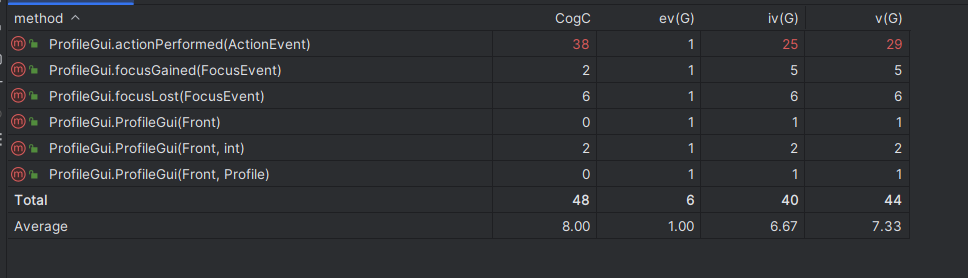
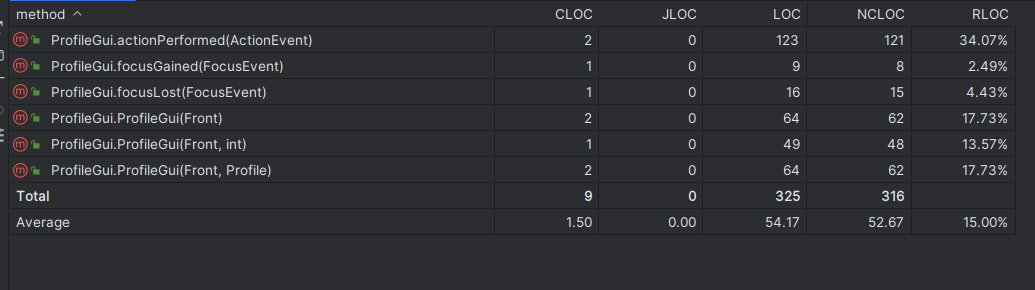
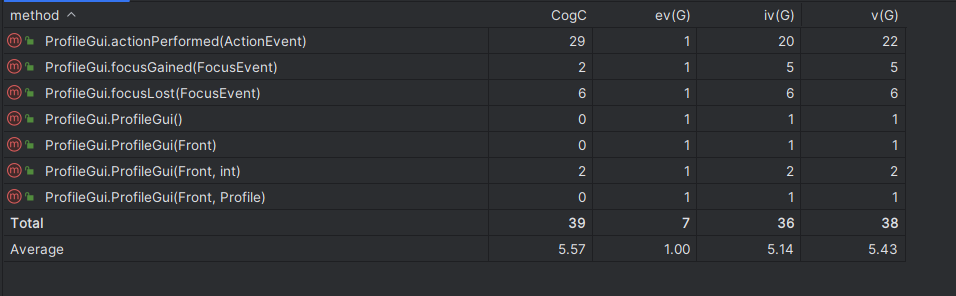
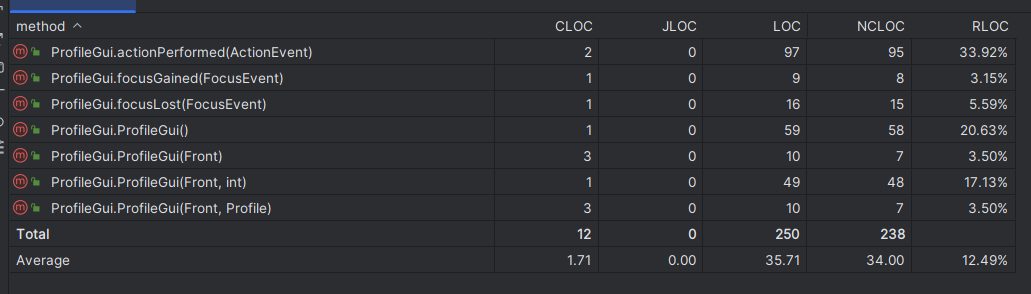
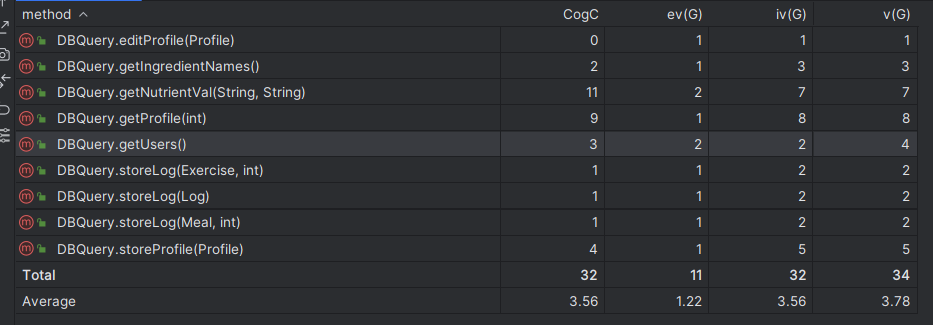
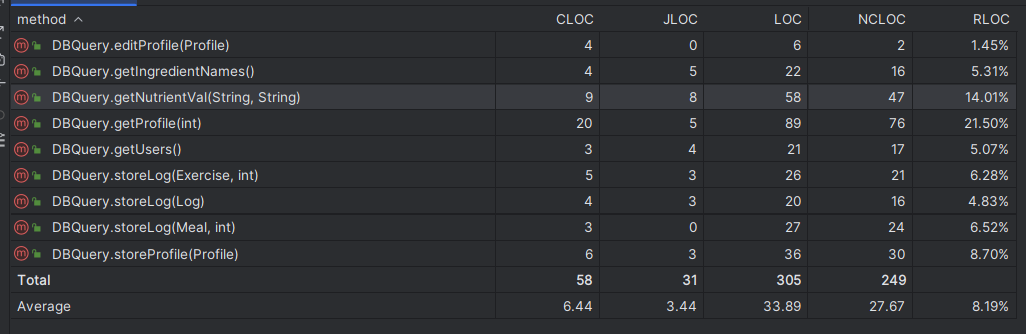
**Use case 1 + Database Queries:**

