Objective-C Basics

iPhone SDK

- Enrolled students will be invited to developer program
  - Login to Program Portal
  - Request a Certificate
  - Download and install the SDK

The First Program in Objective-C

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {

    NSLog(@"Hello, World!");

    return 0;
}
```

OOP Vocabulary

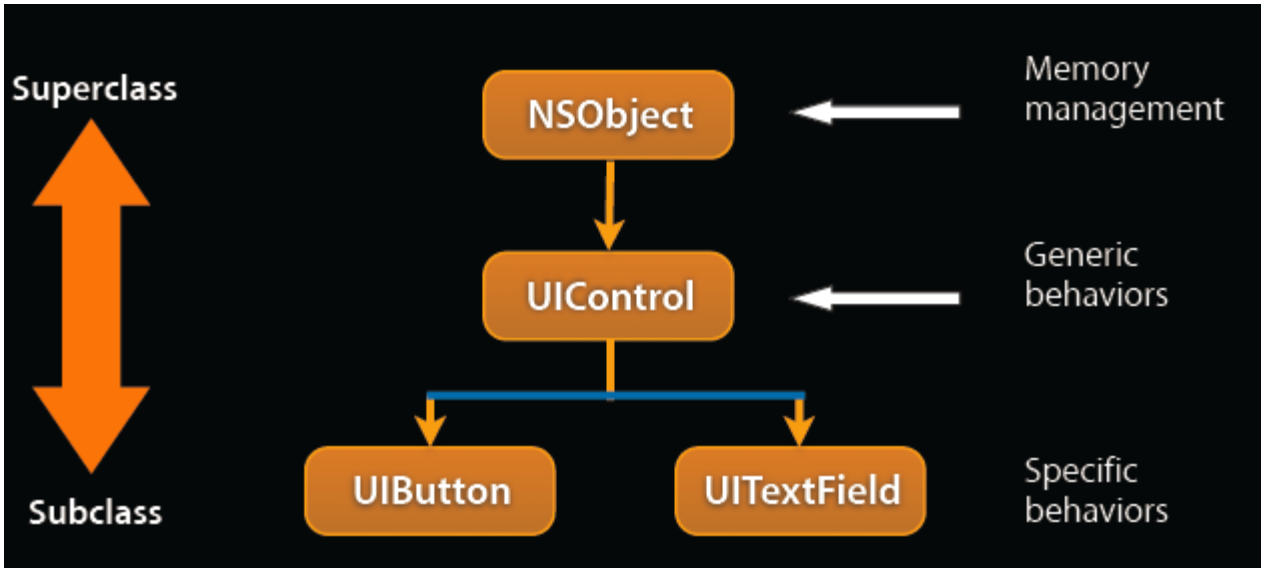
- **Class**: defines the grouping of data and code, the “type” of an object
- **Instance**: a specific allocation of a class
- **Method**: a “function” that an object knows how to perform
- **Instance Variable (or “ivar”)**: a specific piece of data belonging to an object

OOP Vocabulary

- Encapsulation
  - keep implementation private and separate from interface
- Polymorphism
  - different objects, same interface
- Inheritance
  - hierarchical organization, share code, customize or extend behaviors

Inheritance

- Hierarchical relation between classes
- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods



More OOP Info?

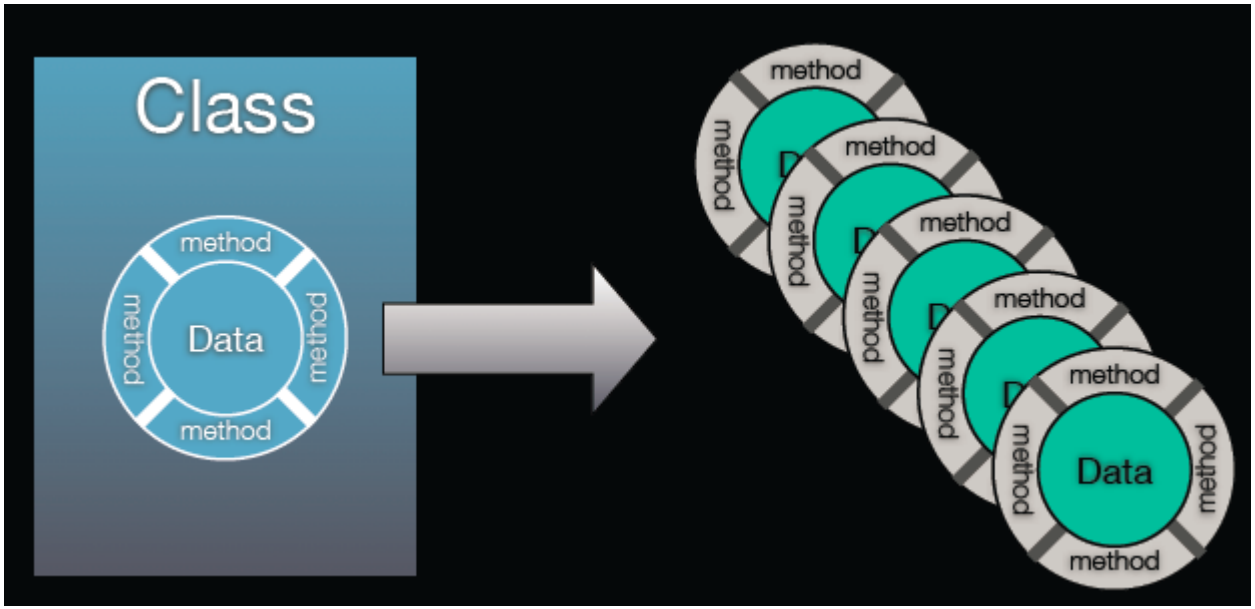
- Tons of books and articles on OOP
- Most Java or C++ book have OOP introductions
- Objective-C 2.0 Programming Language
  - http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC

Objective-C

- Strict superset of C
  - Mix C with ObjC
  - Or even C++ with ObjC (usually referred to as ObjC++)
- A very simple language, but some new syntax
- Single inheritance, classes inherit from one and only one superclass
- Protocols define behavior that cross classes
- Dynamic runtime

Classes and Instances

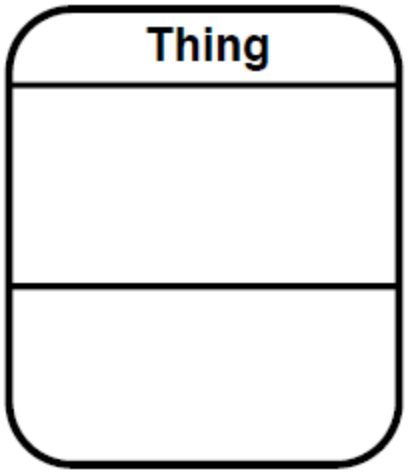
- In Objective-C, classes and instances are both objects
- Class is the blueprint to create instances



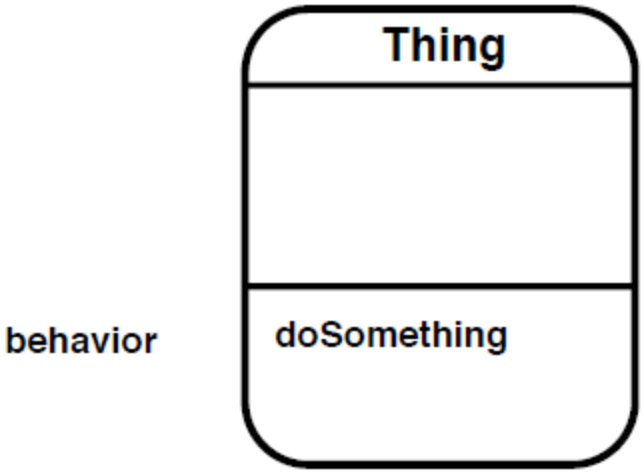
Classes and Objects

- Classes declare state and behavior
- State (data) is maintained using instance variables
