National University of Singapore
School of Computing
CS3216: Software Development on Evolving Platforms
AY2011/2012, Semester 2

Problem Set 1: **Hello iPad!**

Issue date: 12 January 2012
Due date: **15 January 2012, 23:59**

Introduction

In this assignment, you will be introduced to Xcode, which is the IDE for Mac OS and iOS development. Also, you will take the opportunity to learn the basics of the Objective-C programming language.

The following files are provided with this problem set:

- `cs3217_logo.png`, an image file that you can use for the first part of the assignment;
- `PEShape.h` and `PERectangle.h`, containing declarations for the classes used in the second part of the assignment;
- `Overlaps.m` and `PERectangle.m`, containing stubs for the implementation of the class used in the second part of the assignment.

You can download these files as a zip file from the IVLE Workbin. Let's begin...

Reminder:	Please read the entire assignment before starting.

Getting Started

Xcode is already installed in the iMacs in Programming Lab 1, so there is no need for you to install Xcode if you work in the lab. If you plan to work on this assignment and subsequent assignments on your own iMac or Macbook, you will have to install Xcode yourself. If so, we will leave you to figure out how to download and install Xcode.

This week's assignment has two parts. The goal of the first part of the assignment is to create your first iPad application, *HelloIPad*. You do not have to worry, because you will be given detailed step-by-step instructions for this first assignment.

Important Note:	Even though it is possible to complete the first part of the assignment just by following instructions, it is important for you to understand what you are doing in each step, since in the next assignments you will be required to use the same tools.

As the weeks go by, you will be required to be increasingly independent and figure out more things for yourself. Part of the learning experience is to learn how to figure things out even if you are not told explicitly what to do.

In the second part of this assignment, the goal is to get used to programming Objective-C. For this part, your goal is to develop a Mac command line tool. During this part, you will have to remember the C programming language (or learn C really quickly) and augment your knowledge with Objective-C syntax, constructs, and base classes.

Objective-C and Cocoa

Objective-C is the primary language that will used to write Mac and iPhone/iPad software. During this semester, all coding assignments will be done using Objective-C. If you're comfortable with basic object-oriented concepts and the C language, the extensions provided by Objective-C should be quite natural (though the syntax might seem a little odd) and it should be easy to pick up Objective-C during CS3217. More information and resources about Objective-C can be found on the Apple developer pages at `http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf`.

Cocoa is Apple's native object-oriented application programming interfaces (APIs) for the Mac OS X operating system. For iPad programming, we will be using **Cocoa Touch**, a lightweight and touch-oriented API that provides the key frameworks for developing applications on devices running iOS. Similar to Cocoa, Cocoa Touch follows a *Model-View-Controller (MVC)* software architecture. More information about Cocoa and Cocoa Touch is found on the Apple developer pages at `http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/Introduction/Introduction.html%23//apple_ref/doc/uid/TP40002974`.

The iOS SDK and Tools

To develop applications for iOS, you need a Mac OS X computer running the Xcode tools. Xcode is Apple's suite of development tools that provide support for project management, code editing, building executables, source-level debugging, source-code repository management, performance tuning, and so on. The **Xcode** application provides the basic source-code development environment (IDE). **Interface Builder** is the tool you use to assemble your applications user interface visually. Using Interface Builder, you assemble your applications window by dragging and

dropping preconfigured components onto it. Finally, **Instruments** allows you analyze the performance of iOS applications while running in the simulator or on a device. You can find information about the iOS development using the SDK on the Apple developer pages at `http://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf`.

Note also that there are several free Apple developer e-books that you can download and read with iBooks on your iPad.

Part 1: Hello, iPad!

