الجمهورية العربية السورية
جامعة حمص
الكلية التطبيقية
قسم تقانات الحاسوب
السنة الرابعة

الفصل الاول
٢٠٢٥- ٢٠٢٦

هندسة برمجيات متقدمة – القسم العملي

اختبار البرمجيات (JUnit)

المحاضرة السابعة

مدرسو المقرر

م.روز المشرقي

م.جنان الكردي

ما هو اختبار البرمجيات؟

- هي إحدى العمليات في دورة حياة البرمجية وتهدف إلى:
  - التحقق من أن المنتج البرمجي يقوم بما هو مطلوب منه
  - اكتشاف الأخطاء قبل وضع المنتج قيد الاستخدام
- يتم أثناء عملية الاختبار تشغيل البرنامج ببيانات اصطناعية Artificial Data
- يتم بعد ذلك تفقد النتائج للعثور على:
  - الأخطاء Errors
  - الشذوذ Anomalies
  - معلومات حول الصفات غير الوظيفية Non-Functional Attributes

أهمية اختبار البرمجيات

- يُعتبر اختبار البرمجيات عملية في غاية الأهمية لأنه يسمح باكتشاف الأخطاء والنقص في المنتج البرمجي وإيجاد حلول لها قبل تسليم المنتج للزبون
- تتميز المنتجات البرمجية التي تم اختبارها بشكل صحيح بعدة صفات منها
  - الموثوقية Reliability
  - الأمن Security
  - الأداء العالي High Performance
- يؤدي ذلك في نهاية الأمر إلى **تقليل زمن التطوير وتكلفته**، بالإضافة إلى زيادة **رضا الزبون** حول المنتج

أهداف اختبار البرمجيات

- للإثبات للمطور والزبون أن البرنامج يفي بالمتطلبات المتفق عليها
- لاكتشاف المواقف التي يكون فيها سلوك البرنامج **غير صحيح** أو **غير مرغوب فيه** أو **لا يتوافق مع مواصفاته**
  - تفاعل غير مرغوب فيه مع أنظمة أخرى
  - انهيارات في النظام System Crashes
  - حسابات غير صحيحة Incorrect Computations
  - فساد البيانات Data Corruption

Validation vs Verification

- **Verification**
  - هل نبني المنتج **الصحيح**؟
  - تتضمن التحقق من الوثائق والتصاميم والكود البرمجي
  - لا تتضمن تنفيذ الكود البرمجي (عملية ساكنة Static)
- **Validation**
  - هل نبني المنتج **بشكل صحيح**؟
  - تتضمن التحقق من المنتج بشكل فعلي
  - تتضمن تنفيذ الكود البرمجي (عملية ديناميكية Dynamic)

