Hyperiondev

TASK

Unit Testing in Eclipse Using JUnit

Visit our website

Introduction

WELCOME TO THE UNIT TESTING IN ECLIPSE USING JUNIT TASK!

Earlier on in this bootcamp you were introduced to the software development process. Testing is a crucial part of this process. This task introduces how to test Java classes using JUnit.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with a code reviewer. You can also schedule a call or get support via email.

Our expert code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

JUNIT

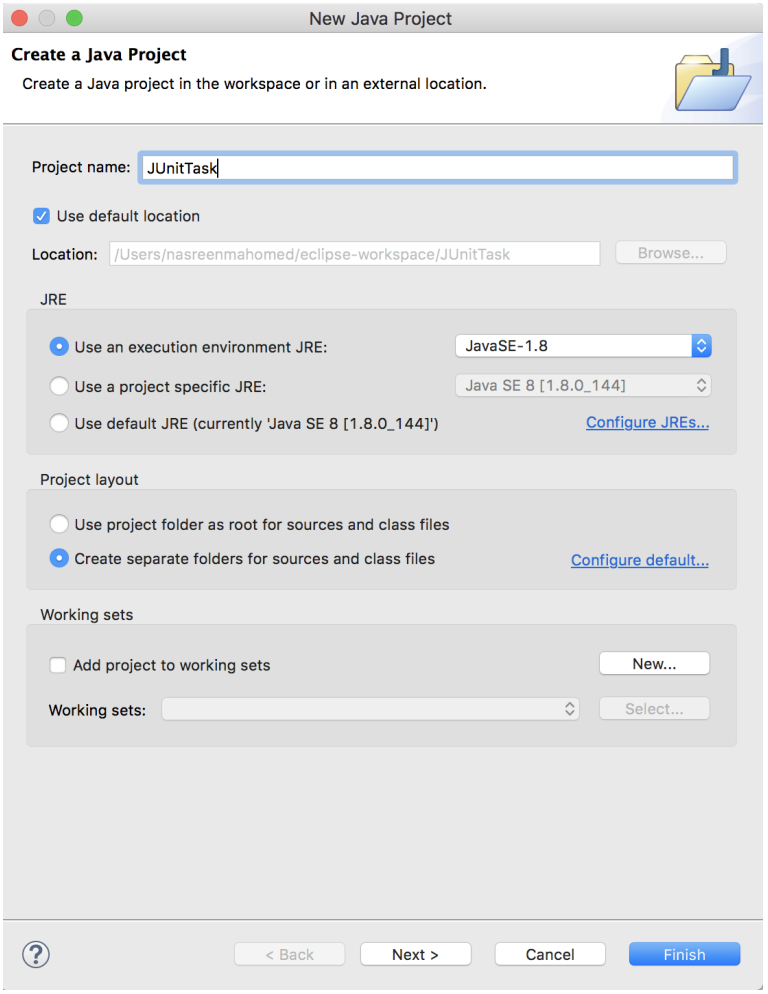
JUnit is an open-source framework for Java. It was developed by Erich Gamma and Kent Beck and has been an important factor in the evolution of test-driven development, which is part of a larger software design paradigm, Extreme Programming (XP). With test-driven development, developers must write and execute unit tests first before any code is written.

As the name implies, JUnit helps you perform unit testing. Unit testing involves examining a small "chunk" (or “unit”) of software to verify that it meets its expectations or specification. This small “chunk” of software is usually a single class.

JUnit is not part of the standard Java class libraries. However, it is included with Eclipse. JUnit can be downloaded for free from the JUnit **website** if you don’t have Eclipse. For this task, however, we will assume that you have already downloaded and installed Eclipse.

CREATING A JUNIT TEST CASE IN ECLIPSE

- Create a new project called JUnitTask as follows:
    - Choose *File → New → Java Project*
    - In the ‘New Java Project’ window enter **JUnitTask** in the Project name field and click on *Finish*.



