1 Introduction

What's Mutation Testing? Mutation testing is a software testing technique where the source code is intentionally modified (mutated) to create faulty versions (mutants) of the tests of a program. Doing this enables the user to evaluate the effectiveness of their test suite by determining if the tests can detect the introduced faults, thereby ensuring high test coverage and identifying weak spots in testing. (Testing the Tests)

The default implementation provided by PiTest requires multiple steps to see the desired results and runs on a complete project by default. This makes the progress of using PiTest tedious and slows down developers. To ease the workflow and to increase the productivity of developers, the PIT MutationMate plugin was developed.

2 Workflow

2.1 Obtaining the plugin

The plugin can be downloaded from the JetBrains Marketplace by following these steps:

1. Open IntelliJ IDEA: Launch IntelliJ IDEA on your computer.
2. Navigate to the Marketplace: In the IntelliJ IDEA menu, go to "File" -> "Settings" (or "IntelliJ IDEA" -> "Preferences" on macOS). In the Settings/Preferences dialog, select "Plugins" from the left-hand menu.
3. Access the JetBrains Marketplace: In the Plugins settings, click on the "Marketplace" tab. Click on the "Browse repositories" button to access the JetBrains Marketplace.
4. Search for the Plugin: Use the search bar to find your specific plugin by name or keywords related to its functionality, search for "pitmutationmate".
5. Install the Plugin: Once you locate your plugin in the search results, click on it to view details. Click the "Install" button to initiate the installation process.
6. Review Permissions (if any): If the plugin requires certain permissions, carefully review them and click "Accept" if you agree.
7. Restart IntelliJ IDEA: After the installation is complete, you'll likely be prompted to restart IntelliJ IDEA. Save any open work and restart the IDE.

2.1.1 Adding the companion plugin

To be able to use the PIT MutationMate plugin effectively, the user will have to add a separate Gradle plugin within the receiving project. To do so, follow these steps:

1. Locate the projects build file (typically `build.gradle` or `build.gradle.kts`) in your project and open it.
2. Within the file look for the "plugins" section and add the following line:

```
id "io.github.amos-pitmutationmate.pitmutationmate.override" version "1.3"
// for a `build.gradle` file
id("io.github.amos-pitmutationmate.pitmutationmate.override") version
"1.3" // for a `build.gradle.kts` file
```

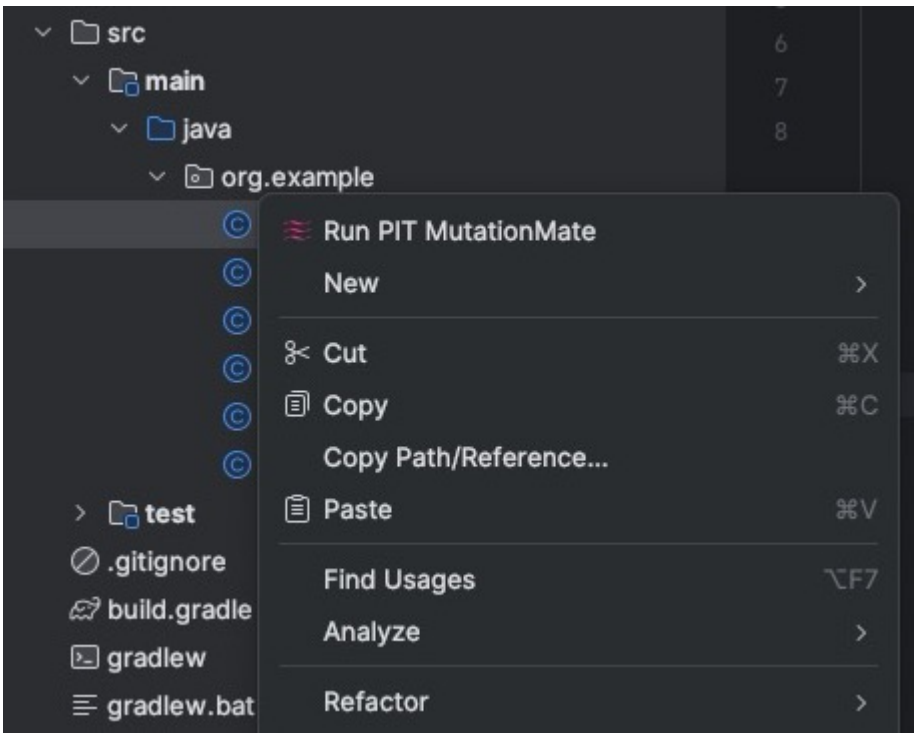
You can find more information on how to add it to your project on it's gradle repository page

2.2 Running the plugin

The plugin can be run in various options.

