**SEPQM ASSIGNMENT – 2018**

**Software Engineering – 3rd Year – Semester 1**

|  |  |
| --- | --- |
| **Student ID** | **Name with initials** |
| IT16113978 | L.M.R. Jayasinghe |
| IT16139640 | K.A.N.W. Hemachandra |
| IT16061712 | Gigarthan Vijayakumaran |
| IT16127838 | L.C.E. De Silva |

Contents

[**Task 1 - Static Analysis Tool** 2](#_Toc514589444)

[**Tool Introduction** 2](#_Toc514589445)

[**Pros and Cons** 3](#_Toc514589446)

[**Scenario** 4](#_Toc514589447)

[**Scenario Execution** 4](#_Toc514589448)

[**Task 2 - Bug/Issue Tracking Tool** 5](#_Toc514589449)

[**Tool Introduction** 5](#_Toc514589450)

[**Pros and Cons** 6](#_Toc514589451)

[**Scenario** 7](#_Toc514589452)

[**Scenario Execution** 7](#_Toc514589453)

[**Task 3 - Code Coverage Tool** 8](#_Toc514589454)

[**Tool Introduction** 8](#_Toc514589455)

[**Pros and Cons** 9](#_Toc514589456)

[**Scenario** 10](#_Toc514589457)

[**Scenario Execution** 10](#_Toc514589458)

# **Task 1 - Static Analysis Tool**

## **Tool Introduction**

**PMD**

PMD is an open source, rule based static analysis tool for JAVA source code. Tool comes with a default set of rules.

It checks for potential errors such as,

1. Dead code (unused local variables, parameters, private methods)
2. Duplicate code (copy-pasted code)
3. Possible bugs (empty try/catch blocks, finally/switch statements)
4. Overcomplicated expressions (unnecessary if statements/ for loops that could be while loops)
5. Suboptimal code (wasteful String/StringBuffer usage)

Therefore, the PMD issues reports for inefficient code and/or bad programming habits which can reduce the performance and maintainability of the program.

Additionally, it allows users to customize analysis rules in a convenient manner.

The tool can be used in command line interface and also in integrated development environments as Eclipse.

Tool is generally quite effective and has proven itself as a useful static analysis tool for both large and small code bases.

<http://cecs.wright.edu/~pmateti/Courses/7140/Lectures/SecureProg/static-analysis.html>

<http://www.cs.cmu.edu/~aldrich/courses/654/tools/hsu-pmd-07.pdf>

## **Pros and Cons**

**Pros**

1. Good documentation helps the installation process much easier.
2. Can add/modify your own rulesets, run mixtures of rulesets for each run, use optional arguments to run PMD on codebases using new (and old, for regression testing) JDKs, etc.
3. Quite scalable, runs well for very large source code bases.

**Cons**

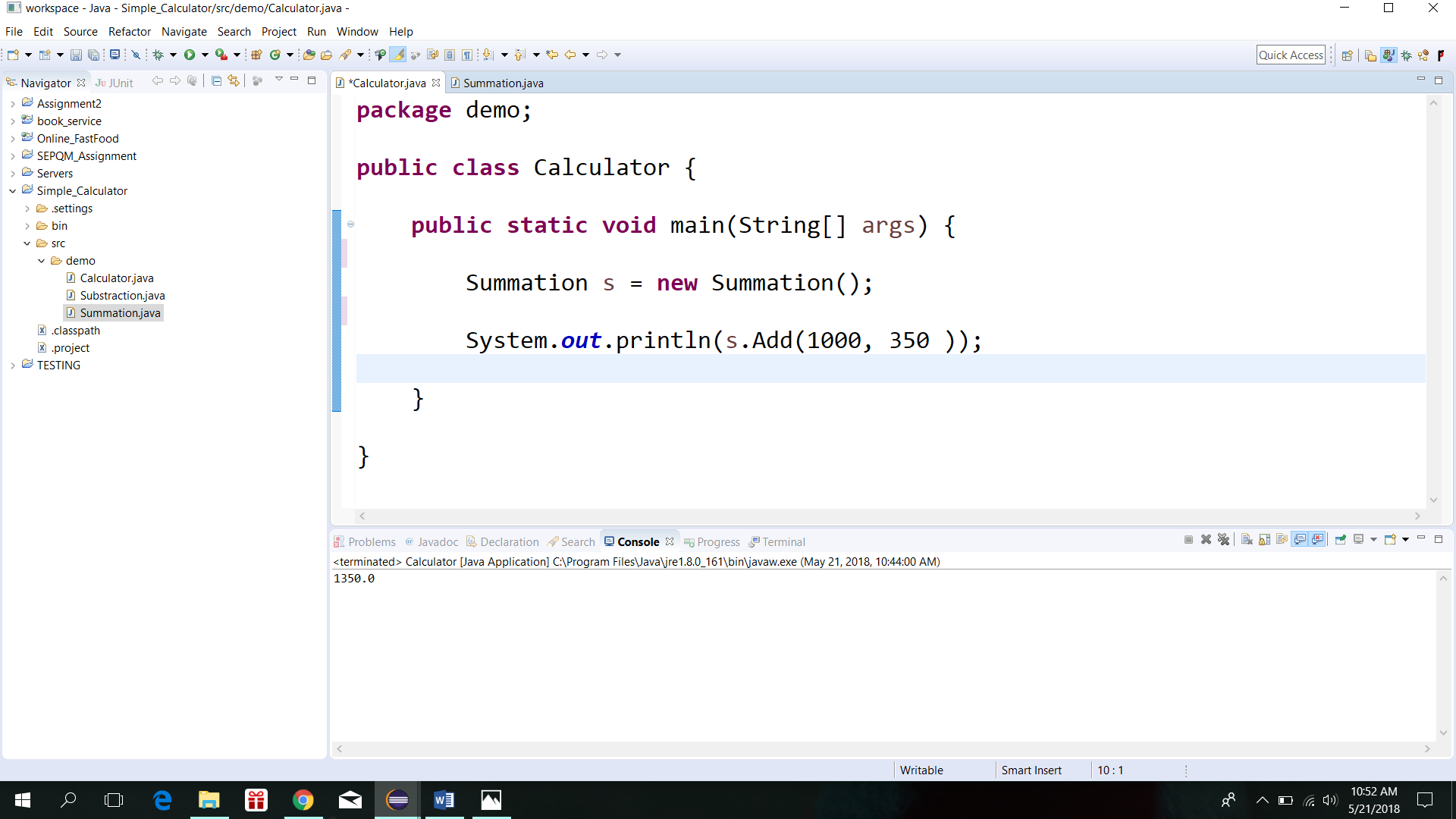
1. Adding code to create a new rule requires an understanding of ASTs, since that's what it's based on.
2. Cannot find runtime issues. Works strictly only for code.
3. Does not validate synchronization mechanisms and threading concepts

## **Scenario**

We have implemented a simple calculator using java to demonstrate static analysis in code review.

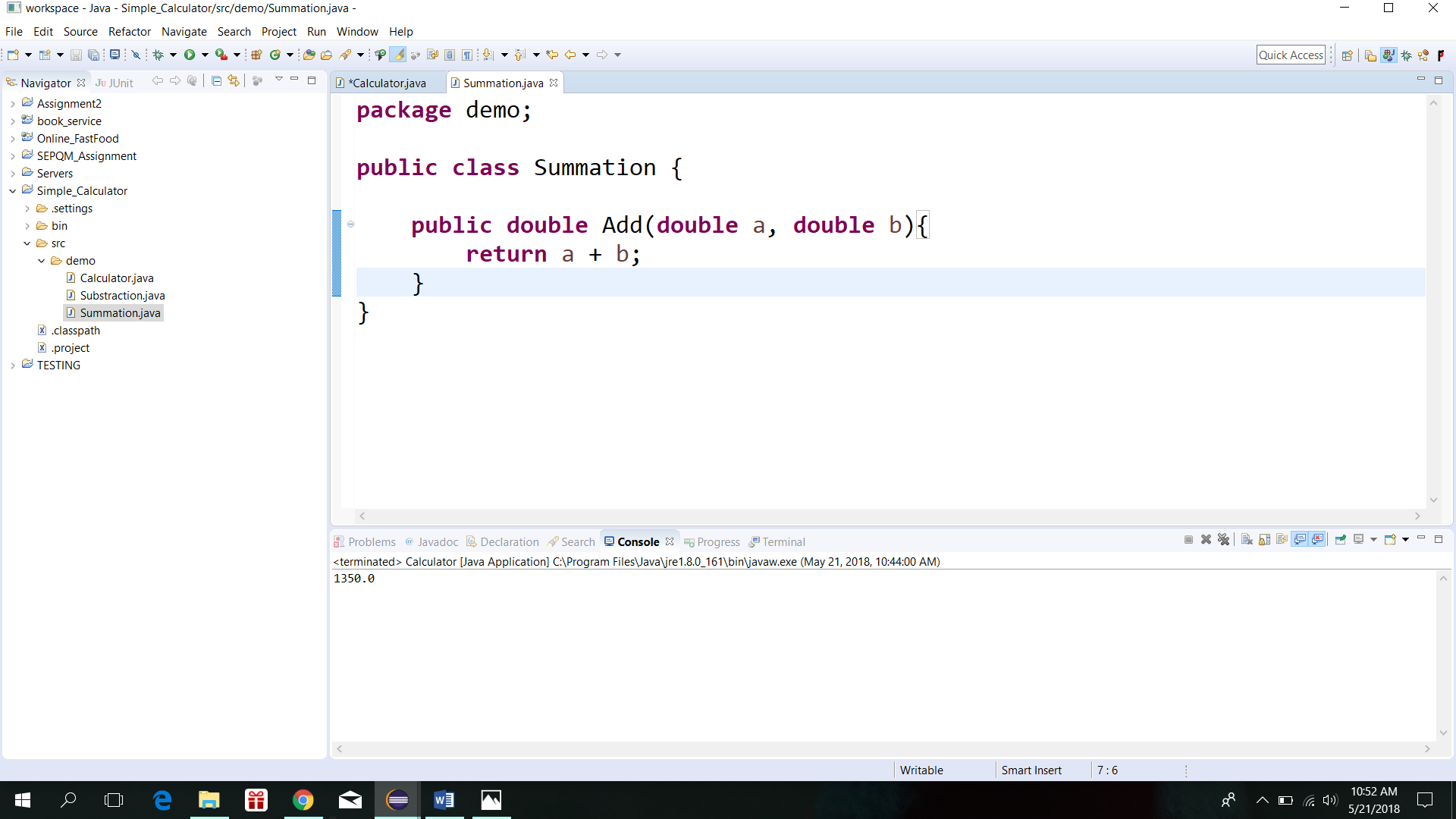
In here we demonstrate how code review is applied using the static analysis tool PMD, in the ADD method of the calculator class.