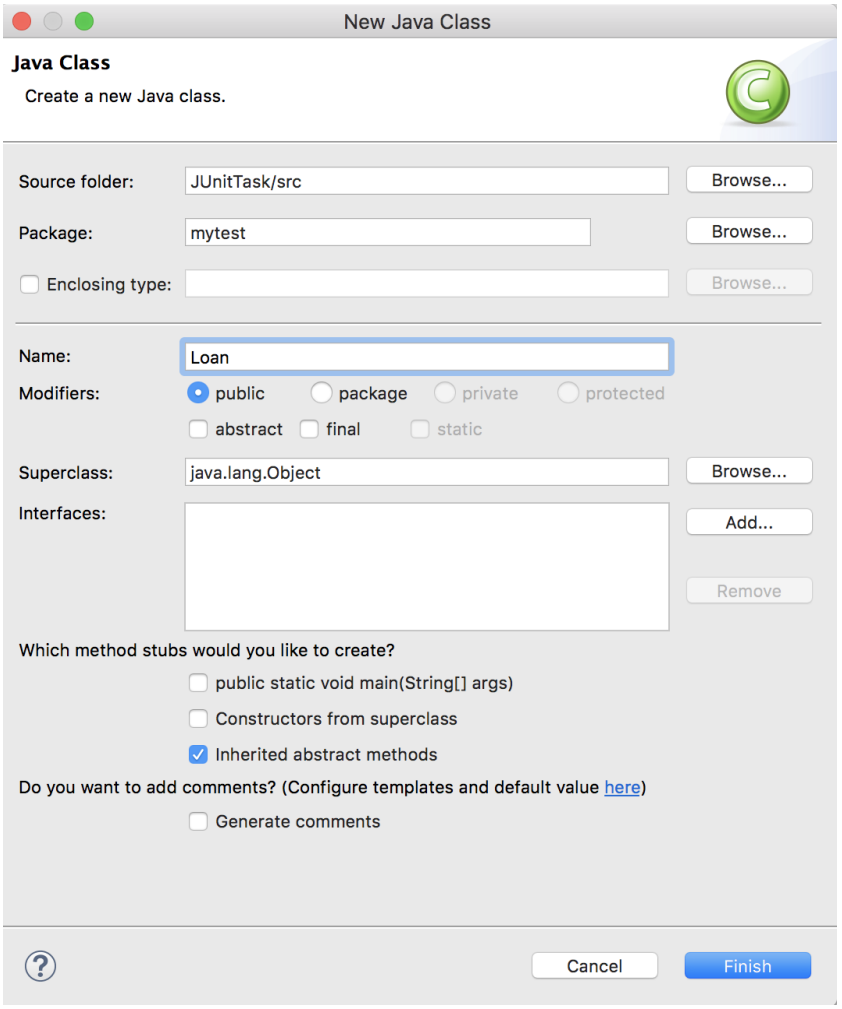
'New Java Project' window

- We now need to create a class to be tested. For this example let's create a class called Loan.
    - Right-click on the JUnitTask project in the Project Explorer.
    - Select *New → Class* from the menu
    - In the New Java Class dialogue box, enter **mytest** in the *Package* field and **Loan** in the *Name* field.
    - Click on *Finish* to create the class
    - Copy the code below to the *Loan* class and make sure the first line is `package mytest;`