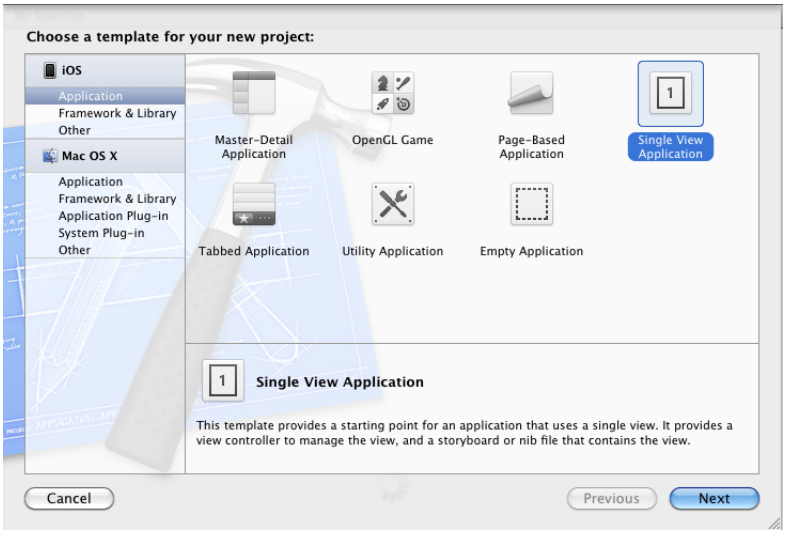
In this assignment, you will use Xcode to create a project and Interface Builder to construct a very simple user interface. This assignment requires no coding. It is intended to provide you with a quick introduction to the tools that you'll be using during the course. Follow this walkthrough and build your first HelloIPad application.

Milestone 1:	Create a new project in Xcode. *(Not graded.)*

a. Launch `/Developer/Application/Xcode`.

b. From the initial screen, choose *Create new Xcode project*.

c. In the *New Project* dialog, select *Application* under the iOS tab in the left, choose *Single-View-based Application*, and select *iPad* from the Product drop-down list.



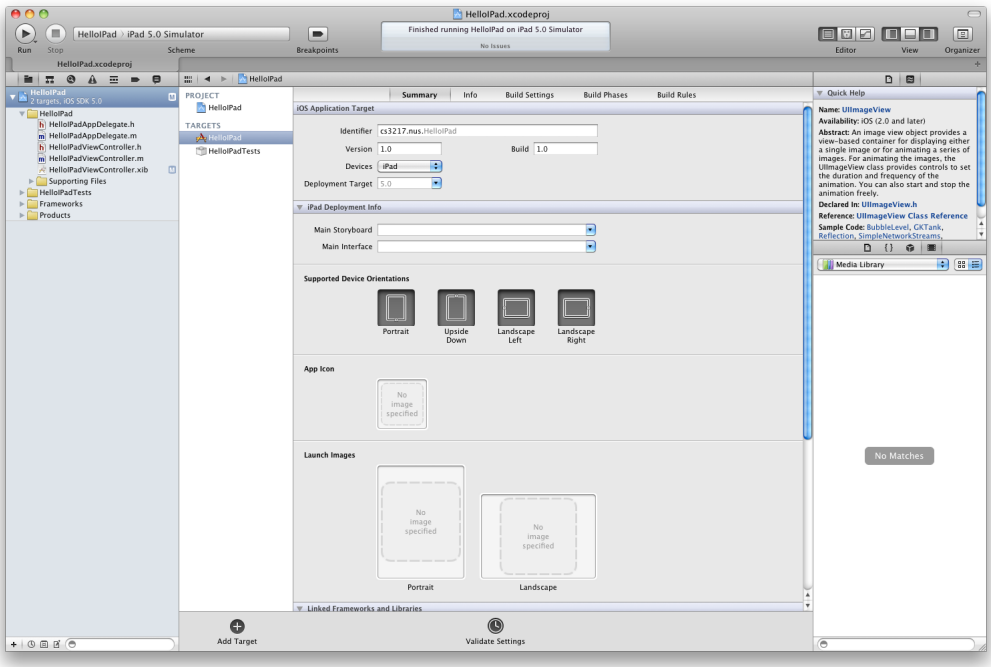
The *Single View Application* is a Xcode template that creates an application containing one window, and in that window is included an empty **View**. The template also creates a subclass to be the **Controller**. The subclass already has some template code for the common object operations.

d. Xcode will ask for options to configure the project, you can fill in Company Identifier as nus.cs3217 and leave prefix empty. You should **deselect** *Use Storyboard*. For other options, you can follow the default configurations.

e. In the file chooser dialog screen that is presented next, navigate to a folder where you want to place the project, and in the *Save As:* field type `HelloIPad`, the name of the application.

With iOS 5 SDK, Apple introduces Storyboard as an alternative to visualize workflow and UI design. Feel free to explore Storyboard and you use it for other assignments.

Congratulations, you have successfully created your first iPad project! You will be prompted by the project window, which looks like the figure below.



Milestone 2: Run your project in the simulator. *(Not graded.)*

After creating the project, you can already run your application in the simulator.