## **Scenario Execution**



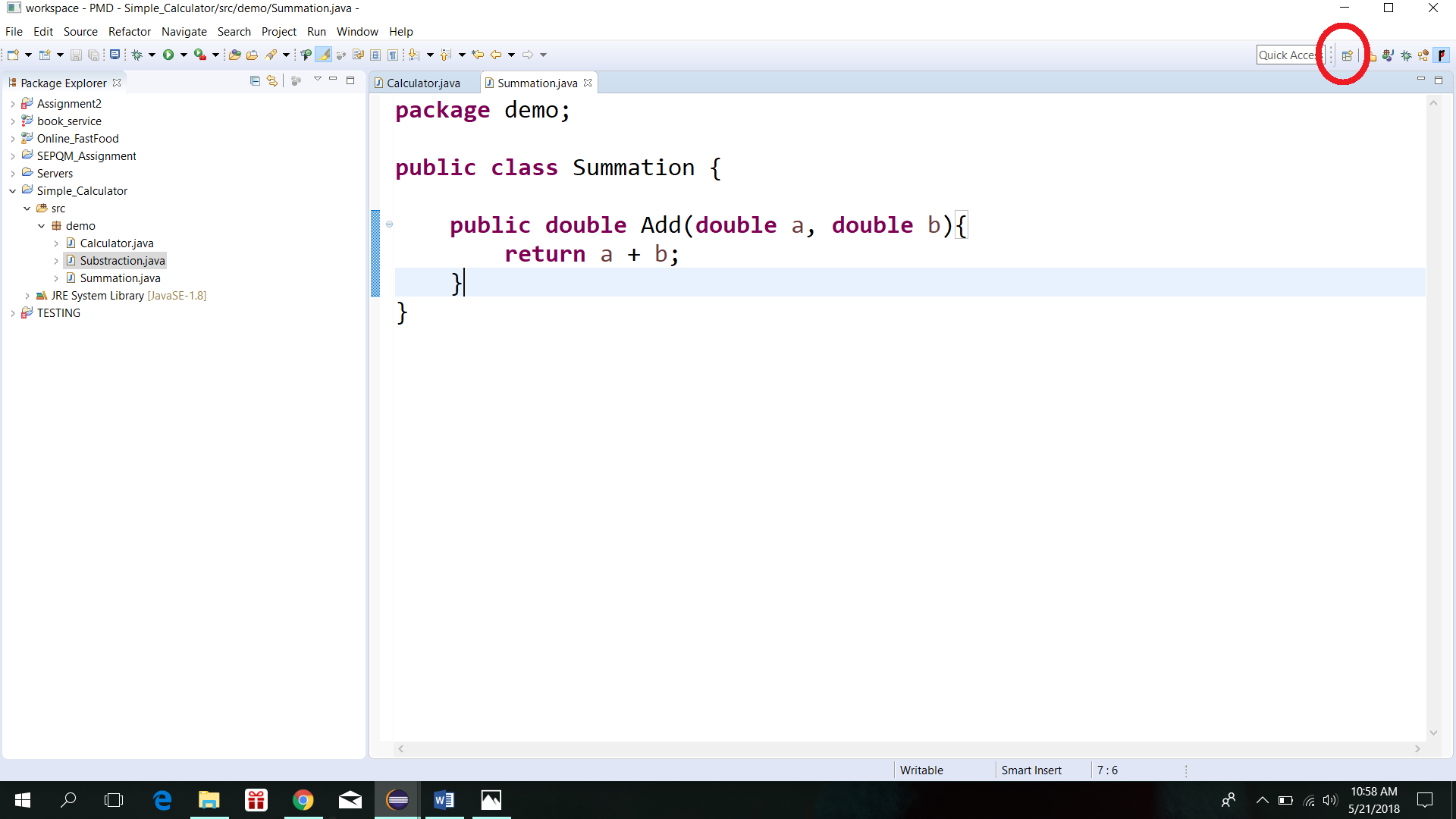
**Main class, Calculator which is used to call the ADD method**