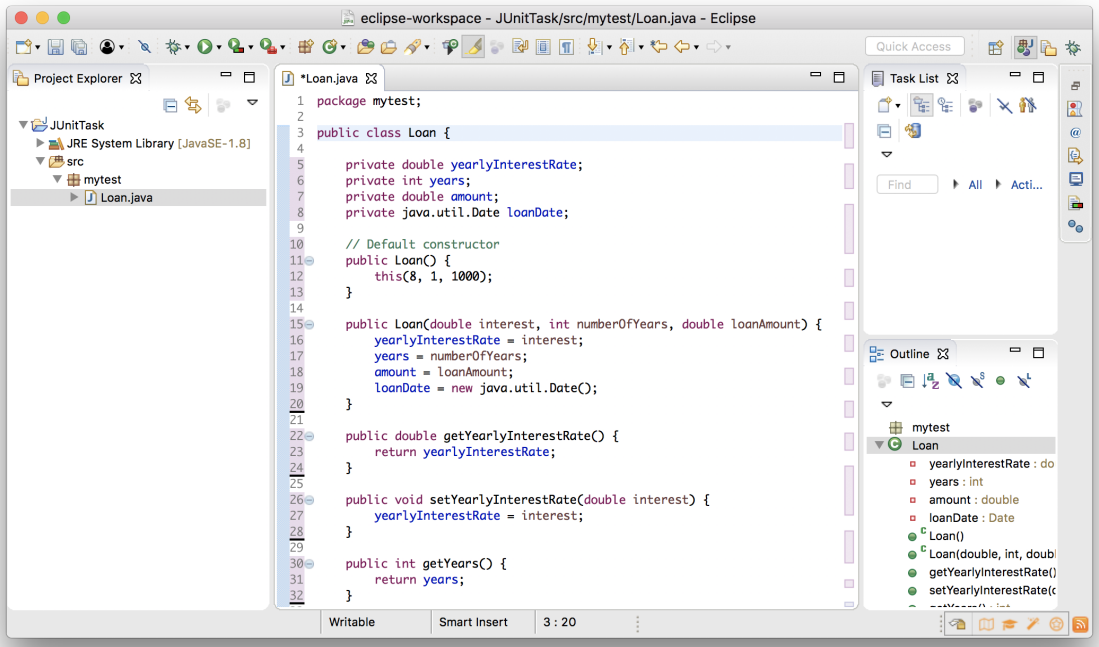
```
public class Loan {

    private double yearlyInterestRate;
    private int years;
    private double amount;
    private java.util.Date loanDate;

    //Default constructor
```

```
  public Loan() {
    this(8, 1, 1000);
  }

  public Loan(double interest, int numberOfYears,
      double loanAmount) {
    yearlyInterestRate = interest;
    years = numberOfYears;
    amount = loanAmount;
    loanDate = new java.util.Date();
  }


  public double getYearlyInterestRate() {
    return yearlyInterestRate;
  }


  public void setYearlyInterestRate(double interest) {
    yearlyInterestRate = interest;
  }


  public int getYears() {
    return years;
  }


  public void setYears(int numberOfYears) {
    years = numberOfYears;
  }


  public double getAmount() {
    return amount;
  }


  public void setAmount(double loanAmount) {
    amount = loanAmount;
  }

  //Calculate monthly payment
  public double getMonthlyPayment() {
    double monthlyInterestRate = yearlyInterestRate / 1200;
    double monthlyPayment = amount * monthlyInterestRate / (1 -
      (1 / Math.pow(1 + monthlyInterestRate, years * 12)));
    return monthlyPayment;
```

```
    }

    // Calculate total payment
    public double getTotalPayment() {
      double totalPayment = getMonthlyPayment() * years * 12;
      return totalPayment;
    }
}
```

