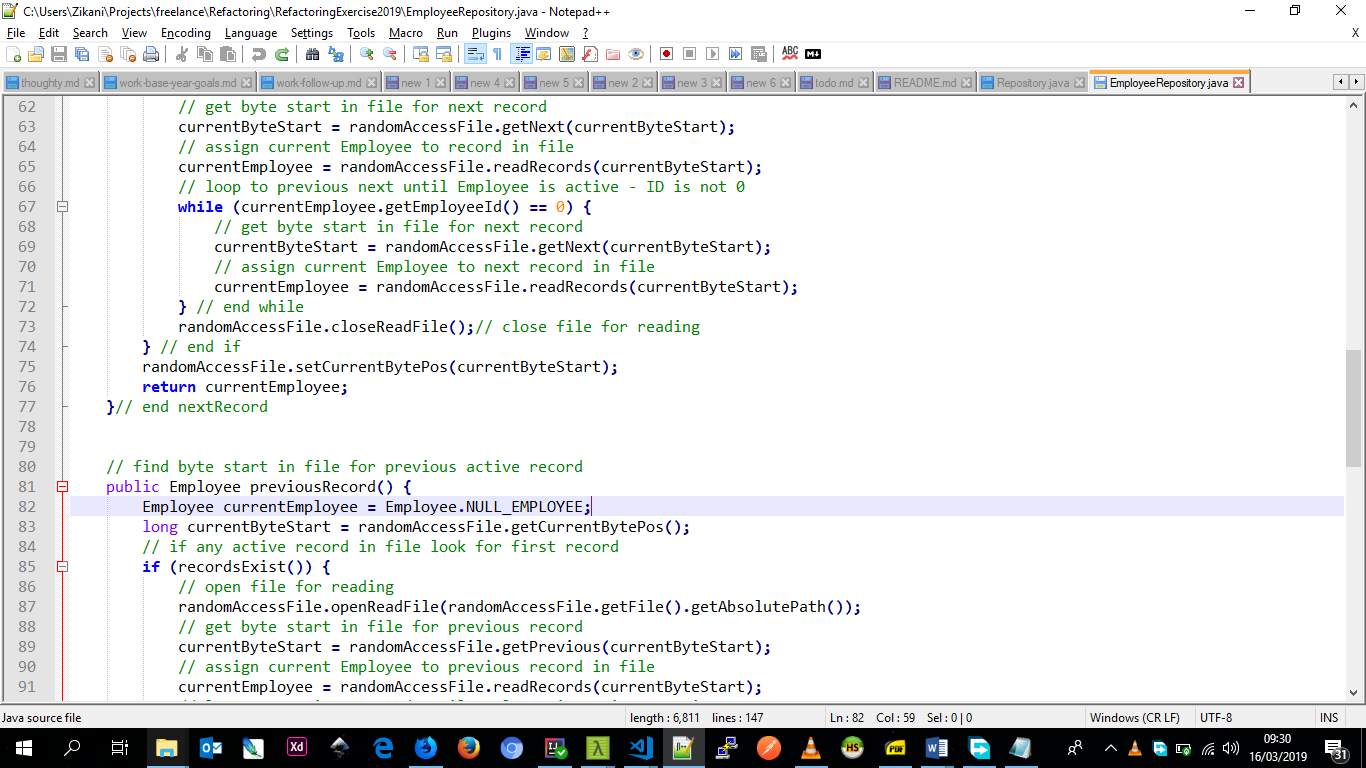
The figures below show the interface (**Repository)** and implementation class (**EmployeeRepository**):





