Academy for iOS App Development

Dr. William C. Bulko

View Frame and Bounds

Core Graphics Fundamental Structures

- `CGPoint`: a structure that contains a point in a two-dimensional coordinate system.

  Ex. `let pt = CGPoint(x:3, y:-5)`

- `CGSize`: a structure that contains width and height values.

  Ex.
  ```
  let mySize = CGSize(width:10,
                      height:5)
  ```

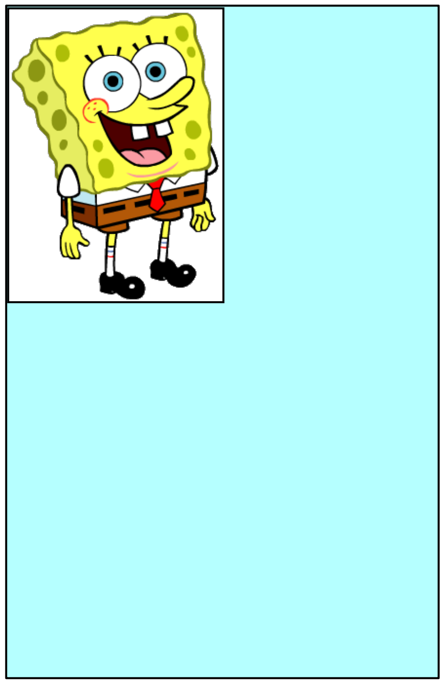
- `CGRect`: a structure that contains the location and dimensions of a rectangle.

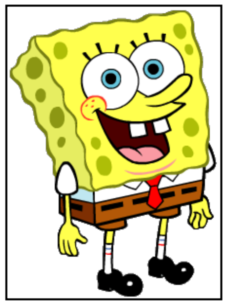
  Ex.
  ```
  let rect = CGRect(x: 3, y: 5,
                    width: 10, height: 5)
  ```
  or
  ```
  let rect = CGRect(origin:pt,
                    size:mySize)
  ```

Frame and Bounds

- *Frame* and *Bounds* are fundamental concepts for all of the elements in the UI.
- Each view has both a frame and a bounds structure. The structure is a CGRect and consists of 4 floats.
  - The **frame** of an UIView is the rectangle, expressed as a location (x,y) and size (width,height) **relative to the superview it is contained within**.
  - The **bounds** of an UIView is the rectangle, expressed as a location (x,y) and size (width,height) **relative to its own coordinate system** (0,0).

Frame and Bounds

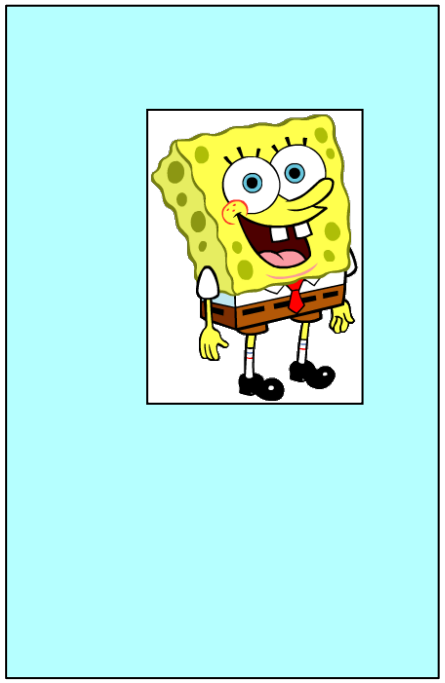


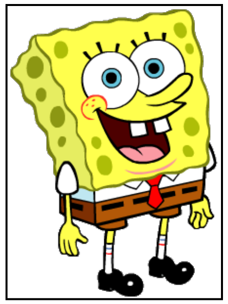


```
Frame
   origin = (0,0)
   width = 219
   height = 300
```

```
Bounds
   origin = (0,0)
   width = 219
   height = 300
```

Frame and Bounds





```
Frame
   origin = (71,50)
   width = 219
   height = 300
```

```
Bounds
   origin = (0,0)
   width = 219
   height = 300
```

Scroll Views

Scroll Views

- Scroll Views provide a way to present content larger than a single screen.
  - Critical for phones since they have limited screen real estate
  - Also helpful for iPads
- Scroll Views provide a way for moving within the content to view various parts of it.

