TODAY'S AGENDA

1. PRIMITIVES
2. FOUNDATION CLASSES
3. CLASS FILES
   - A. PROPERTIES
   - B. METHODS

01 PRIMITIVES

ints, floats, booleans, char & more

PRIMITIVES

NAME	TYPE	EXAMPLE

PRIMITIVES

NAME	TYPE	EXAMPLE

PRIMITIVES

NAME	TYPE	EXAMPLE
void	Void	Nothing

PRIMITIVES

NAME	TYPE	EXAMPLE
void	Void	Nothing
int	Integer	...-1, 0, 1, 2...

PRIMITIVES

NAME	TYPE	EXAMPLE
void	Void	Nothing
int	Integer	...-1, 0, 1, 2...
unsigned	Unsigned Integer	0, 1, 2....

PRIMITIVES

NAME	TYPE	EXAMPLE
void	Void	Nothing
int	Integer	...-1, 0, 1, 2...
unsigned	Unsigned Integer	0, 1, 2....
float	Floating point number	-0.33, 0.5, 1.223

PRIMITIVES

NAME	TYPE	EXAMPLE
void	Void	Nothing
int	Integer	...-1, 0, 1, 2...
unsigned	Unsigned Integer	0, 1, 2....
float	Floating point number	-0.33, 0.5, 1.223
double	Double precision float point number	0.2342134213421340

PRIMITIVES

NAME	TYPE	EXAMPLE
void	Void	Nothing
int	Integer	...-1, 0, 1, 2...
unsigned	Unsigned Integer	0, 1, 2....
float	Floating point number	-0.33, 0.5, 1.223
double	Double precision float point number	0.2342134213421340
char	Character	h, a, l

PRIMITIVES

NAME	TYPE	EXAMPLE
void	Void	Nothing
int	Integer	...-1, 0, 1, 2...
unsigned	Unsigned Integer	0, 1, 2....
float	Floating point number	-0.33, 0.5, 1.223
double	Double precision float point number	0.2342134213421340
char	Character	h, a, l
BOOL	Boolean	0, 1; TRUE, FALSE; Yes, No

BOOLEAN

```
BOOL isBool = YES;

NSLog(@"%d", isBool);
NSLog(@"%@", isBool ? @"YES" : @"NO");
```

CHAR

```
char aChar = 'a';

NSLog(@"The letter %c", aChar);
```

INTEGER

```
int anInt = 24;

NSLog(@"%i", anInt);
```

FLOAT

```
float aFloat = -21.09f;

NSLog(@"%f", aFloat);
NSLog(@"%8.2f", aFloat);
```

ID TYPE

The id type is the generic type for all Objective-C objects. You can think of it as the object-oriented version of C's void pointer.

```
id mysteryObject = @"An NSString object";
NSLog(@"%@", [mysteryObject description]);

mysteryObject = @{@"model": @"Ford", @"year": @1967};
NSLog(@"%@", [mysteryObject description]);
```

NSLOG CHART

bit.ly/1dG8rHy

MATHEMATICAL OPERATIONS

MATHEMATICAL OPERATIONS

+	for addition

MATHEMATICAL OPERATIONS

+	for addition
-	for subtraction

MATHEMATICAL OPERATIONS

+	for addition
-	for subtraction
/	for division

MATHEMATICAL OPERATIONS

+	for addition
-	for subtraction
/	for division
*	for multiplication

MATHEMATICAL OPERATIONS

+	for addition
-	for subtraction
/	for division
*	for multiplication
%	for modulus

MODULUS

The modulus operator is not a percentage calculation. The result of the % operator is the remainder from the integer division of the first operand by the second (if the value of the second operand is zero, the behavior of % is undefined).

02 FOUNDATION FRAMEWORK

NSString, NSNumber, NSDictionary, NSSet, NSArray

NSNUMBER

NSNUMBER

The NSNumber class is a lightweight, object-oriented wrapper around C's numeric primitives. It's main job is to store and retrieve primitive values, and it comes with dedicated methods for each data type.