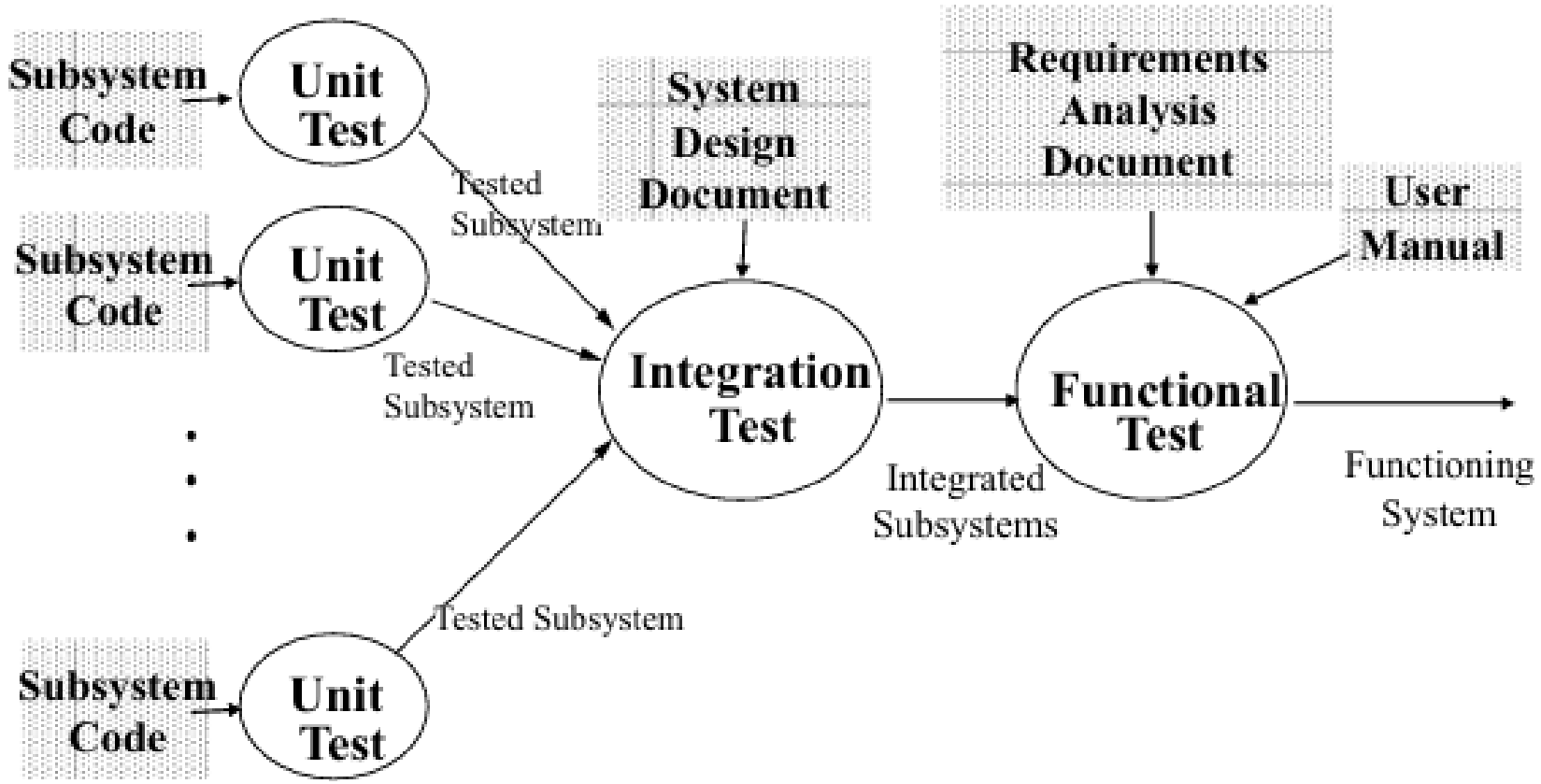
أنواع اختبار البرمجيات

- هناك العديد من أنواع الاختبارات على البرمجيات، ولكل منها أهداف واستراتيجيات خاصة بها. نذكر منها:
  - **اختبار القبول Acceptance Testing:** تقنية اختبار تستخدم لتحديد اذا كان النظام يحتوي على المتطلبات المحددة في وثيقة المتطلبات
  - **اختبار النظام System Testing:** اختبار النظام ككل ضمن بيئة شبيهة بالتي سيعمل ضمنها
  - **اختبار التكامل Integration Testing:** تقنية لاختبار أن مكونات النظام البرمجي تعمل معاً بشكل صحيح
  - **اختبار الوحدات Unit Testing:** تقنية للتحقق من أن كل وحدة في البرمجية تعمل كما يجب. علماً أن الوحدة هي أصغر مكون قابل للاختبار في النظام البرمجي

أنواع اختبار البرمجيات

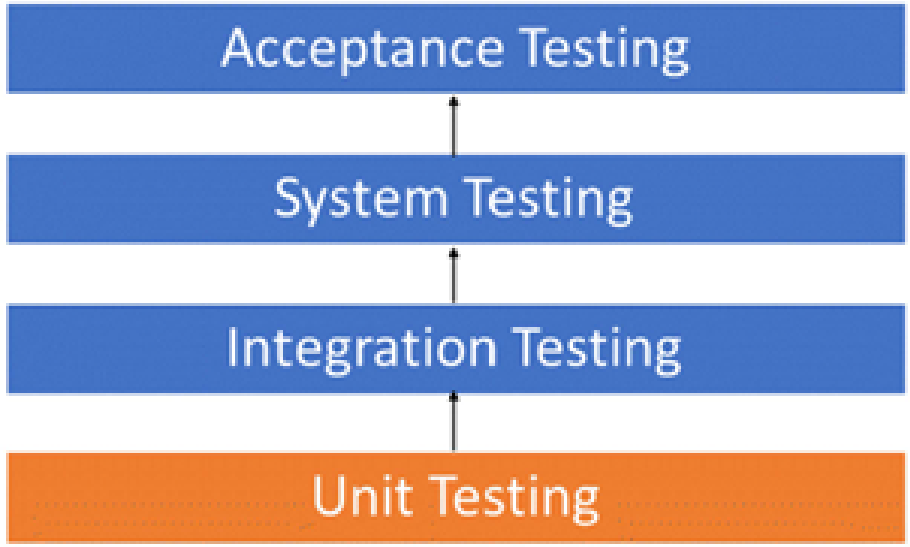
- **اختبار الأداء Performance Testing:** اختبار أداء البرنامج في ظل أعباء العمل المختلفة. يستخدم اختبار الحمل Load Test، على سبيل المثال، لتقييم الأداء في ظل ظروف الحمل الواقعية، بينما يُستخدم اختبار الضغط Stress Test لمعرفة مدى الضغط الذي يستطيع النظام تحمله قبل أن ينهار
- **اختبار قابلية الاستخدام Usability Testing:** التحقق من مدى قدرة الزبون على استخدام النظام لتحقيق هدف معين
- **اختبار التوافقية Compatibility Testing:** التحقق من عمل وظائف البرمجية على برامج وعتاد وشبكات ومتصفحات مختلفة

مستويات الاختبار :



اختبار الوحدات (Unit Testing)

- هو اختبار يجري فيه التحقق من مكونات البرمجية بهدف التأكد من أن كل وحدة في الكود البرمجي تعمل بشكل صحيح وكما هو متوقع منها
- أول مستوى اختبار قبل اختبار التكامل
- يُجرى اختبار الوحدات أثناء مرحلة التطوير
- قد تكون الوحدة
  - Function
  - Object
  - Module
  - ...



JUnit

- هو إطار (Framework) مفتوح المصدر (Open-Source) لإجراء اختبار الوحدات في لغة جافا
- يستطيع المطور باستخدامه بناء **حالات اختبار Test Cases** لاختبار الكود البرمجي الذي قام بتطويره
- حالة الاختبار هي كود برمجي يكتبه المطور لاختبار منطق عمل البرنامج وضمان أنه يعمل بالشكل المطلوب
- تتضمن الحزمة org.junit العديد من الواجهات والصفوف المستخدمة في اختبار الوحدات كما سنرى

ميزات JUnit

- إطار برمجي مفتوح المصدر
- يوفر مجموعة من ال annotations لتحديد دوال الاختبار
- يوفر مجموعة من ال assertions لاختبار النتائج المتوقعة
- يسمح بكتابة الكود بسرعة مما يزيد من جودة المنتج البرمجي
- يمكن تنظيم الاختبارات ضمن مجموعة Test Suite
- يمكن تنفيذ الاختبارات بشكل أوتوماتيكي والحصول على تقارير فورية

