Overview of key features in Objective C

Fall 2016 Class #2

- Questions from last weeks class

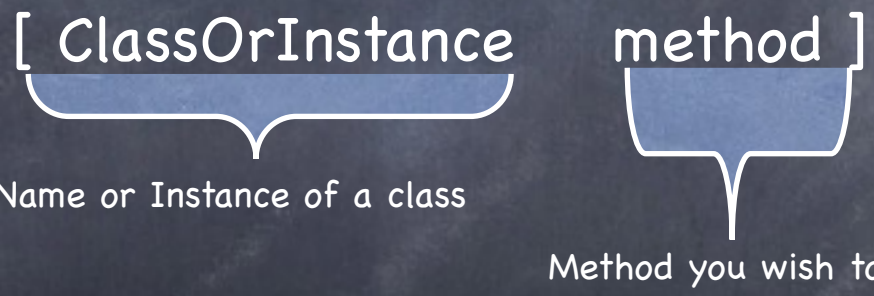
Objects, Classes, and Methods

Object is a thing. OO programming is something you want to do to that thing.

Class is a unique occurrence of an object.

Method can be applied to an instance of a class or class.

Car is an object – your car is an instance of an object that has specific characteristics about your car such as color, miles on car, dents on car etc. You perform methods on car such as fill up gas, drive, clean etc.



jitinCar = [car new]

InsuranceCo = [jitinCar insuredBy]

Slew of action methods to fill gas, report last oil change, etc. etc.

Xcode and Objective C using Fraction example

- Class Exercise to create project files can be found in Kohn book - chapter 2 in case you need to review steps
- Be sure to pick Mac OS application -> Command Line tool. On next selection page be sure to pick "Foundation" for type of app otherwise xcode template will generate something that won't work for the exercise.

Example of Simple Object compiled for Mac OS

- Refer to the exercise "fraction" on xcode.
- @interface – describes the class, its data components, and its methods. @end signals end of interface description for the interface section.
- @implementation – contains the code that implements the interface.
- @program – contains the program code to carry out the purpose of the program.
- Typically professional programs will keep interface in one file, and implementation of individual interfaces in separate files. For this class we will keep all three in one file most of the time to easily look up things.
- Rules for naming variables, classes etc. Xcode will help you check with an incorrect name starting with $name. Reserved words such as int, short etc. can't obviously be used. Once again Xcode will help
- Pick meaningful names and follow a convention to make your program read like a good story.

Classes and Methods

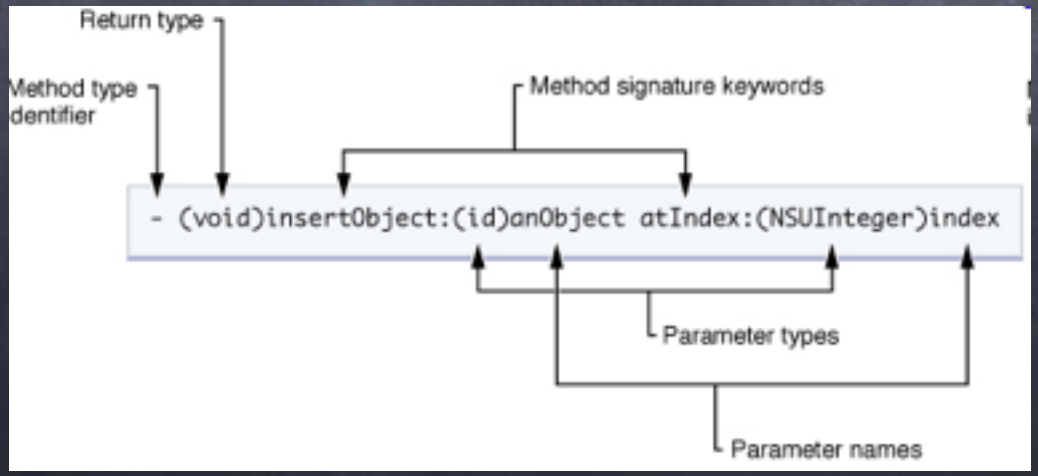
The leading (-) sign implies an instance method of a class

The leading (+) sign implies a class factory method. They combine allocation and initialization in one step and return the created object. Methods that create array for e.g. allocates space, initialized and returns the array object.

Instance Variables - every class instance would have its own - typical OO paradigm

Return value - like most programming languages do what you expect. (void) keyword in interface section would signal compiler that no return statement is expected. return -> returns 0.

- Method declaration syntax



Method Syntax

- Different than C, C++, Java, etc. Trips people off first few weeks
- Parameters are positional (Refer to pic on previous slide)
- There is space between parameters as opposed to more conventional comma separator
- Variable name is optional. You can pass parameters to method with colon and values.
- Invoking methods on an object is by sending messages. You send a message (method name of the object) along with parameters if any.
- For e.g. in our previous class we used [alert show] statement. It sends "show" message to alert class instance. Simple translation: invoke show method on alert object.

