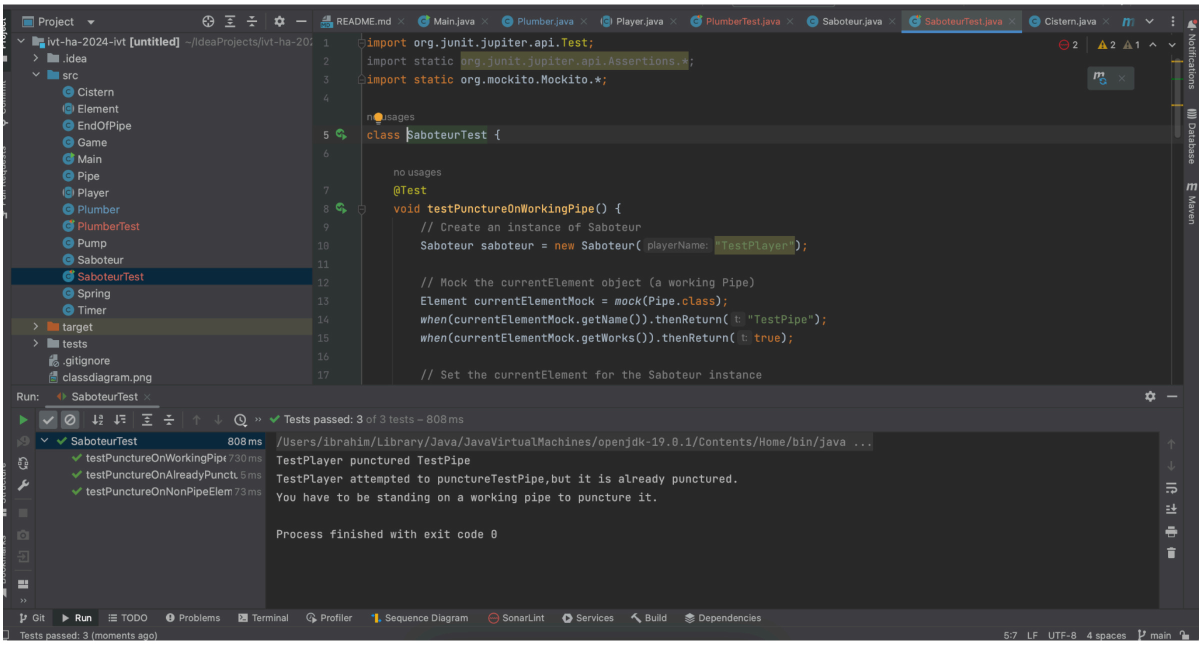
These tests are like check-ups for a pretend plumbing system to make sure it works well. They carefully look at how two main parts of the system, the "Plumber" and the "Saboteur," do their jobs in the computer program.

First, let's talk about the "Plumber" tests. They focus on checking if the plumber can fix problems with pipes and pumps without any trouble. One test called "testFixPipe" looks at how well the plumber can repair a broken pipe. It checks not only if the repair is done right but also if the plumber communicates well after fixing it. Another test called "testFixPump" checks if the plumber can fix pump issues properly.

Now, onto the "Saboteur" tests. These tests make sure that when someone tries to mess up the plumbing system in the program, it happens correctly and doesn't cause extra problems. For example, in a test called "testPunctureOnWorkingPipe," it watches to see if the saboteur can poke a hole in a working pipe without making things worse. Another test called "testPunctureOnAlreadyPuncturedPipe" looks at what happens when the saboteur tries to break a pipe that's already broken, making sure it doesn't cause more damage. There's also a test called "testPunctureOnNonPipeElement" to check that the saboteur only messes with pipes and doesn't mess up anything else in the plumbing system.

