EXERCISES RELATED TO iOS PROGRAMMING

Kari Laitinen
http://www.naturalprogramming.com
2017-08-30 File created.
2017-09-21 Last modification.

EXERCISES WITH PROGRAM Animals.swift

With these exercises you will learn some basics about the Swift programming language.

Animals.swift is a simple program that contains a class named `Animal`. This program shows how `Animal` objects are created and how methods are called for those objects.

Exercise 1:

When a method is written to a Swift class, it begins with the reserved word `func`. The compiler recognizes methods and functions with this reserved word (keyword).

Write a new method named `make_stomach_empty()` to the `Animal` class. This method should empty the ’stomach’ of the `Animal` object so that an empty string `""` is assigned to stomach contents. It should be able to call the new method in the following way.

```
cat_object.make_stomach_empty()
dog_object.make_stomach_empty()
```

You should call the method `make_speak()` in order to see that the stomach is really emptied.

Exercise 2:

In the terminology of the Swift programming language constructors are called *initializers*. An initializer is invoked automatically when an object is created. `init()` is the name for a Swift initializer (constructor). An initializer that calls another initializer is said to be a *convenience initializer.*

Add to the `Animal` class a new default initializer with which it will be possible to create an `Animal` object without giving any parameters, for example, in the following way

```
    var default_animal = Animal()
```

The data field `species_name` can be assigned the string "default animal". The default initializer can be made by using a convenience initializer. By studying the example program **Windows.swift** you can see how different kinds of initializers can be written.

Also in this exercise you should call method `make_speak()` to ensure that your program works as required.

Exercise 3:

Data fields or data members of classes can also be called instance variables. Objects are often called instances. In the original **Animals.swift** program there are two instance variables, which have names `species_name` and `stomach_contents`. In this exercise you have to add a new data field (instance variable) to the `Animal` class. The new data field should be named `animal_name`, and it should contain the name of the animal.

You have to modify the constructor so that an `Animal` object can be created with a statement like

```
var cat_object  =  Animal( "cat", "Arnold" )
```

In the Swift language it is possible to give external names (argument labels) to method parameters. When an external name is not used, an underscore character `_` must be written before the local name of the parameter. By studying programs **Olympics.swift** and **Windows.swift** you will find out how initializers and methods can be written.

In this exercise you must also modify the copy constructor (initializer) so that the new data field will be copied. The method `make_speak()` must be modified so that it prints the animal name in the following way.

```
Hello, I am a cat named Arnold.
I have eaten: ...
```

In the default initializer the new data field can be given the value "nameless".

Exercise 4:

Modify method `make_speak()` so that it prints

```
    Hello, I am a ... named ...
    My stomach is empty.
```

in the case when `stomach_contents` refers to an empty string. The stomach is empty if method `feed()` has not been called for the object. One way to find out if a string is an empty string is to use `String.CharacterView` property `count`. Studying the example program **StringReverse.swift** might help you in this exercise.

If the stomach is not empty, the `make_speak()` method should produce the original output.

EXERCISES WITH APPLICATION SquareBallRectangle

Among the Swift-based iOS sample applications you can find an app named SquareBallRectangle, which displays a square, a ball, or a rectangle, depending on the selection made with a `UISegmentedControl` object.

You can start working with this app by downloading the file **SquareBallRectangle.zip** and unzipping it on local computer. Unzipping can be done with double-click. Safari sometimes unzips automatically.

Please, ensure that the application works before you start doing the following exercises.

Exercise 1:

Modify the program so that it draws a triangle in place of the rectangle.

There is no ready-to-use method to draw a triangle. However, at the end of file **SquareBallRectangleView.swift** you will find, in comments, the necessary statements that specify a drawing path to make a triangle.