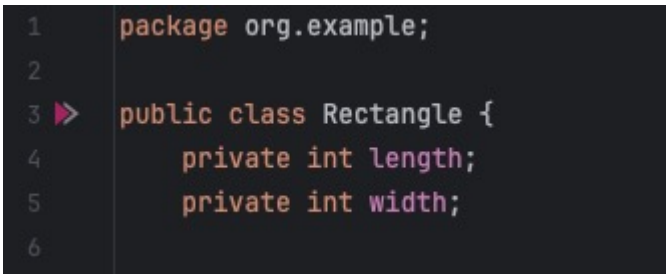
2.2.1 From the Context Menu

To start a PiTest run from the Context menu either right-click within an open file, on a class in the file-tree or a folder in the file-tree. If the right-click was performed somewhere, where a PiTest run can be performed the the option "Run PIT MutationMate" will be presented. Click this to start a PiTest run for the underlying classes.



2.2.2 From the Gutter

To start a PiTest run from the Gutter, navigate to a class that supports PiTest. In the Gutter, next to the desired class a PIT MutationMate Run icon will appear. Click this to start a PiTest run for the specified class.



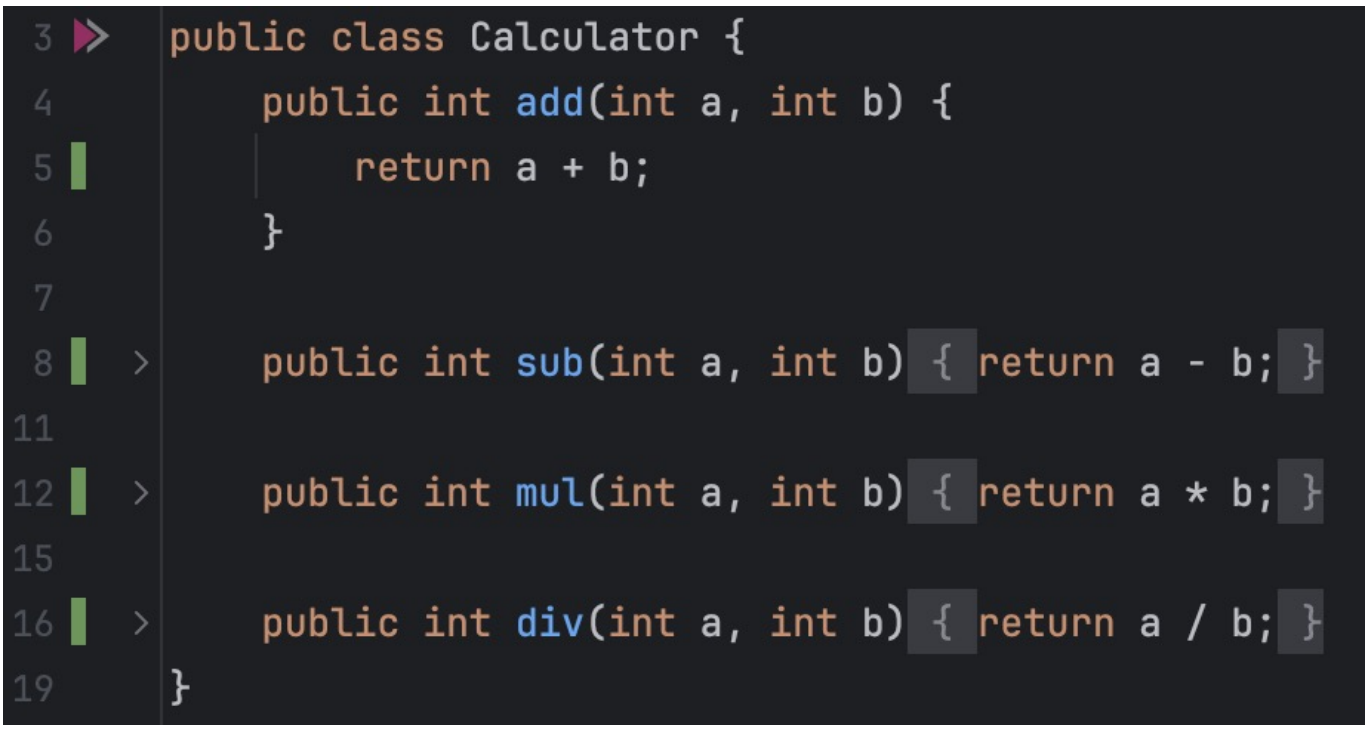
```
package org.example;

public class Rectangle {
    private int length;
    private int width;

```

2.3 Working with the results

Run Mutation Testing: Follow the previously mentioned workflow to run mutation testing on a specific class using your plugin. The results will be viewable in the following ways.

2.3.1 Editor

Within the file-editor PITMutationMate results will be shown in the gutter as color-bars. The possible colors correspond with the PiTest colors found in a standard PiTest run, and give the user information about the coverage on a line-specific level. These annotations can be removed through the context menu in the editor or by clicking on the gutter.