Classes and Methods

- Pointer to the class
  - Car *myCar; -> myCar is variable that holds pointer to a class (Car in this case)
  - myCar = [Car alloc] -> allocates memory of size needed for the Car to myCar. Technically [Car alloc] sends an alloc message to the Car to do what it needs to do
  - alloc, init etc. are derived from base class NSobject - refer to interface section to see definition
  - alloc allocates zeroed memory to the class. init initializes the class, variables etc. Don't worry about specifics yet.
  - Short cuts such as Car *myCar = [[Car alloc] init]; work. Go easy while you are learning syntax
  - [pool drain] and autorelease are things of past!

Data Types

- Basic data types – int, float, double, and char
- constant – single char, any number, or a character string
- Int can be expressed in octal (preceding with o) or hexadecimal (preceding with 0x). Print formatter options like %o or %x give you value in format you desire.
- Float can be expressed in scientific notation. Store using e to express exponent in an expression. For e.g. 2.25e-3. Use print formatter option like %g %e or %f to print in desired format. Try some NSlog statements if you have not used this syntax before.
- Double gives you guess what – double storage space. Be sure to check machine dependent implementation to figure range. Going over range will produce unexpected results.
- Qualifiers like long, long long, short, unsigned, signed give you what you are typically used to in other languages.

Data Types continued

- NSlog expects NSString object. Adding "@" in front of a string "I am a string" make @"I am a string" a NSString. So add @ character in front of a double quoted c-style string for right variable type for NSlog and other objects.
- id - generic object type. It's a basis for features like polymorphism - which we will briefly mention during the course.

Arithmetic Expressions

- Binary arithmetic operators - +, -, * and /
- Precedence of operators – left to right or right to left evaluation by the compiler depending on the precedence of an operator. Check language reference when in doubt. I write a small program to check my logic.
- unary operator(+/-) – changes polarity of variable when assigned. Doesn't change the variable – for e.g. -a. Unary minus has higher precedence than all other operators
- modulus operator % – remainder of the division
- Type casting – generally abused to silence complier or code audit tool warning. Useful when class variables find unexpected uses.
- Assignment operators – combine arithmetic operators with assignment operators e.g. +=, -=, /=, *= %= etc.
- Effect of count += 10 is =? count = ?

Loop Control

Legacy loop control statements from C. All control statements will use relational operators: ==, !=, <, <=, >, and >=

Syntax of the for statement is:

for (init_expression; loop_condition, loop_expression) followed by the program statements.

if loop condition is not met on the first evaluation - will program statement be executed at least once?

Nesting of the loop is allowed.

Use "for" statement for a loop that will be executed pre-determined number of times or the initial expression and looping condition involve same variables.

Increment and decrement operators: ++varName, varName- does what you expect. More readable after you get used to it.

break and continue statements. Works just like c.

Loop Control Continued

The while statement identical to c.
Syntax: while (expression) –execute program statements within while space.
program statements will keep executing till "expression" evaluates to TRUE.
Use "while" statement when loop control is based on a logic condition that's expected to change by program statements within the loop.
break and continue for additional control of the loop beyond "expression".
Class question – what will continue do?

Loop Control Continued

- The "do" loop. For and while may not execute if conditions are not met. In that case the program statements within "for" or a "while" scope will not execute.

```
do
    statements
while (expression)
```

- In "do" loop, statements execute once and then loop condition is evaluated to determine if statement block should be re-executed.
- Statements within the loop scope manipulate loop condition "expression"
- Break and continue: Class question what will break statement do within do scope?

Decision Statements

Most common statement in C are decision statements.

If (expression) followed by a block of program statement within if scope is the syntax.

If-else provides a scope for statements that do not meet "expression" condition.

```
If (expression)
    statements
    else if
    statements
    else
    statements
```

Switch statement

- A special type of if-then-else language construct. Very useful when you will be maintaining the program over a life cycle and adding more decision points.
- Only one case value will be matched. Be careful with break statement.

```
switch ( expression )
{
case value1:
program statement
program statement
...
break;
case value2:
program statement
program statement
...
break;
... // will this statement execute?
case valuen:
program statement
program statement
...
break; // what if this break statement is missing
default:
program statement
program statement
break;
}
```

The conditional operator

- The most over rated operator.
- You can accomplish same thing with if statement.
- Programmer use it to show their prowess. 99% of the time reader of code needs to refer back to the syntax.
- Seen more bugs with this statement than any other statement in my professional life.
- Many coding standards outright ban it.
- Complier will generate the same size of binary with if statement.
- Enough said!

