Programming in Objective-C Day 2

Methods / Objects and Classes a deeper look

Programming in Objective-C

- Here's an example of where a method takes an *argument* that specifies a particular value that may differ from one method call to the next:
- [yourCar setSpeed: 55]; *set the speed to 55 mph*
- Your sister, Sue, can use the same methods for her own instance of a car:
- [suesCar drive]; [suesCar wash];
- [suesCar getGas];

Programming in Objective-C

- Applying the same methods to different objects is one of the key concepts of object-oriented programming, and you'll learn more about it later.

- You probably won't need to work with cars in your programs. Your objects will likely be computer-oriented things, such as windows, rectangles, pieces of text, or maybe even a calculator or a playlist of songs.

- And just like the methods used for your cars, your methods might look similar, as in the following:

+
Programming in Objective-C

`[myWindow erase];`	Clear the window
`theArea = [myRect area];`	Calculate the area of the rectangle
`[userText spellCheck];`	Spell-check some text
`[deskCalculator clearEntry];`	Clear the last entry
`[favoritePlaylist showSongs];`	Show the songs in a playlist of favorites
`[phoneNumber dial];`	Dial a phone number
`[myTable reloadData];`	Show the updated table's data
`n = [aTouch tapCount];`	Store the number of times the display was tapped

+
Programming in Objective-C

- Now it's time to define an actual class in Objective-C and learn how to work with instances of the class.
- Once again, you'll learn procedure first.
- As a result, the actual program examples might not seem very practical.
- We get into more practical stuff later.
- Suppose that you need to write a program to work with fractions.
- Maybe you need to deal with adding, subtracting, multiplying, and so on.

+ Programming in Objective-C

```
// Simple program to work with fractions

#import <Foundation/Foundation.h>

int main (int argc, char * argv[])
{
   @autoreleasepool {
      int  numerator = 1;
      int  denominator = 3;
      NSLog (@"The fraction is %i/%i", numerator, denominator);
   }
   return 0;
}
```

Programming in Objective-C

- n Program 3.1, the fraction is represented in terms of its numerator and denominator.
- After the @autoreleasepool directive, the two lines in main both declare the variables numerator and denominator as integers and assign them initial values of 1 and 3, respectively.
- This is equivalent to the following lines:
- int numerator, denominator;
- numerator = 1;
- denominator = 3;

Programming in Objective-C

- We represented the fraction 1/3 by storing 1 in the variable numerator and 3 in the variable denominator.
- If you needed to store a lot of fractions in your program, this could be cumber- some.
- Each time you wanted to refer to the fraction, you'd have to refer to the corresponding numerator and denominator.
- And performing operations on these fractions would be just as awkward.

Programming in Objective-C

- It would be better if you could define a fraction as a single entity and collectively refer to its numerator and denominator with a single name, such as myFraction.
- You can do that in Objective-C, and it starts by defining a new class.
- Program 3.2 duplicates the functionality of Program 3.1 using a new class called Fraction.
- Here, then, is the program, followed by a detailed explanation of how it works.

+
Programming in Objective-C

```
// Program to work with fractions – class version

#import <Foundation/Foundation.h>

//---- @interface section ----

@interface Fraction: NSObject

-(void)   print;
-(void)   setNumerator: (int) n;
-(void)   setDenominator: (int) d;

@end
```

+
Programming in Objective-C

```
//---- @implementation section ----

@implementation Fraction
{
   int  numerator;
   int  denominator;
}
-(void) print
{
   NSLog (@"%i/%i", numerator, denominator);
}

-(void) setNumerator: (int) n
{
   numerator = n;
}

-(void) setDenominator: (int) d
{
   denominator = d;
}

@end
```

+
Programming in Objective-C

```
int main (int argc, char * argv[])
{
   @autoreleasepool {
      Fraction  *myFraction;

      // Create an instance of a Fraction

      myFraction = [Fraction alloc];
      myFraction = [myFraction init];

      // Set fraction to 1/3
```

+ Programming in Objective-C

```
        // Set fraction to 1/3

        [myFraction setNumerator: 1];
        [myFraction setDenominator: 3];

        // Display the fraction using the print method

        NSLog (@"The value of myFraction is:");
        [myFraction print];
    }
    return 0;
}
```

+
Programming in Objective-C

- the program is logically divided into three sections:
- @interface section
- @implementation section
- program section
- @interface section describes the class and its methods
- @implementation section describes the data

- The program section contains the program code to carry out the intended purpose of the program.

