ITP 342
Mobile App Dev

Animation

Core Animation

- Introduced in Mac OS X Leopard
- Uses animatable "layers" built on OpenGL
- UIKit supports Core Animation out of the box
  - Every UIView has a CALayer behind it

View Animations

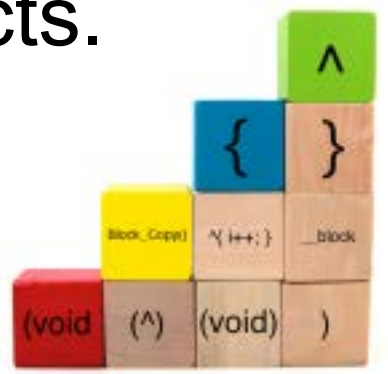
- UIKit makes basic animation easy
- Multiple view properties are animatable
  - frame (position and size)
  - transform
  - alpha (transparency)
  - background color

Animation Methods

- UIView has several class methods for animation:
  - `animateWithDuration: animations:`
  - `animateWithDuration: animations: completion:`
  - `animateWithDuration: delay: options: animations: completion:`
  - `animateWithDuration: delay: usingSpringWithDamping: initialSpringVelocity: options: animations: completion:`

Using Blocks to Animate

- UIView has various class methods that we can use to animate, but they use blocks.
- So we need to learn about blocks.
- A block is a chunk of code that can be executed at some future time.
- Blocks are functions that are objects.



Working with Blocks

- Blocks are a language-level feature added to C, Objective-C and C++, which allow you to create distinct segments of code that can be passed around to methods or functions as if they were values.
- Blocks are Objective-C objects, which means they can be added to collections like NSArray or NSDictionary.
- They also have the ability to capture values from the enclosing scope, making them similar to closures or lambdas in other programming languages.

Block Syntax

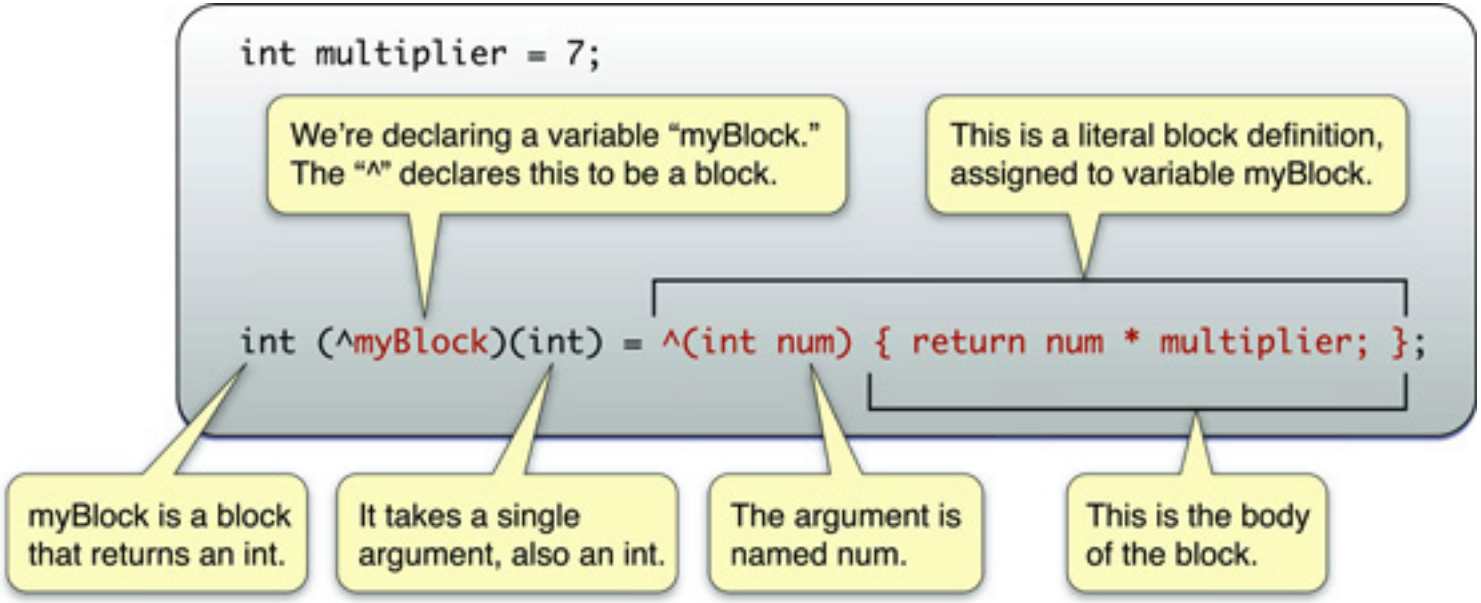
- The syntax to define a block literal uses the caret symbol (^), like this:

```
^{
   NSLog(@"This is a block");
}
```

- As with function and method definitions, the braces indicate the start and end of the block.
- In this example, the block doesn't return any value, and doesn't take any arguments.