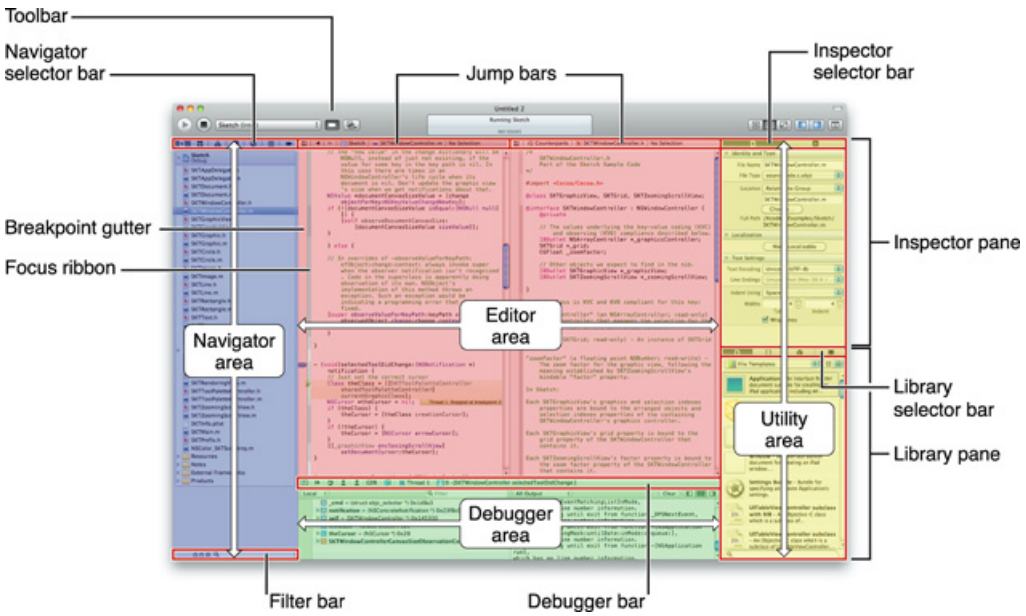
a. Select *iPad 5.0 Simulator* in the drop-down menu on the left of the project window toolbar.

b. Press the *Run* button in the toolbar. Xcode will compile your project, after which the simulator will start, install your application and run it.

A blank screen will appear in the iPad simulator like in the figure below. This screen is from the view that is created by the Xcode template.

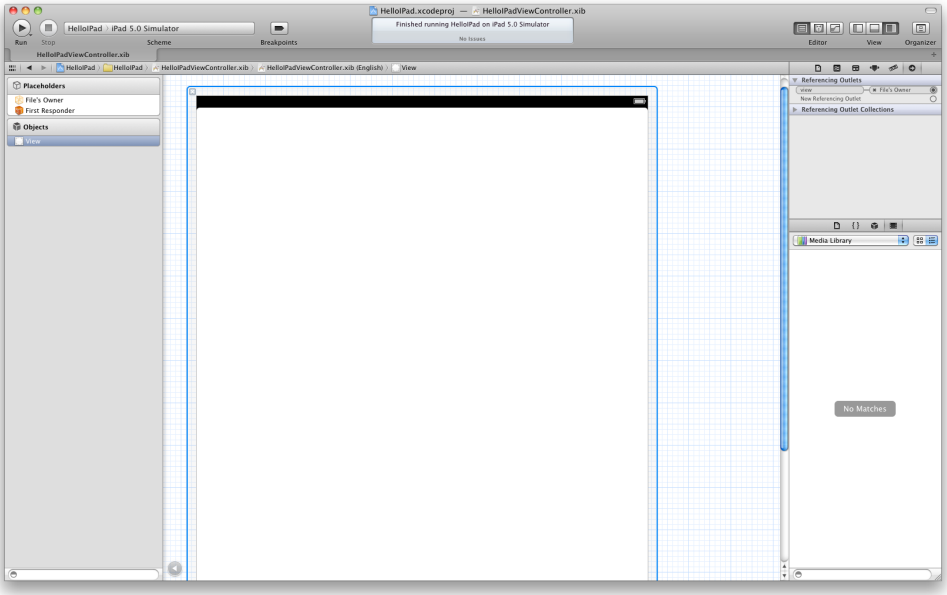
Milestone 3: Add library interface elements to your application. *(Not graded.)*

The next step is to add some interface elements to your application. Go back to the Xcode project window. Notice in the left side of the window the structure of your project, grouped under *Groups & Files*.