+
Programming in Objective-C

- When you define a new class, you have to tell the Objective-C compiler where the class came from.
- That is, you have to name its *parent* class.
- Next, you need to define the type of operations, or *methods,* that can be used when working with objects from this class.
- The general format of this section looks like this:

```
@interface NewClassName: ParentClassName
   propertyAndMethodDeclarations;
@end
```

+
Programming in Objective-C

- By convention, class names begin with an uppercase letter, even though it's not required.
- This enables someone reading your program to distinguish class names from other types of variables by simply looking at the first character of the name.
- Let's take a short diversion to talk a little about forming names in Objective-C.

+

Programming in Objective-C

- The rules for forming names are quite simple: They must begin with a letter or underscore (_), and they can be followed by any combination of letters (uppercase or lowercase), underscores, or the digits 0 through 9.
- sum
- pieceFlag
- i
- myLocation
- numberOfMoves
- sysFlag
- ChessBoard

Programming in Objective-C

- However, the following names are not valid for the stated reasons:

- sum$value $—Is not a valid character.
- piece flag—Embedded spaces are not permitted.
- 3Spencer—Names cannot start with a number.
- int—This is a reserved word.

+

Programming in Objective-C

- Instance variables, objects, and method names, however, typically begin with lowercase letters.
- To aid readability, capital letters are used inside names to indicate the start of a new word, as in the following examples:
- AddressBook—This could be a class name.
- currentEntry—This could be an object.
- addNewEntry—This could be a method name.
- When deciding on a name, keep one recommendation in mind: Don’t be lazy.

+

Programming in Objective-C

- Here, again, is the @interface section from Program 3.2:

```
//---- @interface section ----

@interface Fraction: NSObject

-(void) print;
-(void) setNumerator: (int) n;
-(void) setDenominator: (int) d;

@end
```

Programming in Objective-C

- You have to define methods to work with your Fractions. You need to be able to set the value of a fraction to a particular value.
- Because you won't have direct access to the internal representation of a fraction (in other words, direct access to its instance variables), you must write methods to set the numerator and denominator.
- You'll also write a method called print that will display the value of a fraction.

+
Programming in Objective-C

- Here’s what the declaration for the print method looks like in the interface file:
- -(void) print;
- The leading minus sign (-) tells the Objective-C compiler that the method is an instance method. The only other option is a plus sign (+), which indicates a class method.
- A class method is one that performs some operation on the class itself, such as creating a new instance of the class.

Programming in Objective-C

- An instance method performs some operation on a particular instance of a class, such as setting its value, retrieving its value, displaying its value, and so on.
- Referring to the car example, after you have manufactured the car, you might need to fill it with gas.
- The operation of filling it with gas is performed on a particular car, so it is analogous to an instance method.

+

Programming in Objective-C

- When you declare a new method, you have to tell the Objective-C compiler whether the method returns a value and, if it does, what type of value it returns.
- You do this by enclosing the return type in parentheses after the leading minus or plus sign.
- So this declaration specifies that the instance method called currentAge returns an integer value:
- **–(int) currentAge;**

Programming in Objective-C

- Similarly, this line declares a method that returns a double precision value.
- –(double) retrieveDoubleValue;
- A value is returned from a method using the Objective-C return statement, similar to the way in which we returned a value from main in previous program examples.
- If the method returns no value, you indicate that using the type void, as in the following:
- –(void) print;

