

Assignment 2A - WhatATool (Part II)

Due Date

This assignment is due by **11:59 PM, October 8.**

Assignment

Now that we have some Objective-C experience under our belt, we'll dive into the world of custom classes and more advanced Objective-C language topics. You will define and use a new custom class, and learn about Objective-C categories.

We will use the same WhatATool project from last week as our starting point. A few more sections will be added to the existing structure.

The basic layout of your program should look something like this:

```
#import <Foundation/Foundation.h>

// sample function for one section, use a similar function per section
void PrintPathInfo() {
    // Code from path info section here
}

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    PrintPathInfo();           // Section 1
    PrintProcessInfo();       // Section 2
    PrintBookmarkInfo();      // Section 3
    PrintIntrospectionInfo(); // Section 4
    PrintPolygonInfo();       // Section 6 (No function for section 5)

    [pool release];
    return 0;
}
```

Testing

In most assignments testing of the resulting application is the primary objective. In this case, testing/grading will be done both on the output of the tool, but also on the code of each section.

We will be looking at the following:

1. Your project should build without errors or warnings.
2. Your project should run without crashing.
3. Each section of the assignment describes a number of log messages that should be printed. These should print.
4. Each section of the assignment describes certain classes and methodology to be used to generate those log messages – the code generating those messages should follow the described methodology, and not be simply hard-coded log messages.

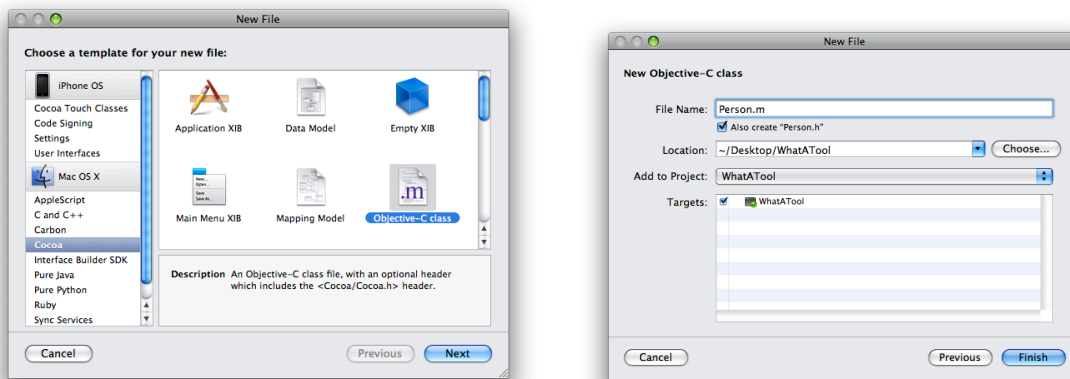
To help make the output of your program more readable, it would be helpful to put some kind of header or separator between each of the sections.

Assignment Walkthrough

Section 5: Creating a new class

Create a PolygonShape class. To create the files for the new class in Xcode:

1. Choose File > New File...
2. In the Cocoa section under Mac OS X, select the 'Objective-C class' template
3. Name the file PolygonShape.m – Be certain the checkbox to create a header file is also checked



Note that your new class inherits from NSObject by default, which happens to be what we want.

Now that we've got a class, we need to give it some attributes. Objective-C 2.0 introduced a new mechanism for specifying and accessing attributes of an object. Classes can define "properties" which can be accessed by users of the class. While properties usually allow developers to avoid having to write boilerplate code for setting and getting attributes in their classes, they also allow classes to express how an attribute can be used or what memory management policies should be applied to a particular attribute. For example, a property can be defined to be read-only or that an when an object property is set how the ownership of that object should be handled.

In this section you will add some properties to your PolygonShape class.

1. Add the following properties to your PolygonShape class
 - numberOfSides – an int value
 - minimumNumberOfSides – an int value
 - maximumNumberOfSides – an int value
 - angleInDegrees – a float value, readonly
 - angleInRadians – a float value, readonly
 - name – an NSString object, readonly
2. The numberOfSides, minimumNumberOfSides and maximumNumberOfSides properties should all be backed by instance variables of the appropriate type. These properties should all be synthesized. When a property is "synthesized" the compiler will generate accessor methods for the properties according to the attributes you specify in the @property declaration. For example, if you specified a property as "readonly" then the compiler will

only generate a getter method but not a setter method.