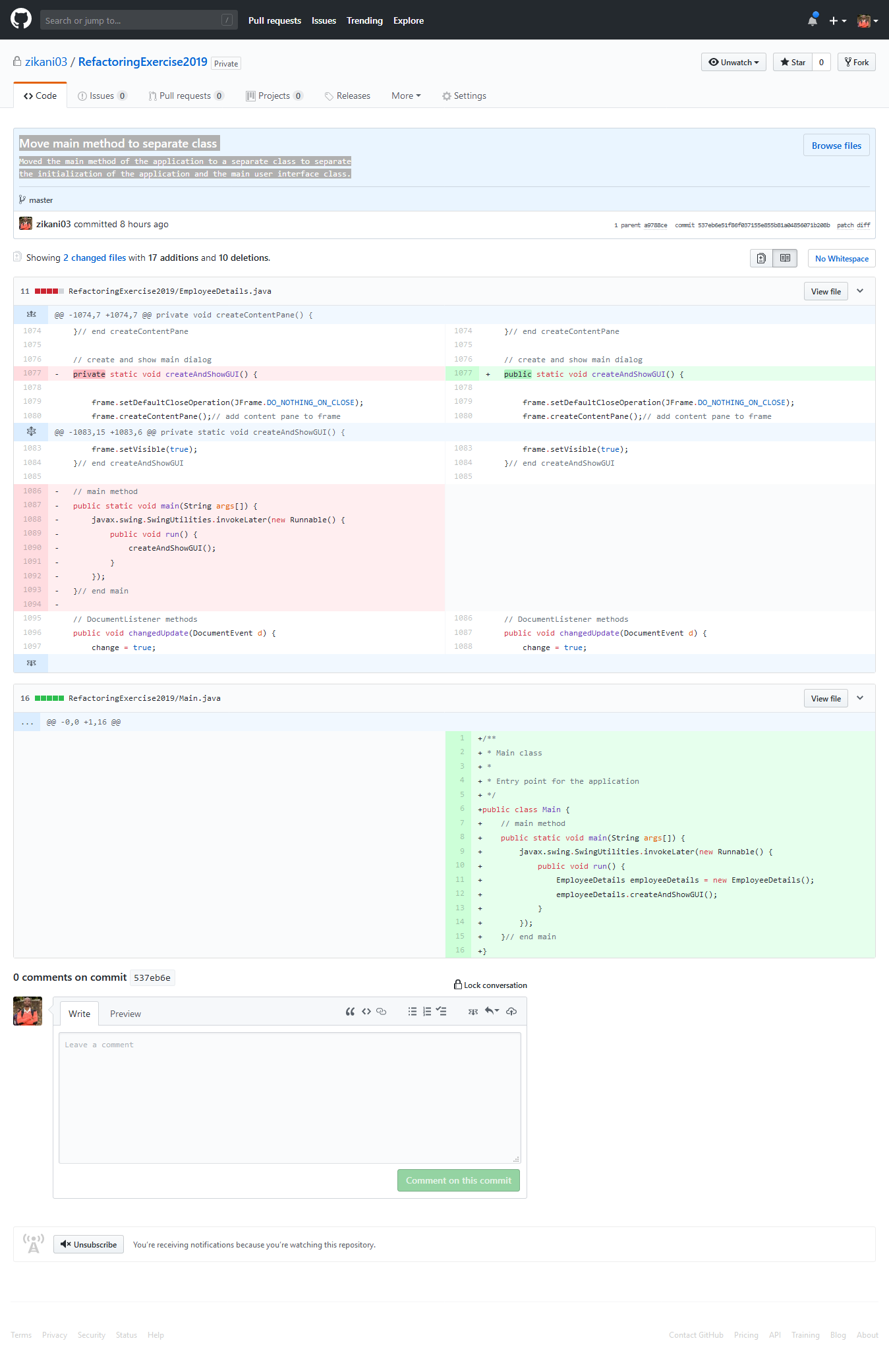






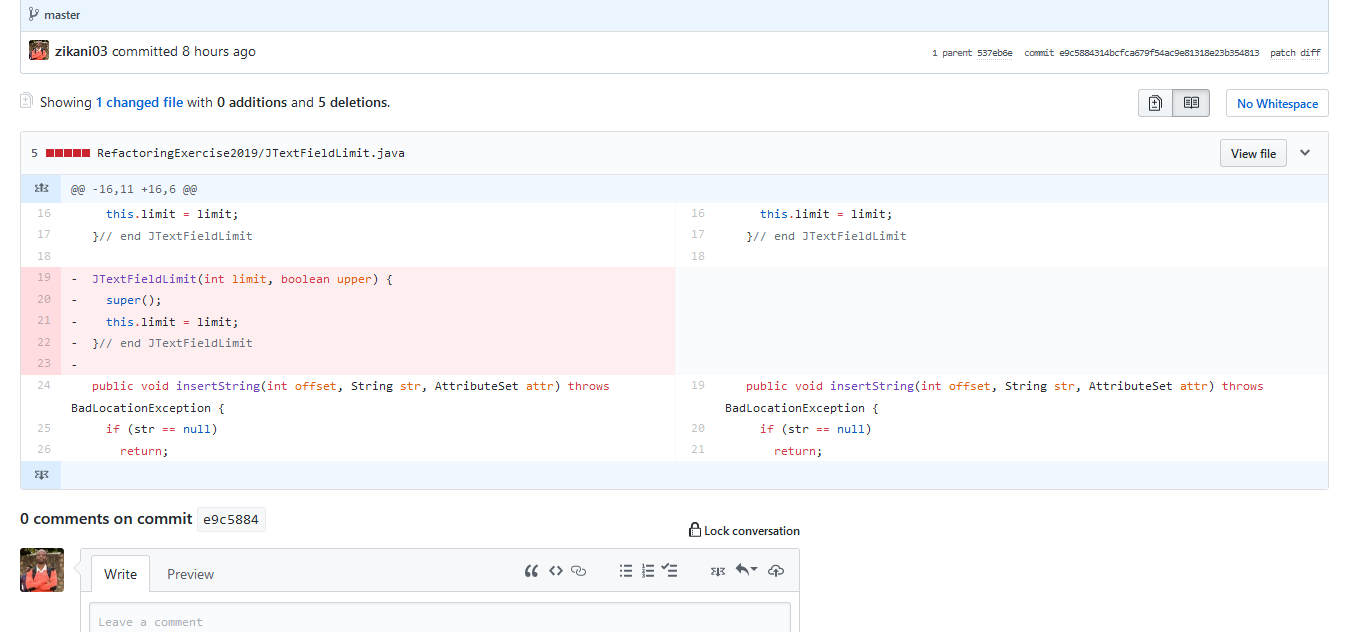
## Move main method to separate class

Moved the main method of the application to a separate class to separate the initialization of the application and the main user interface class.