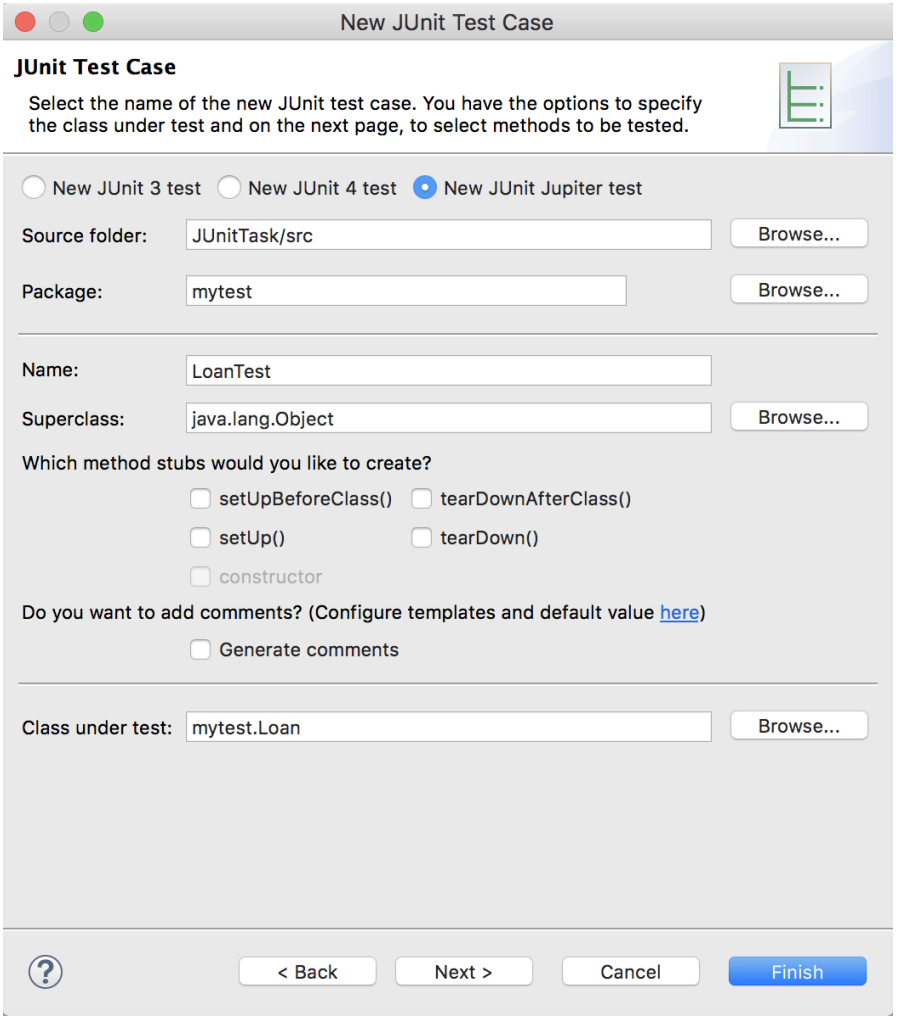


'New Java Class' window

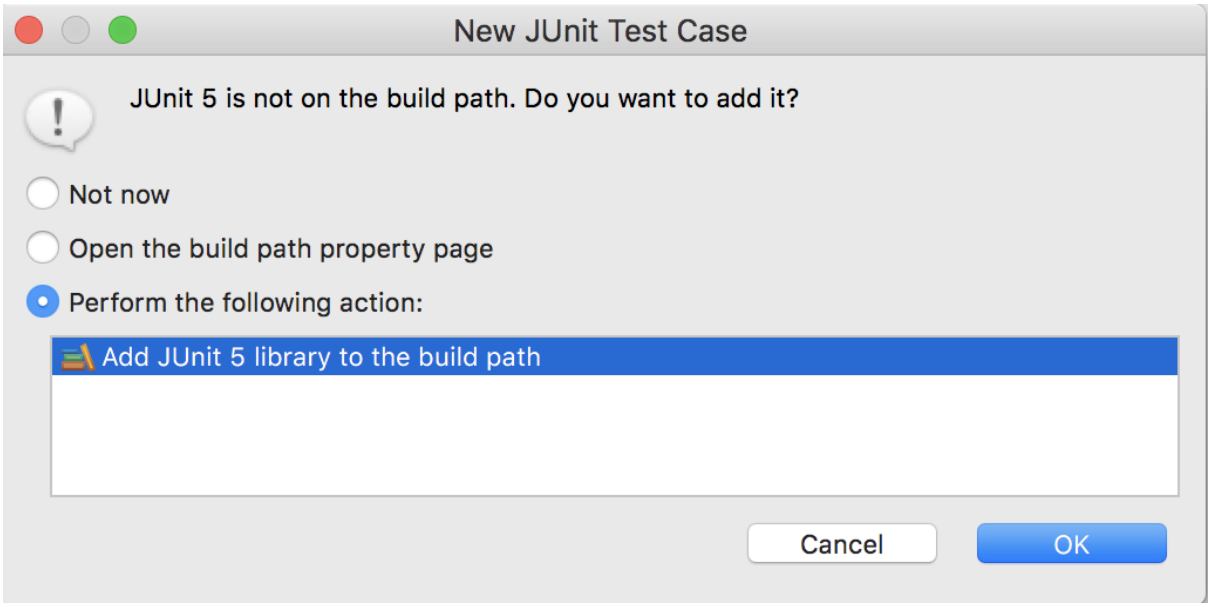


The Loan class shown in Eclipse

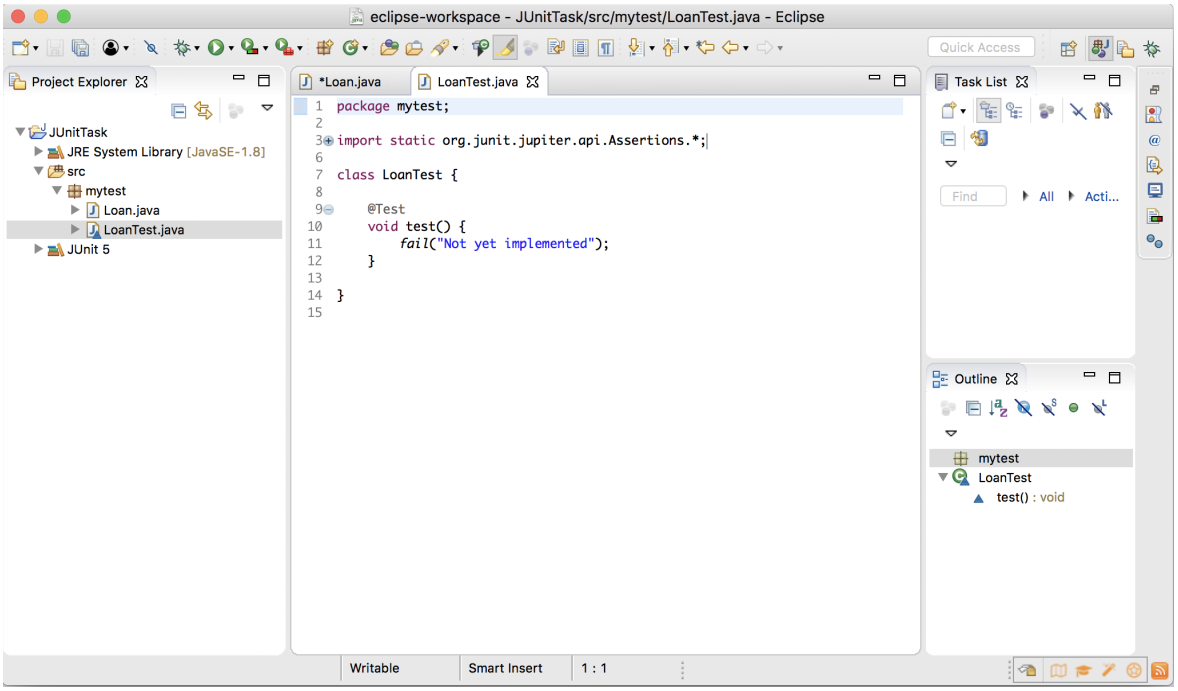
- You can now create a test class to test the Loan class.
    - Right-click *Loan.java* in the Project Explorer
    - Select *New* → *Other* from the menu
    - In the New window, select the JUnit folder and then *JUnit Test Case* and select *Next*.
    - In the New JUnit Test Case dialogue box should now appear. Simply select *Finish*.
    - After you select *Finish*, a dialogue will appear prompting you to add JUnit 5 to the project build path. Click *OK* to add it.
    - A test class named **LoanTest** should now be created.