3. Implement setter methods for each of the number of sides properties and enforce the following constraints:

- `numberOfSides` – between the minimum and maximum number of sides
- `minimumNumberOfSides` – greater than 2
- `maximumNumberOfSides` – less than or equal to 12

Attempts to set one of these properties outside of the constraints should fail and log an error message. For example, if you have a polygon that is configured with a `maximumNumberOfSides` set to 5 and you attempt to set the `numberOfSides` property to 9 you should log a message saying something like:

```
Invalid number of sides: 9 is greater than the maximum of 5 allowed
```

4. Implement a custom initializer method that takes the number of sides for the polygon:

```
- (id)initWithNumberOfSides:(int)sides minimumNumberOfSides:(int)min  
maximumNumberOfSides:(int)max;
```

Your initializer should set the minimum and maximum number of sides first (to establish the constraints) and then set the number of sides to the value passed in.

5. Implement a custom `init` method (overriding the version implemented in `NSObject`) which calls your custom initializer with default values. For example, your generic `-[PolygonShape init]` method might create a 5 sided polygon with min of 3 sides and max of 10.
6. The `angleInDegrees` and `angleInRadians` properties should **not** be stored in an instance variable since they are all properties derived by the `numberOfSides`. These properties do not need to be synthesized and you should implement methods for each of them which return the appropriate values. We're being boring and using regular polygons so the angles are all the same.
7. Similarly, the `name` property should also not be synthesized (nor stored in an instance variable) and you should implement a method for it. The name of the polygon should be a descriptive name for the number of sides. For example, if a polygon has 3 sides it is a "Triangle". A 4-sided polygon is a "Square".
8. Give your `PolygonShape` class a `-description` method. Example output from this method:

```
Hello I am a 4-sided polygon (aka a Square) with angles of 90 degrees  
(1.570796 radians).
```
9. In order to verify your memory management techniques, implement a `dealloc` method and include an `NSLog` statement indicating that `dealloc` is being called.

Section Hints:

- You can find a list of names for polygons on the web. Wikipedia has a good one.
- The formula for computing the internal angle of a regular polygon in degrees is $(180 * (\text{numberOfSides} - 2) / \text{numberOfSides})$.
- Remember your trigonometry: 360° is equal to 2π .

- You can use the preprocessor value `M_PI` for π .

Section 6: Using the PolygonShape class

Write a C function (e.g. `PrintPolygonInfo`) called by `main()` that uses your newly created `PolygonShape` class.

You will have to add an import statement to the `WhatATool.m` file:

```
#import "PolygonShape.h"
```

IMPORTANT: In this section, you are expected to use `+alloc` and `-init` methods to create the polygons and the array that they are put into. You must practice correct memory management techniques of releasing the polygon objects when you are done with them.

1. Create a mutable array (using `alloc/init`).
2. Create 3 (or more) `PolygonShape` instances with the following values:

| Min number of sides | Max number of sides | Number of sides |
|---------------------|---------------------|-----------------|
| 3 | 7 | 4 |
| 5 | 9 | 6 |
| 9 | 12 | 12 |

When allocating your polygons, use a mixture of vanilla `alloc/init` then set the properties along with using your custom initializer method that takes all of the properties in the initializer method.

3. As you create each polygon, add them to the array of polygons and emit a log with the polygon's description. There should be at least 3 descriptions logged.
4. Test the constraints on your polygons. Iterate over the array of polygons and attempt to set their `numberOfSides` properties to 10. This should generate two logs indicating that the number of sides doesn't fall within the constraints (for the first two polygons in the table above).
5. Verify that your polygon objects are being deallocated correctly. If you have followed the rules correctly with regard to memory management you should see 3 logs from the `dealloc` method of your polygons. If you do not see these logs, review your `alloc/init` and `release` calls to make sure they are correctly balanced for your polygon objects as well as the array you are putting them into. Remember, `alloc/init` and `release` work like `malloc` and `free` in C.