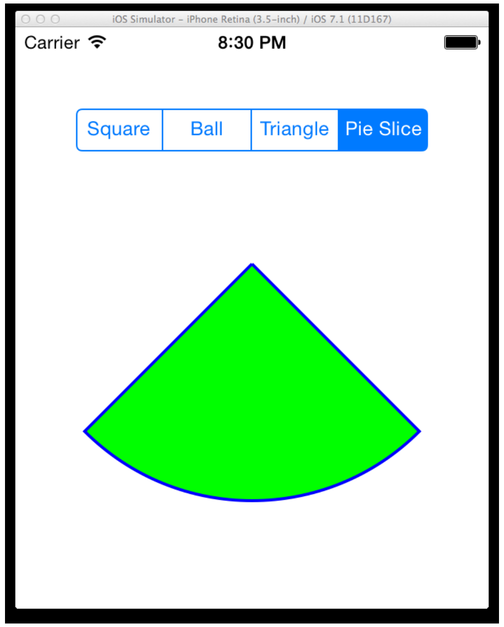
When you make the triangle with the statements provided in comments, the triangle will be drawn into the center of the View, and you do not need to modify the zero point of the coordinate system.

In this exercise, you need also to modify the user interface so that the word "Triangle" is put in place of "Rectangle".

Exercise 2:

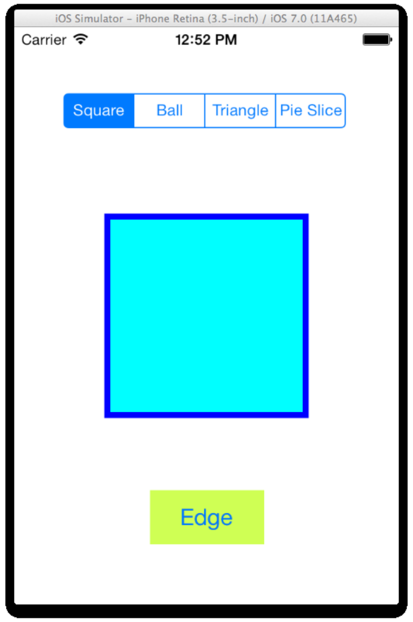
Modify the program so that it provides the possibility to select a fourth shape, which you can name as "Pie Slice". You must use Interface Builder so that the `UISegmentedControl` will have four selectable segments. In addition you must modify the drawing method so that it will draw the "Pie Slice". You could study the application named DrawingDemo to find out how a "Pie Slice" can be drawn.

After these modifications, the app should look like the following when "Pie Slice" is selected.



Exercise 3:

Improve the application so that it will have a button, with which the thickness of the edges of the shapes can be adjusted. The edge thickness (or width) can grow with two (pixels) when the button is pressed once. After the edge thickness has reached value 10, it should go back to original value 2. After this modification, the UI could have a button with text "Edge" in the following way.



In the original program the line width used in drawing is set at the beginning of the drawing method **`draw()`**. Now the line width can be stored in a data field (instance variable) that can be declared in the following way.

```
var edge_line_width : CGFloat = 2
```

The program must have a method that is called when the new Edge button is pressed. This method can be written inside the class that is derived from `UIView`. You should use the Interface Builder to make the new button and establish a connection between the button and the method. By studying the example application named ButtonDemo you can find out what kind of method is needed to react to the pressing of a button.

EXERCISES WITH APPLICATION GesturesDemo

In iOS it is possible to write methods that are called automatically when certain gestures are made on the touch screen. These gesgures include the following.

- Tap: short touch on the screen
- Swipe: some kind of finger movement, hard to simulate
- Long Press: finger remains on the screen for a longer time
- Pan: finger movement, can be easily simulated
- Pinch: scaling with two fingers, can be simulated
- Rotation: rotation with two fingers, can be simulated

GesturesDemo is an application that can be used to study when and how these gestures are generated.

Please, study and use this application first, and do then the following exercises. You can keep the original features of the program as they can be helpful when you add new features.

Exercise 1:

Modify the program so that it shows a ball that is an object of class `Ball`.

You can copy the `Ball` class from an app named MovingBallsWithFinger in the following way.

First, copy the MovingBallsWithFinger app from the web so that you unzip its **.zip** file in the usual way.

After this you can do in Xcode in your GesturesDemo project so that you select `File -> Add Files to "GesturesDemo" ...` When you are asked to select a file in a dialog, search the file **Ball.swift** among the files of MovingBallsWithFinger app, and add the file to your project.