+
Programming in Objective-C

- This declares an instance method called print that returns no value. In such a case, you do not need to execute a return statement at the end of your method.
- Alternatively, you can execute a return without any specified value, as in the following: return;
- Two other methods are declared in the @interface section from Program 3.2:
- –(void) setNumerator: (int) n;
- –(void) setDenominator: (int) d;

Programming in Objective-C

- These are both instance methods that return no value.
- Each method takes an integer argument, which is indicated by the (int) in front of the argument name.
- In the case of setNumerator, the name of the argument is n.
- This name is arbitrary and is the name the method uses to refer to the argument.
- Therefore, the declaration of setNumerator specifies that one integer argu- ment, called n, will be passed to the method and that no value will be returned.

Programming in Objective-C

- This is similar for setDenominator, except that the name of its argument is d.
- Notice the syntax of the declaration for these methods.
- Each method name ends with a colon, which tells the Objective-C compiler that the method expects to see an argument.

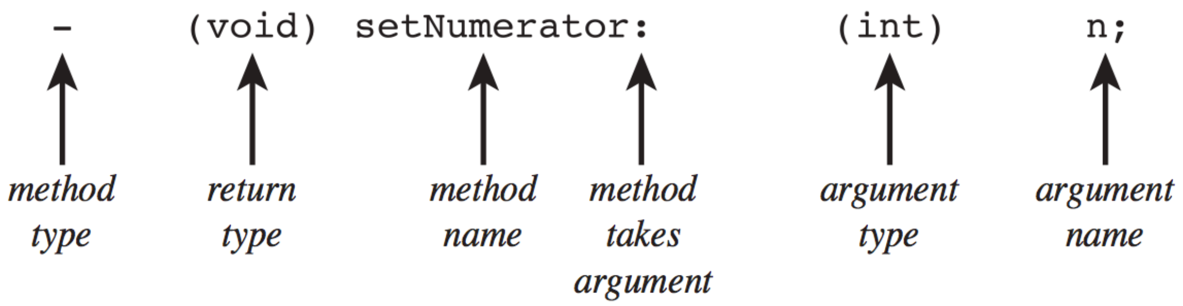


Figure 3.1 Declaring a method

Programming in Objective-C

- When a method takes an argument, you also append a colon to the method name when referring to the method.
- Therefore, setNumerator: and setDenominator: is the correct way to identify these two methods, each of which takes a single argument.
- Also, identifying the print method without a trailing colon indicates that this method does not take any arguments.
- In Chapter 7, "More on Classes," you'll see how methods that take more than one argument are identified.

+

Programming in Objective-C

- the @implementation section contains the actual code for the methods you declared in the @interface section.
- You have to specify what type of data is to be stored in the objects of this class.
- That is, you have to describe the data that members of the class will contain.
- These members are called the *instance variables.*

+

Programming in Objective-C

- Just as a point of terminology, you say that you declare the methods in the @interface section and that you *define* them (that is, give the actual code) in the @implementation section.

```
@implementation NewClassName
{
   memberDeclarations;
}
  methodDefinitions;
@end
```

+
Programming in Objective-C

- *NewClassName* is the same name that was used for the class in the @interface section.
- You can use the trailing colon followed by the parent class name, as we did in the @interface section:
- @implementation Fraction: NSObject
- However, this is optional and typically not done.
- The *memberDeclarations* section specifies what types of data are stored in a Fraction, along with the names of those data types.

Programming in Objective-C

- As you can see, this section is enclosed inside its own set of curly braces.
- For your Fraction class, these declarations say that a Fraction object has two integer members, called numerator and denominator:
- int numerator;
- int denominator;
- The members declared in this section are known as the *instance variables.*

+
Programming in Objective-C

- Each time you create a new object, a new and unique set of instance variables also is created.
- Therefore, if you have two Fractions, one called fracA and another called fracB, each will have its own set of instance variables—that is, fracA and fracB each will have its own separate numerator and denominator.
- The *methodDefinitions* part of the @implementation section contains the code for each method specified in the @interface section.