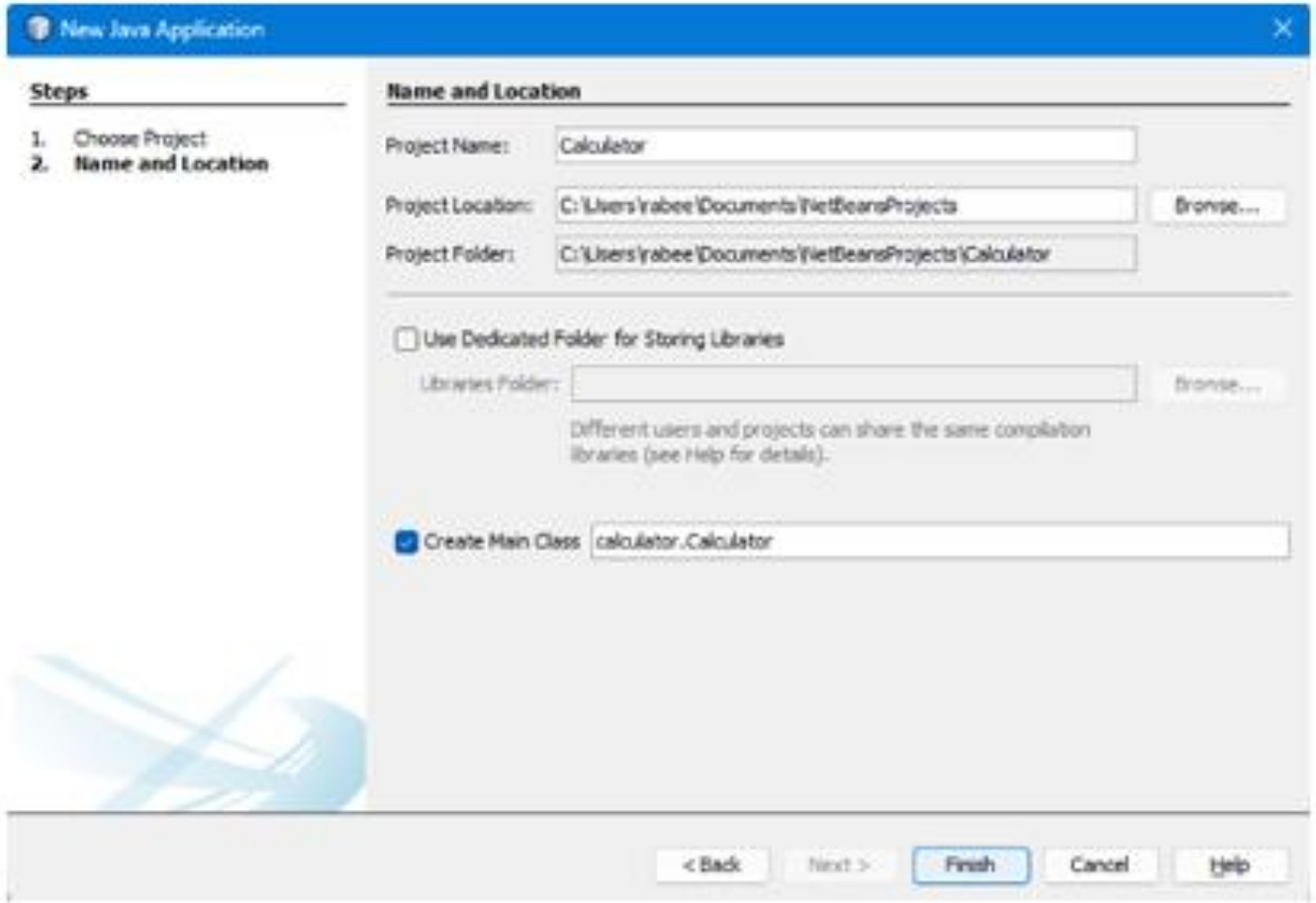
حالة اختبار الوحدة (Unit Test Case)

- هي جزء من الكود البرمجي، هدفه التحقق من أن جزء آخر من الكود (دالة مثلًا) يعمل بالشكل المطلوب
- تتصف حالة الاختبار المكتوبة بشكل صحيح بأنها تتألف من **دخل معروف known input** و**خرج متوقع expected output**
- يجب أن يكون هناك حالتي اختبار على الأقل لكل متطلب (حالة اختبار إيجابي وحالة اختبار سلبي)

JUnit

- لنقم بداية بإنشاء مشروع جافا جديد في Netbeans
- File ⟶ New Project ⟶ Java Application
- نسمي المشروع الجديد Calculator
- نسمي الصف الرئيسي calculator.Calculator
- ننشئ المشروع