**(before doing static analysis)**

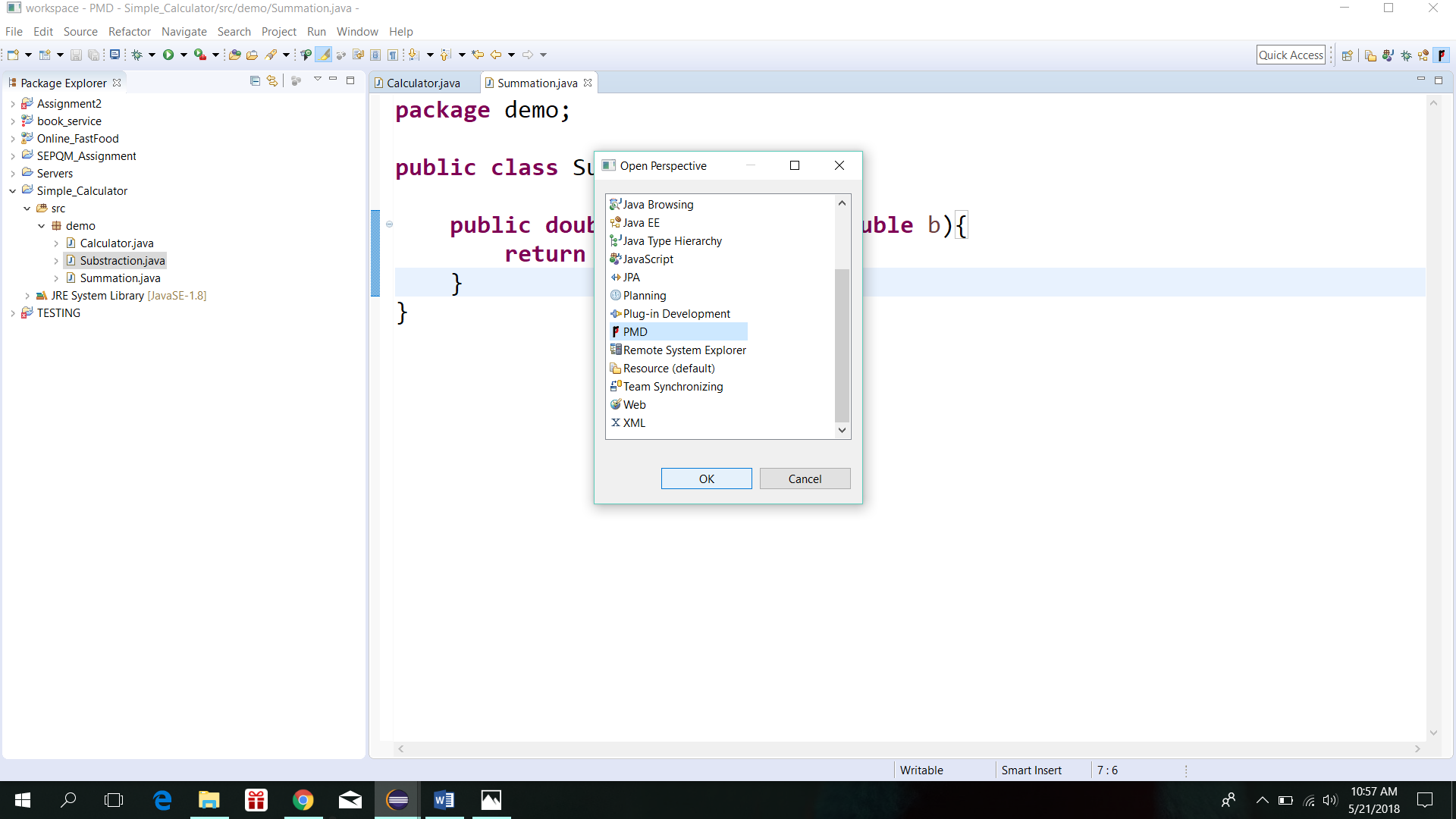


**Summation class, which is used to implement the ADD method**

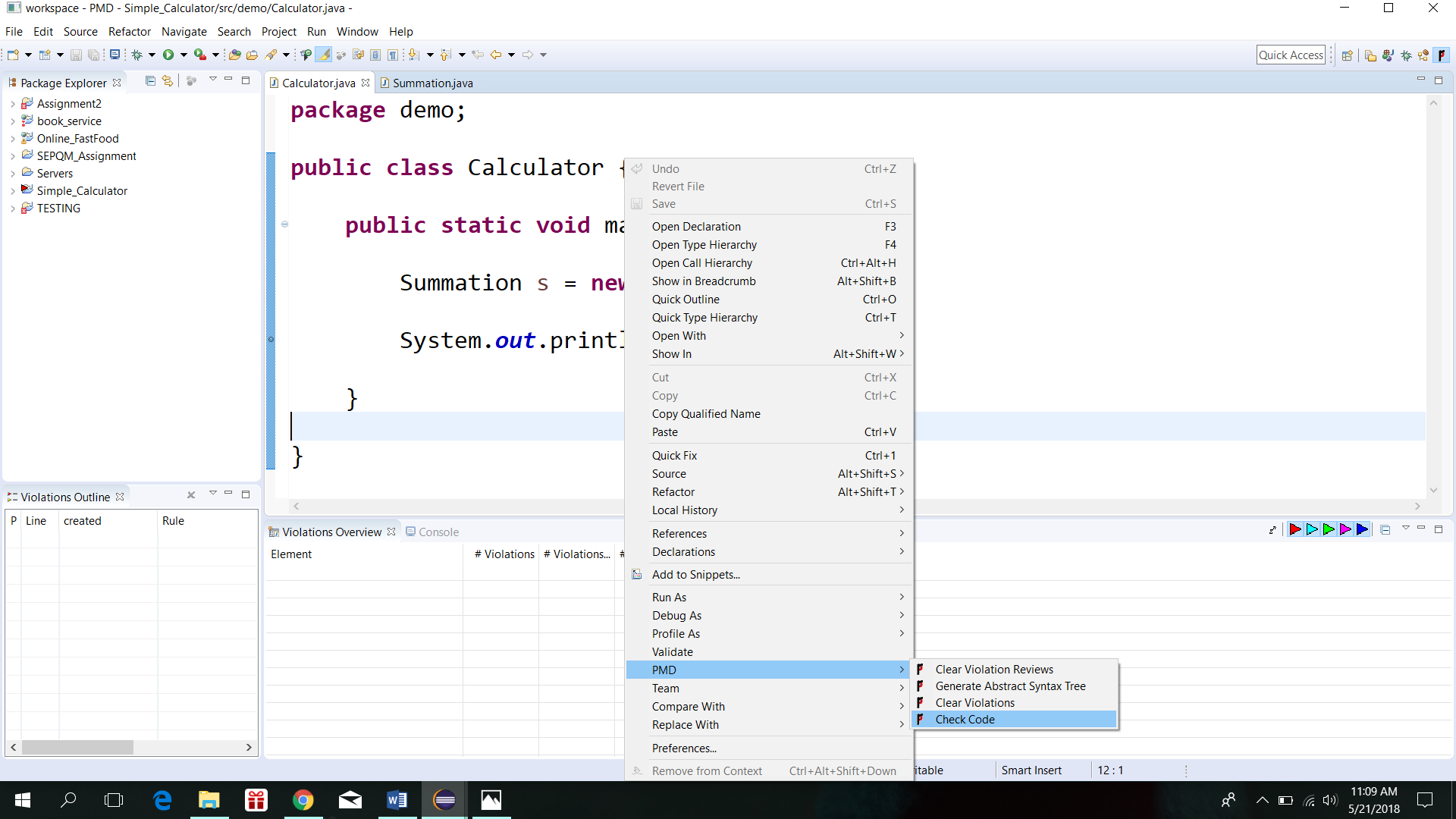
**(before doing static analysis)**

****

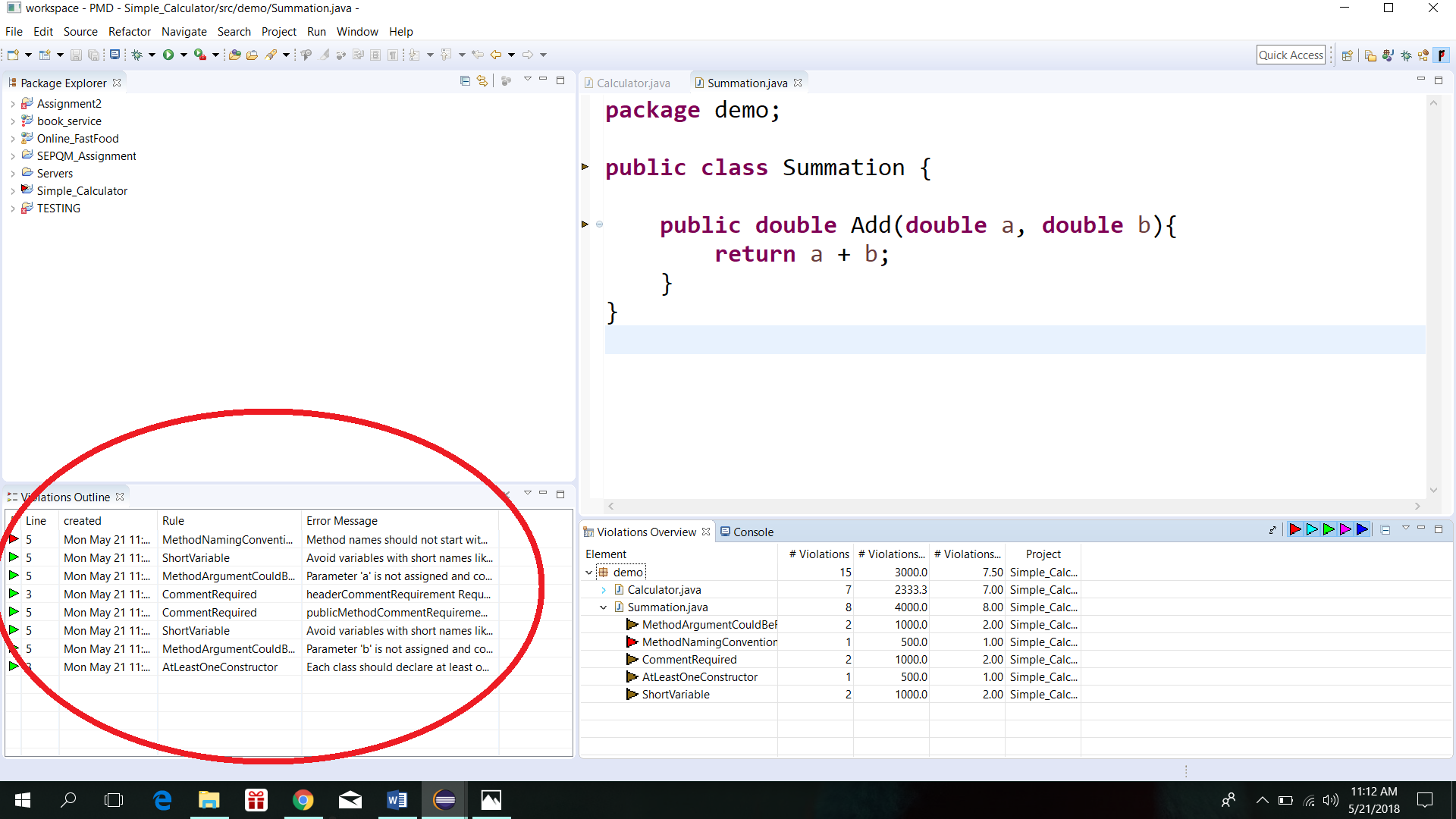