- Behavior is implemented using methods
- Instance variables typically hidden
  - Accessible only using getter/setter methods

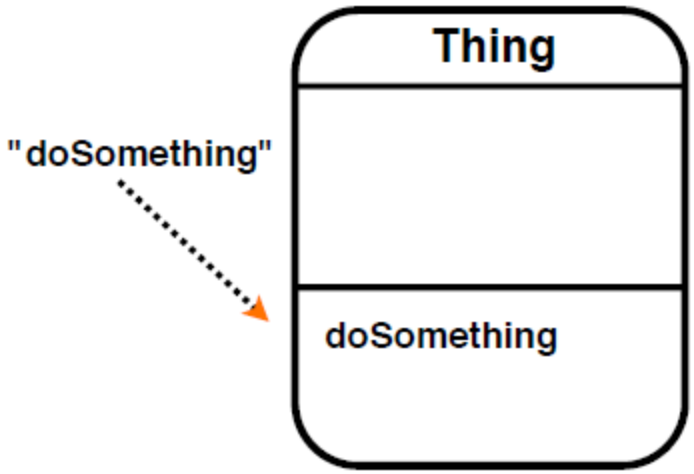
Object



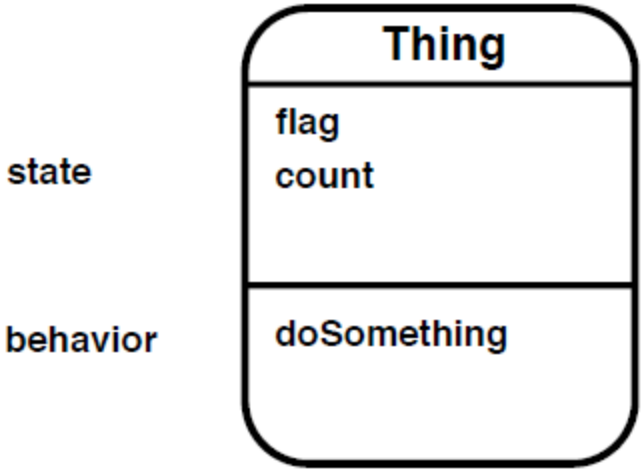
Behavior



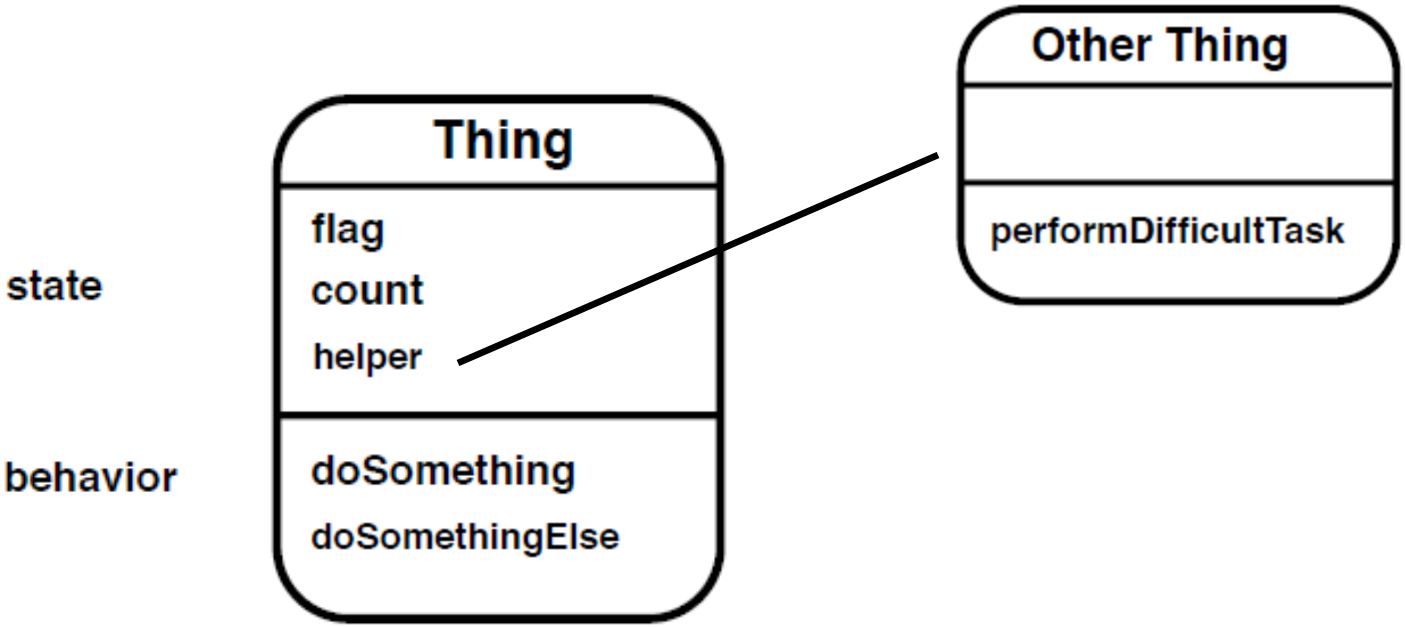
Message



State



Other Objects As State



OOP From ObjC Perspective

- Everybody has their own spin on OOP
  - Apple is no different
- For the spin on OOP from an ObjC perspective:
  - Read the “Object-Oriented Programming with Objective-C” document
  - http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/OOP_ObjC

Class and Instance Methods

- Instances respond to instance methods
  - (id) init;
  - (float) height;
  - (void) walk;
- Classes respond to class methods
  - + (id) alloc;
  - + (id) person;
  - + (Person *) sharedPerson;

Message syntax

- [receiver message]
- [receiver message:argument]
- [receiver message:arg1 andArg:arg2]

Message examples