'New JUnit Test Case' window



Dialogue prompting you to add JUnit 5 to the project build path



The LoanTest test class

WRITING TESTS

In your JUnit test case file, each unit test method should test a small aspect of the behaviour of the "class under test". In our example, the “class under test” is Loan and LoanTest should have testing methods each of which should be short and should test only one specific aspect of the Loan class.

JUnit testing method use assertions. Assertions statements check whether a condition is true or false. If the given condition is false, the test method fails otherwise, if all the given conditions are true the test method passes. Assertions are used to state things that you always expect to be true. It is a great way to detect and correct programming errors. An example of an assertion is **`assertEquals(10, list.size());`**. This statement expects a List to contain exactly 10 elements.

The following assertion methods are provided by JUnit:

`assertEquals(expectedValue, actualValue);`
 	
`assertEquals("message", expectedValue, actualValue)`
	Asserts if two values are equal. If the two values are not equal the test method fails. The expected value is passed in first and actual second. This method uses the *equals* method to compare objects.

`assertNotEquals(value1, value2)` `assertNotEquals("message", value1, value2)`	Asserts if two values are not equal to each other. If the two values are equal to each other the method fails. This method uses the *equals* method to compare objects.

`assertTrue(boolean condition)` `assertTrue(String message, boolean condition)`	AsseRts that a boolean condition is true. If the condition is not true the test method fails.
`assertFalse(boolean condition)` `assertFalse(String message, boolean condition)`	Asserts that a boolean condition is false. If the condition is not false the test method fails.
`assertNull(value)` `assertNull("message", value)`	Asserts that a value is null. If the value is not null the test method fails.
`assertNotNull(value)` `assertNotNull("message", value)`	Asserts that a value is not null. If the value is null the test method fails.
`assertSame(expectedValue, actualValue)` `assertSame("message", expectedValue, actualValue)`	Asserts if two values are the same. Similar to *assertEquals*, however it uses the `==` operator and not the *equals* method to compare values.
`assertNotSame(value1, value2)` `assertNotSame("message", value1, value2)`	Asserts if two values are not the same. Similar to *assertNotEquals*, however it uses the `==` operator and not the *equals* method to compare values.
`fail()` `fail(String message)`	Fails a test

For example look at the following LoanTest class:

```
package mytest;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class LoanTest {


  @Test
  public void testPaymentMethods() {
    double annualInterestRate = 2.5;
    int numberOfYears = 5;
    double loanAmount = 1000;
    Loan loan = new Loan(annualInterestRate, numberOfYears, loanAmount);

    assertTrue(loan.getMonthlyPayment() ==
      getMonthlyPayment(annualInterestRate, numberOfYears, loanAmount));

    assertTrue(loan.getTotalPayment() ==
      getTotalPayment(annualInterestRate, numberOfYears, loanAmount));
  }
```

```
  private double getMonthlyPayment(double annualInterestRate, int
numberOfYears,
    double loanAmount) {


    double monthlyInterestRate = annualInterestRate / 1200;
    double monthlyPayment = loanAmount * monthlyInterestRate /
      (1 - (1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12)));


    return monthlyPayment;

  }

  public double getTotalPayment(double annualInterestRate, int
numberOfYears,
    double loanAmount) {

    return getMonthlyPayment(annualInterestRate, numberOfYears,
      loanAmount) * numberOfYears * 12;
  }
}
```