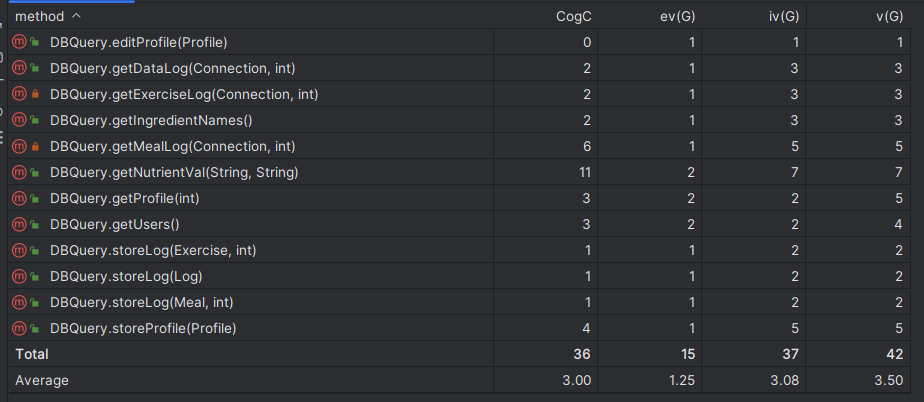
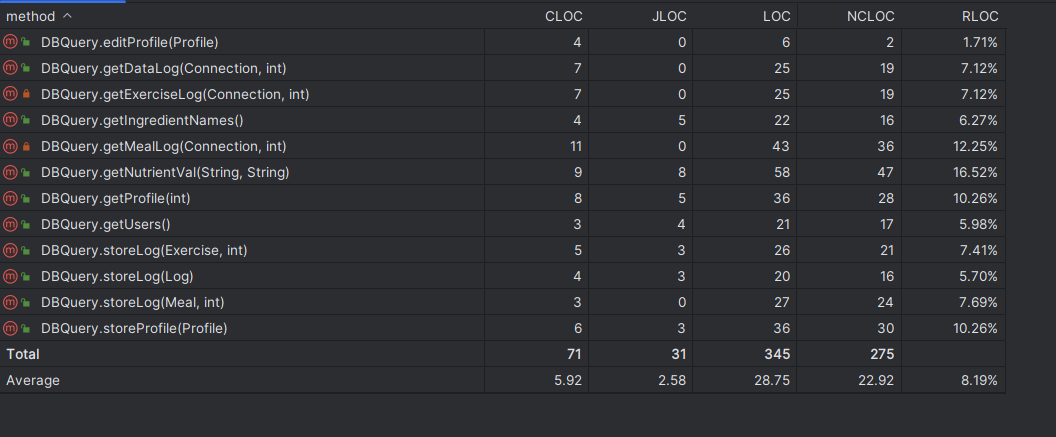
Initial metrics:  
  


After refactoring:  
  


Code Smell: Duplicated Code

* The constructors are used to display the different GUIs associated with the user profile. Of the three, ProfileGui(Front) and ProfileGui(Front, Profile) contain very similar codes.
* The method actionPerformed(ActionEvent) is essential for the functionality of the ProfileGui.java. As such, the size and cyclomatic complexity of the code is expected to be much higher than other methods in the class. However, examining the code shows that there are duplicated code within the method that needed adjustments.
* Solution:
  + The constructors are redefined to be chained from a common constructor instead.
  + The duplicated portions are combined as much as possible to remove the code smell.

Initial metrics:  
  


After refactoring:  
  


Code Smell: Long Method:

* The method getProfile(int) has the highest value of cyclomatic complexity as well as accounting for 21.50% of the code within the class. Further examinations show that the method also violates SRP, being responsible for creating a Profile object from the database but also creating the necessary histories of the user’s profile at the same time, making it confusing for other developers to understand.
* Solution: Extract the creation of each history field into individual methods, preserving SRP as well as adding more reusability to the class.
  + Cyclomatic complexity on average is lower at the cost of increasing the total of the class.
  + The lines of code per method in the class is reduced.

**Use case 2-3:**

Initial Metrics:

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**Code Smell 1:** Duplicated Code

**Description**: The isValidDate(int year, int month, int day) exists in both the MealGUI and ExerciseGui classes. This is unnecessary and could cause multiple issues. If the method needed to be refactored for any reason (either an update to date format, or a bug), both methods in either class would have to be changed.

**Solution**: This code smell was fixed by removing the method from the ExerciseGUI class. Instead, it will call on the same method from the MealGui class. With these changes, if any changes are to be made to the method, no further issues will arise.

**Justification**: This refactoring decreases the amount of methods, and lines of codes in the ExerciseGUI class without making a difference in functionality. A redundant method was removed, which decreases complexity and increases readability.

**Code Smell 2:** Long Method/Duplicated Code

**Description**: Both the exerciseGUI and MealGUI constructor methods are very long. They also share some duplicate code: including the process of creating the combo boxes for the date selector.

**Solution**: This code smell was fixed by creating a new class called DateGUI. This class holds the isValidDate() method, the logThis method, which handles processing the date, as well as a method for setting up the combo boxes for the date selector. Both MealGUI and ExerciseGUI call on this class to access these methods.

**Justification**: The refactoring in this step decreases the complexity of the ExerciseGUI and MealGUI classes, and their methods. It also removes duplicated code, making the code more readable. Also, the maintainability is increased as if any change is made to the way the combo boxes are set up, these changes can be made in DateGUI, which will affect both ExerciseGUI and MealGUI.

**Code Smell 3:** Magic numbers and hard-coded values

**Description**: Originally, when prompted to select the date for a meal or exercise, the was allowed to choose a year between 2020-2025. These hard coded values are not future proof, and would have to be changed.

**Solution**: This code smell was fixed by setting the year combo box to hold 3 values (the current year, and the two previous years). Also, the code was changed so that the combo boxes default to the current date, as that is the most likely date that the user will select.

**Justification**: The refactoring done here improves maintainability by ensuring that the date values will automatically update using the Calendar class in java. The images below show the before and after of this code.  
A computer screen with text