By doing these tests carefully, we can be sure that the pretend plumbing system works well and can be trusted to fix things and handle sabotage properly.



```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class SaboteurTest {

    @Test
    void testPunctureOnWorkingPipe() {
        // Create an instance of Saboteur
        Saboteur saboteur = new Saboteur( playerName: "TestPlayer");

        // Mock the currentElement object (a working Pipe)
        Element currentElementMock = mock(Pipe.class);
        when(currentElementMock.getName()).thenReturn( t: "TestPipe");
        when(currentElementMock.getWorks()).thenReturn( t: true);

        // Set the currentElement for the Saboteur instance
```

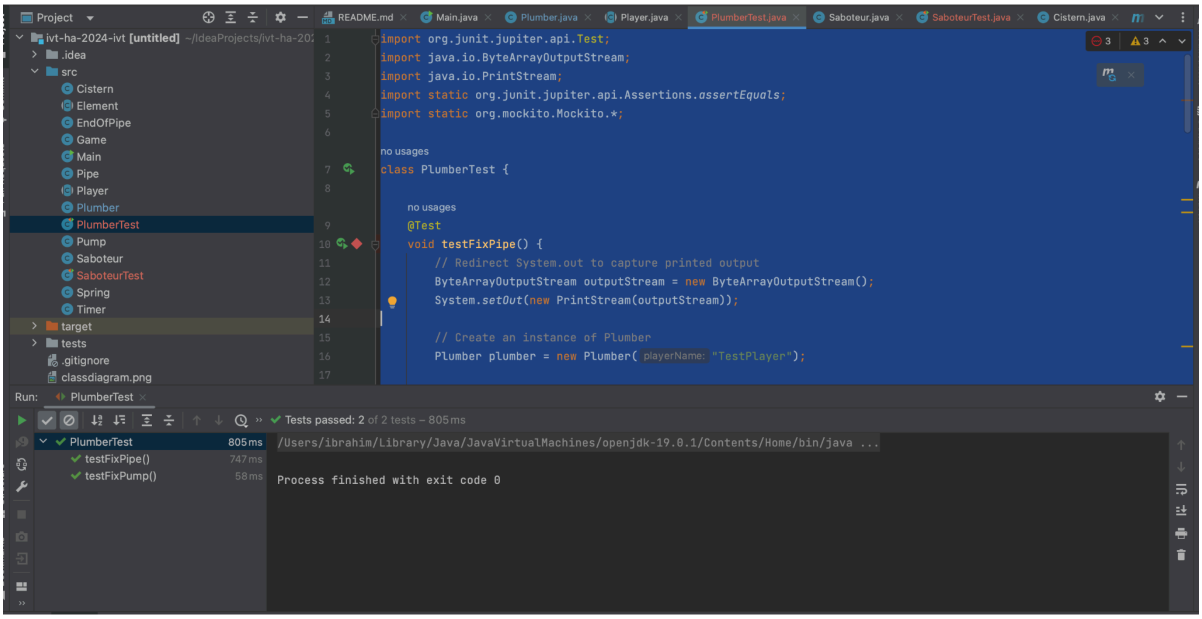
Tests passed: 3 of 3 tests – 808 ms

```
SaboteurTest 808 ms
  testPunctureOnWorkingPipe 730 ms
  testPunctureOnAlreadyPunctu 5 ms
  testPunctureOnNonPipeElem 73 ms
```

```
/Users/ibrahim/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java ...
TestPlayer punctured TestPipe
TestPlayer attempted to punctureTestPipe,but it is already punctured.
You have to be standing on a working pipe to puncture it.

Process finished with exit code 0
```

Tests passed: 3 (moments ago)



```
import org.junit.jupiter.api.Test;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;

no usages
class PlumberTest {

    no usages
    @Test
    void testFixPipe() {
        // Redirect System.out to capture printed output
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outputStream));

        // Create an instance of Plumber
        Plumber plumber = new Plumber( playerName: "TestPlayer");

```

✓ Tests passed: 2 of 2 tests – 805 ms

```
✓ PlumberTest      805 ms
    ✓ testFixPipe()    747 ms
    ✓ testFixPump()     58 ms
```

```
/Users/ibrahim/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java ...

Process finished with exit code 0
```