Note that in the **HelloIPad** section Xcode has created `.h` and `.m` files for two classes: `HelloIPadAppDelegate` and `HelloIPadViewController`. The second class is the source code for the **Controller**, while the **View** can be accessed by opening the file `HelloIPadViewController.xib`, located under *Resources*.

a. Open the view by clicking on the file `HelloIPadViewController.xib`. This will open **Interface Builder**, a graphical tool for designing interfaces. In this case, you can dedicate the full screen to the designer's view by closing the navigation area.



b. Notice the three panels: the left panel containing the view objects hierarchy, the middle panel containing the view composition window, the right panel containing the library of objects and the attributes inspector. Search for "label" in the library of objects, and drag the label object into the view window. This will

add a label containing the text “Label” to your view. Double click on the label and change the text to “Hello iPad”.

c. Save the view. You can go back to Xcode and run the project to see the label in your application.

Milestone 4: Add resource files to your project. *(Not graded.)*

Next, we add the CS3217 logo to the view of the application. The logo is provided as a PNG file, `cs3217_logo.png`, which has to be added to the iPad project.

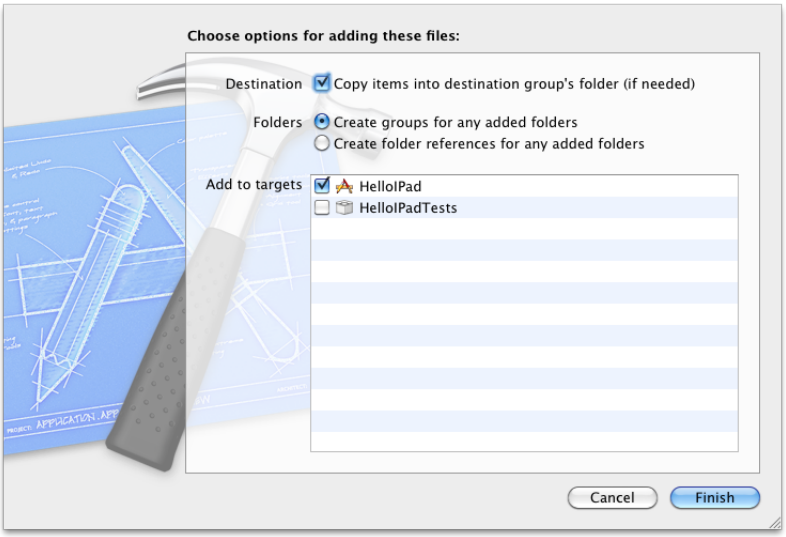
a. Navigate your Mac to locate the logo file using the **Finder** application.

b. Create a new group named *Resources* under *HelloIPad* in the navigation window.

c. Drag the logo file in Xcode over the *Resources* group.

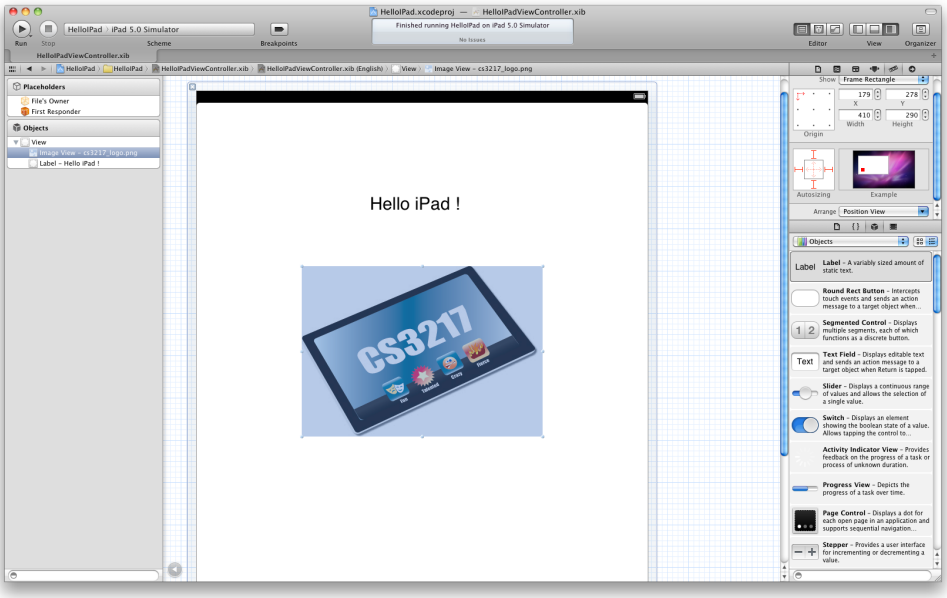
d. From the following dialog, select *Copy items into destination group's folder* to create a copy of the logo in the project folder.