Condition ? expression1 : expression2
If condition is true expression1 is executed otherwise expression2 is executed.

```
If (condition) {
  expression1;
} else {
expression2;
}
```

Achieves same result as
Condition ? Expression1 : expression2

The Fraction Class

- Xcode demo to highlight concepts we talked about.
- Step by step to create project files for earlier iOS version can be found in Kohn book - chapter 2.
- Hint: Be sure to pick Mac OS application -> Command Line tool. On next selection page be sure to pick "Foundation" for type of app otherwise xcode template will generate something that won't work for the exercise.

Bit Operators

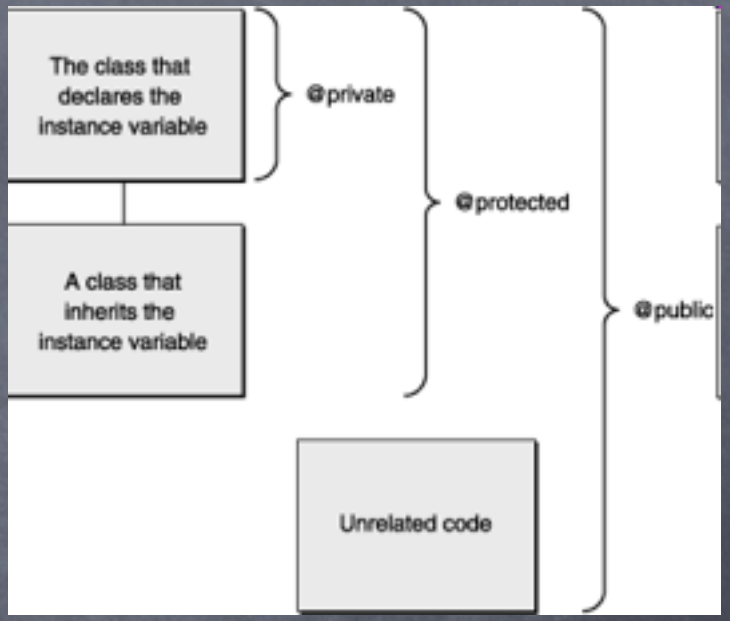
- Implements Boolean algebraic operations. Very powerful when working with logic on hardware. Very brief introduction – use reference and write sample program to understand effect of an operator.
- Bit operators: And is &, inclusive OR - |, OR - ^, one's complement - ~, left shift - <<, right shift - >>
- Additional Data types: _Bool, _Complex, _Imaginary. Complex and Imaginary types come from Fortran legacy and used mostly in modeling engineering problems.

Classes continued

- Local variables: Their values exist only during execution of the method and they can be accessed only from within the method in which they are defined.
- Instance variables: They retain their values through method calls. Declared in the interface section and is contained in every instance of the object – unique to the instance.
- Static Variables: Have default initial value of 0 and retain their values through method or function invocations. C equivalent is global variable which others can access.
- Self keyword: A variable that refers to the object that is the receiver of the current method

Variable Scope, Data Types

- Understand difference between @private, @protected and @public variables
- @package scope is @public inside the image that implements the class but like @private to others.
- A directive applies to all the instance variables listed after it, up to the next directive or end of the list as shown nearby.
- Scope of variables in Worker Class




```
@interface Worker:NSObject {
        char *name;
    @private
        int age;
    char *evaluation;
    @protected
        id job;
    float wage;
    @public
        id boss;
```

Initialization, Self and Super

- Several initialization methods are available in base classes. NSArray has six initialization methods: initWithArray, initWithArray:copyItems, initWithContentsOfFile, initWithContentsOfURL, initWithObjects, initWithObjects:count.
- One strategy is to have one method as preferred/designated initializer and have rest of the methods use it as base and then add as necessary.
- self and super are keywords to compiler to tell where to begin the search.
- [self negotiate] statement in Low will send message to ---- class
- [super negotiate] statement in Mid will send message to ----- class
- [self negotiate] statement in Mid will send message to ----- class
- [self negotiate] statement in High will send message to ----- class
- [super negotiate] statement in Low will send message to ----- class

```
(Fraction *) initWith: (int) n: (int) d
{
    self = [super init];
    if (self) {
        [self setTo: n over: d];
    }
    return self;
}
```

Class Exercise and Home Work

- HW # 1 - due on Class #3 on 9/7. Please put your project folder on USB Drive.

- Update the calculator class described in Kohn text book (program 4.6) to add memory capabilities to the Calculator class from program 4.6 (exercise #10 in the book). Here is the description from Chapter 4, exercise 10 in case you have different edition of the book.

Add a memory capability to the Calculator class from Program 4.6. Implement the following method declarations and test them:

```
-(double) memoryClear;
-(double) memoryStore;
-(double) memoryRecall;
-(double) memoryAdd: (double) value; -(double) memorySubtract: (double) value;
```

```
// clear memory
// set memory to accumulator // set accumulator to memory // add value into memory
// subtract value from memory
```

The last two methods set the accumulator as well as perform the indicated operation on memory. Have all the methods return the value of the accumulator.

Summary