Description automatically generatedA computer screen shot of a program

Description automatically generated

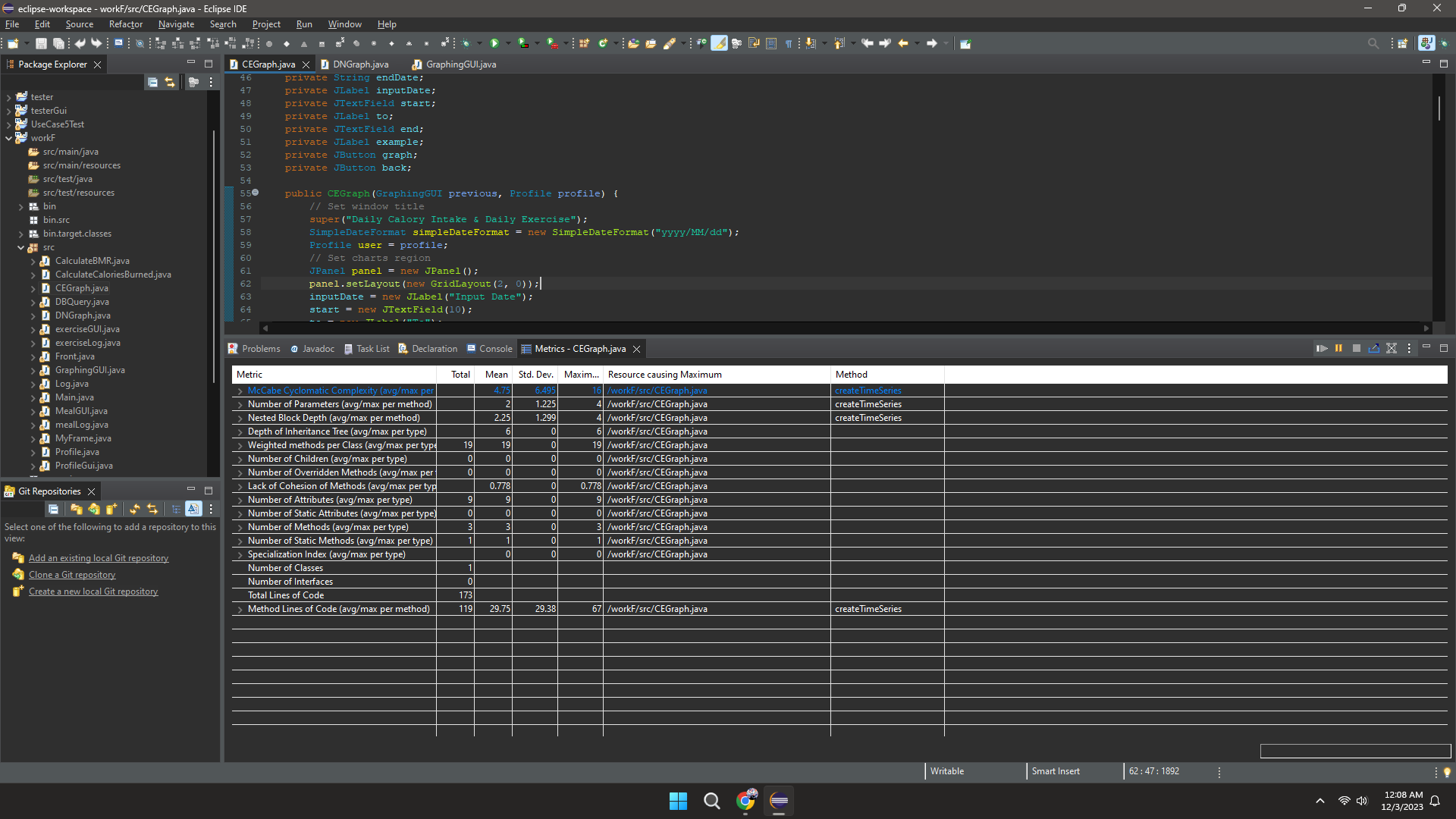
New Metrics:

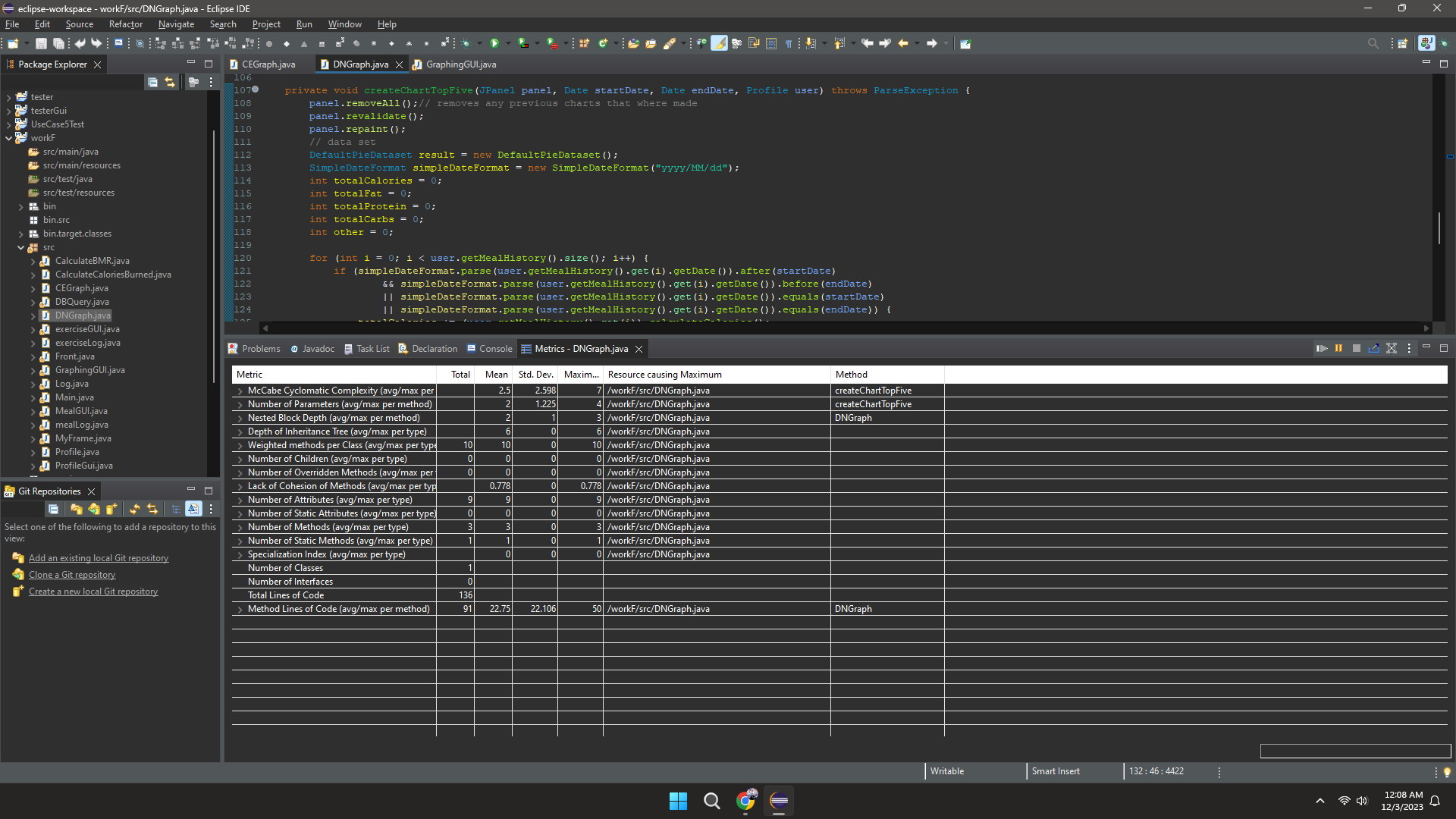
A screenshot of a computer

Description automatically generatedA screenshot of a computer

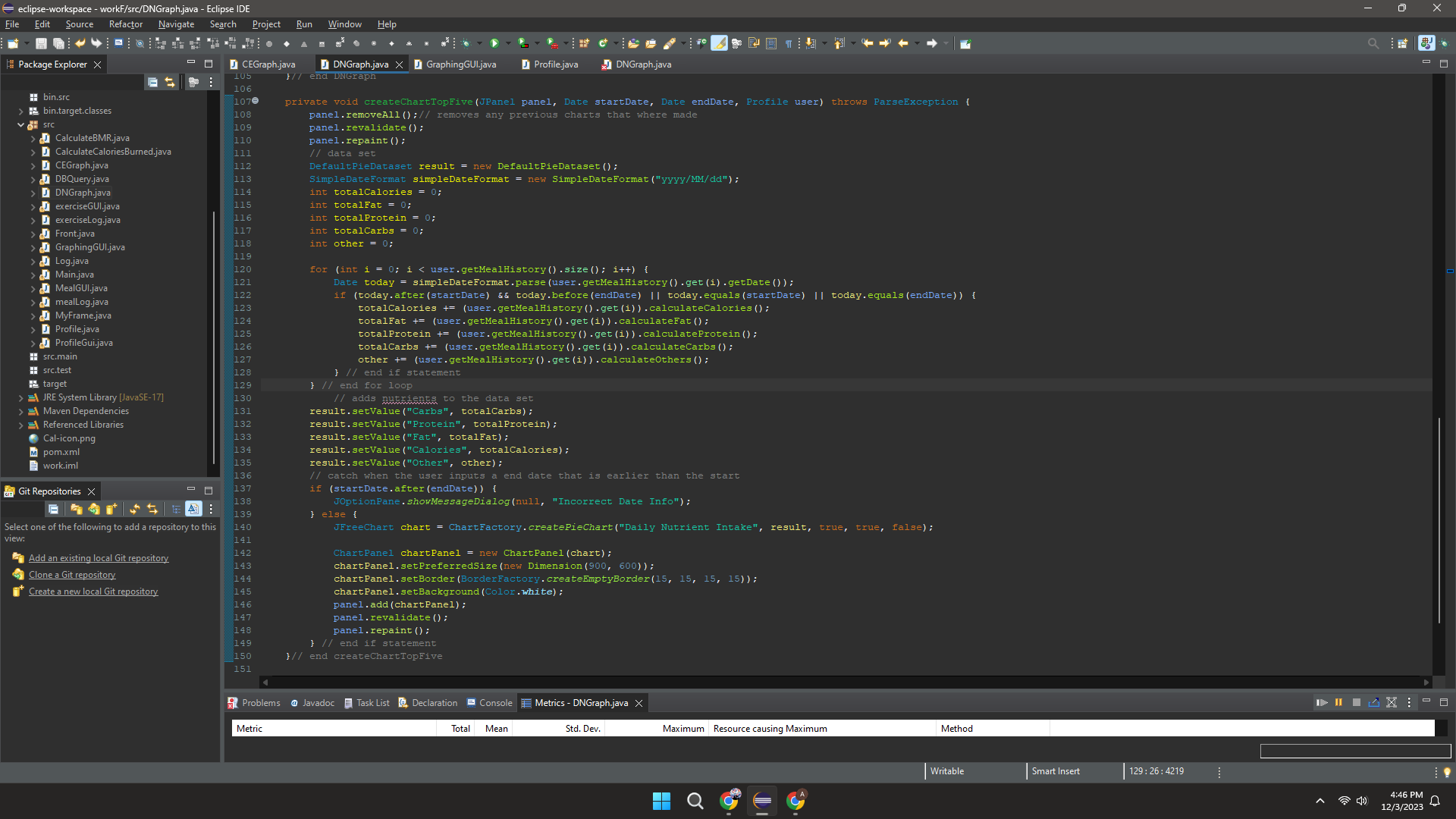
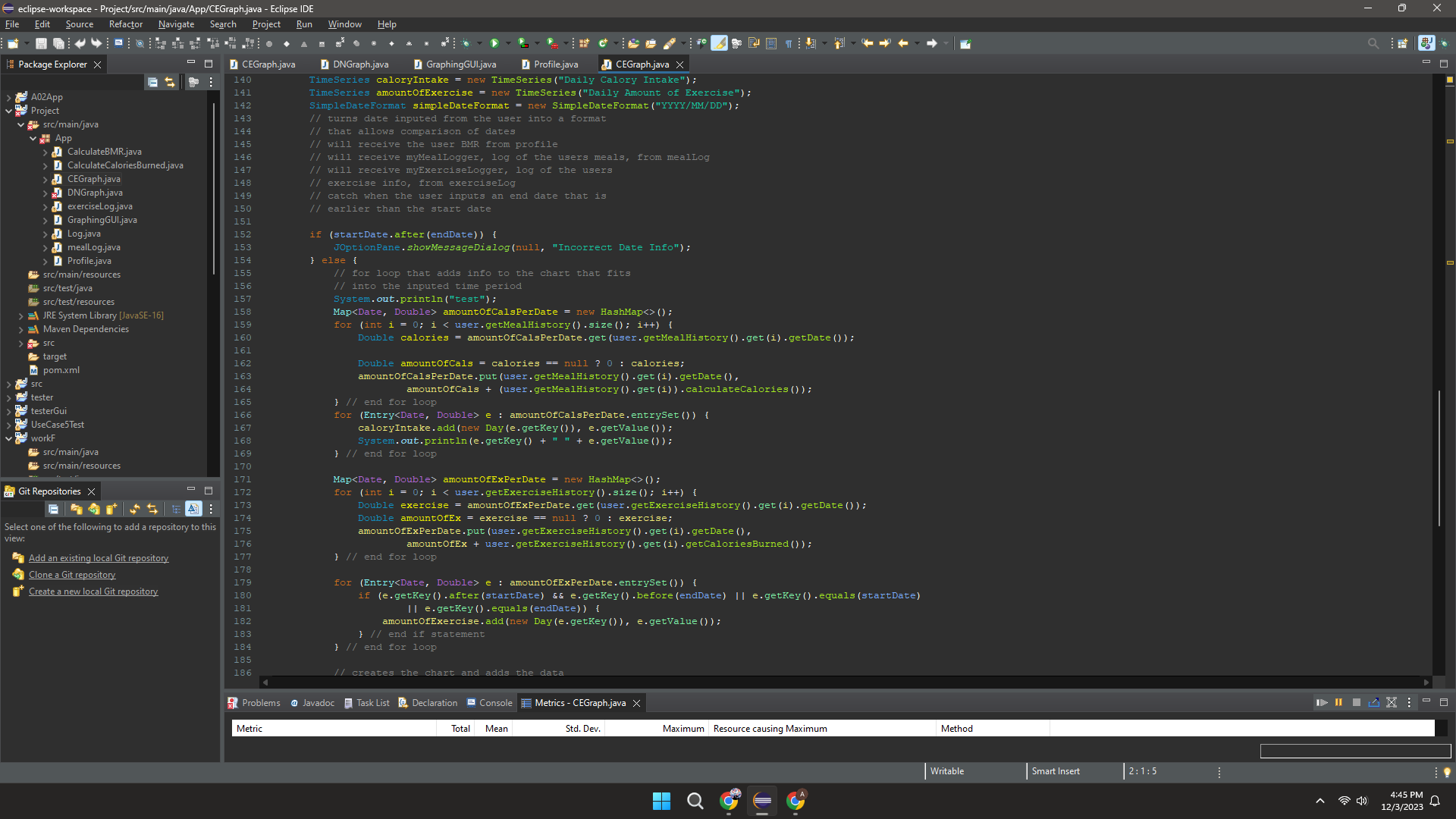
Description automatically generated