NSNUMBER

The NSNumber class is a lightweight, object-oriented wrapper around C's numeric primitives. It's main job is to store and retrieve primitive values, and it comes with dedicated methods for each data type.

**Most used with Core Data so if you plan on using Core Data for anything this is how it is stored. To work with the numbers you will need to "unwrap" them once they get to their destination.

NSNUMBER

The NSNumber class is a lightweight, object-oriented wrapper around C's numeric primitives. It's main job is to store and retrieve primitive values, and it comes with dedicated methods for each data type.

**Most used with Core Data so if you plan on using Core Data for anything this is how it is stored. To work with the numbers you will need to "unwrap" them once they get to their destination.

No garbage collection in Objective C, Objective C uses reference counting
All objects live in the heap
Objects always have pointers (* means pointer)

NSNUMBER

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);
NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];	NSLog(@"%hhu", [aUChar unsignedCharValue]);

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);
NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];	NSLog(@"%hhu", [aUChar unsignedCharValue]);
NSNumber *aShort = [NSNumber numberWithShort:32767];	NSLog(@"%hi", [aShort shortValue]);

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);
NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];	NSLog(@"%hhu", [aUChar unsignedCharValue]);
NSNumber *aShort = [NSNumber numberWithShort:32767];	NSLog(@"%hi", [aShort shortValue]);
NSNumber *aUShort = [NSNumber numberWithUnsignedShort:65535];	NSLog(@"%hu", [aUShort unsignedShortValue]);

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);
NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];	NSLog(@"%hhu", [aUChar unsignedCharValue]);
NSNumber *aShort = [NSNumber numberWithShort:32767];	NSLog(@"%hi", [aShort shortValue]);
NSNumber *aUShort = [NSNumber numberWithUnsignedShort:65535];	NSLog(@"%hu", [aUShort unsignedShortValue]);
NSNumber *anInt = [NSNumber numberWithInt:2147483647];	NSLog(@"%i", [anInt intValue]);

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);
NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];	NSLog(@"%hhu", [aUChar unsignedCharValue]);
NSNumber *aShort = [NSNumber numberWithShort:32767];	NSLog(@"%hi", [aShort shortValue]);
NSNumber *aUShort = [NSNumber numberWithUnsignedShort:65535];	NSLog(@"%hu", [aUShort unsignedShortValue]);
NSNumber *anInt = [NSNumber numberWithInt:2147483647];	NSLog(@"%i", [anInt intValue]);
NSNumber *aUInt = [NSNumber numberWithUnsignedInt:4294967295];	NSLog(@"%u", [aUInt unsignedIntValue]);

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);
NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];	NSLog(@"%hhu", [aUChar unsignedCharValue]);
NSNumber *aShort = [NSNumber numberWithShort:32767];	NSLog(@"%hi", [aShort shortValue]);
NSNumber *aUShort = [NSNumber numberWithUnsignedShort:65535];	NSLog(@"%hu", [aUShort unsignedShortValue]);
NSNumber *anInt = [NSNumber numberWithInt:2147483647];	NSLog(@"%i", [anInt intValue]);
NSNumber *aUInt = [NSNumber numberWithUnsignedInt:4294967295];	NSLog(@"%u", [aUInt unsignedIntValue]);
NSNumber *aFloat = [NSNumber numberWithFloat:26.99f];	NSLog(@"%lu", [aULong unsignedLongValue]);