To implement scrolling:

- Create a `UIScrollView` and define its properties
- Make the `UIScrollView` a subview of the VC's view
- Make the view you want scrollable a subview of the `UIScrollView.`

Gesture Recognizers

Events

There are 4 general types of UI events in iOS:

- **Touch** events: the most common
- **Motion** events
- **Remote-control** events: allow a responder object to receive commands from an external accessory or headset (usually to manage audio and video)
- **Press** events: represent interactions with a game controller, AppleTV remote, or other device that has physical buttons

Gestures

Gestures refer to touches and touch events.

- Central to the modern smart phone experience
- A core built-in capability in iOS

A *touch* is an instance of the user putting a finger on the screen.

The OS and the hardware work together to know when a finger touches the screen, where it is, when it moves, and when it is no longer touching the screen.

Its location at any point in time is reduced to a single appropriate point.

Gestures (cont.)

Why are they important?

- They allow us to interact more naturally and intuitively with the application
- It is a *significant* paradigm shift to how humans interact with computers: analogous to what happened when people were first provided GUIs to interact with computers

Gesture Recognizers

Gesture recognizers are high-level mechanisms provided by iOS that takes care of the nitty-gritty of touch events, and makes it very easy to respond to a set of common touch events/sequences.

- They handle touches and movements of one or more fingers that happen on a specific area of the screen
- They are objects derived from the abstract *UIGestureRecognizer* class that are related to a view, and monitor for a predefined gesture made on that view
- There are some predefined subclasses which deal with specific (common) kinds of gestures
- They all perform an action once a valid gesture is detected.

Without gesture recognizers, you would be writing pages of code to handle what takes only a few lines of code *with* gesture recognizers.

Gesture Recognizers

You can set up gesture recognizers in IB or in code.

- A view can contain more than one gesture recognizer
- They are contained in a UIView property (an array) named gestureRecognizers

However, just one gesture can occur at any given point in time.

There are two types of gesture recognizers:

- Discrete: manage a single event; for example, touch to select an object
- Continuous: manage a series of events; for example, dragging an object on the screen

Gesture Recognizers

Predefined gesture recognizer classes:

- `UI`**`Tap`**`GestureRecognizer` (discrete)
- `UI`**`Swipe`**`GestureRecognizer` (discrete)
- `UI`**`Pan`**`GestureRecognizer` (continuous)
- `UI`**`Pinch`**`GestureRecognizer` (continuous)
- `UI`**`Rotation`**`GestureRecognizer` (continuous)
- `UI`**`LongPress`**`GestureRecognizer` (continuous)
- `UI`**`ScreenEdgePan`**`GestureRecognizer` (continuous)

Setting Up a Gesture Recognizer Using IB

- In IB, identify the object that you want to manipulate on the storyboard. Drag a Gesture Recognizer object **on top of the target object.**
- In the Swift file, write a function to handle the gesture.
- In IB, ctrl-drag the Gesture Recognizer object to the View Controller. Choose the name of the function you wrote.
- Click on the target object and go to the Attribute Inspector. Make sure "User Interface Enabled" is clicked on.

Setting Up a Gesture Recognizer Programmatically

- Create a Gesture Recognizer using one of the functions listed on the previous chart.

```
let tapRecognizer =
    UITapGestureRecognizer(target: self, action:
    #selector(handleTap(recognizer:)))
```

- Set up any properties for the Gesture Recognizer that you may want.
- Associate the Gesture Recognizer with the target object.

```
targetObject.addTapRecognizer(tapRecognizer)
```

- In the Swift file, write a function to handle the gesture.

```
@IBAction func handleTap(recognizer:
    UITapGestureRecognizer) {
        <code>
    }
```