Mocking Scanner in JUnit 5 Tests

Scanner is a commonly used class in Java for taking console input or file input. However, it can become really tricky to unit test because of it's reliance on outside input.

The concept of mocking in testing is to create dummy classes or objects with our own test data in place of what the program is expecting. Mocking intercepts what the program would normally execute and gives the program our own set of test data.

There is formal mocking frameworks, like Mockito, but to keep things simple at first we will just be using the concept of mocking to create fake user input from the console. Ours won't be a true mock in the sense of a dummy class or object, but using a similar idea.

Step 1: JUnit Setup

Make sure you have added JUnit as a dependency by following the steps here: JUnit Setup

Step 2: Setting up the Mock

In the case of taking user input from the console, our mock will be hard-coding an input from the user and tricking the scanner to read from that, and not wait for console input.

To do this, you can setup a little helper method inside your test class.

The method accepts a `String` as input so that we can reuse this method for any input scenario we want.

Then we convert the `String` into bytes and save that into a new `ByteArrayInputStream`, because that is the type of data expected from the scanner.

Finally, we set our `ByteArrayInputStream` object as the value for input on the `System` class. This works if you setup your scanner to use `System.in` as the input. Now we've saved a byte stream in the `System.in` and the next time `scanner.nextLine()` runs, it'll see our mocked input in the buffer.

```
private void mockScannerInput(String input) {
        ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes());
        System.setIn(in);
    }
```

Step 3: Using the mockScannerInput Method

Inside any of your tests that may run code that utilizes the `Scanner` class for console input, you can first call the `mockScannerInput` method we wrote above to setup our mock user input BEFORE the scanner code executes.

```
@Test
    public void whenInputCostOfMaterialsIsDouble_thenReturnIsValid() {
        // Calling mockScannerInput with the String I am pretending the user
inputted
        mockScannerInput(String.valueOf(COST_OF_MATERIALS));

        String expected = String.valueOf(COST_OF_MATERIALS);

        // inputCostOfMaterials method using the scanner input. Instead of waiting
for user input, it will use what we set above in the mockScannerInput
        String actual = HarveyHomeRepair.inputCostOfMaterials();

        assertEquals(expected, actual);
    }
```