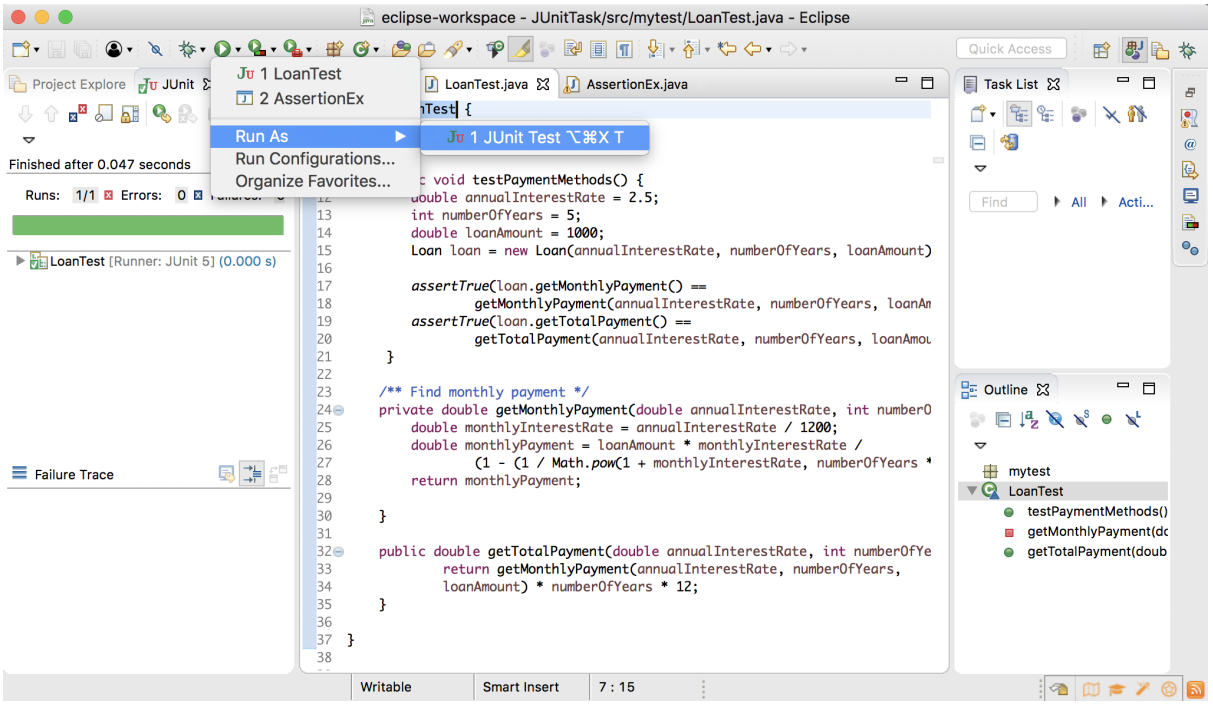
The `@Test` annotation specifies that `testPaymentMethods()` is the test method. The test method `testPaymentMethods()` in LoanTest creates an instance of Loan and tests whether `loan.getMonthlyPayment()` returns the same value as `getMonthlyPayment(annualInterestRate, numberOfYears, loanAmount)`, which is defined in the LoanTest class.

The test method also tests whether `loan.getTotalPayment()` returns the same value as `getTotalPayment(annualInterestRate, numberOfYears, loanAmount)`, which is also defined in the LoanTest class.

Copy and paste this code into LoanTest.

RUNNING YOUR TEST CASE

Once your test methods are written, they need to be run. This can be done in two ways. The first way is to click on the *Run* button in the toolbar at the top of the window then select *Run As → JUnit Test*.



How to run your test case

The second way is to right-click your JUnit test case class and choose Run As → JUnit Test.

On the left side of your screen, a new pane will appear showing the test results for each method. If all the tests are passed a green bar should appear in this pane. If any of the tests failed, a red bar will appear. You can view the details about the failure if any of the tests happen to fail. Simply click on the failed test's name and the details will appear in the pane below.

You might assume that getting a red bar is a very bad thing. However, it is just the opposite! Getting a red bar means that you have successfully identified a potential bug and you can now fix it. By fixing the code and then re-running the test program you can turn the red bar into a green one.

Compulsory Task 1

Follow these steps:

- Create a class called PrimeNumberMethod.
- A prime number is an integer greater than 1, that only has a positive divisor is 1 or itself.

- Create a method called isPrime that determines whether a given number is prime.

- Write a test class to test the method isPrime.

Compulsory Task 2

Follow these steps:

- Write a test class to test the following methods in the **java.lang.String** class:
    - length
    - charAt
    - substring
    - indexOf

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.