When the `Ball` class is available in your project, you can put the following data field to the file **GesturesDemoView.swift**

```
        var ball_on_screen : Ball! = nil
```

and then you can create a `Ball` object in the `draw()` method of `GesturesDemoView` in the following way.

```
    if ball_on_screen == nil
    {
       ball_on_screen =
                  Ball( 100, 220, UIColor.red.cgColor )
    }
```

When you create the `Ball` object in the drawing method, you do not have to think what kind of constructor (initializer) you would need in the class. After the `Ball` object has been created, you can draw it in the drawing method in the following way.

```
    let context: CGContext = UIGraphicsGetCurrentContext()!
    ball_on_screen.draw( context )
```

Exercise 2:

After you have done the previous exercise and a ball is visible on the screen, your task is to modify the program so that with a Tap gesture the ball will be moved into the screen location that was tapped.

The program already has a method that is called after a Tap gesture. You have to modify it so that it will move the ball. The `Ball` class already has a method with which to move the ball. The screen position that was tapped can be found out in the following way.

```
    let tapped_point = recognizer.location( in: self )
```

By studying the MovingBallsWithFinger application you can find out how to get the screen coordinates from a `CGPoint` object.

Exercise 3:

In this exercise you have to modify the program so that always when the screen is tapped, a new ball appears in the tapped position on the screen. A new **`Ball`** object needs to be created in the method that reacts to Tap gesgures. The **`Ball`** objects should be stored into an array that can be specified as a data member in the **`GesturesDemoView`** class in the following way

```
var balls_on_screen : [ Ball ] = [ ]
```

An array like this can store **`Ball`** objects and in the beginning the array is empty.

A **`Ball`** object can be added to the end of the array with method **`append()`**. You need to modify the **`tap_detected()`** method so that, instead of moving a ball, the method creates a new **`Ball`** and adds it to the end of the array.

The **`Ball`** objects contained in the array can all be drawn by using a **`for-in`** loop like

```
for  ball in balls_on_screen
{
   ball.draw( context )
}
```

Exercise 4:

Modify the program so that the balls will have different colors. You can try to create random colors, but another way is to specify a set of colors and put them into an array in the following way

```
   var ball_colors : [ UIColor ] =

      [ UIColor.red, UIColor.green, UIColor.yellow,
        UIColor.cyan, UIColor.magenta, UIColor.blue,
        UIColor.brown, UIColor.gray, UIColor.orange ]
```

When you have this kind of an array, you can take, for example, the last color and put it back as the first color in the array. A **`UIColor`** object can be removed from the end of the array, and put to the beginning with the following statements.

```
        let color_for_new_ball = ball_colors.removeLast()

        ball_colors.insert( color_for_new_ball, at: 0 )
```

When you use this kind of ’rotating’ set of colors, an ’old’ color will be used only after all other colors have been used once. A **`UIColor`** must be converted to a **`CGColor`** before a **`Ball`** object can be created.

Exercise 5:

Improve the program further so that with a Pan gesture you can remove a ball from the screen if the Pan gesture takes place on the ball area. First you could find the Pan gesture location in the following way

```
      let pan_point = recognizer.location( in: self )
```

After this you can examine the array containing the `Ball` objects. You should examine the array starting from the end, and then delete the `Ball` that contains the pan point. When the array is processed starting from the end, you will delete the ball that is the topmost on the screen. In practice, you need to delete only a single ball as a lot of Pan gestures will be recognized when the finger moves on the screen. You can use the following kind of loop to search for a ball that should be deleted:

```
      var ball_index = balls_on_screen.count
      var ball_to_delete_is_found = false

      while  ball_index > 0 && ball_to_delete_is_found == false
      {
         ball_index -= 1

         if  balls_on_screen[ ball_index ].contains_point( pan_point )
         {
            // Here you need to delete the ball from the array.

            ball_to_delete_is_found = true
         }
      }
```

Exercise 6:

Improve the program so that with a Pinch gesture it will be possible to modify the sizes of all balls. `UIPinchGestureRecognizer` objects have a property named `scale` that can be checked.

