

iPhone Development Workshop

Handout

Julian Dolce

Agenda

9:30 am – 10:00 am – Welcome, introduction

10:00 am – 11:30 am – Intro to Objective-C

11:30 am – 12:00 pm – Virtual Pet Exercise

12:00 pm – 1:00 pm – Lunch

1:00 pm – 2:00 pm – Flash to iPhone

2:00 pm – 3:00 pm – Interface Builder Walkthrough

3:00 pm – 3:15 pm – Break

3:15 pm – 4:00 pm – UITableViews and UITabBars

4:00 pm – 4:45 pm – Schedule Application exercise

4:45 pm – 5:15 pm – Tips&Tricks, QA, wrap up

NSString Format Specifiers

Specifier	Description
%@	Objective-C object, printed as the string returned by <code>descriptionWithLocale:</code> if available, or <code>description</code> otherwise. Also works with <code>CTypeRef</code> objects, returning the result of the <code>CFCopyDescription</code> function.
%%	'%' character
%d, %D, %i	Signed 32-bit integer (<code>int</code>)
%u, %U	Unsigned 32-bit integer (<code>unsigned int</code>)
%hi	Signed 16-bit integer (<code>short</code>)
%hu	Unsigned 16-bit integer (<code>unsigned short</code>)
%qi	Signed 64-bit integer (<code>long long</code>)
%qu	Unsigned 64-bit integer (<code>unsigned long long</code>)
%x	Unsigned 32-bit integer (<code>unsigned int</code>), printed in hexadecimal using the digits 0–9 and lowercase a–f
%X	Unsigned 32-bit integer (<code>unsigned int</code>), printed in hexadecimal using the digits 0–9 and uppercase A–F
%qx	Unsigned 64-bit integer (<code>unsigned long long</code>), printed in hexadecimal using the digits 0–9 and lowercase a–f
%qX	Unsigned 64-bit integer (<code>unsigned long long</code>), printed in hexadecimal using the digits 0–9 and uppercase A–F
%o, %O	Unsigned 32-bit integer (<code>unsigned int</code>), printed in octal
%f	64-bit floating-point number (<code>double</code>)
%e	64-bit floating-point number (<code>double</code>), printed in scientific notation using a lowercase e to introduce the exponent
%E	64-bit floating-point number (<code>double</code>), printed in scientific notation using an uppercase E to introduce the exponent
%g	64-bit floating-point number (<code>double</code>), printed in the style of %e if the exponent is less than –4 or greater than or equal to the precision, in the style of %f otherwise
%G	64-bit floating-point number (<code>double</code>), printed in the style of %E if the exponent is less than –4 or greater than or equal to the precision, in the style of %f otherwise
%c	8-bit unsigned character (<code>unsigned char</code>), printed by <code>NSLog()</code> as an ASCII character, or, if not an ASCII character, in the octal format <code>\\ddd</code> or the Unicode hexadecimal format <code>\\udddd</code> , where d is a digit
%C	16-bit Unicode character (<code>unichar</code>), printed by <code>NSLog()</code> as an ASCII character, or, if not an ASCII character, in the octal format <code>\\ddd</code> or the Unicode hexadecimal format <code>\\udddd</code> , where d is a digit
%s	Null-terminated array of 8-bit unsigned characters. %s interprets its input in the system encoding rather than, for example, UTF-8.

<code>%S</code>	Null-terminated array of 16-bit Unicode characters
<code>%p</code>	Void pointer (<code>void *</code>), printed in hexadecimal with the digits 0–9 and lowercase a–f, with a leading <code>0x</code>
<code>%L</code>	Length modifier specifying that a following <code>a</code> , <code>A</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , or <code>G</code> conversion specifier applies to a <code>long double</code> argument
<code>%a</code>	64-bit floating-point number (<code>double</code>), printed in scientific notation with a leading <code>0x</code> and one hexadecimal digit before the decimal point using a lowercase <code>p</code> to introduce the exponent
<code>%A</code>	64-bit floating-point number (<code>double</code>), printed in scientific notation with a leading <code>0X</code> and one hexadecimal digit before the decimal point using an uppercase <code>P</code> to introduce the exponent
<code>%F</code>	64-bit floating-point number (<code>double</code>), printed in decimal notation
<code>%z</code>	Length modifier specifying that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a <code>size_t</code> or the corresponding signed integer type argument
<code>%t</code>	Length modifier specifying that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a <code>ptrdiff_t</code> or the corresponding unsigned integer type argument
<code>%j</code>	Length modifier specifying that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a <code>intmax_t</code> or <code>uintmax_t</code> argument

Virtual Pet Exercise

Take the following example from Essential ActionScript 3.0 book and convert it to Objective-C. Below is the code to use when porting the setInterval portion of the sample.

```
//create the timer
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:1.0
target:self selector:@selector(digest) userInfo:nil repeats:YES];

//start the timer in 1 second
[timer performSelector:@selector( fire ) withObject:nil
afterDelay:1.0];

//call this when you want to stop the timer
[timer invalidate];
```

