



Auto Logic Manual

Automated Tool Support for Logic Coverage of Java Code

Bradley Rumball & Harley Everett

What is AutoLogic?

AutoLogic is an automated tool for performing logical testing on Java methods. It automatically analyses conditions present in classes, instruments these via log statements, then generates appropriate test data through a genetic algorithm. Finally, creating compilable JUnit code for the desired class. AutoLogic is supplied with multiple test classes to experiment on and to show the capabilities of the system.

In this guide, we will discuss how to correctly compile and run AutoLogic as well as what limitations and features it presents.

However, before knowing how to run AutoLogic, it is important to understand the structure of the program to see “under the hood” and gain a proper understanding of its features and how it can help a human tester automate logical testing.



To generate the tests the file must first be turned into a Compilation Unit (CU) which is all handled by JavaParser. This CU is then passed to a modified visitor which visits each branch within the classes methods and injects them with log statements for example:

To generate the tests the file must first be turned into a Compilation Unit (CU) which is all handled by JavaParser. This CU is then passed to a modified visitor which visits each branch within the classes methods and injects them with log statements for example:

if (methodLogger.log(0, side1, side2, BinaryExpr.Operator.GREATER))

These log statements are important as they instrument the class ready for coverage percentage calculation as well as generate fitness for the Genetic Algorithm (GA). Once injected, the Genetic Algorithm *Host()* object is run and values are calculated for each condition using fitness. This fitness helps guide how far away or close the values are to satisfying the condition. Once values have been generated, the *Host* class calls the *JUnitOutputManager* class, which constructs individual tests via a loop with the help of *OutputElements* objects. Finally this generates a file to the desired output path location specified by the user at the start of runtime.

How to run AutoLogic?

There are two ways you can run AutoLogic, through an IDE (easier) or through the command line (faster). Please make sure you have **Java 8** installed on your system for it to run correctly. Both ways of running AutoLogic also require small manual changes onto the outputted test file to allow the tests to run correctly on your system. This will be discussed after we have run AutoLogic, in the subsection *Prepping the Junit Test file for compiling*.

AutoLogic can also be used as an API, users can call new AutoLogic objects with paths as parameters on their own projects.

Through an IDE (IntelliJ)

This is probably the easiest way to run AutoLogic for an inexperienced user, as it requires little interaction with a command line input.

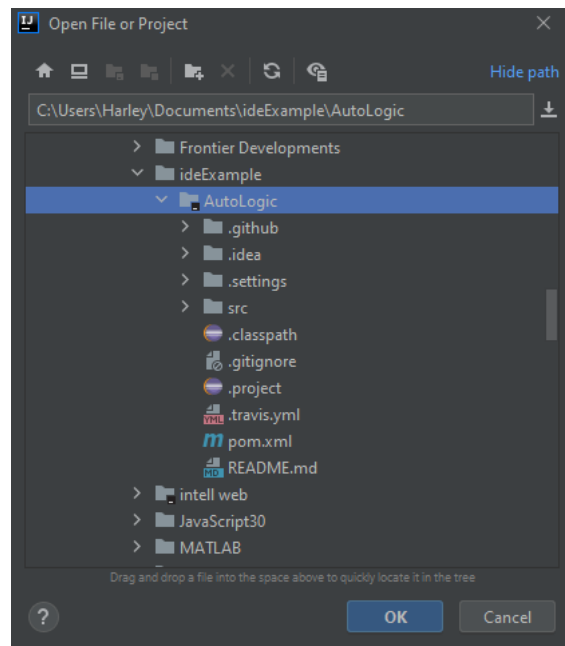
- Clone the repo
- Open in IntelliJ
- Edit the configurations for the AutoLogic class main method - this can be done by right-clicking on the main method and selecting *Create 'AutoLogic main()'* then by clicking and changing the *Program arguments* to the ones desired
- The *Program arguments* should contain the **targeted file absolute path** and the **desired output directory absolute path** for the Junit tests, with a space separating them
- Press the green play button
- If all done correctly, you should see the success message with coverage percentage and the Junit file should now be in the targeted directory