Please be aware that Xcode attempts to put all files in one directory. You should learn to organize your files in Xcode project,for example, create folders and link groups with folders.



e. Go back and edit the view in **Interface Builder**.

f. In the right panel, go to the *Media Library* tab, where you can find the logo after it was added into the Xcode project.



g. Drag the logo into the view and position it under the label. Save the work.

> **Hint:** You can align to center vertically and horizontally the interface elements using the *Arrange* tools in the *Size Inspector* window (see figure above).

> **Milestone 5:** Build and run your first iPad application. *(Not graded.)*

Hit the *Run* button. The "Hello iPad" application will run in the simulator. Congratulations, you complete the first part of this assignment.

You are strongly encouraged to explore the Xcode 4 and some learning resources are available here (You may need to log in with Apple ID to watch the videos.)

- Apple WWDC 2010 Talks - There are multiple tutorials about Xcode 4 `http://developer.apple.com/videos/wwdc/2010/`.
- Apple WWDC 2011 Talks - Effective Debugging with Xcode 4 `http://developer.apple.com/videos/wwdc/2011/`
- Apple Xcode user guide `http://developer.apple.com/library/mac/#documentation/ToolsLanguages/Conceptual/Xcode4UserGuide/Introduction/Introduction.html` You can find many tips on how to configure Xcode well for your need.

- Since Xcode has *git* as the default software control management tool, `http://sixrevisions.com/resources/git-tutorials-beginners/` includes a list of tutorials for you to pick up this awesome tool.

Optional:	If you have the iPad issued and the developer certificates already installed, you can change the build target to *iOS Device* and test the application on the iPad.

Part 2: Introduction to Objective-C (100 points)

The assignment for part two is to develop an Objective-C program that, given two rectangles, determines if the rectangles overlap each-other. You are provided with stubs that can compile, but does nothing. You are required to fill in the required C and Objective-C code to implement the required functions. The input of the program are the coordinates for two rectangles, and the output is a text indicating if the rectangles overlap.

> **Hint:** You can find the **Terminal** application in `/Applications/Utilities`. For this assignment, you can edit the provided files using the Xcode IDE, or another text editor of your choice. For command-line junkies, you can find **vi** already installed, and **emacs** or **mcedit** may be installed using either *Fink* or *Macports*.

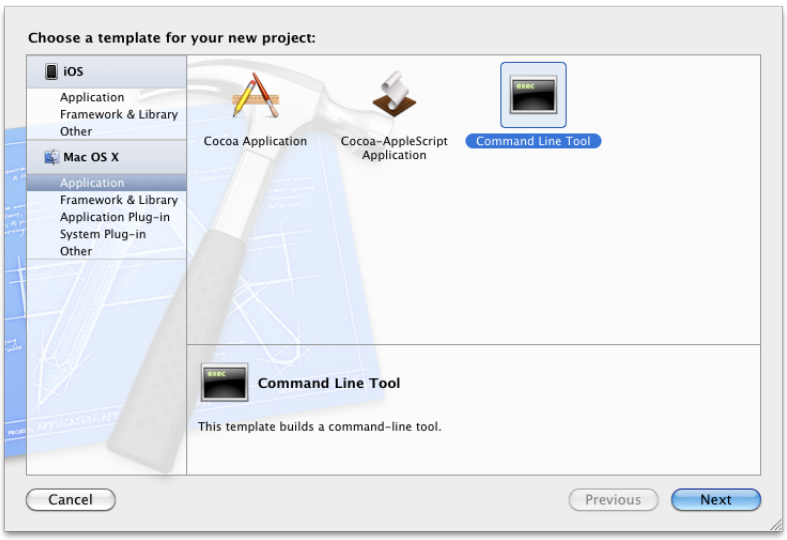
Problem 1: [Warm Up] Determine if two rectangles overlap/touch. *(10 points)*