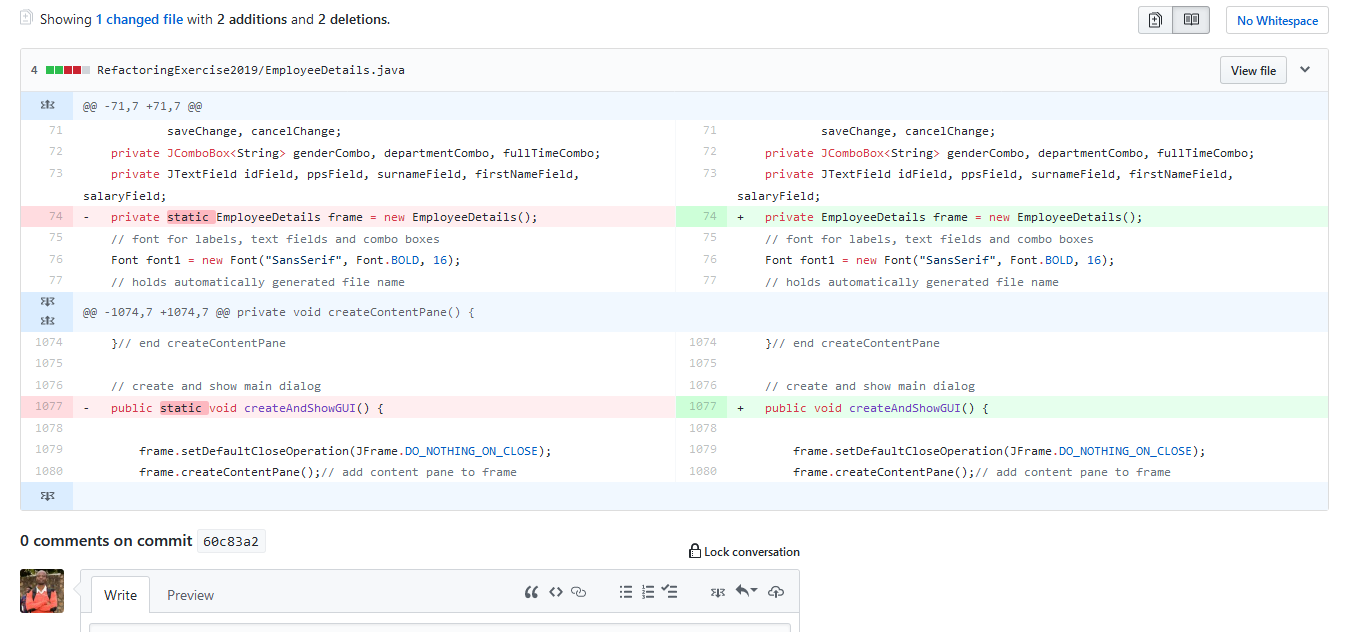
## Remove unused constructor

Removed constructor in JTextFieldLimit that was not being used.



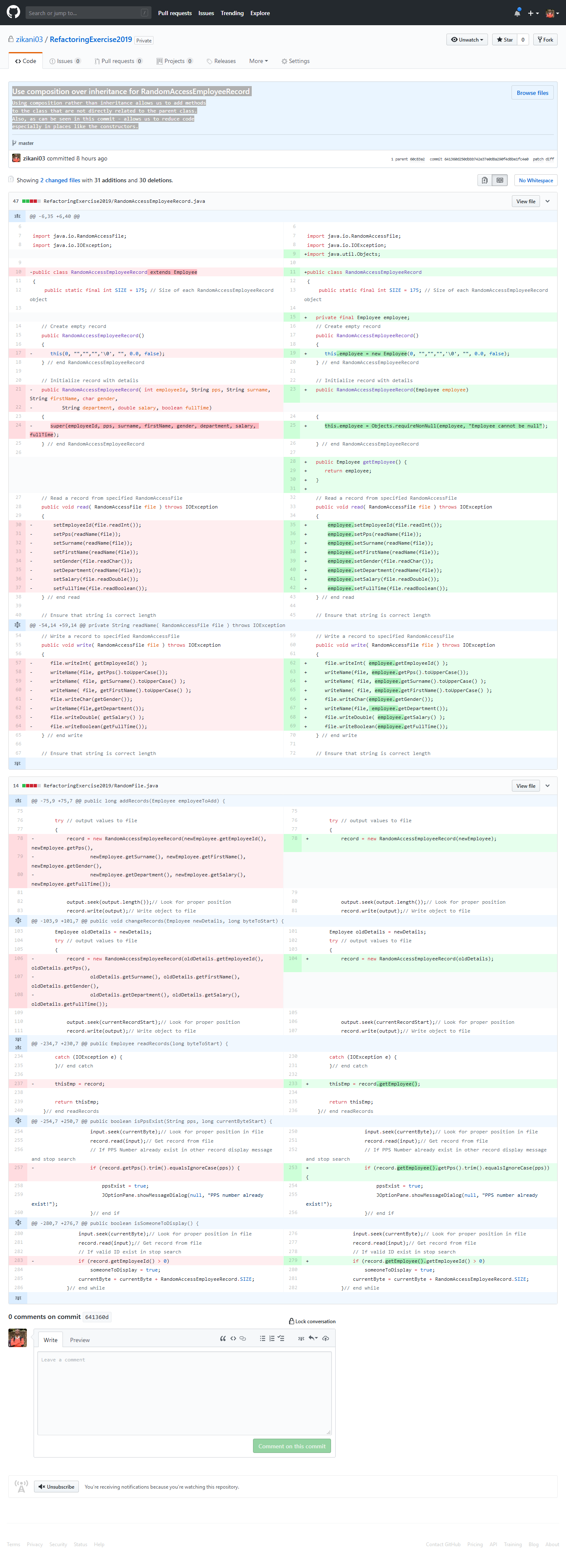
## Make frame instance and createAndShowGUI methods non-static

Prefer instance variables to reduce dependency on global state in the application. Using static variables introduces global state which is not a best practice.