Through an IDE (IntelliJ) - Example

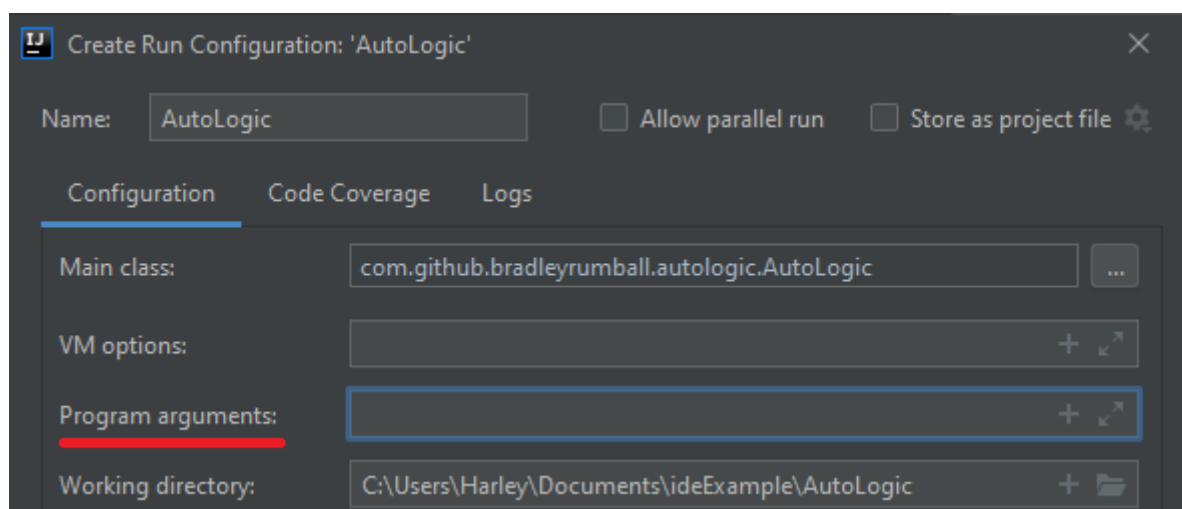
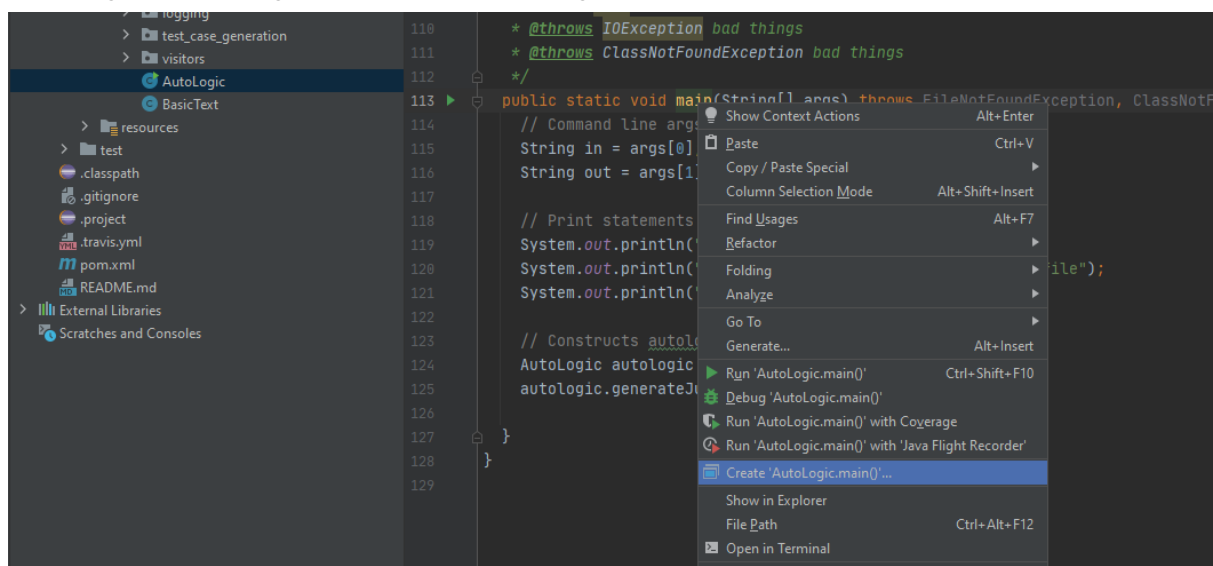
Cloning the repo...

```
C:\Users\Harley\Documents\ideExample>git clone https://github.com/bradleyrumball/AutoLogic.git
Cloning into 'AutoLogic'...
remote: Enumerating objects: 1001, done.
remote: Counting objects: 100% (1001/1001), done.
remote: Compressing objects: 100% (499/499), done.
Receiving objects: 100% (1001/1001), 372.24 KiB | 1.88 MiB/s, done. 0 eceiving objects: 97% (971/1001)
Resolving deltas: 100% (349/349), done.
C:\Users\Harley\Documents\ideExample>
```