Declaring a Block

- You use the ^ operator to declare a block variable and to indicate the beginning of a block literal.
- The body of the block itself is contained within {}.



Pass Blocks as Arguments

- It's common to pass blocks to functions or methods for invocation elsewhere.
- Blocks are also used for callbacks, defining the code to be executed when a task completes.
  - You can define the callback behavior at the time you initiate the task.
- Several methods in the Cocoa frameworks take **a block as an argument**, typically either to perform an operation on a collection of objects, or to use as a callback after an operation has finished.

Block Should Be Last

- It's best practice to use only one block argument to a method.
- If the method also needs other non-block arguments, the block should come last:

```
- (void)beginTaskWithName: (NSString *) name
   completion: (void(^)(void)) callback;
```

- This makes the method call easier to read when specifying the block inline, like this:

```
[self beginTaskWithName: @"MyTask"
   completion: ^{
      NSLog(@"The task is complete");
   }
];
```

Back to Animation

- UIView has several class methods for animation:
  - `animateWithDuration: animations:`
  - `animateWithDuration: animations: completion:`
  - `animateWithDuration: delay: options: animations: completion:`
  - `animateWithDuration: delay: usingSpringWithDamping: initialSpringVelocity: options: animations: completion:`

Animate

- Animate changes to one or more views using the specified duration.

```
+ (void) animateWithDuration:
    (NSTimeInterval) duration
    animations: (void (^)(void)) animations
```

- Parameters
  - **`duration`**: The total duration of the animations, measured in seconds. If you specify a negative value or 0, the changes are made without animating them.
  - **`animations`**: A block object containing the changes to commit to the views. This is where you programmatically change any animatable properties of the views in your view hierarchy. This block takes no parameters and has no return value. This parameter must not be NULL.

Example – 1 Animation

- In the ViewController, create a method that passes in a new name (as a NSString) and fades it in.

```
- (void) fadeInStudent: (NSString *) name {
    // Alpha = 0 means the text is transparent
    self.nameLabel.alpha = 0;
    self.nameLabel.text = name;

    [UIView animateWithDuration:1.0 animations:^{
        // Fade in the text of the label
        self.nameLabel.alpha = 1;
    } ];
}
```

Animate

- What we have:
  - The old string disappears and the new string fades in.
- What we want:
  - We want the old string to fade out first.
- We need 2 animations
  - First one to fade out old string
  - Second one to fade in new string after the first one finishes

Animate

- Animate changes to one or more views using the specified duration and completion handler.

```
+ (void) animateWithDuration: (NSTimeInterval)
    duration animations: (void (^)(void)) animations
    completion:(void (^)(BOOL finished))completion
```

- Parameters
  - **`duration`**: The total duration of the animations, measured in seconds.
  - **`animations`**: A block object containing the changes to commit to the views.
  - **`completion`**: A block object to be executed when the animation sequence ends. This block has no return value and takes a single Boolean argument that indicates whether or not the animations actually finished before the completion handler was called.

Example – 2 Animations

- Create a method that fades in the new text.
  - The example in class was the `animateStudent: firstName:` method.
- Create a method that fades out the old text of the label.
  - In the completion argument for the animation, call the method that fades in the new text.
  - The example in class was the `displayStudent: firstName:` method.

Example – 2 Animations

- Create a method that fades in the new text.

```
- (void) animateStudent: (NSString *) name {
    self.nameLabel.text = name;

    [UIView animateWithDuration:1.0
        animations:^{
            self.nameLabel.alpha = 1;
        }
    ];
}
```

Example – 2 Animations

- Create a method that fades out the old text of the label.
  - In the completion argument for the animation, call the method that fades in the new text.

```
- (void) displayStudent: (NSString *) name {
    [UIView animateWithDuration:1.0
        animations:^{
            // Fade out old text of label
            self.nameLabel.alpha = 0;
        }
        completion:^(BOOL finished) {
            // Upon completion, call animateStudent:
            [self animateStudent: name];
        }
    ];
}
```

Example – Colors

- Add code to rotate between 2 colors

```
- (void) animateStudent: (NSString *) name {
    self.nameLabel.text = name;

    // If black, set to USC cardinal red
    if (self.nameLabel.textColor == UIColor.blackColor) {
        self.nameLabel.textColor = [UIColor
            colorWithRed:(153.0f/255.0f)
            green:0.0 blue:0.0 alpha:1.0];
    } else {
        self.nameLabel.textColor = UIColor.blackColor;
    }

    [UIView animateWithDuration:1.0
        animations:^{
            self.nameLabel.alpha = 1;
        }
    ];
}
```

Resources

- Apple's Developer Site:
  - https://developer.apple.com/library/ios/documentation/windowsviews/conceptual/viewpg_iphoneos/animatingviews/animatingviews.html
  - https://developer.apple.com/library/ios/DOCUMENTATION/Cocoa/Conceptual/Blocks/Articles/00_Introduction.html
- The Pragmatic Studio's Using Blocks:
  - https://pragmaticstudio.com/blog/2010/7/28/ios4-blocks-1