Unit Testing with JUnit	start time:

This activity will introduce key ideas & issues in testing the behavior of individual methods. We will use the Java language, the BlueJ environment, and the JUnit test framework, but the same concepts apply to many other languages, environments, & frameworks.

Before you start, complete the form below to assign a role to each member. If you have 3 people, combine Speaker & Reflector.

Team	**Date**
Team Roles	**Team Member**
Recorder: records all answers & questions, and provides copies to team & facilitator.	
Speaker: talks to facilitator and other teams.	
Manager: keeps track of time and makes sure everyone contributes appropriately.	
Reflector: considers how the team could work and learn more effectively.	

Notes

1. When a question asks you to **explain** something, always answer in **complete sentences**.
2. **Recorder**: Write legibly & neatly so that everyone can read & understand your responses.
    a. Note the time whenever your team starts a new section or question.
3. You will need a computer with BlueJ open and ready to use below.
    a. In BlueJ, go to Tools > Preferences > Editor to adjust the font size.
    b. Consider using an external monitor or projector so everyone can see BlueJ easily.

(15 min) VI. JUnit	start time:

JUnit is a software framework (toolkit) to help people develop & run unit tests in Java.
Similar frameworks are available for many other programming languages.

In BlueJ, find the *Preferences* menu and find the place to check "Show unit testing tools".
(Apple and Windows may have these in slightly different places.)
Next, right-click on the icon for class `GC` and select "Create Test Class"
to create a corresponding unit test class. Open it and study the contents.
Notice that the test class starts with a set of import statements. If this is unfamiliar,
don't worry – this allows `GCTest` to use methods defined in those classes.

1. (2 min) What is the purpose of the methods **`setUp()`** & **`tearDown()`**?
The **`@Before`** and **`@After`** are **annotations** that tag these methods
so JUnit can find them. (Don't worry if you haven't seen annotations before.)
Can you guess why this important? If not, don't worry – we will discuss this later.

2. (3 min) To define an **empty unit test**, create a method in `GCTest`
with a **`@Test`** annotation, no return value, no parameters,
and a name that includes the tested method and the test case, e.g.:

```
    @Test public void gToP_A() { }
```

JUnit automatically finds every **`@Test`** method like this and runs them as unit tests.
Create an empty test method for **each test case** you identified in Section II.

3. (1 min) Once `GCTest` compiles successfully, right-click on its icon and select "Test All".
What happens?

4. (5 min) JUnit provides a set of methods to simplify unit tests.
For example, to check whether a method returns a correct value,
use an `assertEquals` methods, which take an **expected value** and an **actual value**.
If the values are **equal**, the test passes; otherwise, the test fails.

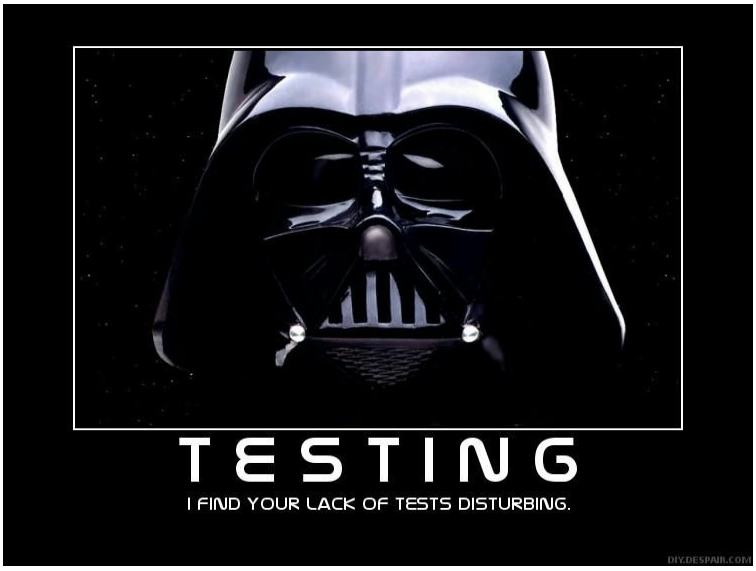
```
// these are EXAMPLES - do NOT copy into BlueJ

// 2+2 should equal 4
assertEquals( 4  , 2+2 );
// floating point calculations involve rounding
// 1.0/3.0 should be within 0.01 of 0.33
assertEquals( 0.33, 1.0/3.0, 0.01 );

assertEquals( "A", GC.gToP(100.0) );
```

Add an appropriate assertion to each of the **empty test methods** you wrote above,
run the tests, and describe what happens.

5. Give a **copy** of `class TCTest` to the facilitator to review (printout or email).
If you are editing this document electronically, paste the code below or at the end.
JUnit provides a variety of other methods and tools, which you will learn about later.



Handout: JUnit Assert Methods

```
// methods to check boolean conditions
void assertTrue   (            boolean cond)
void assertFalse  (            boolean cond)
void assertTrue   (String msg, boolean cond)
void assertFalse  (String msg, boolean cond)

// methods to compare values
void assertEquals (            long      expected, long   actual)
void assertEquals (            double    expected, double actual,
                                                   double delta )
void assertEquals (String msg, long      expected, long   actual)
void assertEquals (String msg, double    expected, double actual,
                                                   double delta )

// methods to compare object contents (including Strings)
void assertEquals (            Object    expected, Object actual)
void assertEquals (String msg, Object    expected, Object actual)

// methods to compare object references
void assertNotSame(            Object  unexpected, Object actual)
void assertNotSame(String msg, Object  unexpected, Object actual)
void assertSame   (            Object    expected, Object actual)
void assertSame   (String msg, Object    expected, Object actual)

// methods to check for null references
void assertNull   (            Object  o)
void assertNull   (String msg, Object  o)
void assertNotNull(            Object  o)
void assertNotNull(String msg, Object  o)

// methods that always fail
void          fail(            )
void          fail(String msg  )
```