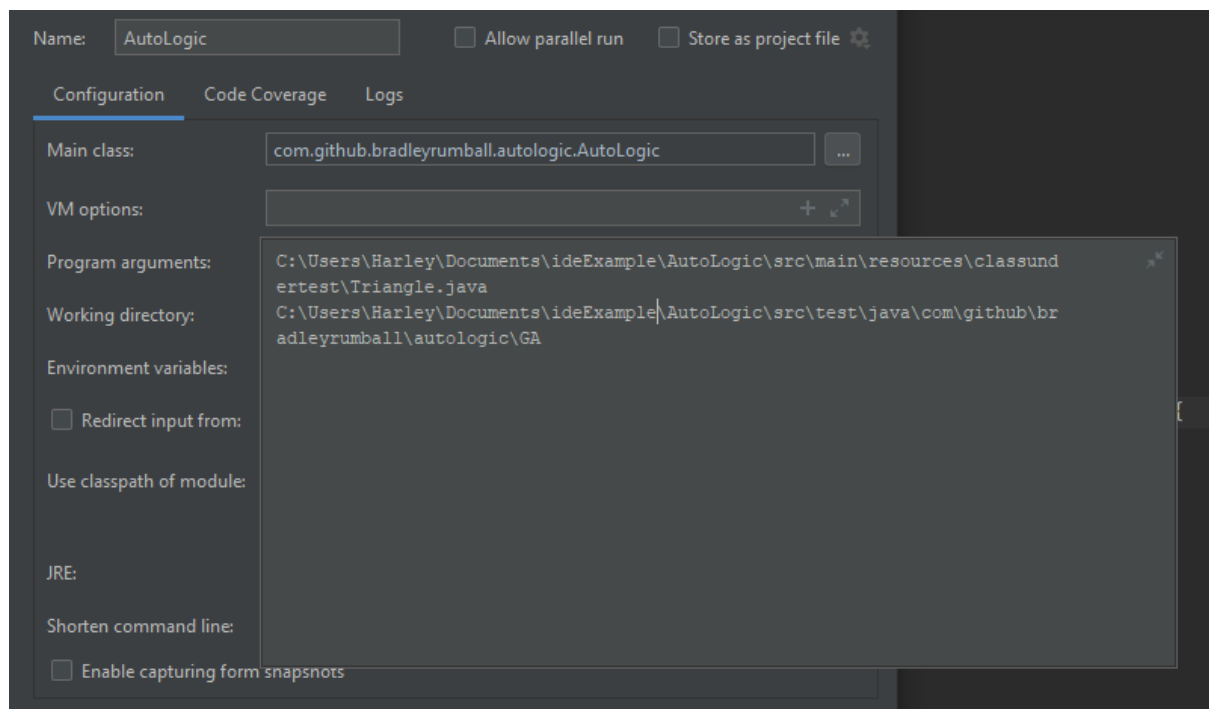
Opening the cloned repo in IntelliJ...



