Stephen G. Kochan

Second Edition

Programming in Objective-C 2.0

A complete introduction to the Objective-C language for Mac OS X and iPhone development

Developer's Library

Programming in Objective-C 2.0

Using the `@class` directive is more efficient because the compiler doesn't need to process the entire `XYPoint.h` file (even though it is quite small); it just needs to know that `XYPoint` is the name of a class. If you need to reference one of the `XYPoint` classes methods, the `@class` directive does not suffice because the compiler would need more information; it would need to know how many arguments the method takes, what their types are, and what the method's return type is.

Let's fill in the blanks for your new `XYPoint` class and `Rectangle` methods so you can test everything in a program. First, Program 8.4 shows the implementation file for your `XYPoint` class.

First, Program 8.4 shows the new methods for the `Rectangle` class.

Program 8.4 `Rectangle.m` Added Methods

```
#import "XYPoint.h"

-(void) setOrigin: (XYPoint *) pt
{
   origin = pt;
}

-(XYPoint *) origin
{
   return origin;
}
@end
```

Following are the complete XYPoint and Rectangle class definitions, followed by a test program to try them out.

Program 8.4 `XYPoint.h` Interface File

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

@interface XYPoint: NSObject
{
   int  x;
   int  y;
}
@property int x, y;

-(void) setX: (int) xVal andY: (int) yVal;
@end
```

Program 8.4 `XYPoint.m` Implementation File

```
#import "XYPoint.h"


@implementation XYPoint

@synthesize x, y;
-(void) setX: (int) xVal andY: (int) yVal
{
    x = xVal;
    y = yVal;
}
@end
```

Program 8.4 `Rectangle.h` Interface File

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


@class XYPoint;
@interface Rectangle: NSObject
{
   int   width;
   int   height;
   XYPoint *origin;
}

@property int width, height;

-(XYPoint *) origin;
-(void)  setOrigin: (XYPoint *) pt;
-(void)  setWidth: (int) w andHeight: (int) h;
-(int)   area;
-(int)   perimeter;
@end
```

Program 8.4 `Rectangle.m` Implementation File

```
#import "Rectangle.h"


@implementation Rectangle

@synthesize width, height;

-(void) setWidth: (int) w andHeight: (int) h
```

```
{
   width = w;
   height = h;
}

-(void) setOrigin: (XYPoint *) pt
{
    origin = pt;
}

-(int) area
{
    return width * height;
}

-(int) perimeter
{
    return (width + height) * 2;
}

-(XYPoint *) origin
{
    return origin;
}
@end
```

Program 8.4 Test Program

```
#import "Rectangle.h"
#import "XYPoint.h"

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

   Rectangle *myRect = [[Rectangle alloc] init];
   XYPoint   *myPoint = [[XYPoint alloc] init];

   [myPoint setX: 100 andY: 200];

   [myRect setWidth: 5 andHeight: 8];
   myRect.origin = myPoint;

   NSLog (@"Rectangle w = %i, h = %i",
      myRect.width, myRect.height);
```

```
   NSLog (@"Origin at (%i, %i)",
      myRect.origin.x, myRect.origin.y);

   NSLog (@"Area = %i, Perimeter = %i",
      [myRect area], [myRect perimeter]);
   [myRect release];
   [myPoint release];

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

Program 8.4 Output

```
Rectangle w = 5, h = 8
Origin at (100, 200)
Area = 40, Perimeter = 26
```

Inside the `main` routine, you allocated and initialized a rectangle identified as `myRect` and a point called `myPoint`. Using the `setX:andY:` method, you set `myPoint` to `(100, 200)`. After setting the width and the height of the rectangle to `5` and `8`, respectively, you invoked the `setOrigin` method to set the rectangle's origin to the point indicated by `myPoint`. The three `NSLog` calls then retrieve and print the values. The expression

```
myRect.origin.x
```

takes the `XYPoint` object returned by the accessor method `origin` method and applies the dot operator to get the x-coordinate of the rectangle's origin. In a similar manner, the following expression retrieves the y-coordinate of the rectangle's origin:

```
myRect.origin.y
```

Classes Owning Their Objects

Can you explain the output from Program 8.5?

Program 8.5

```
#import "Rectangle.h"
#import "XYPoint.h"

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

   Rectangle *myRect = [[Rectangle alloc] init];
   XYPoint   *myPoint = [[XYPoint alloc] init];
```

```
   [myPoint setX: 100 andY: 200];

   [myRect setWidth: 5 andHeight: 8];
   myRect.origin = myPoint;

   NSLog (@"Origin at (%i, %i)",
      myRect.origin.x, myRect.origin.y);
   [myPoint setX: 50 andY: 50];
   NSLog (@"Origin at (%i, %i)",
      myRect.origin.x, myRect.origin.y);
   [myRect release];
   [myPoint release];

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

Program 8.5 Output

```
Origin at (100, 200)
Origin at (50, 50)
```

You changed the `XYPoint myPoint` from `(100, 200)` in the program to `(50, 50)`, and apparently it also changed the rectangle's origin! But why did that happen? You didn't explicitly reset the rectangle's origin, so why did the rectangle's origin change? If you go back to the definition of your `setOrigin:` method, perhaps you'll see why:

```
-(void) setOrigin: (XYPoint *) pt
{
   origin = pt;
}
```

When the `setOrigin:` method is invoked with the expression

```
myRect.origin = myPoint;
```

the value of `myPoint` is passed as the argument to the method. This value points to where this `XYPoint` object is stored in memory, as depicted in Figure 8.5.

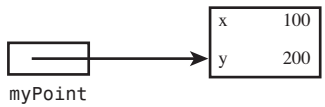

Figure 8.5 The **`XYPoint myPoint`** in memory

That value stored inside `myPoint`, which is a pointer into memory, is copied into the local variable `pt` as defined inside the method. Now both `pt` and `myPoint` reference the same data stored in memory. Figure 8.6 illustrates this.

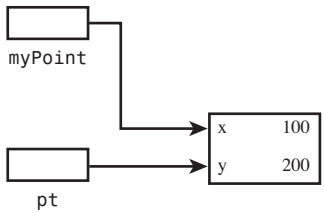

Figure 8.6 Passing the rectangle's origin to the method

When the origin variable is set to `pt` inside the method, the pointer stored inside `pt` is copied into the instance variable `origin`, as depicted in Figure 8.7.

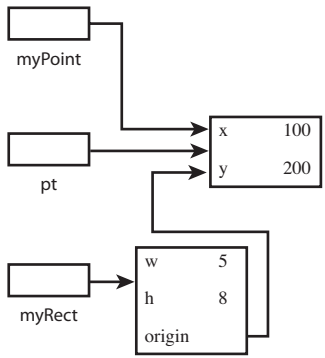

Figure 8.7 Setting the rectangle's origin

Because `myPoint` and the origin variable stored in `myRect` reference the same area in memory (as does the local variable `pt`), when you subsequently change the value of `myPoint` to `(50, 50)`, the rectangle's origin is changed as well.

You can avoid this problem by modifying the `setOrigin:` method so that it allocates its own point and sets the `origin` to that point. This is shown here:

```
-(void) setOrigin: (XYPoint *) pt
{
   origin = [[XYPoint alloc] init];

   [origin setX: pt.x andY: pt.y];
}
```

The method first allocates and initializes a new `XYPoint`. The message expression