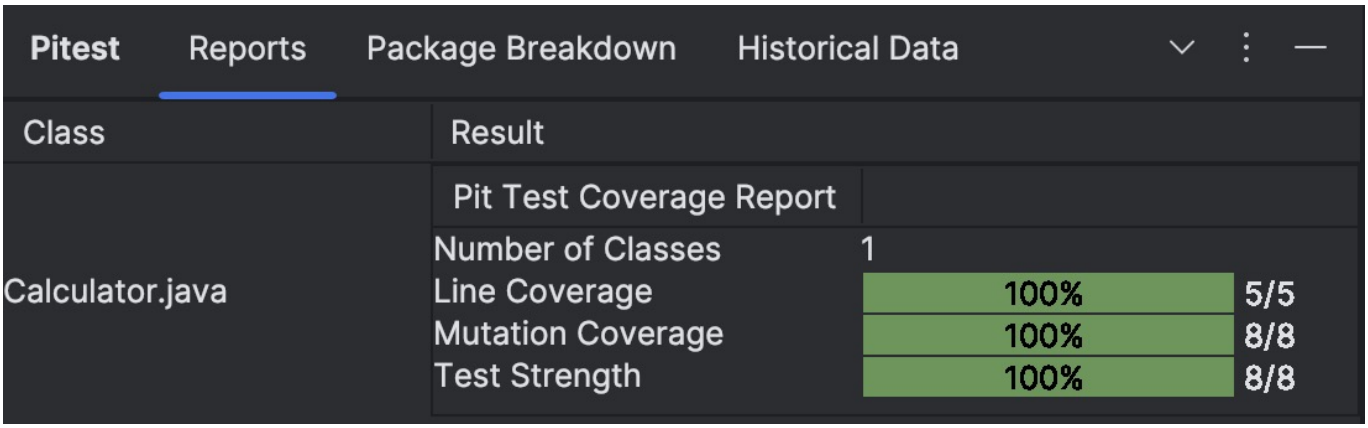


```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int sub(int a, int b) { return a - b; }

    public int mul(int a, int b) { return a * b; }

    public int div(int a, int b) { return a / b; }
}
```

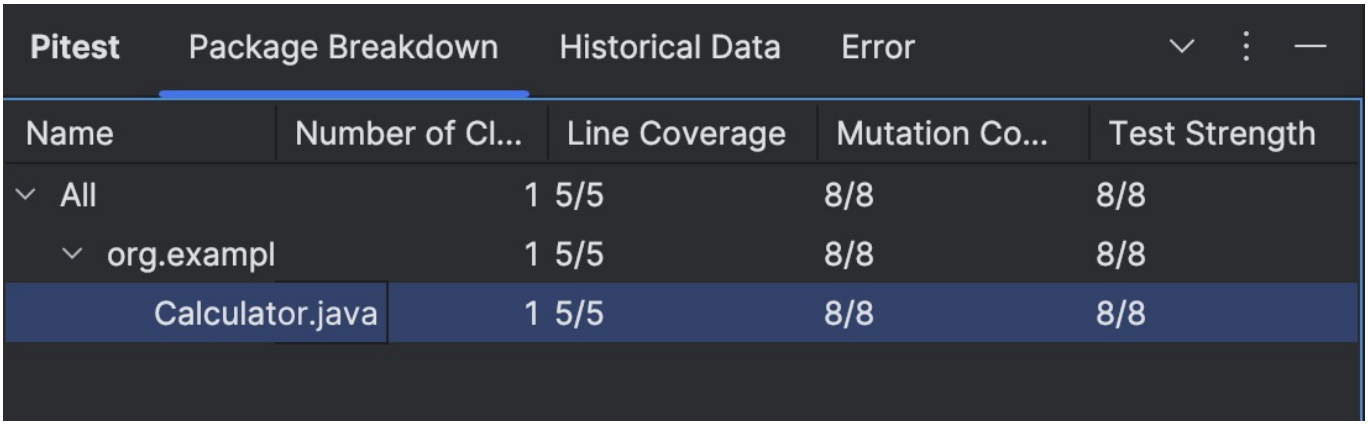
2.3.2 Results Window

To get a more detailed view of the results, users can inspect the results in a dedicated results-window containing the panels "Reports", "Package Breakdown" and "Historical Data".

The panel "Reports" contains a breakdown of the 5 most recent PiTest class results followed by a summary of the complete PiTest run for the whole project. Here the user can view the line and mutation coverage as well as the overall test strength for each of the 5 classes and for the project as a whole.



The panel "Package Breakdown" presents the same information in the same tree structure found in the file-tree in the IDE.



The panel "Historical Data" shows information about all executed runs.