**Click on open perspective .**



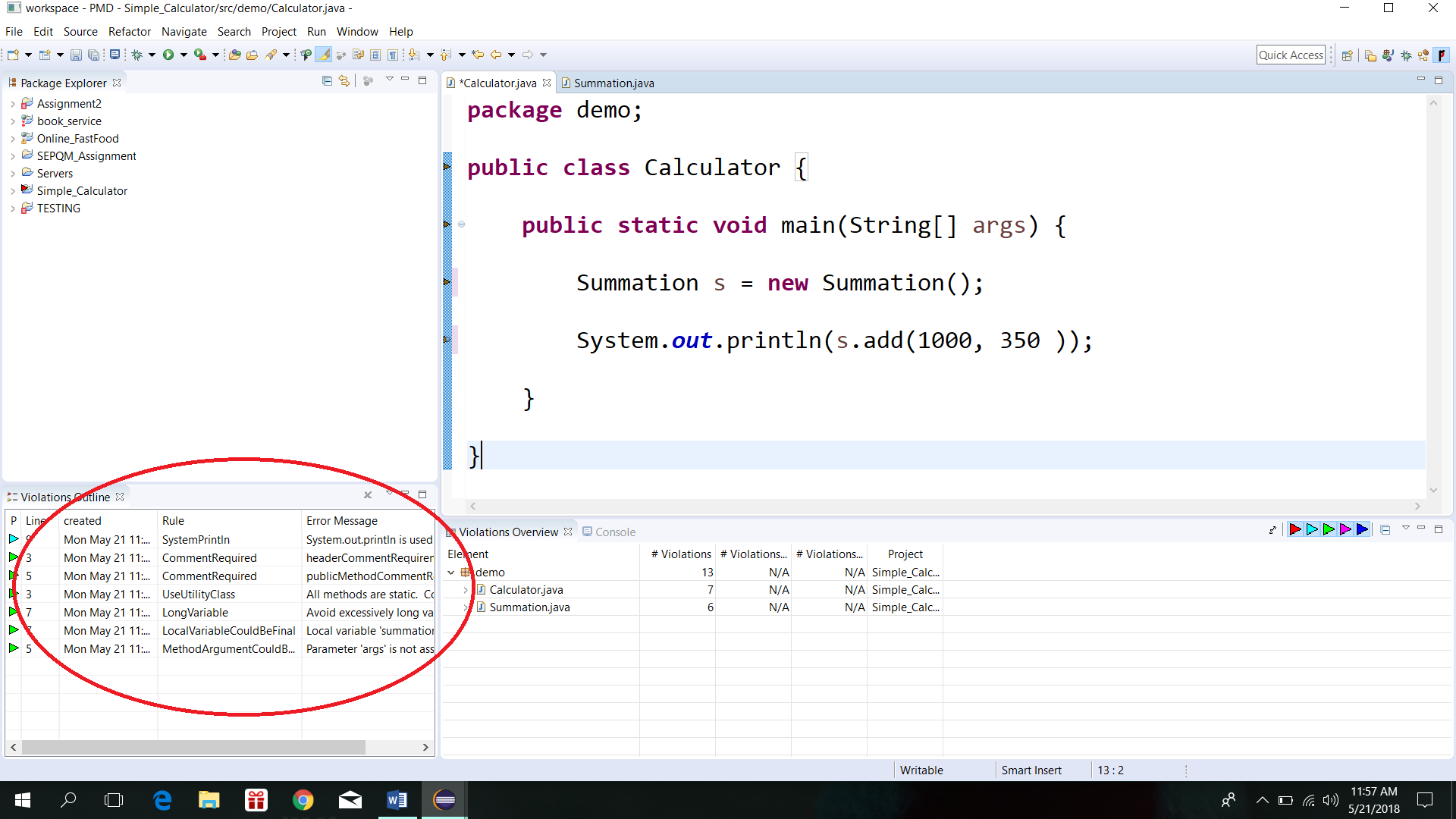
**Select PMD and click ‘OK’ to open the PMD perspective in Eclipse.**



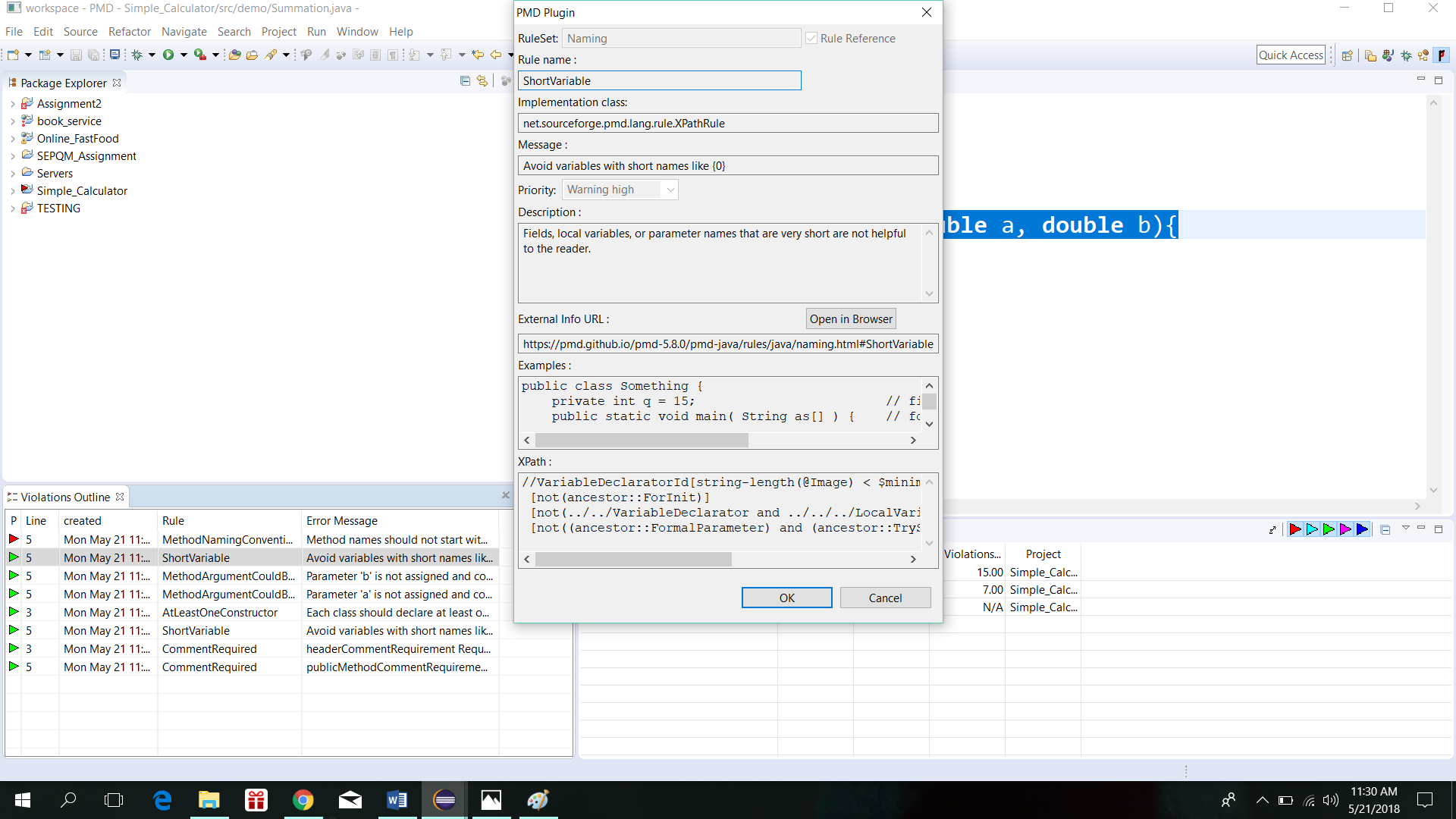
**To run the static analysis right click and select PMD , Check Code.**