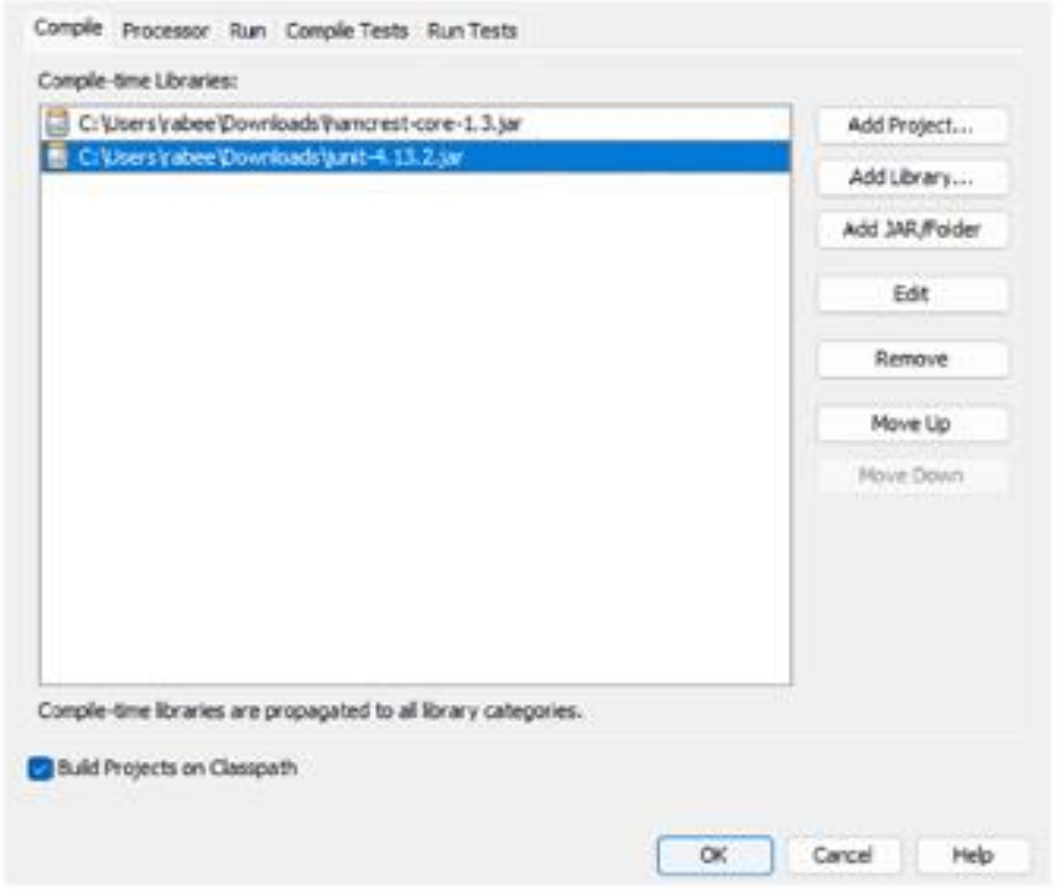
إنشاء مشروع

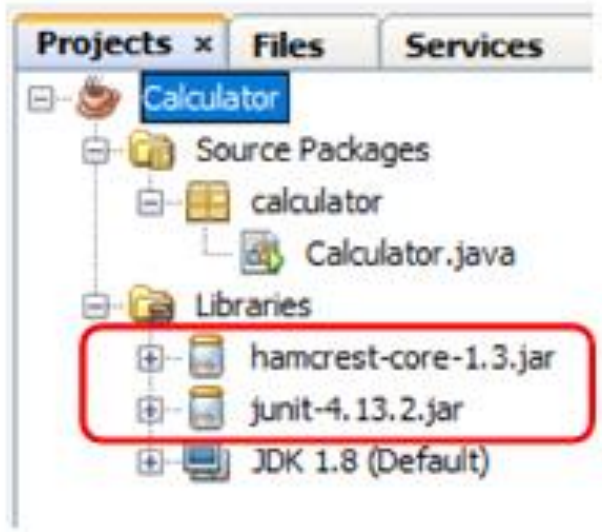

New Java Application
Steps
1. Choose Project
2. Name and Location
Name and Location
Project Name: Calculator
Project Location: C:\Users\rabee\Documents\NetBeansProjects
Browse...
Project Folder: C:\Users\rabee\Documents\NetBeansProjects\Calculator
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Browse...
Different users and projects can share the same compilation libraries (see Help for details).
Create Main Class calculator.Calculator
< Back
Next >
Finish
Cancel
Help

إضافة JUnit

- لإضافة JUnit إلى المشروع:
  - نقرة يمينية على المشروع Calculator
  - نختار Properties، تظهر لنا نافذة خصائص المشروع
  - نختار Libraries
  - ننقر على زر Add JAR/Folder
  - نختار الملفين **junit-4.13.2.jar** و**hamcrest-core-1.3.jar**
  - ننقر Open ثم Ok

إضافة JUnit


Compile
Processor
Run
Compile Tests
Run Tests
Compile-time Libraries:
C:\Users\rabee\Downloads\hamcrest-core-1.3.jar
C:\Users\rabee\Downloads\junit-4.13.2.jar
Add Project...
Add Library...
Add JAR/Folder
Edit
Remove
Move Up
Move Down
Compile-time libraries are propagated to all library categories.
Build Projects on Classpath
OK
Cancel
Help


Projects
Files
Services
Calculator
Source Packages
calculator
Calculator.java
Libraries
hamcrest-core-1.3.jar
junit-4.13.2.jar
JDK 1.8 (Default)