Creating a run configuration for the AutoLogic main method...



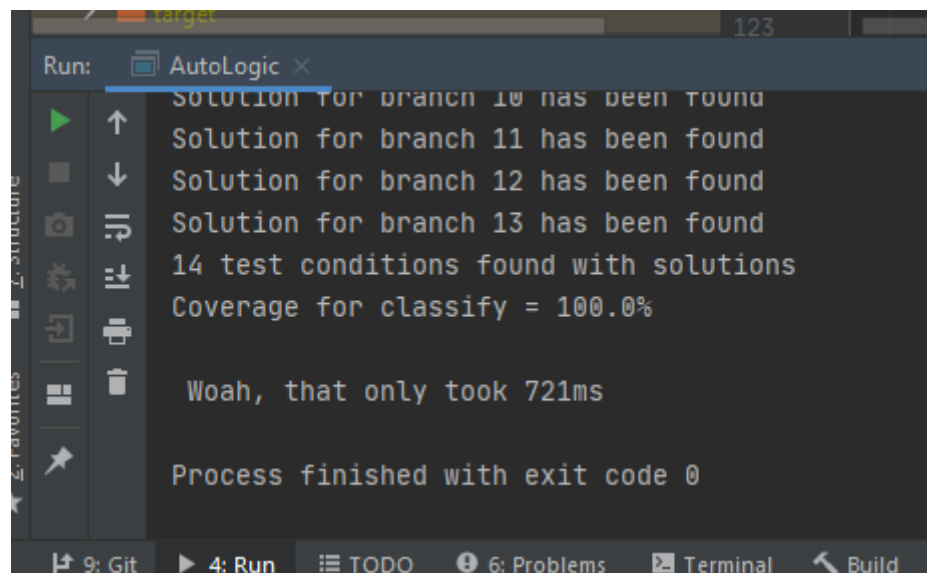
Changing the program arguments to the desired **targeted file location** and the **desired output directory**, below is an example using the supplied triangle class and outputting the tests to the test. As you can see they are absolute paths.



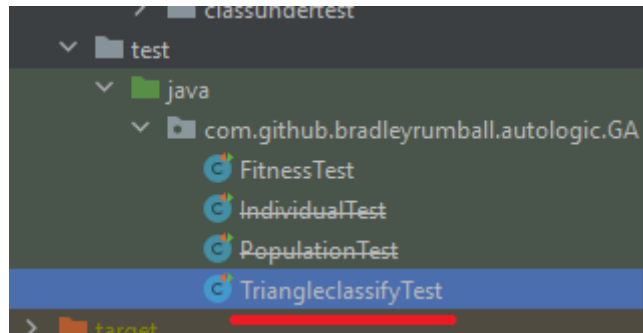
Running the configuration...



Success!



The test file in the location we desired...



Command line with maven

Please make sure you have maven installed on your system before attempting to run AutoLogic in this way, it can be installed from here <https://maven.apache.org/>

- Clone the repo
- Navigate to the directory *AutoLogic* on a chosen command line interpreter
- Run the command `mvn clean` - This command will delete all previously compiled Java .class files
- Run the command `mvn package` - This command will make a *target* directory with a "fat-jar" of AutoLogic
- Once the green **BUILD SUCCESS** appears we have correctly built the .jar file
- Navigate to the newly created *target* directory
- When in the target directory we can run the following command by replacing **ARG1** with the targeted file path and **ARG2** with the output directory path desired for the Junit tests

```
java -jar AutoLogic-1.0-SNAPSHOT.jar ARG1 ARG2
```

- If all done correctly, you should see the success message with coverage percentage and the Junit file should now be in the targeted directory.

