JUnit Setup Instructions

Setting up JUnit for your project involves two main steps:

1. Adding JUnit dependencies to your dependency manager (we're using Maven in this class)
2. Setting up a class to run tests

Step 1: Adding JUnit as a dependency to your project

Maven is a depedency manager that will look at a list of packages and download them so they are ready to run in your project. That list of packages is inside the file called pom.xml which is created automatically when you create a new IntelliJ project with the Maven box checked.

The pom.xml file looks intimidating at first, but it's really simple once you know your way around it.

It uses the XML format, which uses tags to organize data. If you've never seen XML, you have seen HTML and the tag setup works the same ways.

You'll need to create a new tag inside the project tag called dependencies.

```
<dependencies>
    <!-- Dependencies go here -->
</dependencies>
```

Inside that dependency tag you can place all your project dependencies. Dependencies can be found from a maven repository like https://mvnrepository.com/

Go to the URL above and search for: JUnit

1. Select JUnit Jupiter API
2. Select 5.10.2 (or whatever the latest version is)
3. Make sure the Maven tag is selected, and copy the XML from the box.
4. Paste XML into your pom.xml inside your `<depedencies>` tag

In the end it should look like:

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.10.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

You'll notice IntelliJ might pop up a little M icon with a refresh symbol. Click it to reload you pom.xml and download dependencies from Maven.

If that little icon does not appear, click the Maven tab on the side of IntelliJ and reload dependencies yourself.

Step 2: Create a Java test class

Creating a class for your JUnit tests is simple. Under your src.test.java package, right-click and create a new class. You'll want to name it using the name of the class you are testing, and the word Tests at the end. Like this:

`BasketBuildTests.java`

If you have methods you are testing inside multiple classes, you should create a test class for each class. This will keep your tests organized.

Inside your test class you will utilize both JUnit annotations, and JUnit methods.

To create your first test, create a new method called whatever you want (I like to follow the convention of saying what you are testing, and what the expected result is). Make sure it is public void and accepts no parameters. Annotate with `@Test` right before the method.

```
@Test
public void whenDefaultConstructorIsCalled_thenDefaultValuesAreCorrect() {

}
```

Inside the method is where you will setup test data, and test an assertion. The result of that assertion determines if the test passes or fails. There are several assert methods you can use from JUnit, but `assertEquals()` is the most common.

AssertEquals takes two arguments: the expected value, and the actual value.

The following example test is testing the code to make sure the correct values are set when the default constructor is used.

```
@Test
public void whenDefaultConstructorIsCalled_thenDefaultValuesAreCorrect() {
    // Setting up my data
    Estimate defaultEstimate = new Estimate();

    // Running my assertions. This is what tests the code.
    assertEquals("Sample", defaultEstimate.getJobName());
    assertEquals(500, defaultEstimate.getCostOfMaterials());
    assertEquals(1, defaultEstimate.getLaborHours());
    assertEquals(1, defaultEstimate.getTravelHours());
}
```

A test should only test one scenario at a time, and one specific part of the class. You should create many test methods inside your test class to test as many scenarios and methods in your classes as possible.