إضافة منطق العمل

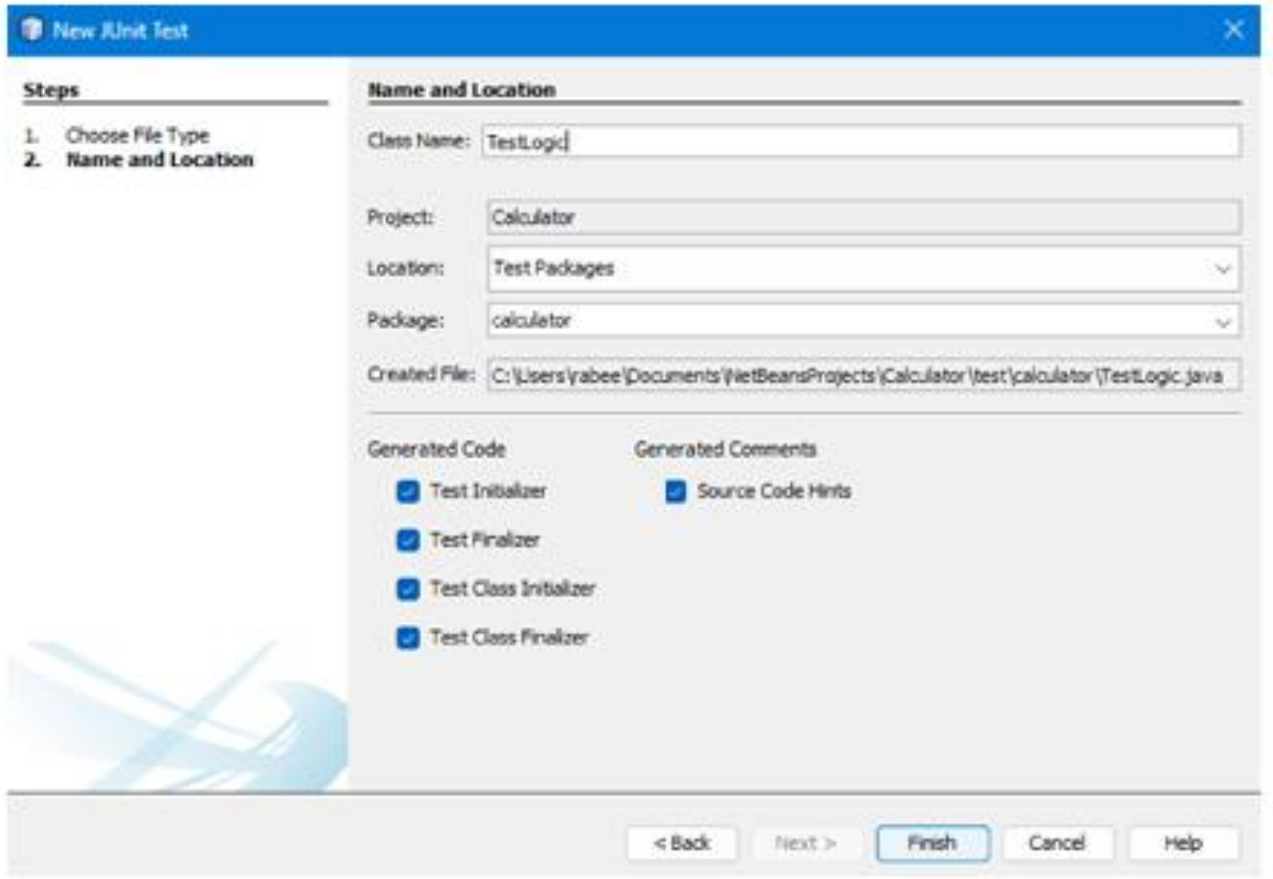
- لنكتب دالة تقوم بإيجاد القيمة العظمى Max من مصفوفة قيم:

```
public static int findMax(int arr[]){
    int max = 0;
    for(int i=1; i<arr.length; i++){
        if(max < arr[i])
            max=arr[i];
    }
    return max;
}
```

إنشاء حالات الاختبار

- هناك عدة طرق لإنشاء حالات الاختبار
- نقرة يمينية على المشروع Calculator
- نختار new ← other ← JUnit Test (من مجموعة Unit Tests)
- نحدد اسم صف الاختبار والحزمة والخيارات الأخرى
- Finish

إنشاء حالات الاختبار



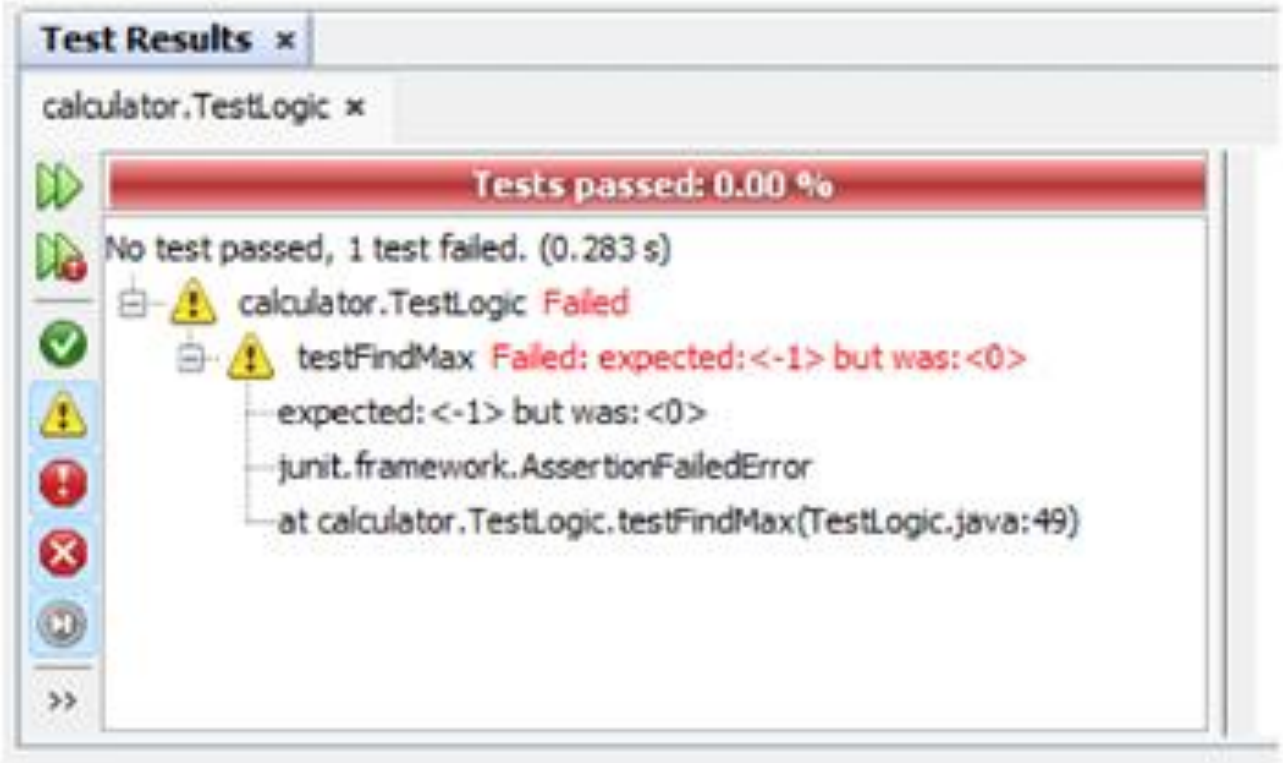
إنشاء حالات الاختبار

- لنقم الآن بإضافة كود الاختبار على الصف TestLogic كما يلي:

```
@Test
public void testFindMax(){
    int[] test_input_a = new int[]{1, 3, 4, 2};
    int[] test_input_b = new int[]{-12, -1, -3, -4, -2};

    int expected_output_a = 4;
    int expected_output_b = -1;

    int actual_output_a = Calculator.findMax(test_input_a);
    int actual_output_b = Calculator.findMax(test_input_b);

    assertEquals(expected_output_a, actual_output_a);
    assertEquals(expected_output_b, actual_output_b);
}
```