**Result after completing static analysis of Summation.java class which shows errors where code quality is low.**

****

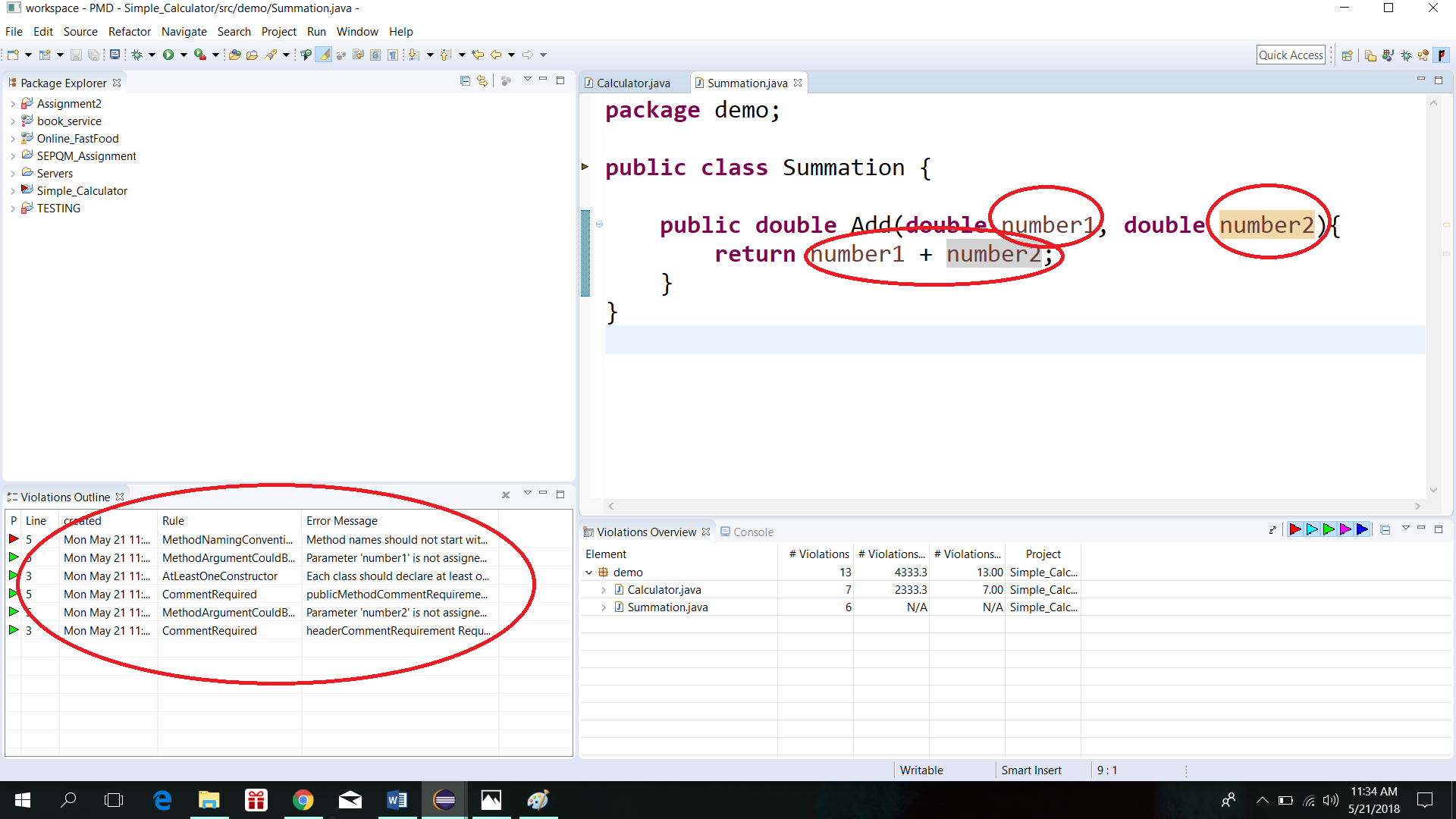
**Result after completing static analysis of Calculator.java class which shows errors where code quality is low.**



**Detailed description of the error can be viewed by right clicking the error on the violation Outline window and selecting show details.**

The above identified errors have to be fixed to improve the quality of the code.

The following shows the code after the errors are fixed.



**As highlighted, after fixing certain errors and performing static analysis those errors will disappear.**

# **Task 2 - Bug/Issue Tracking Tool**

## **Tool Introduction**

**JIRA**

An open source tool built to track and manage bugs in software development. It also provides issue tracking and project management functions. Provides complete set of recording, reporting, workflow and other convenience-related features.

It can be integrated directly to the development environment.

Supports agile projects as well.

<https://www.softwaretestinghelp.com/popular-bug-tracking-software/amp/#4_JIRA>

<https://www.atlassian.com/software/jira/bug-tracking>

## **Pros and Cons**

**Pros**

1. Easy to use.
2. Usable interface for tasks
3. Integrates with other systems
4. Can be accessed anywhere since it is web based

**Cons**

1. Only JIRA users can access that environment
2. Cloud max storage size of 25GB per site
3. Can be overly complex for small teams

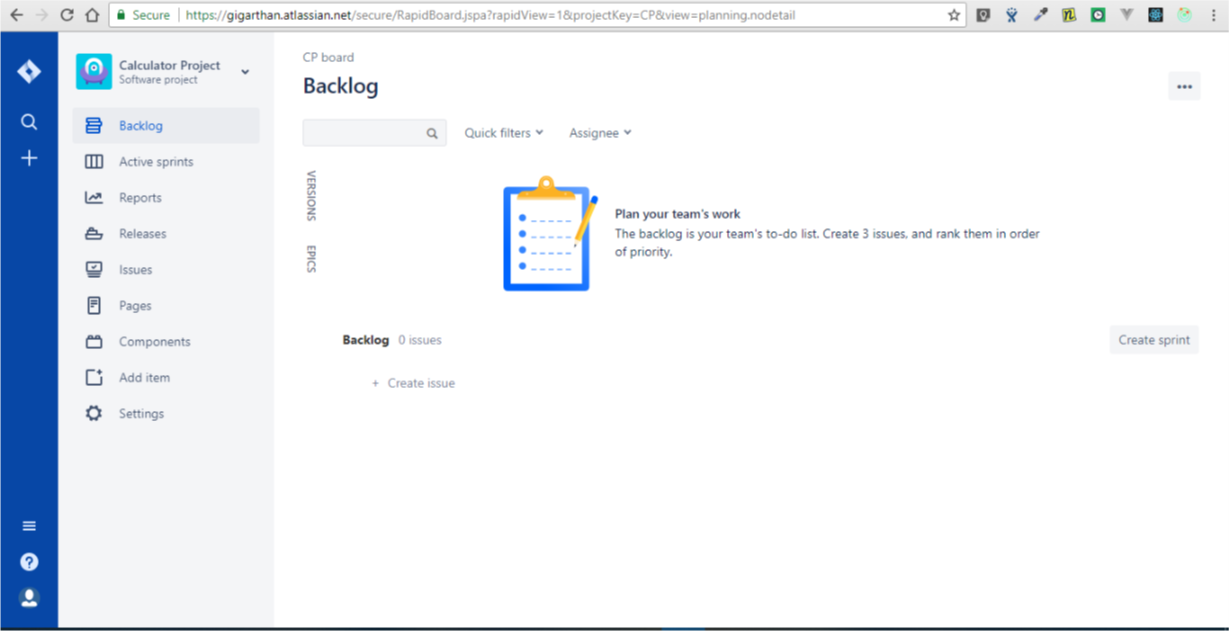
## **Scenario**

Jira Bug tracking system is used to track and manage bugs in the simple calculator app.

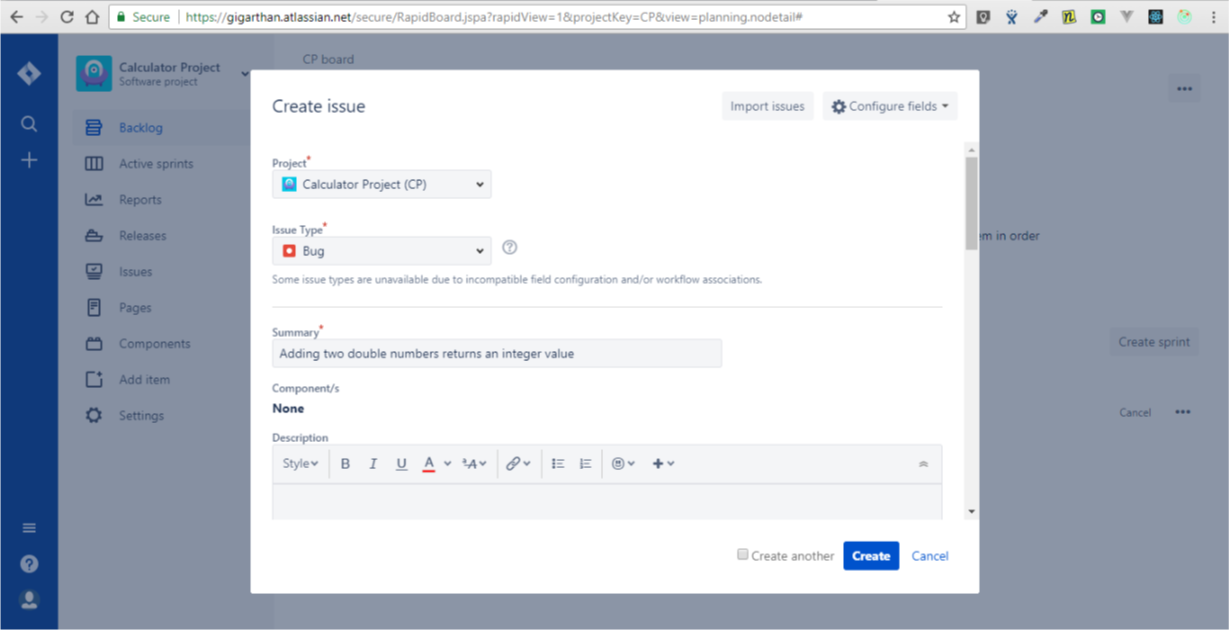
In here, we demonstrate how a bug can be issued and how the workflow goes until a bug is fixed.