Programming in Objective-C

- The program section contains the code to solve your particular problem, which can be spread out across many files, if necessary.

```
//---- program section ----

int main (int argc, char * argv[])
{
   @autoreleasepool {
      Fraction  *myFraction;

      // Create an instance of a Fraction and initialize it

      myFraction = [Fraction alloc];
      myFraction = [myFraction init];

      // Set fraction to 1/3
```

Programming in Objective-C

```
        // Set fraction to 1/3

        [myFraction setNumerator: 1];
        [myFraction setDenominator: 3];

        // Display the fraction using the print method

        NSLog (@"The value of myFraction is:");
        [myFraction print];
    }

    return 0;
}
```

+

Programming in Objective-C

- Inside main, you define a variable called myFraction with the following line: Fraction *myFraction;
- This line says that myFraction is an object of type Fraction; that is, myFraction is used to store values from your new Fraction class.
- Now that you have an object to store a Fraction, you need to create one, just as you ask the factory to build you a new car.
- myFraction = [Fraction alloc];
- alloc is short for *allocate.* You want to allocate memory storage space for a new fraction.

+
Programming in Objective-C

- This expression sends a message to your newly created Fraction class: [Fraction alloc]
- You are asking the Fraction class to apply the alloc method, but you never defined an alloc method, so where did it come from? The method was inherited from a parent class.
- When you send the alloc message to a class, you get back a new instance of that class.
- The alloc method is guaranteed to zero out all of an object's instance variables. However, that doesn't mean that the object has been properly initialized for use. You need to initialize an object after you allocate it.

+
Programming in Objective-C

- myFraction = [myFraction init];
- Again, you are using a method here that you didn't write yourself. The init method initializes the instance of a class.
- Note that you are sending the init message to myFraction.
- That is, you want to initialize a specific Fraction object here, so you don't send it to the class; you send it to an instance of the class.
- The init method also returns a value—namely, the initialized object. You store the return value in your Fraction variable myFraction.

+
Programming in Objective-C

- The two-line sequence of allocating a new instance of class and then initializing it is done so often in Objective-C that the two messages are typically combined, as follows:
- myFraction = [[Fraction alloc] init];
- This inner message expression is evaluated first:
- [Fraction alloc]
- As a final shorthand technique, the allocation and initialization is often incorporated directly into the declaration line, as in the following:
- Fraction *myFraction = [[Fraction alloc] init];

Programming in Objective-C

- Returning to Program 3.2, you are now ready to set the value of your fraction. These program lines do just that:
- // Set fraction to 1/3
- [myFraction setNumerator: 1];
- [myFraction setDenominator: 3];
- The first message statement sends the setNumerator: message to myFraction.
- The argument that is supplied is the value 1.
- Control is then sent to the setNumerator: method you defined for your Fraction class.

+

Programming in Objective-C

- Now you're ready to display the value of your fraction, which you do with the following lines of code from Program 3.2:
- // Display the fraction using the print method
- NSLog (@"The value of myFraction is:");
- [myFraction print];
- The NSLog call simply displays the following text: The value of myFraction is:
- The following message expression invokes the print method: [myFraction print];

Programming in Objective-C

- Let’s go back for a second to the declaration of myFraction
  Fraction *myFraction;
- and the subsequent setting of its values.
- The asterisk (*) in front of myFraction in its declaration says that myFraction is actually a reference (or *pointer*) to a Fraction object.
- We can conceptualize myFraction as a box that holds a value. Initially the box contains some undefined value, as it hasn’t been assigned any value, as shown in Figure 3.2.