تنفيذ الاختبار

- لنقم الآن بتنفيذ حالة الاختبار السابقة
- نقرة يمينة على TestLogic ← Test file



نتائج الاختبار

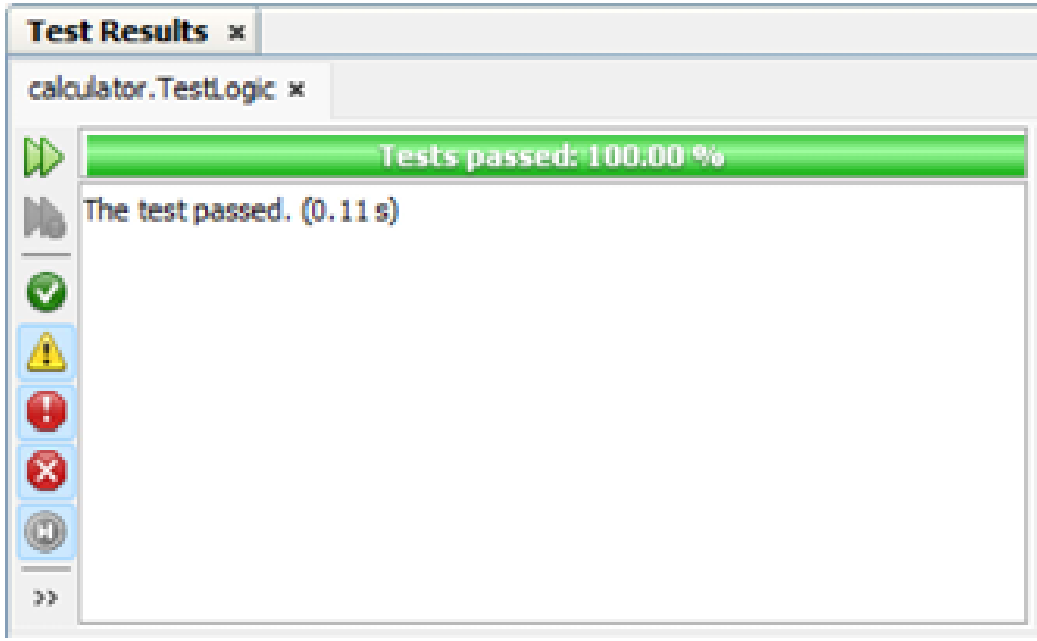
- نلاحظ مما سبق فشل الاختبار مع رسالة الخطأ

expected:<-1> but was:<0>

- يعود سبب ذلك إلى خطأ في منطق عمل دالة إيجاد القيمة العظمى، إذ أننا قمنا بإسناد قيمة ابتدائية للمتحول max تساوي ٠، بالتالي فإن الدالة سوف تفشل عند محاولة إيجاد قيمة عظمى بين مجموعة قيم سالبة
- لنقم بتصحيح الدالة findMax ثم إعادة تنفيذ الاختبار

تصحيح منطق العمل

```
public static int findMax(int arr[]){
    int max = arr[0];
    for(int i=1; i<arr.length; i++){
        if(max < arr[i])
            max=arr[i];
    }
    return max;
}
```



JUnit Annotations

- تعتمد JUnit على مجموعة من annotations أثناء كتابة حالات الاختبار

Annotation	الشرح
@Test	تحدد أن الدالة هي دالة اختبار
@BeforeClass	تحدد أنه سيتم تنفيذ الدالة مرة واحدة فقط قبل جميع الاختبارات. يمكن استخدامها لإجراء العمليات المكلفة زمنياً، مثل الاتصال مع قاعدة بيانات
@Before	تحدد أنه سيتم تنفيذ الدالة قبل كل اختبار. يمكن استخدامها لتحضير بيئة الاختبار (مثل قراءة بيانات الدخل، تهيئة الصف، ...)

JUnit Annotations

- تعتمد JUnit على مجموعة من annotations أثناء كتابة حالات الاختبار

Annotation	الشرح
@After	تحدد أنه سيتم تنفيذ الدالة بعد كل اختبار. يمكن استخدامها لعمليات تنظيف بيئة الاختبار (مثل حذف البيانات المؤقتة، إعادة تعيين القيم الافتراضية، ...). كما يمكن لها تحسين استهلاك الذاكرة المؤقتة بحذف هياكل البيانات الضخمة من الذاكرة
@AfterClass	تحدد أنه سيتم تنفيذ الدالة مرة واحدة فقط بعد انتهاء جميع الاختبارات. يمكن استخدامها لإجراء عمليات التنظيف مثل إنهاء الاتصال مع قاعدة البيانات

JUnit Annotations

- لنقم باختبار تسلسل تنفيذ الدوال المضاف لها annotations وذلك بإضافة الدوال التالية:

```
@BeforeClass
public static void setUpClass() {
    System.out.println("Before Class");
}


@AfterClass
public static void tearDownClass() {
    System.out.println("After Class");
}
```