**Use case 4-5:**

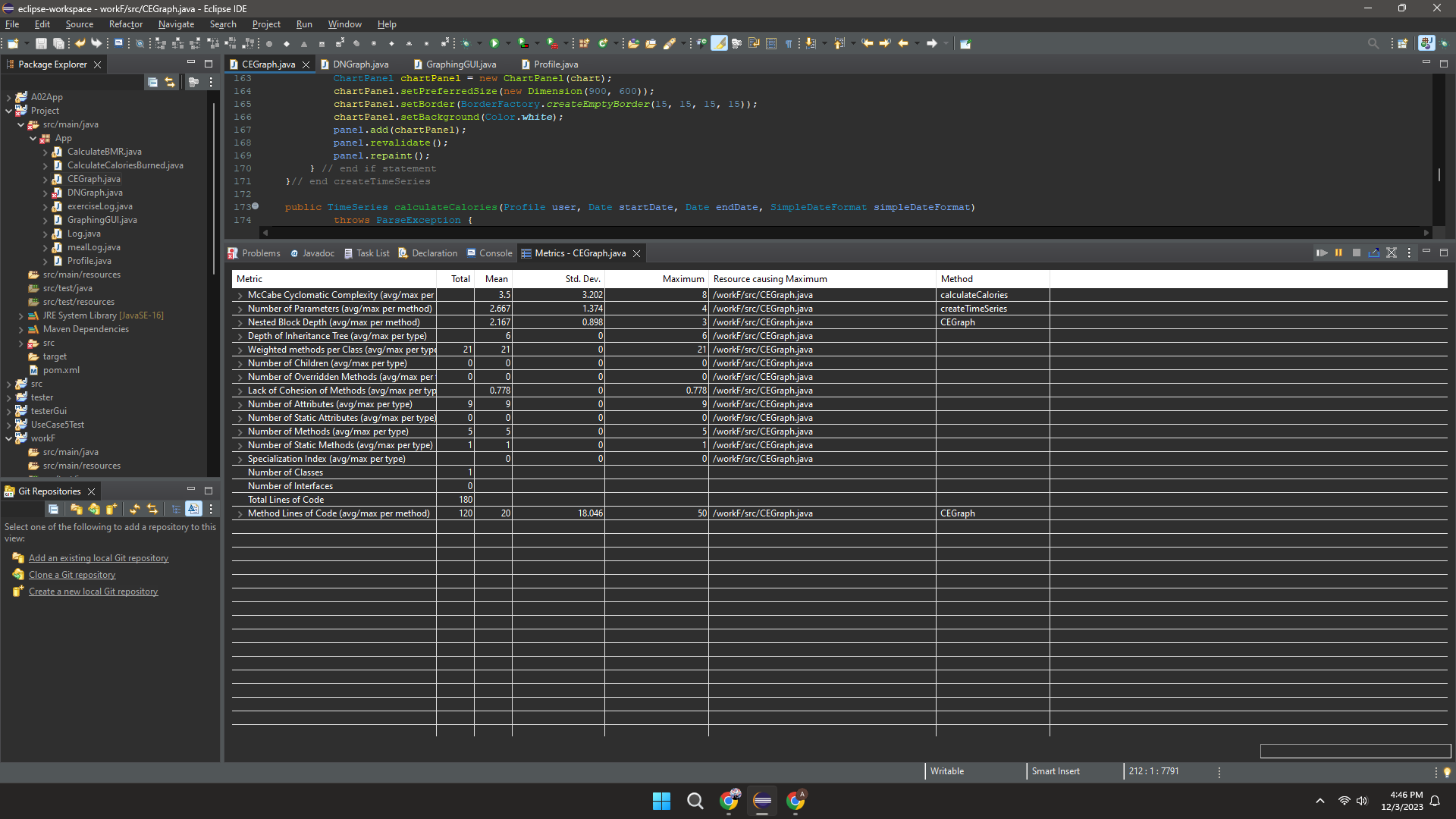
Initial metrics

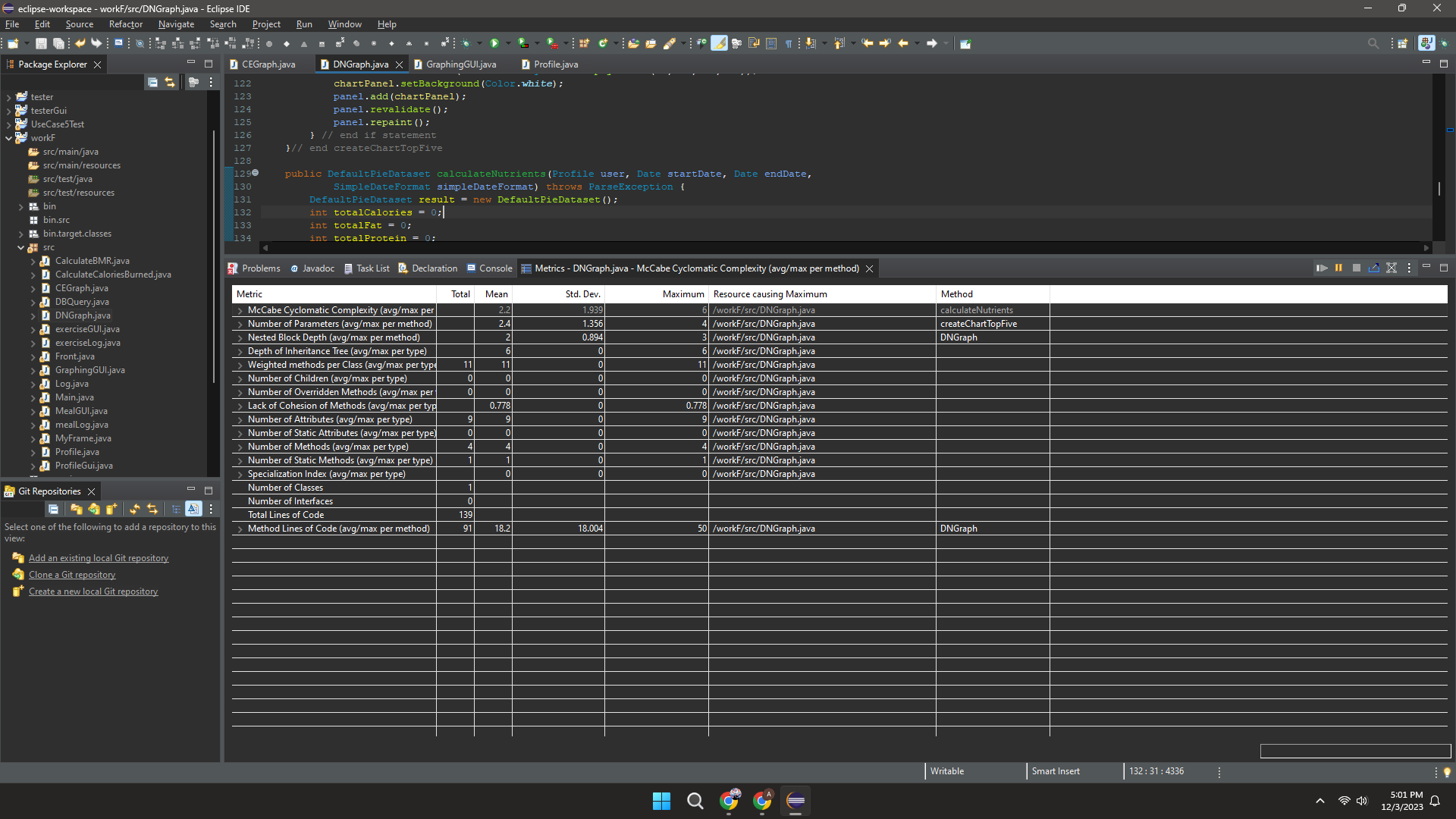


Initial Code

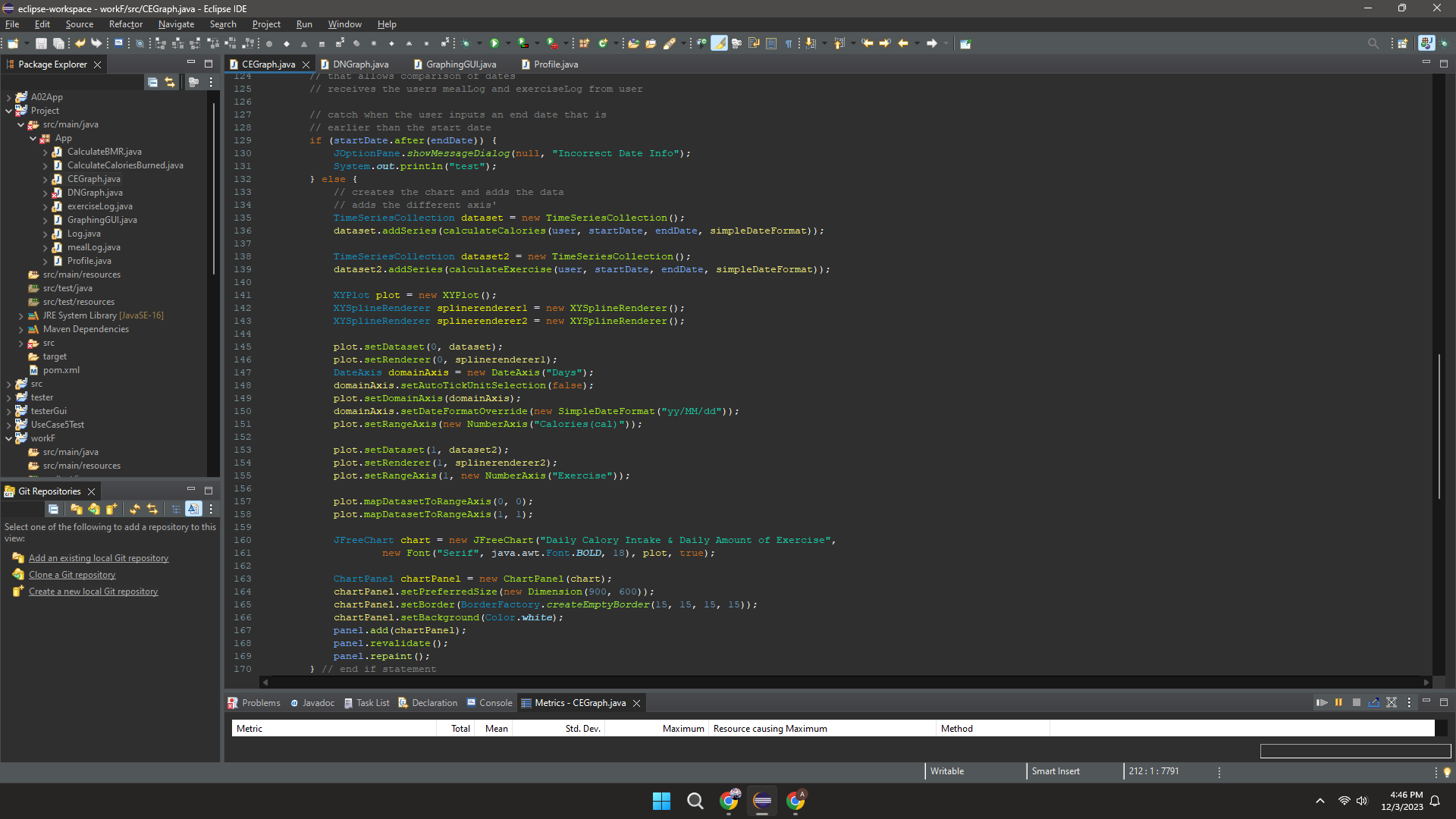


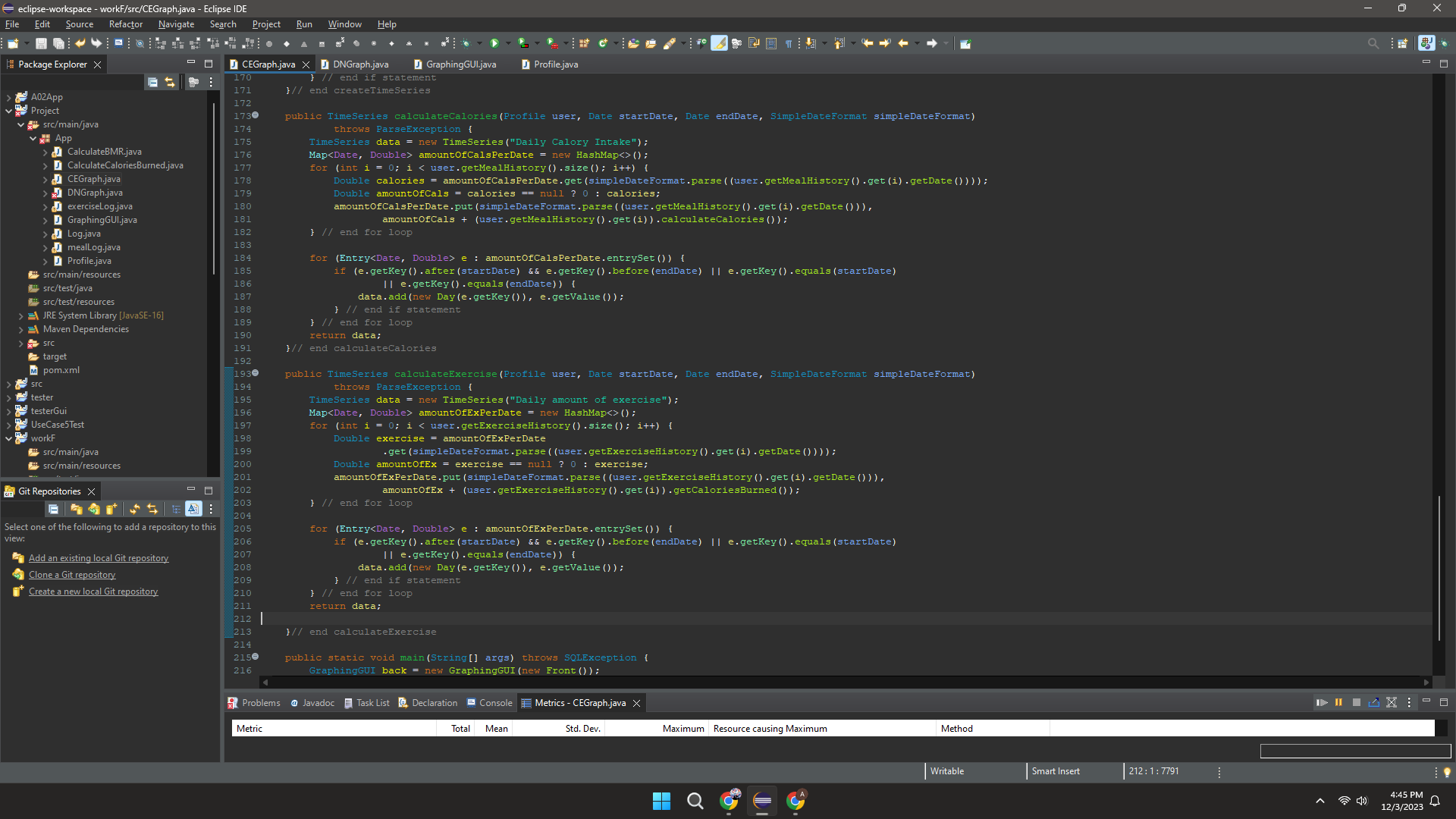
New Metrics

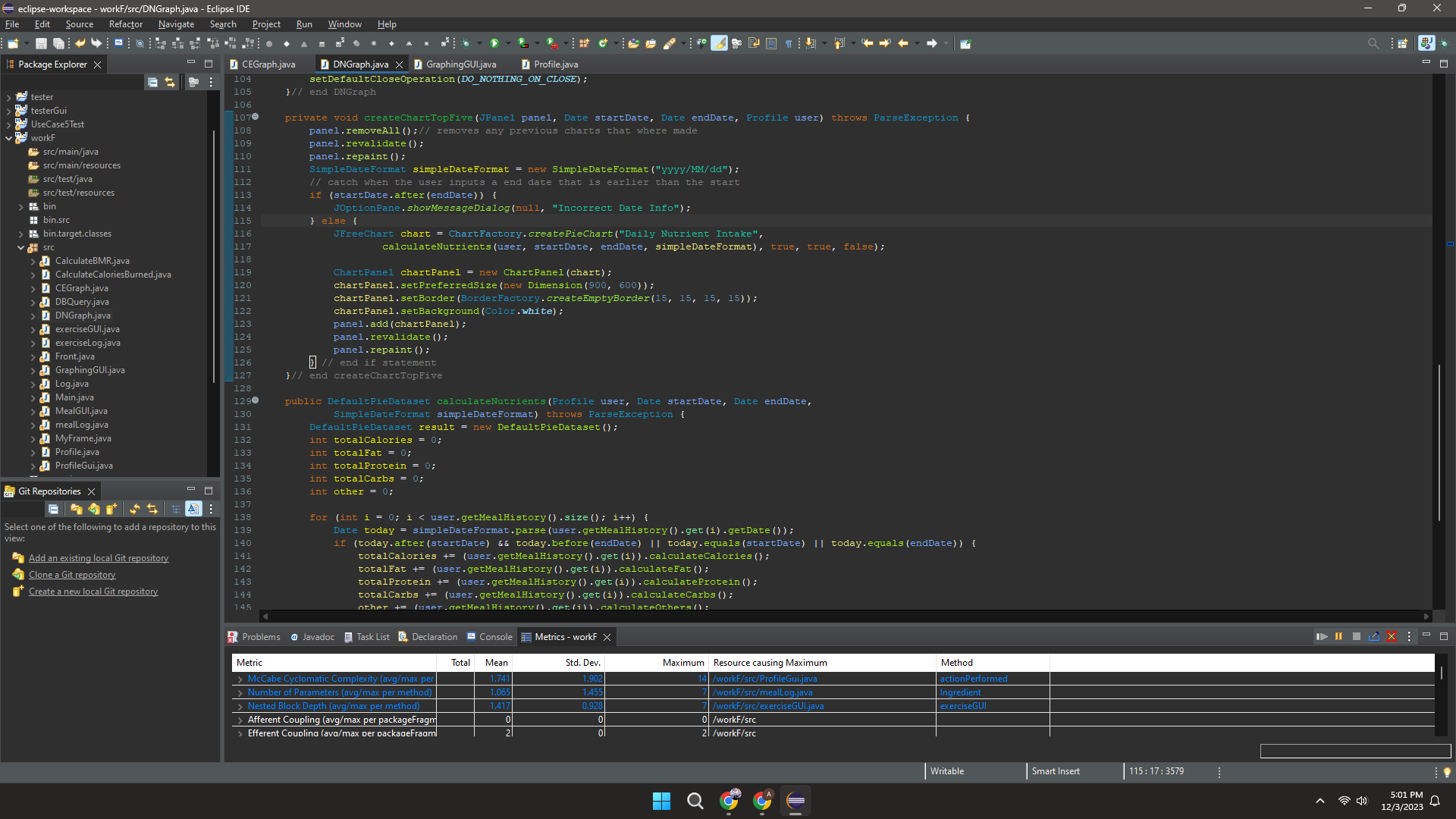
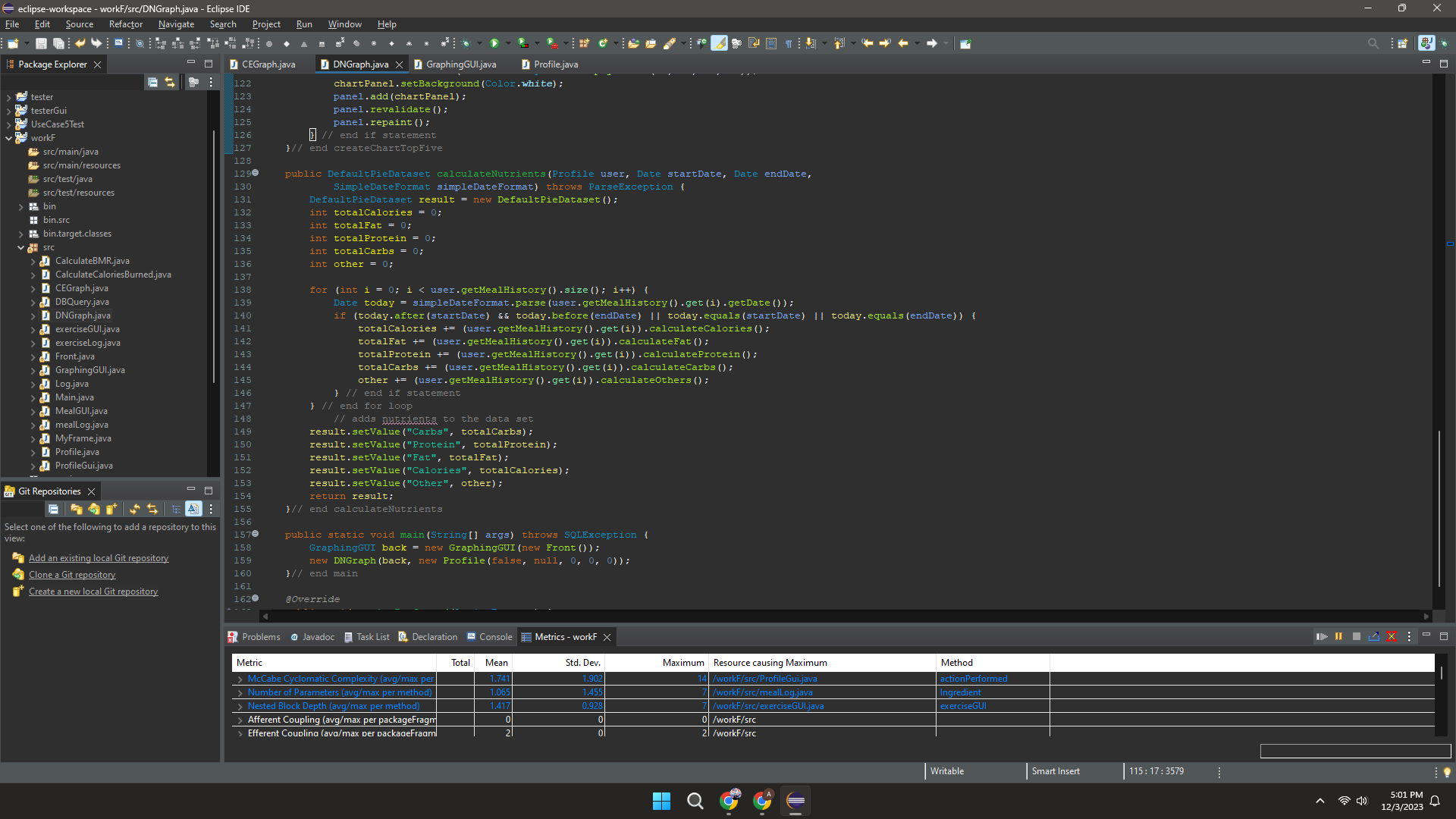




New Code







Code smells

Duplicated Code in CEGraph -> extract method

Long Method for both -> extract method

CEGraph’s method createChart repeats and calculates data in multiple places in the system so it is duplicated code. This can make the code harder to read for others and can mean that it is difficult to alter the code in the future. I used the extract method