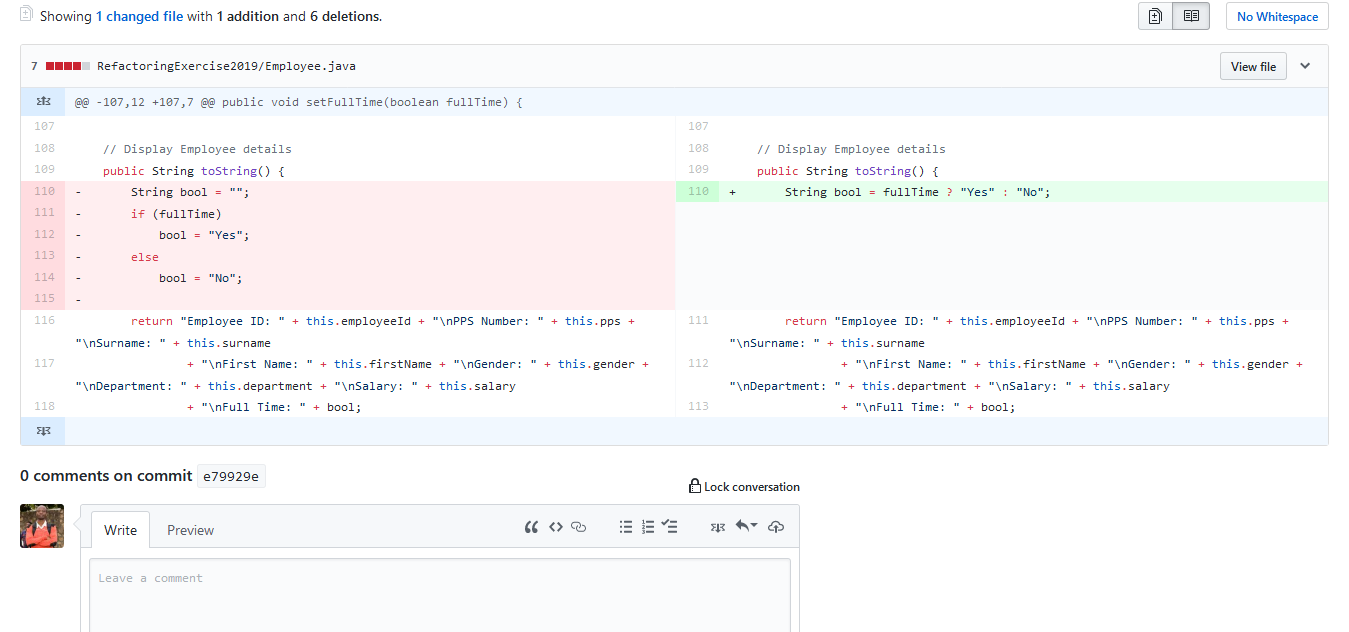
## Use composition over inheritance for RandomAccessEmployeeRecord

Using composition rather than inheritance allows us to add methods to the class that are not directly related to the parent class. Also, as can be seen in the screenshot bellow – it allows us to reduce code especially in places like the constructors.