Command line with maven - Example

Cloning the repo...

```
C:\Users\Harley\Documents\ideExample>git clone https://github.com/bradleyrumball/AutoLogic.git
Cloning into 'AutoLogic'...
remote: Enumerating objects: 1001, done.
remote: Counting objects: 100% (1001/1001), done.
remote: Compressing objects: 100% (499/499), done.
Receiving objects: 100% (1001/1001), 372.24 KiB | 1.88 MiB/s, done. 0 eceiving objects: 97% (971/1001)
Resolving deltas: 100% (349/349), done.
C:\Users\Harley\Documents\ideExample>
```

Navigating to the AutoLogic directory...


```
C:\Users\Harley\Documents\mavenexample>cd AutoLogic
C:\Users\Harley\Documents\mavenexample\AutoLogic>.
```

Running *mvn clean* command...

```
C:\Users\Harley\Documents\mavenexample\AutoLogic>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----< org.sheffield:AutoLogic >-----
[INFO] Building AutoLogic 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ AutoLogic ---
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 0.308 s
[INFO] Finished at: 2021-04-10T18:30:16+01:00
[INFO]
C:\Users\Harley\Documents\mavenexample\AutoLogic>
```

Running *mvn package* command...

```
[WARNING] See http://maven.apache.org/plugins/maven-shade-plugin/
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing C:\Users\Harley\Documents\mavenexample\AutoLogic\target\AutoLogic-1.0-SNAPSHOT.jar with C:\Users\Harley\Documents\mavenexample\AutoLogic\target\AutoLogic-1.0-SNAPSHOT-shaded.jar
[INFO] Dependency-reduced POM written at: C:\Users\Harley\Documents\mavenexample\AutoLogic\dependency-reduced-pom.xml
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.774 s
[INFO] Finished at: 2021-04-10T18:31:23+01:00
[INFO]
C:\Users\Harley\Documents\mavenexample\AutoLogic>.
```

Navigating to the target directory just created...

```
C:\Users\Harley\Documents\mavenexample\AutoLogic>cd target
C:\Users\Harley\Documents\mavenexample\AutoLogic\target>
```

Running the *java -jar AutoLogic-1.0-SNAPSHOT.jar* *ARG1 ARG2* command, in this example we ran...

```
java -jar AutoLogic-1.0-SNAPSHOT.jar
C:\Users\Harley\Documents\mavenexample\AutoLogic\src\main\resources\classundertest\Triangle.java
C:\Users\Harley\Documents\mavenexample\AutoLogic\src\test\java\com\github\bradleyrumball\autologic\GA
```

```
C:\Users\Harley\Documents\mavenexample\AutoLogic\target>java -jar AutoLogic-1.0-SNAPSHOT.jar C:\Users\Harley\Documents\mavenexample\AutoLogic\src\main\resources\classundertest\Triangle.java C:\Users\Harley\Documents\mavenexample\AutoLogic\src\test\java\com\github\bradleyrumball\autologic\GA
Welcome to AutoLogic :D!
You are currently testing C:\Users\Harley\Documents\mavenexample\AutoLogic\src\main\resources\classundertest\Triangle.java file
With output to C:\Users\Harley\Documents\mavenexample\AutoLogic\src\test\java\com\github\bradleyrumball\autologic\GA
package com.github.bradleyrumball.autologic;
```

Success!

```

Solution for branch 8 has been found
Solution for branch 9 has been found
Solution for branch 10 has been found
Solution for branch 11 has been found
Solution for branch 12 has been found
Solution for branch 13 has been found
14 test conditions found with solutions
Coverage for classify = 100.0%





Woah, that only took 661ms

C:\Users\Harley\Documents\mavenexample\AutoLogic\target>_

```

The test file in the location we desired...