## 

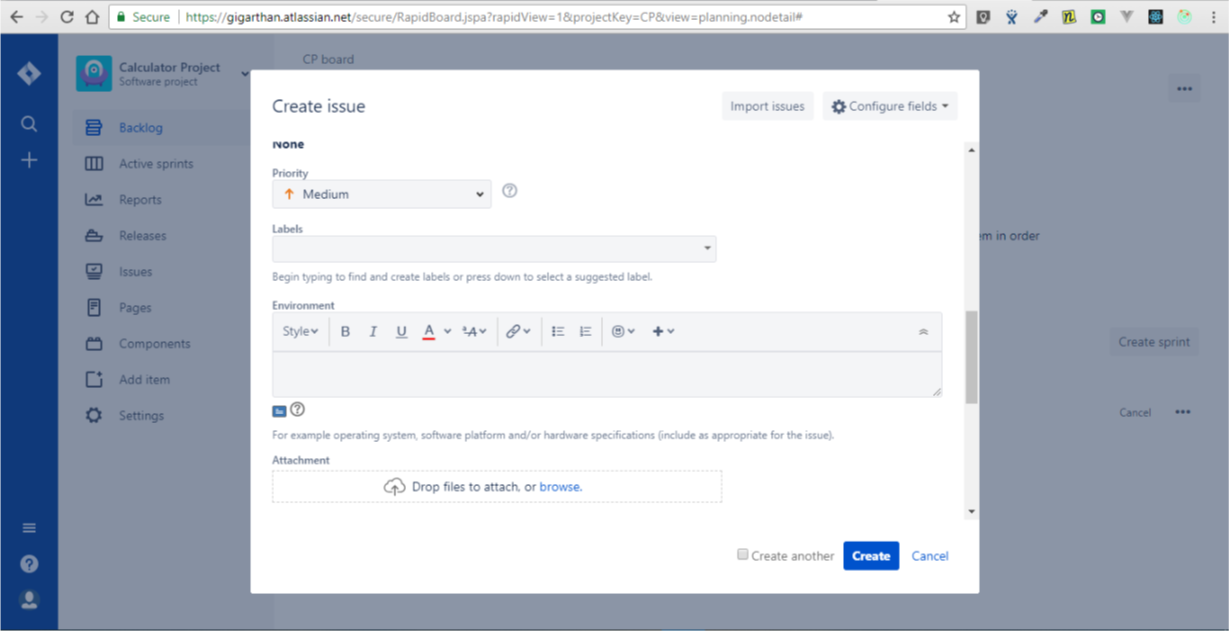
## **Scenario Execution**



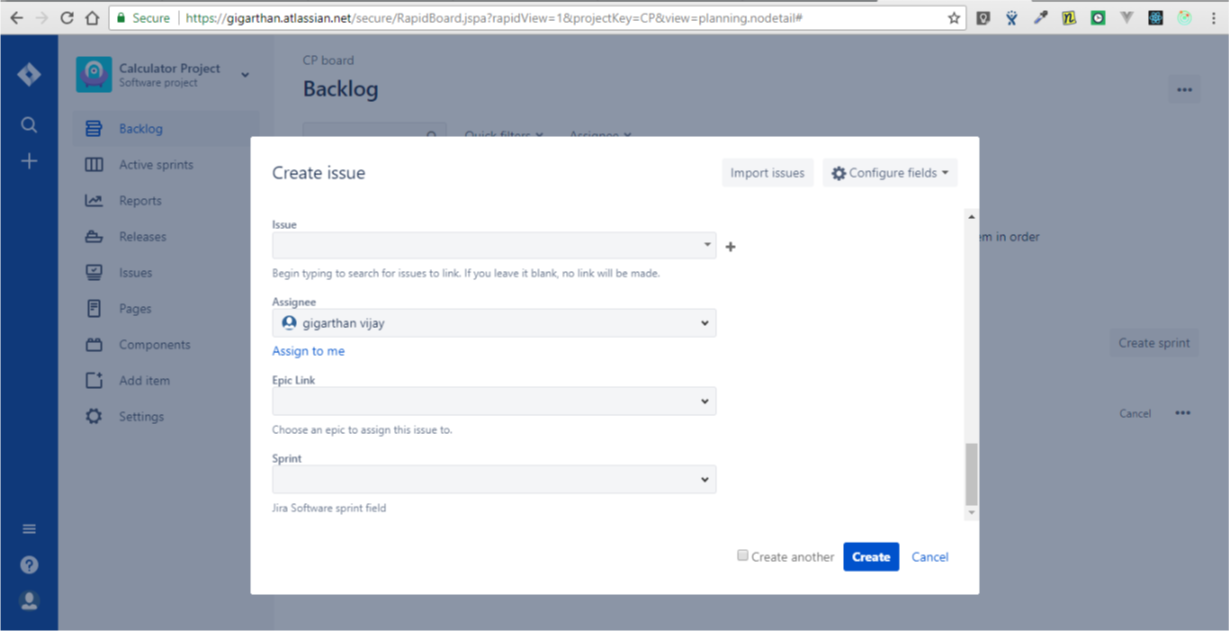
This is the dashboard page for our calculator project in Jira. To create a sprint, we need to create some issues in the backlog and assign them to a sprint. To create an issue click on the create issue placeholder under the Backlog in bottom half of the page.



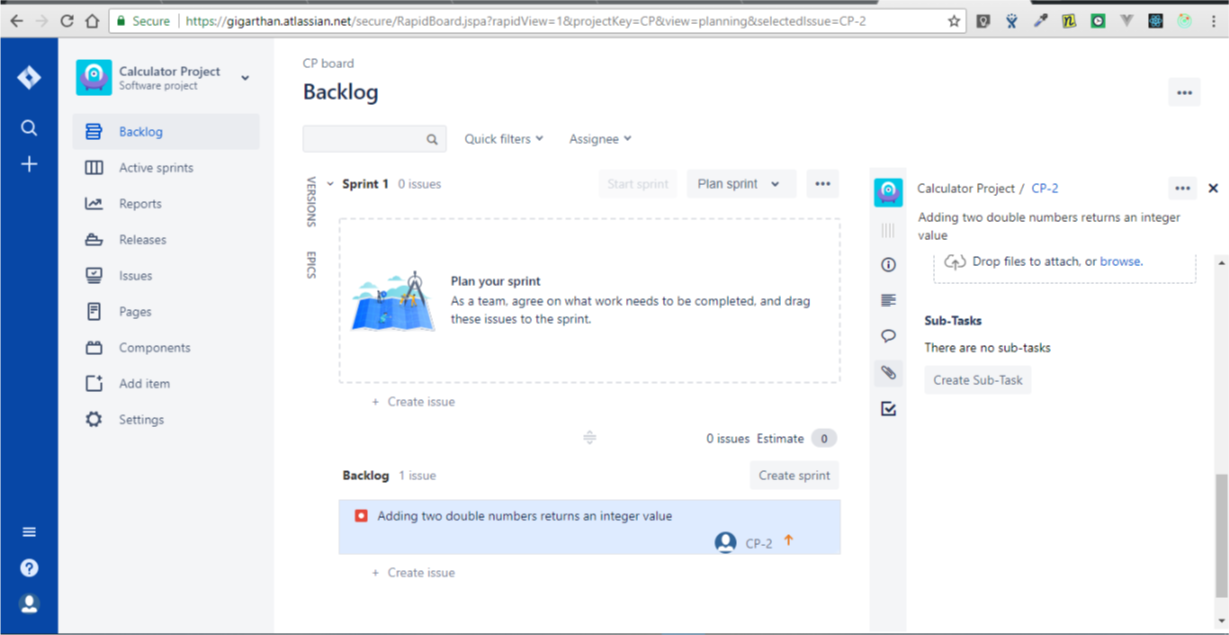
The details of the issue can be provided here. First we need to assign which project the bug is in, in this our project is the Calculator Project. Next we need to select the issue type, there are three types of issues in Jira. Story, Bug, Epic are those. We will select the type Bug here. Next we could give a summary about the bug.