The `scale` value is greater than one when fingers are moved farther from each other, and it is less than one when fingers move closer to each other. You can use the `scale` value to determine if the balls should be made bigger or smaller. The `Ball` class has methods `enlarge()` and `shrink()` which can be used in this task. You can use a loop and go through all balls and either enlarge or shrink them depending on what kind of Pinch gesture was recognized.

The Pinch gesture can be generated with iOS Simulator when you use the Alt key and ’mouse’ button simultaneously.

EXERCISES WITH APPLICATION ChangingViewsNoIB

Exercise 1:

Add a new `UIButton` to the view controlled by `MainViewController`. You should add the button below the current button. The new button should have the same size as the existing button. You could introduce the new button with the following name

```
      var button_demo_button : UIButton!
```

This name is good as in the next exercise your task will be to make the new button display the view that is used in the ButtonDemoNoIB app.

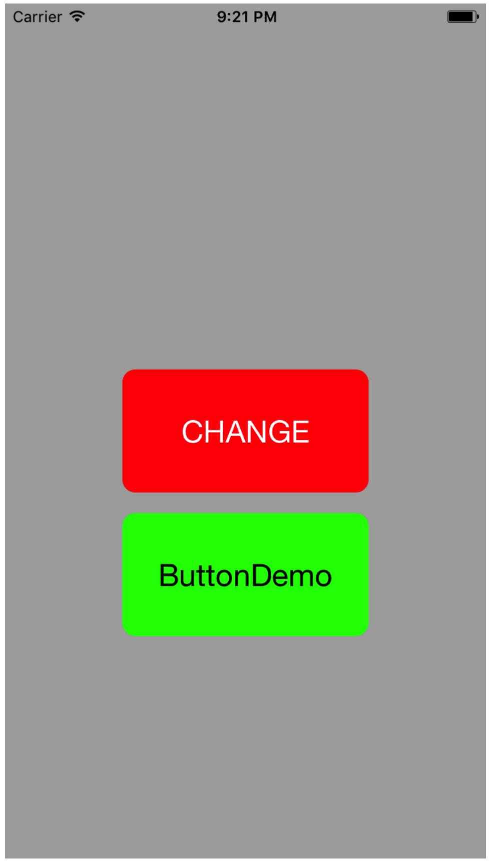
You can easily specify the features of the new button by first copying the code lines that specify the features of the existing button. You could use different colors for the new button. The new button can, in this exercise, activate the same view as the existing button.

With the following constraint definitions the new button will be placed right below the existing button:

```
button_demo_button.centerXAnchor.constraint(
       equalTo: view.layoutMarginsGuide.centerXAnchor ).isActive = true

button_demo_button.centerYAnchor.constraint(
                        equalTo: view.layoutMarginsGuide.centerYAnchor,
constant: 112 ).isActive = true
```

After having done this exercise, `MainViewController` might show the following view:



Exercise 2:

Now your task is to make the new button activate the `ButtonDemoViewController` of the ButtonDemoNoIB app. You should first download and unzip this app. Then, in Xcode, you can select **File -> Add files to "ChangingViewsNoIB" ...**, and you could just add the files **ButtonDemoViewController.swift** and **ButtonDemoView.swift** to your project.

In `MainViewController`, you could create an object in the following way

```
 let button_demo_view_controller = ButtonDemoViewController()
```

You need to write a new method that presents this view controller, and you need to specify that the pressing of the new button will result in a call to the new method.

First you do not need to modify the files **ButtonDemoViewController.swift** or **ButtonDemoView.swift**. `ButtonDemoView` should become visible when the new button is pressed. However, to go back to the `MainViewController`, you need to make modifications.

Please, study `AnotherViewController` and make a Back button to `ButtonDemoView` so that you can go back to the main view.

Exercise 3:

Add a ButtonDemo button also to the view controlled by `AnotherViewController`. You do not need to modify `ButtonDemoView` or `ButtonDemoViewController` in this exercise.

After this modification, it should be possible to go from `MainViewController` to `AnotherViewController` and from there to `ButtonDemoViewController`, and with Back buttons it should be possible to come back using the same 'route'.