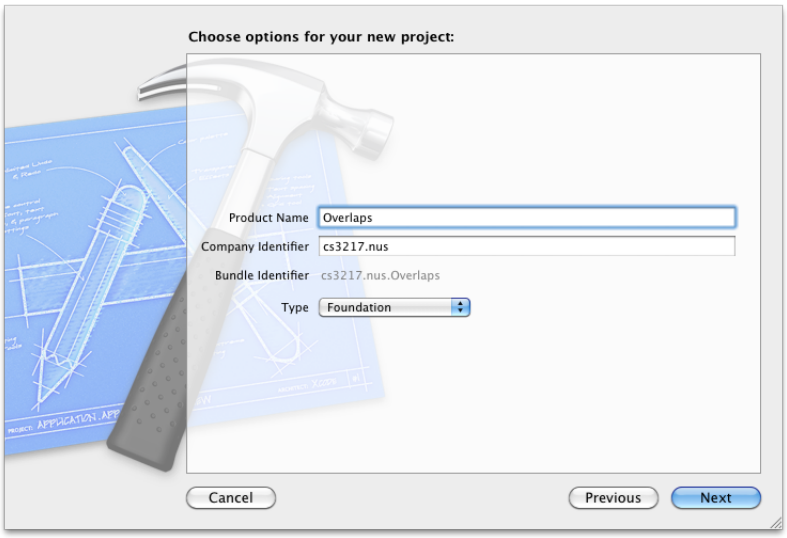
For this problem, you are **strictly to use only C**. The required stubs are found in the file `Overlaps.m`. First, you have to define a `struct` that specifies a rectangle. A rectangle is specified using four **integer** values that represent its origin (a point in the 2D cartesian plane that is the **top-left corner** of the rectangle), and its size (the width and height). Next, you have to write the `main` function that reads the values for the two rectangles from `stdin`, checks if the two rectangles overlap, and prints an appropriate message. Finally, you have to implement the `overlap` function that checks if two rectangle structures overlap/touch and return an appropriate value.

Sample execution:

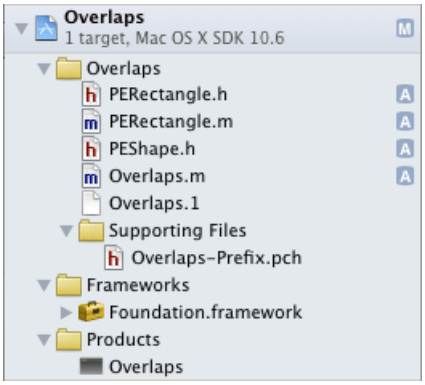
```
$ ./Overlaps
Input <x y> coordinates for the origin of the first rectangle: 0 100
Input width and height of the first rectangle: 100 200
Input <x y> coordinates for the origin of the second rectangle: 150 100
Input width and height of the second rectangle: 100 100
The two rectangles are not overlapping!
```

You may use Xcode as IDE to solve the problem. In this case, you should create a new **Command Line Tool** project in Xcode and choose the `Foundation` type.





Then you can drag in the provided files `Overlaps.m`,`PERectangle.h`,`PERectangle.m` and `PEShape.h` into this project. There is a main function in `Overlaps.m` so you should either remove the `main.m` generated by Xcode or unselect it from target membership. If you remove `main.m`, the file structure in Xcode should be similar to the screenshot below. Just for your information, instead of **gcc**,Xcode uses Apple LLVM Compiler as default compiler to compile C or Objective-C code.



Hint: To compile the program, **gcc** is also available as part of the development tools. Since for the second part of this assignment you will write Objective-C code, the program is linked to the *Foundation* framework by including the `Foundation.h` header file, which contains base class definitions. To link against frameworks, you can compile your program using the `-framework` option: **gcc -framework Foundation Overlaps.m -o Overlaps**

Problem 2: Determine if two rotated rectangle objects overlap/touch. *(90 points)*

The C implementation done in the previous part does not take into account rotation, i.e. rectangles may be rotated **around their center of mass**. Thus, a better specification for a rectangle needs to allow operations such as translation, rotation and scaling to be applied. In this problem, you will define and use rectangle **objects** using Objective-C. Since rectangles are a kind of shape, you are provided with stubs for a general shape interface and the required rectangle class. The stubs are found in the files `PEShape.h` and `PERectangle.m`.