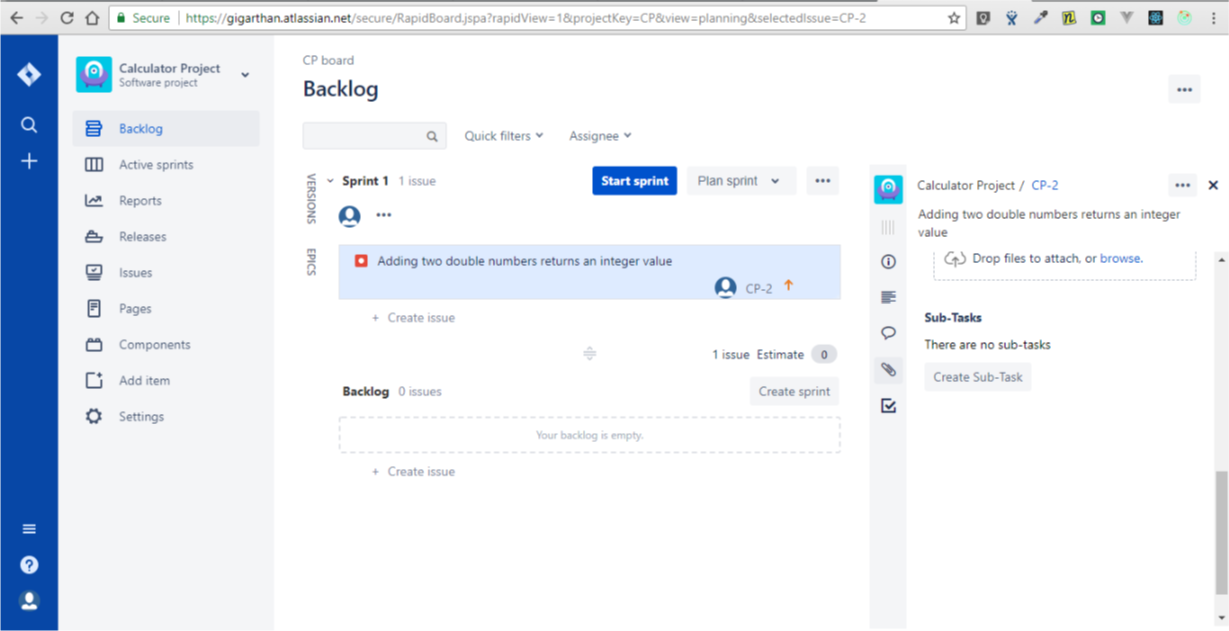
Next, bug priority should be selected in the Priority dropdown. We can choose between Highest, High, Medium, Low, and Lowest. After that we could select any labels, environment in which the bug was found, any other attachments.



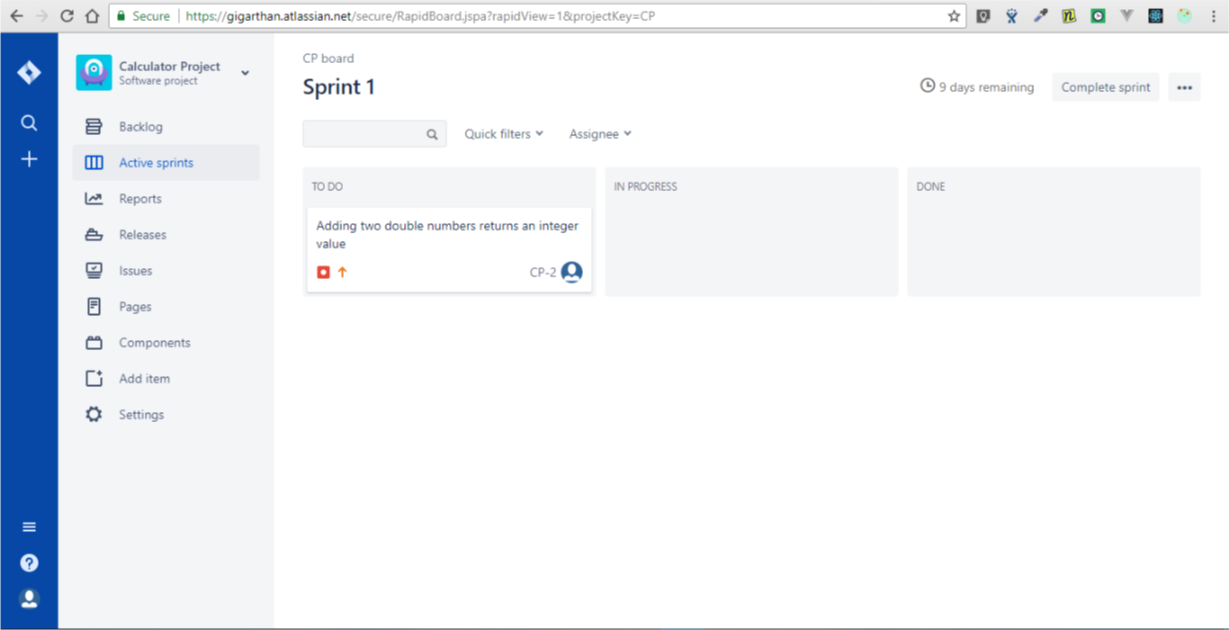
Once we scroll down, we can include more details like any related issues with this bug and we can assign a developer to debug the bug. Once important details are filled, we can create the issue by clicking the create button.



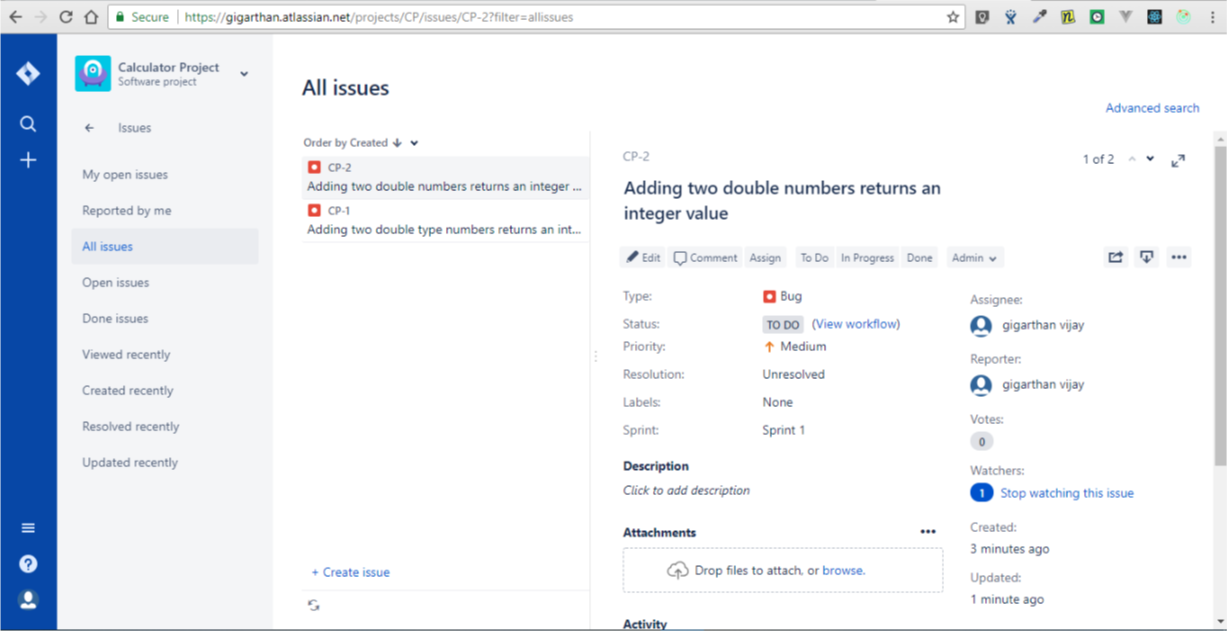
We can check whether our bug has been created in the backlog and we can see from the above image that our bug has been created. Next we need to assign the bug to a sprint to start its development.



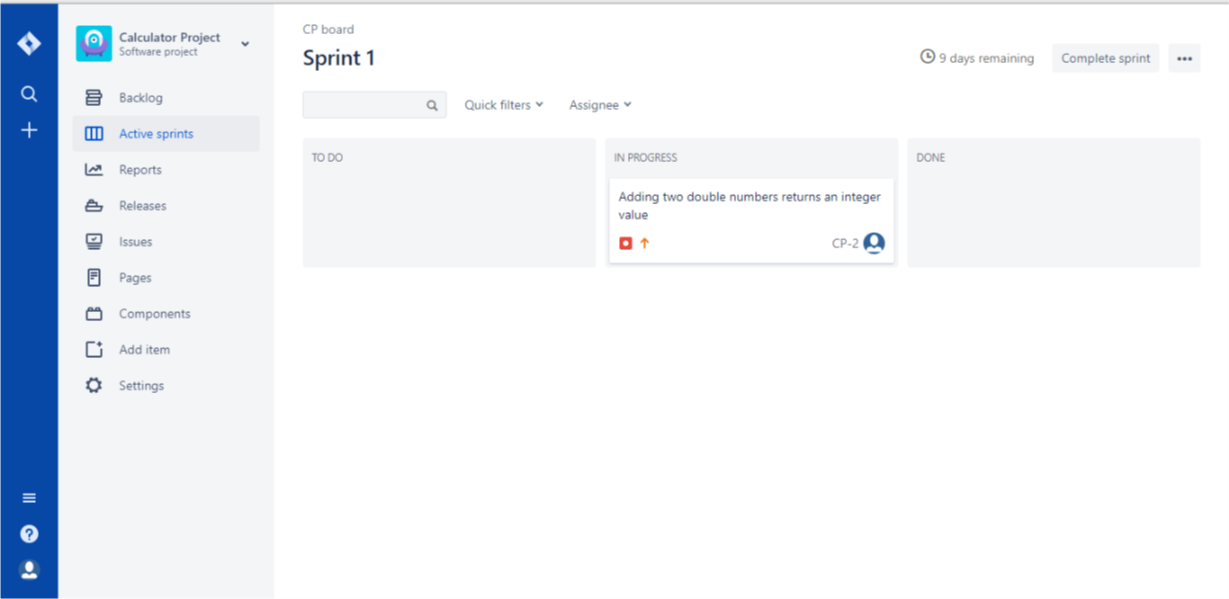
We have assigned the bug to Sprint 1 by dragging and dropping into Sprint 1. Now we can start our sprint by clicking the start sprint button in top right corner.



