TDD for the iPhone

using Google Toolbox for Mac

Where to get it: http://code.google.com/p/google-toolbox-for-mac/wiki/iPhoneUnitTesting

Setup (instructions also available on website):

1. Create a new iPhone Target (Cocoa Touch Application) via "Project Menu > New Target...". Choose a name that makes some sense such as "Unit Test".
2. Right click the target and choose to add Existing Files.
3. Add google-toolbox-for-mac/UnitTesting/GTMIPhoneUnitTestMain.m to your target
4. Add google-toolbox-for-mac/UnitTesting/GTMIPhoneUnitTestDelegate.m to your target
5. Add google-toolbox-for-mac/UnitTesting/GTMIPhoneUnitTestDelegate.h to your target
6. Add google-toolbox-for-mac/UnitTesting/GTMSenTestCase.m to your target
7. Add google-toolbox-for-mac/UnitTesting/GTMSenTestCase.h to your target
8. Add google-toolbox-for-mac/GTMDefines.h to your target
9. Add a new 'run script' build phase as the last step of your target build via "Project Menu > New Build Phase > New Run Script Build Phase", and dragging it to the end of the build steps if needed.
10. Edit your Run Script Build Phase by double clicking it, and set the shell to "/bin/sh" and the script to "PATH_TO_GTM/UnitTesting/RunIPhoneUnitTest.sh", where PATH_TO_GTM is the path to your local version of google-toolbox-for-mac.
11. Build! Note that if you choose build and go you will see your unit tests executed twice, once as part of the build script, and once being run

Your target should now build cleanly, and if you check the build log you should see something like: "Executed 0 tests, with 0 failures (0 unexpected) in 0.001 (0.001) seconds" at the end.

Common Assertions:

STAssertFalse
STAssertTrue
STAssertEquals
STAssertNotEquals
STAssertEqualObjects
STAssertNULL
STAssertNotNULL

The full list of assertions can be found in the GTMSenTestCase file.

C++ to Objective-C cheat sheet

Objective-C developers often brag that its simple to switch from C based languages, Java this means you too, to Objective-C. I'm not sure I'd call it simple, as the thought process is very different, but there is a simple conversion for many common tasks in C/C++. Keep in mind that Objective-C is an offshoot of C, not C++, and as such many C++ constructs are not available to you. It does mean that C code can be, and frequently is, interspersed in Objective-C. Now on to the cheat sheet:

	C++	Objective-C

Creating a class	`class A : public B {`
	`public:`
	`  int method();`
	`  int A;`
	`};`
		`@interface A : B`
		`{`
		`  int A;`
		`}`
		
		`-(int) method;`
		
		`@end`

Implementing a class	`#include "test.h"`
	
	`int A::method() {`
	`  return 0;`
	`}`
		`@implementation A`
		
		`-(int) A {`
		`  return 0;`
		`}`
		
		`@end`

Header File Extension	.h	.h
Implementation File Extension	.cpp	.m
Self references	`this`	`self`
Private	Declared with the private: keyword.	The concepts of private/protected/public do not exist in Objctive-C. All the "ivars" - instance variables – are inaccessible without an accessor method. You can define a method in a classes implementation and not in its corresponding header file, which will compiler warnings if it is called in another file, but it can still be called
Calling methods	`object.method();`	`[object method];`
Multiple Parameters	`object.method(1, 2);`	`[object method: A with: B];`
Creation	`Object* object = new Object();`	`Object* object = [Object alloc];`

Constructors	`ObjectName();`	`- (id)init`
		`{`
		`  [super init];`
		
		`  return self;`
		`}`
		
		Note: Objective-C methods do not implicitly call any constructors, a common pattern is:
		
		`Object* object =`
		`  [[Object alloc] init];`

Destruction	`~ObjectName();`	`- (void)dealloc` `{`

	Don't forget to make it virtual.	[super dealloc]; }
Deleting Objects	`delete object;`	`[object release];`

Memory Management

Memory Management in Objective-C is non-trivial subject and is out of the scope of this handout. For a thorough treatment it is highly recommended you read the Apple Guidelines.