`PEShape.h` contains an Objective-C **protocol**, which is similar to a Java interface. The protocol declares the methods that can be implemented by any class. The `PEShape` protocol declares three shape properties: origin point, center of mass point, and rotation angle, together with several methods representing general operations applicable to shapes, such as initialization, rotation, translation, and a method that checks if shapes overlap/touch. Explore the documentation and make use of already-existing structures from Objective-C (you will find that the structure you defined in *Problem 1* is already defined as `CGRect`)

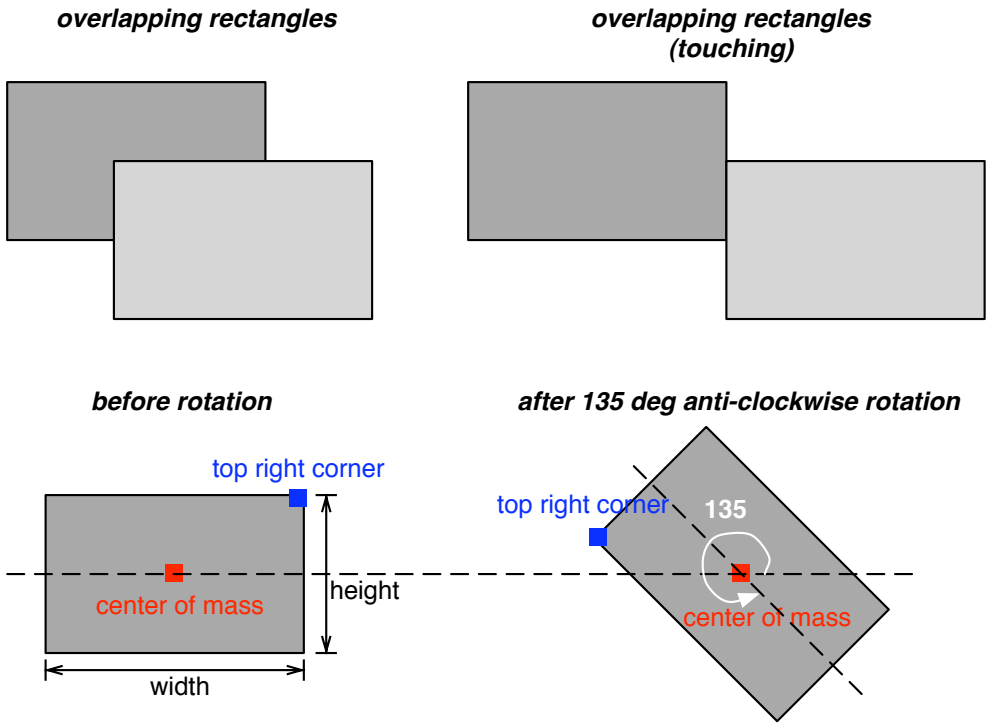
Your tasks are as follows:

a. Implement getters and setters for the rectangle properties (you can use the *@synthesize* directive where you think appropriate). Read about the properties attributes and choose appropriate attributes for the rectangle properties. Usually, properties are used in conjunction with class variables, but can be used also without a corresponding variable. **Do NOT store the *center* shape property in a variable. Compute the center of mass of the shape in the getter method using the origin point and the rotation angle.** *(10 points)*

b. Write the implementation for the class default constructor, `initWithOrigin:(CGPoint)o width:(CGFloat)w height:(CFloat)h rotation:(CGFloat)r`, and for the convenience constructor `initWithRect:(CGRect)rect`. *(5 points)*

c. Write the implementation for the `translate` and `rotate` methods. Note that rotation is done around the center of mass of the shape. Note: positive values indicate anti-clockwise rotation. *(10 points)*

d. Methods used inside the class implementation are not required to be declared in the header file. An example is the method `-(CGPoint)cornerFrom:(int)corner`, which is not declared in `PERectangle.h`. Implement this method, which returns the coordinates (after rotation) of the corner of the rectangle specified by `corner`. Note that we refer to a corner relative to the center of mass and without considering rotation. **To ensure that there is no ambiguity, the figure below shows the center of mass and top-right corner before and after a rotation.**



Declare the rectangle property `corners`, which is already defined in `PERectangle.m`. *(10 points)*

e. Look at the implementation for the protocol method `overlapsWithShape:(id <PEShape>)shape` and add the missing code that checks if two rectangles overlap. Note that two shapes are defined to be *overlapped* as long as they share at least one common point (pixel). *(20 points)*

f. Augment the main function in `Overlaps.m` (by appending to the implementation from *Problem 1*), so that in addition to asking for the coordinates and sizes of two rectangles, it will ask the user for rotation angles (in degrees, positive values for anti-clockwise and negative values for clockwise) for the two rectangles. Next, rotate the rectangles and check if they are overlapping after rotation. *(5 points)*