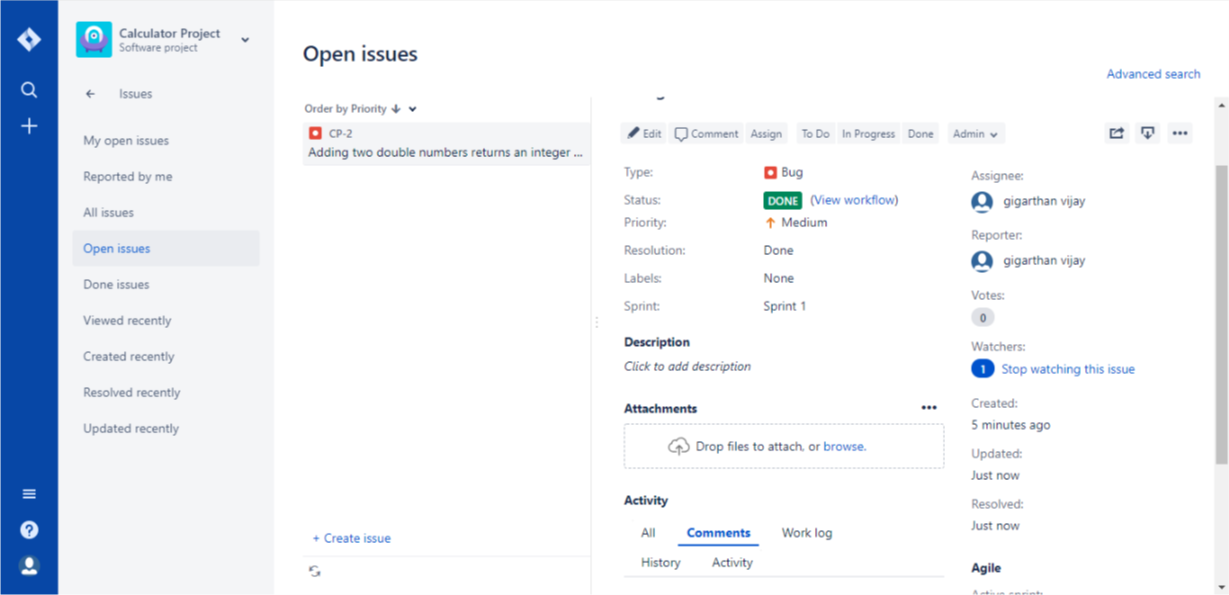
We can check our active sprints by clicking on the Active Sprints button on the left navigation menu. In the above image, we can see the Sprint 1 and issues that has been assigned to it. Currently our bug is in to do state.



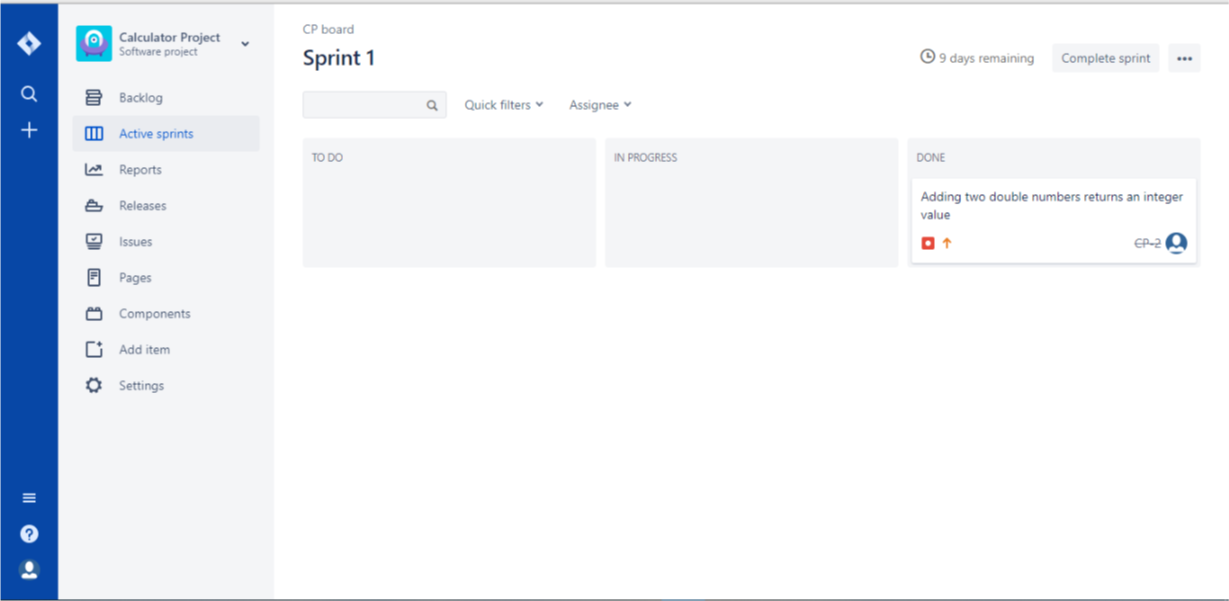
We can see all the issues by navigating into issues -> All issues / My open issues in the navigation bar. We can see all the issues and details of the issues in the above image. We can change the state of the bug by clicking on the “In Progress” button.



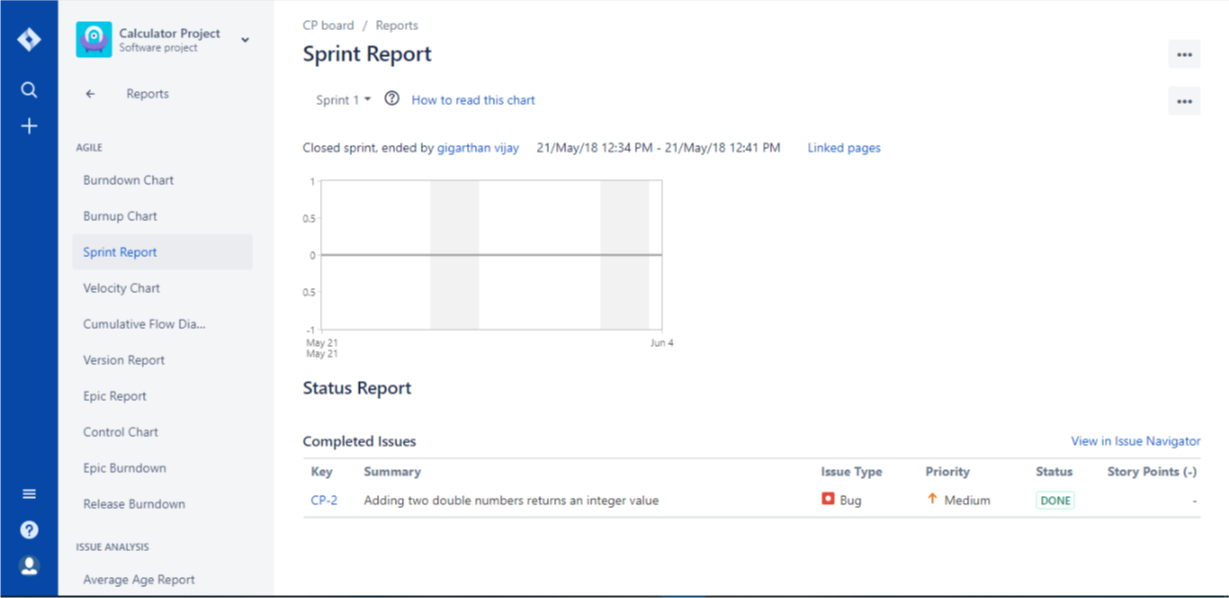
We can come back to active sprints tab to check and we can see that our bug has been moved from “to do” to “In Progress”



Once the bug has been fixed, we can change its state by clicking the done button.



And once again, we can come back to active sprints and can see that our bug has been moved to “Done” state. Once all the bugs are finished we can complete the sprint by clicking the “Complete sprint” button on the top right corner.



Once we complete a sprint, we will automatically route to the see the sprint reports and status by Jira.

# **Task 3 - Code Coverage Tool**

## **Tool Introduction**

## **Pros and Cons**

**Pros**

**Cons**

## **Scenario**

## **Scenario Execution**