to separate the createChart method and create separate methods to calculate calorie intake and exercise.

DNGraph’s method createPieChart has a lot of lines of code and high complexity so it is a long method. Again this makes the code harder to read and update in the future and can cause future maintenance problems. I used the extract method to be able to separate the methods. One creates the chart and calls the other method to calculate the data and add it to the dataset.

With the new adjustments the complexity of both CEGraph and DNGraph has decreased.

Code Smell**:** Large Class

Before

A screenshot of a computer

Description automatically generated

After

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Description**:** The class is very long and contains the code for the Gui of profile creation and profile display. Having big methods and lots of code makes it hard to follow along the code as well as making any edits to the code.

Solution: By breaking down the ProfileGui into Profile creation and Profile display into two different classes which would make the class smaller and better modularity.

Justification: Having two classes will enable for better code readability and ensure that the code sticks to the Single Responsibility Principle.

Code Smell: Long Method

Before

A screenshot of a computer

Description automatically generated

After

A screenshot of a computer

Description automatically generated

Description**:** The calculateBMR method contains the logic for calculating all version of the BMR. Having all the code in one method puts a lot of responsibility on it which can cause bugs to occur and makes it harder to spot out errors and bugs.

Solution: The logic for the three BMR calculations can be split into three different methods which calculateBMR can call. Thus sticking to the separation of concerns and single responsibility princple.

Justification: This shifts some responsibility away from calculateBMR to the other methods, which allows for a more comprehensive code that is easy to edit if need be.