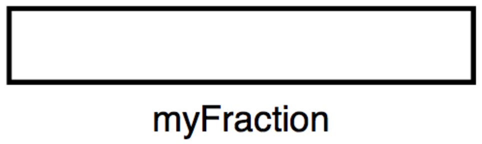


Figure 3.2 Declaring `Fraction *myFraction;`

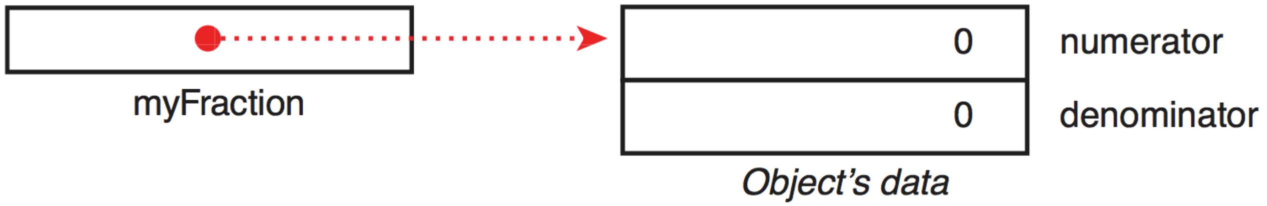
Programming in Objective-C

- When you allocate a new object (using alloc, for example) enough space is reserved in memory to store the object's data, which includes space for its instance variables, plus a little more.

- The location of where that data is stored (the reference to the data) is returned by the alloc routine, and assigned to the variable myFraction.

```
myFraction = [Fraction alloc];
```

The allocation of the object and the storage of the reference to that object in `myFraction` is depicted in Figure 3.3.



+ Programming in Objective-C

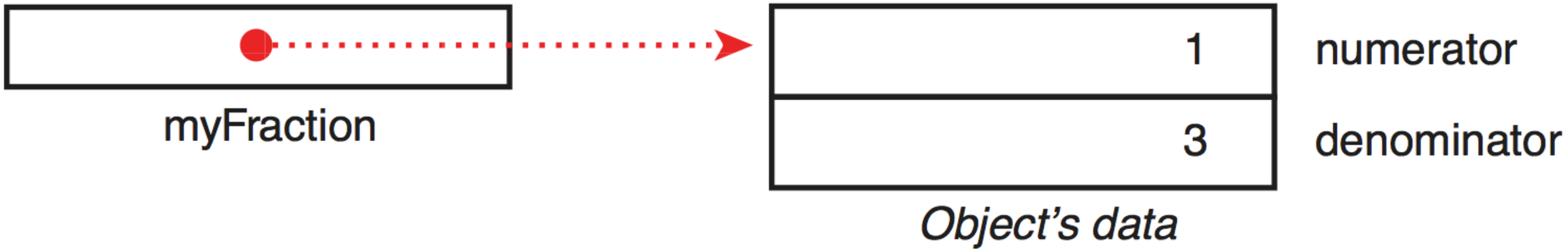


Figure 3.4 Setting the fraction's numerator and denominator

+
Programming in Objective-C

- Data encapsulation provides a nice layer of insulation between the programmer and the class developer.
- You can access your instance variables in a clean way by writing special methods to set and retrieve their values.
- We wrote setNumerator: and setDenominator: methods to set the values of the two instance variables in our Fraction class.
- To retrieve the values of those instance variables, you need to write two new methods.
- For example, create two new methods called, appropriately enough, numerator and denominator to access the corresponding instance variables of the Fraction that is the receiver of the message.

+

Programming in Objective-C

- The result is the corresponding integer value, which you return. Here are the declarations for your two new methods:
- –(int) numerator;
- –(int) denominator;
- And here are the definitions:

```
-(int) numerator
{
   return numerator;
}
```

```
-(int) denominator
{
   return denominator;
}
```

+
Exercises

- On page 49 of the current text complete the following exercises.

2. Based on the example of the car in this chapter, think of an object you use every day. Identify a class for that object and write five actions you do with that object.

4. Imagine that you own a boat and a motorcycle in addition to a car. List the actions you perform with each of these. Do you have any overlap between these actions?

7. Define a class called `XYPoint` that will hold a Cartesian coordinate (x, y), where x and y are integers. Define methods to individually set the x and y coordinates of a point and retrieve their values. Write an Objective-C program to implement your new class and test it.