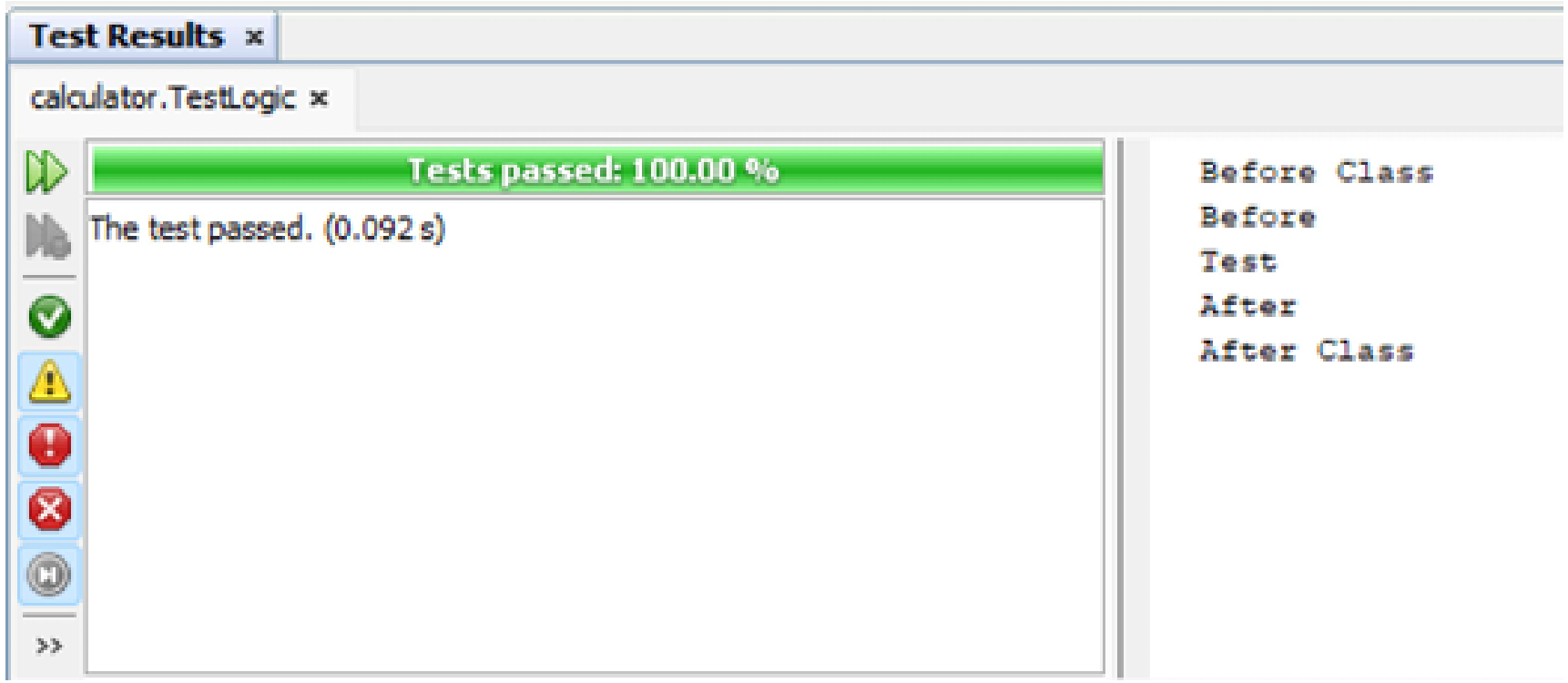
JUnit Annotations

- لنقم باختبار تسلسل تنفيذ الدوال المضاف لها annotations وذلك بإضافة الدوال التالية:

```
@Before
public void setUp() {
    System.out.println("Before");
}


@After
public void tearDown() {
    System.out.println("After");
}
```

JUnit Annotations

- نتيجة تنفيذ الاختبار



JUnit Assert

- تتضمن JUnit تعليمات Assert هدفها إجراء الاختبارات والتحقق من منطق العمل

الدالة	شرح عملها
assertTrue([msg], boolean condition)	تتحقق فيما إذا كان الشرط **محققاً**. ([msg] وسيط اختياري، يمكن من خلاله تخصيص الرسالة التي تظهر عند فشل الاختبار). مثال: `assertTrue("condition is True", actual_output > 5);`
assertFalse([msg], boolean condition)	تتحقق فيما إذا كان الشرط **غير محقق**. مثال: `assertFalse("condition is False", actual_output > 5);`

JUnit Assert

- تتضمن JUnit تعليمات Assert هدفها إجراء الاختبارات والتحقق من منطق العمل

الدالة	شرح عملها
assertEquals([msg], expected, actual)	تتحقق فيما إذا كانت القيمة الحقيقية الناتجة عن عملية ما **actual** مطابقة للقيمة المتوقعة **expected**. مثال: `assertEquals("Same", 5, actual_value);` ينجح الاختبار في حال كانت قيمة actual_value مساوية ل ٥
assertEquals([msg], expected, actual, [delta])	تتحقق فيما إذا كانت قيمتين من نوع double أو float متطابقتين. يحدد الوسيط delta مقدار الخطأ المسموح به. ينجح الاختبار طالما أن $\|expected - actual\| \leq delta$ مثال: `assertEquals(3.33, x, 0.01);`