```
Person *voter; //assume this exists
[voter castBallot];
int theAge = [voter age];
[voter setAge:21];
if ([voter canLegallyVote]) {
// do something voter-y
}
[voter registerForState:@”OH" party:@"Independant"];
NSString *name = [[voter spouse] name];
```

Terminology

- Message expression

  [receiver method: argument]
- Message

  [receiver method: argument]
- Selector

  [receiver method: argument]
- Method

  The code selected by a message

The First OO-Program in Objective-C

Shape-Procedure: Suppose a program draws a bunch of geometric shapes on the screen: circle, square, egg-shaped (Color and Boundary)

Let us first take a look of the procedure-C program!

```
NSString *colorName (ShapeColor color)
{
    switch (color) {
            case kRedColor:
                        return @"red";
                        break;

                    …
        return @"no clue";

} // colorName
```

```
void drawCircle (ShapeRect bounds,
ShapeColor fillColor)
{
    NSLog (@"drawing a circle at (%d %d
%d %d) in %@",
                    bounds.x, bounds.y,
bounds.width, bounds.height,
                    colorName (fillColor));
} // drawCircle
```

Class Definition

```
@interface Circle: NSObject
{
        ShapeColor fillColor;
        ShapeRect  bounds;
}

- (void) setFillColor: (ShapeColor) fillColor;

- (void) setBounds: (ShapeRect) bound;

- (void) draw;

@end // Circle;
```

Class Implementation

```
@implementation Circle

- (void) setFillColor: (ShapeColor) c
{
        fillColor = c;
} //setFillColor

- (void) setBounds: (ShapeRect) b
{
        bounds=b;
} //setBound

- (void) draw
{
        NSLog (@"drawing a circle at (%d %d %d %d) in %@",
                bounds.x, bounds.y, bounds.width, bounds.height,
                colorName (fillColor));
} //draw

@end
```

Message Examples

```
void drawShapes (id shapes[], int count)
{
        int i;

        for (i=0; i< count; i++) {
                [shapes[i] draw];
        }

} // drawShapes
```

```
int main (int argc, const char * argv[]) {
        id shapes[3];

        ShapeRect rect0 = {0, 0, 10, 30};
        shapes[0] = [Circle new];
        [shapes[0] setBounds: rect0];
        [shapes[0] setFillColor:kRedColor];

        ShapeRect rect1 = {30, 40, 50, 60};
        shapes[1] = [Rectangle new];
        [shapes[1] setBounds: rect1];
        [shapes[1] setFillColor: kGreenColor];

        ShapeRect rect2 = {15, 19, 37, 29};
        shapes[2] = [OblateSphereoid new];
        [shapes[2] setBounds: rect2];
        [shapes[2] setFillColor: kBlueColor];

        drawShapes (shapes, 3);

        return 0;
}
```

Inheritance (Common Class)

```
@interface Shape: NSObject
{
        ShapeColor fillColor;
        ShapeRect  bounds;
}

- (void) setFillColor: (ShapeColor) fillColor;

- (void) setBounds: (ShapeRect) bound;

- (void) draw;

@end // Shape;
```

```
@implementation Shape

- (void) setFillColor: (ShapeColor) c
{
        fillColor = c;
} //setFillColor

- (void) setBounds: (ShapeRect) b
{
        bounds=b;
} //setBound

- (void) draw
{
} //draw

@end
```

Inheritance (Common Class)

```
@interface Circle: Shape
@end // Circle;


@interface Rectangle: Shape
@end // Rectangle;

@interface OblateSphereoid: Shape
@end // OblateSphereoid;
```

```
@implementation Circle

- (void) setFillColor: (ShapeColor) c
{
        if (c==kRedColor) {
                c=kGreenColor;
        }

        [super setFillColor:c];
} // setFillColor


- (void) draw
{
        NSLog (@"drawing a circle at (%d
%d %d %d) in %@",
                bounds.x, bounds.y,
bounds.width, bounds.height,
                colorName (fillColor));
} //draw

@end
```

Dynamic and static typing

- Dynamically-typed object

  id anObject

  – Just id

  – Not id * (unless you really, really mean it...)
- Statically-typed object

  Person *anObject

- Objective-C provides compile-time, not runtime, type checking
- Objective-C always uses dynamic binding

The null object pointer

- Test for nil explicitly

```
if (person == nil) return;
```

- Or implicitly

```
if (!person) return;
```

- Can use in assignments and as arguments if expected

```
person = nil;
[button setTarget: nil];
```

- Sending a message to nil?

```
person = nil;
[person castBallot];
```

```
Bool areIntsDifferent (int thing1, int thing2)
{
    return (thing1-thing2);
}

if (areIntsDifferent(23,5)==YES) {
}

if (areIntsDifferent(23,5)){
}
```

BOOL typedef

- When ObjC was developed, C had no boolean type (C99 introduced one)
- ObjC uses a typedef to define BOOL as a type

```
BOOL flag = NO;
```

- Macros included for initialization and comparison: YES and NO

```
if (flag == YES)
if (flag)
if (!flag)
if (flag != YES)
flag = YES;
flag = 1;
```