PC > Documents > mavenexample > AutoLogic > src > test > java > com > github > bradleyrumball > autologic > GA

Name	Date modified	Type	Size
 FitnessTest	10/04/2021 18:28	JAVA File	10 KB
 IndividualTest	10/04/2021 18:28	JAVA File	2 KB
 PopulationTest	10/04/2021 18:28	JAVA File	2 KB
 <u>TriangleclassifyTest</u>	10/04/2021 18:34	JAVA File	2 KB

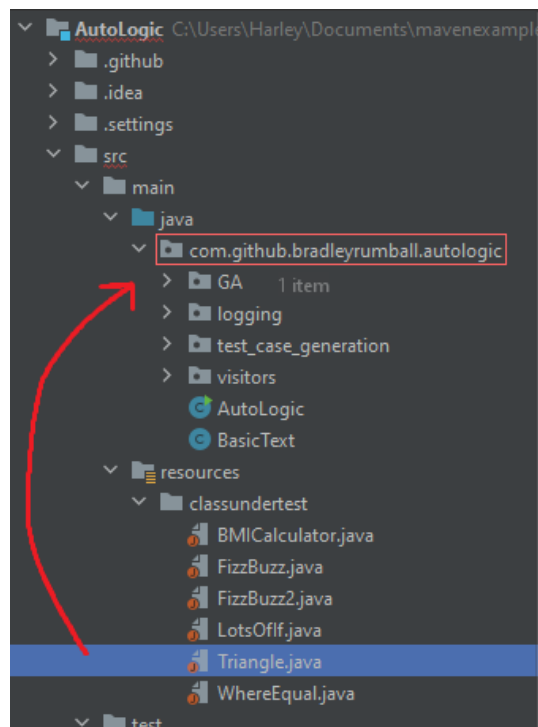
Prepping the Junit Test file for compiling

After creating the test file via your chosen way we now need to make some small changes to the file to allow it to run and compile. The best way to start this process is to open the project in your favourite IDE, for this guide we'll be using IntelliJ.

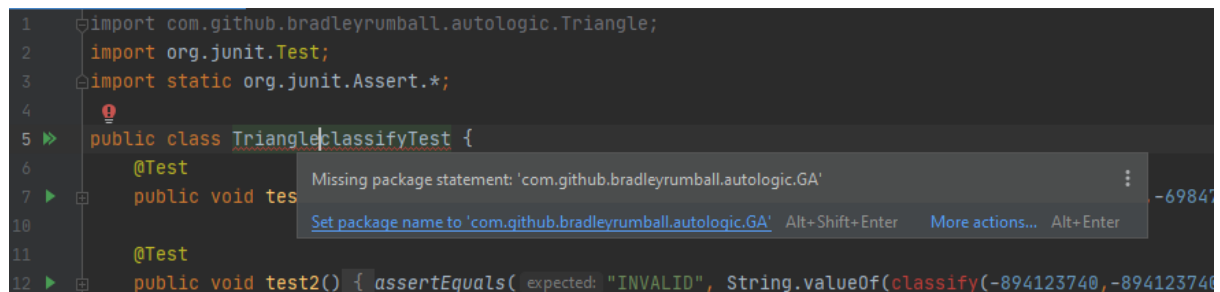
- When you open your chosen test file you'll notice a few things are in red, first we need to make sure the class we are testing is in the main file structure. So make sure to move it into the appropriate area. In this case we moved it into the package *com.github.bradleyrumball.autologic* package.
- Next we want to set the package, this can be done by clicking on the red underlined class name and selecting *Set package name* to it should then automatically put the package name at the top of the file
- Next we want to import static methods for the tests, to do this we click on the red text on the method names, click *More actions* then *Import static method*.
- **Note:** Some primitive types may require manual casting, to do this hover over the parameter and click *Cast parameter to X*
- With all this done correctly there should no longer be any red on the test file
- Run all tests by clicking on the double green arrow next to the class name
- You should now have a fully compiled and correct test suite!
- **Note:** You may need to delete the tests generated before generating tests for a new class

Prepping the Junit Test file for compiling - Example

Moving the file we want to test into the main package, in this example we are using *Triangle.java*...



Setting the package name by clicking on the red class name...



Now we need to import the static methods used in the tests...