VirtualZoo.as

```
package zoo
{
    import flash.display.Sprite;

    public class VirtualZoo extends Sprite
    {
        public function VirtualZoo()
        {
            var pet = new VirtualPet( "Stan" );
            pet.eat( new Apple( ) );
            pet.eat( new Sushi( ) );
        }
    }
}
```

Virtual Pet.as

```
package zoo {
    import flash.utils.setInterval;
    import flash.utils.clearInterval;

    internal class VirtualPet {
        private static var maxLength = 20;
        private static var maxCalories = 2000;
        private static var caloriesPerSecond = 100;

        private var petName;
        private var currentCalories = VirtualPet.maxCalories/2;
        private var digestIntervalID;

        public function VirtualPet (name) {
            setName(name);
            digestIntervalID = setInterval(digest, 1000);
        }
    }
}
```

```
public function eat (foodItem) {
    if (currentCalories == 0) {
        trace(getName() + " is dead. You can't feed it.");
        return;
    }

    var newCurrentCalories = currentCalories +
foodItem.getCalories();
    if (newCurrentCalories > VirtualPet.maxCalories) {
        currentCalories = VirtualPet.maxCalories;
    } else {
        currentCalories = newCurrentCalories;
    }
    trace(getName() + " ate some " + foodItem.getName() + "."
        + " It now has " + currentCalories + " calories
remaining.");
}

public function getHunger () {
    return currentCalories / VirtualPet.maxCalories;
}

public function setName (newName) {
    // If the proposed new name has more than maxNameLength
characters...
    if (newName.length > VirtualPet.maxNameLength) {
        // ...truncate it
        newName = newName.substr(0, VirtualPet.maxNameLength);
    } else if (newName == "") {
        // ...otherwise, if the proposed new name is an empty
string,
        // then terminate this method without changing petName
        return;
    }

    // Assign the new, validated name to petName
    petName = newName;
}

public function getName () {
    return petName;
}
```

```

private function digest () {
  // If digesting more calories would leave the pet's
currentCalories at
  // 0 or less...
  if (currentCalories - VirtualPet.caloriesPerSecond <= 0) {
    // ...stop the interval from calling digest()
clearInterval(digestIntervalID);
    // Then give the pet an empty stomach
currentCalories = 0;
    // And report the pet's death
trace(getName() + " has died.");
  } else {
    // ...otherwise, digest the stipulated number of calories
currentCalories -= VirtualPet.caloriesPerSecond;

    // And report the pet's new status
trace(getName() + " digested some food. It now has "
      + currentCalories + " calories remaining.");
  }
}
}
}
}
}

```

Food.as

```

package zoo {
  public class Food {
    private var calories;
    private var name;

    public function Food (initialCalories) {
      setCalories(initialCalories);
    }

    public function getCalories () {
      return calories;
    }

    public function setCalories (newCalories) {
      calories = newCalories;
    }
  }
}

```

```
public function getName () {
    return name;
}

public function setName (newName) {
    name = newName;
}
}
```

Sushi.as

```
package zoo {
    public class Sushi extends Food {
        private static var DEFAULT_CALORIES = 500;

        public function Sushi (initialCalories = 0) {
            if (initialCalories <= 0) {
                initialCalories = Sushi.DEFAULT_CALORIES;
            }
            super(initialCalories);

            setName("Sushi");
        }
    }
}
```

Apple.as

```
package zoo {
    public class Apple extends Food {
        private static var DEFAULT_CALORIES = 100;

        public function Apple (initialCalories = 0) {
            if (initialCalories <= 0) {
                initialCalories = Apple.DEFAULT_CALORIES;
            }
            super(initialCalories);

            setName("Apple");
        }
    }
}
```

Positioning Objects with x and y

The iPhone SDK objects don't position objects based on x and y. Instead they use the center property, which takes a CGPoint object. To move a UIView with x and y properties to help when porting Flash projects to iPhone, use the following functions in a UIView subclass

```
-(void)x:(float)_x
{
    self.center = CGPointMake( _x + ( self.bounds.size.width / 2 ),
self.center.y );
}
```

```
-(void)y:(float)_y
{
    self.center = CGPointMake( self.center.x, _y + (
self.bounds.size.height / 2 ) );
}
```

```
-(float)y
{
    return self.center.y - ( self.bounds.size.height / 2 );
}
```

```
-(float)x
{
    return self.center.x - ( self.bounds.size.width / 2 );
}
```

Animating UIView properties

There are several different ways to animate a property of a UIView. One way is to use Core Animation and the built in UIView methods. The following code takes a UIView instance and animates it's alpha property.

```
[UIView beginAnimations:nil context:NULL];  
[UIView setAnimationDuration:.5];  
[UIView setAnimationCurve:UIViewAnimationCurveEaseOut];  
myview.alpha = 0;  
[UIView commitAnimations];
```