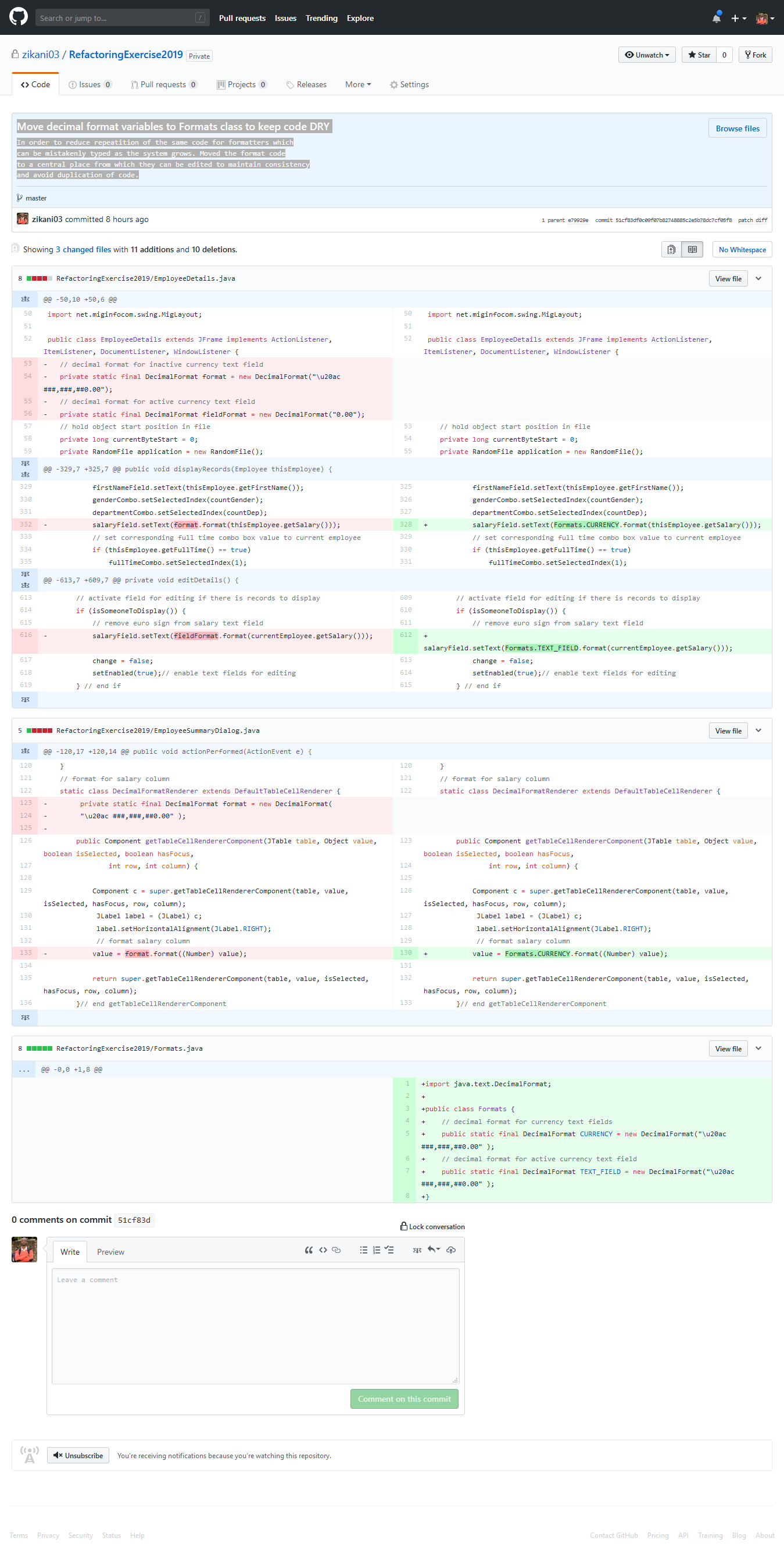
## Use ternary operator for binary condition

The refactoring made the code simpler by using a feature of the Java programming language to make the code shorter and easier to read.



## Move decimal format variables to Formats class to keep code DRY

In order to reduce repetition of the same code for formatters which can be mistakenly typed as the system grows. Moved the format code to a central place from which they can be edited to maintain consistency and avoid duplication of code.



## Centralize code for showing a JOptionPaneDialog for an error message

Centralized the code that allows us to show error messages in a dialog in one method. The reasoning is that we can use this method for things like logging or if we want to style the dialog differently we get to do it in one place.



## Introduction of Tests for verifying functionality

Another refactoring activity carried out was the introduction of Tests for some classes in the code to verify that they behave as expected.

# References

1. What is Refactoring? Website: <http://wiki.c2.com/?WhatIsRefactoring>, 2019, Accessed: 2019-03-14
2. Repository Design Pattern. Per-Erik Bergman, 2019, Website: <https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30> Accessed: 2019-03-14