Hint:	For who wants to use **gcc** for compilation in **Terminal**: **gcc -framework Foundation Overlaps.m PERectangle.m -o overlaps**

Sample Execution:

```
$ ./Overlaps
Input <x y> coordinates for the origin of the first rectangle: 0 100
Input width and height of the first rectangle: 100 200
Input <x y> coordinates for the origin of the second rectangle: 150 100
Input width and height of the second rectangle: 100 100
```

```
The two rectangles are not overlapping!
Input rotation angle for the first rectangle: 90
Input rotation angle for the second rectangle: 0
The two rectangle objects are overlapping!
```

g. **Testing**. Testing is an important concept for software engineering. As an exercise, write simple test cases in the function `test` to test all the functions that you were asked to implement above. You are to do this as systematically and comprehensively as possible. *(20 points)*

h. **Alternative Representations**. In addition to representing the rectangle as a origin coordinate, height and width, can you come up with another plausible representation for the rectangle? Discuss the pros and cons of each representation. Submit the answer to this question in the form of a comment appended at the end of `Overlaps.m`. *(10 points)*

i. **Reflection (Bonus Question)**. Please answer the following questions:

   (a) How many hours did you spend on each problem of this problem set?

   (b) In retrospect, what could you have done better to reduce the time you spent solving this problem set?

   (c) What could the CS3217 teaching staff have done better to improve your learning experience in this problem set?

   Submit the answer to this question in the form of a comment appended at the end of `Overlaps.m`. *(3 bonus points)*

Grading Scheme

In this module, you are training to become a good software engineer. The first and basic requirement is that your code must satisfy the requirements and be correct. Above and beyond correctness, you are required to write well-documented code. In real software projects, just ensuring that your code can do the job is **not** sufficient. Remember that if you are doing anything useful at all, code has to be maintained and the probability that some poor soul will have to come along to read and modify your code is very high. Your goal is to minimize the grief of this poor fella and make him love reading your code.

You also want to minimize the grief that is inflicted on your TAs. In particular, we will be looking out for the following:

- Your submission should adhere to the submission format.
- Your files should compile without errors or warnings.
- Your program should run without crashing over the range of all valid inputs.
- Your code should be well-documented, correctly indented and neat. You should not use magic numbers.

Points will be taken off if you fail to comply with these requirements.

Mode of Submission

In this assignment, you will be using GitHub (`https://github.com`) for submitting your code and receiving feedback. The required files `Overlaps.m`, `PEShape.h`, `PERectangle.h` and `PERectangle.m` should be in a single directory called `ps01`. This directory should be inside the root directory of the private repository assigned to you. You must upload all your work to the master branch of this remote repository. You will be graded on the latest commit on the master branch before the deadline.

The following is a typical workflow to set up the integration with XCode.

a. Use the `git clone` command to get a local copy of your remote repository located at `https://github.com/NUSSOC/<yourreponame>`. This is a one-time operation and should not be repeated for every PS. For successive problem sets, you would just be creating appropriate directories inside this repository as mentioned above.

b. Create a new subdirectory called `ps01` in your local repository. Use the `git add` command to track it.

c. Create a new XCode project for this assignment and save it to `ps01`. XCode will automatically inherit the git repo settings, following which you can push/pull/merge etc. to the remote repo.

d. Once you are satisfied with your work, push your changes to the master branch using XCode.

When you build your project, a directory called `build` containing the binaries appears in your project folder. It is needless and time consuming to have git upload these files to the remote repo, so you can ignore this directory when syncing with the remote. The way to do this is add a file called `.gitignore` containing the line `build/` to the root directory of your local repository. This line is called a pattern or rule, and tells git to ignore any directory named `build` anywhere below the current level in the directory tree. You can add more rules to this file to define which files or directories you want git to ignore.

Important Note:	All the required files should be directly inside the `ps01` directory. Even though your TAs will be reading every single line of your code, we will be compiling and doing high-level tests of your code using an automatic script. You must also include all the files we specified or the scripts might fail. If you fail to comply with our instructions, points will be taken off.

Clarifications and questions related to this assignment may be directed to the IVLE Forum under the heading 'Problem Set 1: Hello iPad!'.

Good luck and have fun!