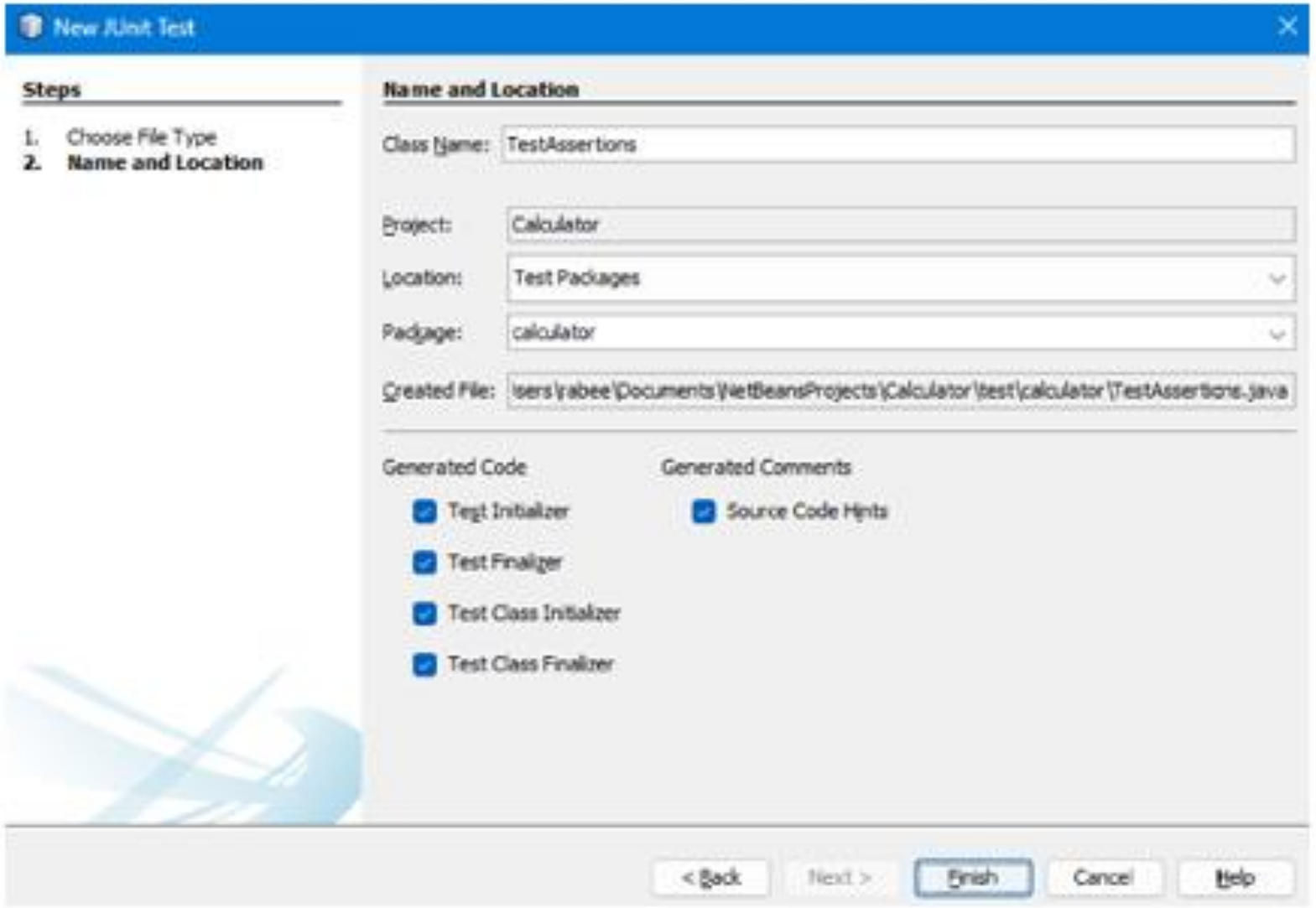
JUnit Assert

- تتضمن JUnit تعليمات Assert هدفها إجراء الاختبارات والتحقق من منطق العمل

الدالة	شرح عملها
assertArrayEquals([msg], expected_array, actual_array)	تتحقق فيما إذا كانت مصفوفتان متطابقتان
assertNull([msg], object)	تتحقق فيما إذا كان الغرض Null
assertNotNull([msg], object)	تتحقق فيما إذا كان الغرض ليس Null
assertSame([msg], object1, object2)	تتحقق فيما إذا كان المتحولان يشيران إلى نفس الغرض
assertNotSame([msg], object1, object2)	تتحقق فيما إذا كان المتحولان يشيران إلى غرضين مختلفين

تمرين ١

- سوف نقوم باختبار جميع تعليمات Assert
- نبدأ بإضافة صف اختبار جديد
- نقرة يمينية على المشروع
- نختار new ← other ← JUnit Test (من مجموعة Unit Tests)
- نحدد اسم صف الاختبار والحزمة والخيارات الأخرى
- Finish


New JUnit Test
Steps
1. Choose File Type
2. Name and Location
Name and Location
Class Name: TestAssertions
Project: Calculator
Location: Test Packages
Package: calculator
Created File: \sers\rabee\Documents\NetBeansProjects\Calculator\test\calculator\TestAssertions.java
Generated Code
Test Initializer
Test Finalizer
Test Class Initializer
Test Class Finalizer
Generated Comments
Source Code Hints
< Back
Next >
Finish
Cancel
Help

لنقم الآن بإضافة الدالة someTests إلى الصف TestAssertions كما يلي:

```
@Test
public void someTests() {
    //test data
    String str1 = new String("abc");
    String str2 = new String("abc");
    String str3 = null;
    String str4 = "abc";
    String str5 = "abc";
    int val1 = 5;
    int val2 = 6;
    double val3 = 10.0;
    double val4 = 3.0;
    String[] expectedArray = {"one", "two", "three"};
    String[] resultArray = {"one", "two", "three"};
```

إضافة تعليمات الاختبار

```
//Check that two objects are equal
assertEquals(str1, str2);

//Check that two float values are equal within delta
assertEquals(3.0, val3/val4, 1);
assertEquals(3.0, val3/val4, 0.1);

//Check that a condition is true
assertTrue(val1 < val2);

//Check that a condition is false
assertFalse(val1 > val2);
```

```
    //Check that an object isn't null
    assertNotNull(str1);

    //Check that an object is null
    assertNull(str3);

    //Check if two object references point to the same object
    assertSame(str4, str5);

    //Check if two object references not point to the same object
    assertNotSame(str1, str3);

    //Check whether two arrays are equal to each other.
    assertArrayEquals(expectedArray, resultArray);
}
```