With all this done correctly there now should be no red left on the file...

```

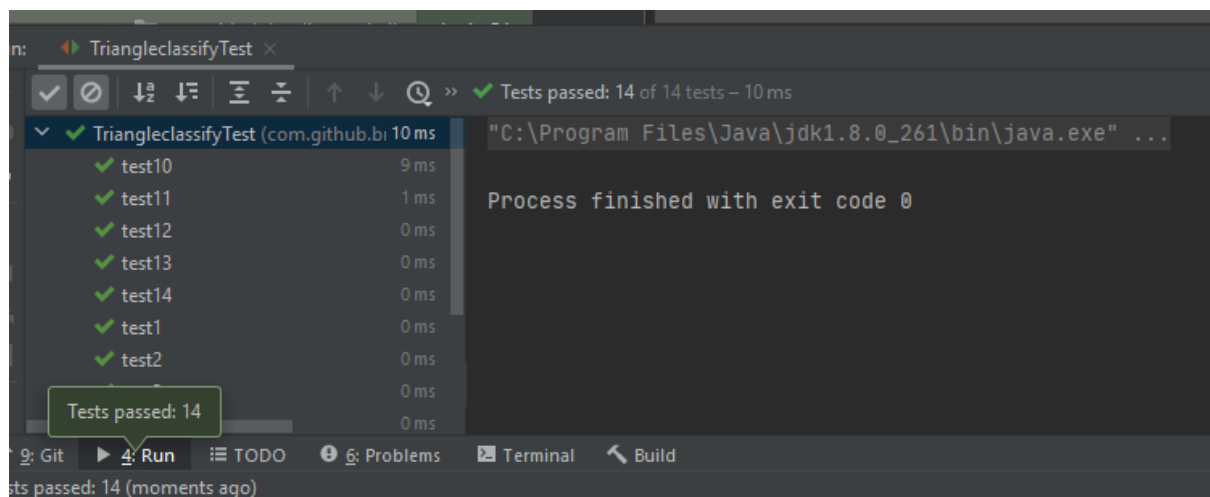
1 package com.github.bradleyrumball.autologic.GA;
2
3 import com.github.bradleyrumball.autologic.Triangle;
4 import org.junit.Test;
5
6 import static com.github.bradleyrumball.autologic.Triangle.classify;
7 import static org.junit.Assert.*;
8
9 public class TriangleclassifyTest {
10     @Test
11     public void test1() { assertEquals( expected: "INVALID", String.valueOf(classify(-14,-698470899,-698470899))); }
12
13     @Test
14     public void test2() { assertEquals( expected: "INVALID", String.valueOf(classify(-894123740,-894123740,-894123740))); }
15
16     @Test
17     public void test3() { assertEquals( expected: "INVALID", String.valueOf(classify(-17,-16,-836900571))); }
18
19     @Test
20     public void test4() { assertEquals( expected: "INVALID", String.valueOf(classify(-894123740,-894123740,-894123740))); }
21
22     @Test
23     public void test5() { assertEquals( expected: "INVALID", String.valueOf(classify(-17,-16,-836900571))); }
24 }

```

Now run the tests by clicking on the double green arrow next to the class name...



Success!



Limitations and Support

Whilst AutoLogic has many features, there are some limitations to its capabilities. One main limitation is that conditions must be binary expressions eg `x == y` so unary expressions eg `!true` cannot be used. Currently it doesn't have support for Arrays or elements from the Java Collections framework. However AutoLogic can test multiple methods within a class in one go! And will generate multiple files for each method.

Below is a table of the data types in Java and AutoLogics level of support for them.

Data Type	Support
Byte	✓
Short	✓
Int	✓
Long	✓
Float	✓
Double	✓
Boolean	✓
Char	✓
String	✓ (~ max length of 8 Alphanumeric Including caps and spaces)

Below is a table of the coverage criterion AutoLogic provides.

Coverage Criterion	Support
Conditional Coverage	✓
MCDC (Restricted)	✓
MCDC (Correlated)	✓