NSNUMBER

NSNumber *aBool = [NSNumber numberWithBool:NO];	NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSNumber *aChar = [NSNumber numberWithChar:'z'];	NSLog(@"%c", [aChar charValue]);
NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];	NSLog(@"%hhu", [aUChar unsignedCharValue]);
NSNumber *aShort = [NSNumber numberWithShort:32767];	NSLog(@"%hi", [aShort shortValue]);
NSNumber *aUShort = [NSNumber numberWithUnsignedShort:65535];	NSLog(@"%hu", [aUShort unsignedShortValue]);
NSNumber *anInt = [NSNumber numberWithInt:2147483647];	NSLog(@"%i", [anInt intValue]);
NSNumber *aUInt = [NSNumber numberWithUnsignedInt:4294967295];	NSLog(@"%u", [aUInt unsignedIntValue]);
NSNumber *aFloat = [NSNumber numberWithFloat:26.99f];	NSLog(@"%lu", [aULong unsignedLongValue]);
NSNumber *aDouble = [NSNumber numberWithDouble:26.99];	NSLog(@"%f", [aFloat floatValue]);

NUMERIC LITERALS

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

```
NSNumber *aBool = @NO;
```

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

```
NSNumber *aBool = @NO;
```

```
NSNumber *aChar = @'z';
```

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

```
NSNumber *aBool = @NO;
NSNumber *aChar = @'z';
NSNumber *anInt = @2147483647;
```

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

```
NSNumber *aBool = @NO;
NSNumber *aChar = @'z';
NSNumber *anInt = @2147483647;
NSNumber *aUInt = @4294967295U;
```

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

```
NSNumber *aBool = @NO;
NSNumber *aChar = @'z';
NSNumber *anInt = @2147483647;
NSNumber *aUInt = @4294967295U;
NSNumber *aLong = @9223372036854775807L;
```

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

```
NSNumber *aBool = @NO;
NSNumber *aChar = @'z';
NSNumber *anInt = @2147483647;
NSNumber *aUInt = @4294967295U;
NSNumber *aLong = @92233720368547758O7L;
NSNumber *aFloat = @26.99F;
```

NUMERIC LITERALS

Xcode 4.4 introduced numeric literals, which offer a much more convenient alternative to the above factory methods. The NSNumber version of BOOL's, char's, int's and double's can all be created by simply prefixing the corresponding primitive type with the @ symbol; however, unsigned int's, long's, and float's must be appended with the U, L, or F modifiers, as shown below.

```
NSNumber *aBool = @NO;
NSNumber *aChar = @'z';
NSNumber *anInt = @2147483647;
NSNumber *aUInt = @4294967295U;
NSNumber *aLong = @9223372036854775807L;
NSNumber *aFloat = @26.99F;
NSNumber *aDouble = @26.99;
```

NUMERIC LITERALS ARITHMETIC

In addition to literal values, it's possible to box arbitrary C expressions using the @() syntax. This makes it trivial to turn basic arithmetic calculations into NSNumber objects:

```
double x = 24.0;
NSNumber *result = @(x * .15);

NSLog(@"%.2f", [result doubleValue]);
```

NSSTRING

NSString class is the basic tool for representing text in an Objective-C application. Aside from providing an object-oriented wrapper for strings, NSString provides many powerful methods for searching and manipulating its contents. It also comes with native Unicode support. The most common way to create strings is using the literal @"Some String" syntax, but the stringWithFormat: class method is also useful for generating strings that are composed of variable values.

NSSTRING (CONT)

```
NSString *make = @"Porsche";
NSString *model = @"911";
int year = 1968;

NSString *message = [NSString stringWithFormat:@"That's a %@ %@
from %d!", make, model, year];

NSLog(@"%@", message);
```

NSSET

An NSSet object represents a static, unordered collection of distinct objects. Sets are optimized for membership checking, so if you're asking a lot of "is this object part of this group?" kind of questions, you should be using a set—not an array.

NSSET (CONT)

```
NSSet *americanMakes = [NSSet setWithObjects:@"Chrysler", @"Ford",
                                             @"General Motors", nil];
NSLog(@"%@", americanMakes);
```

```
NSArray *japaneseMakes = @[@"Honda", @"Mazda",
                           @"Mitsubishi", @"Honda"];
NSSet *uniqueMakes = [NSSet setWithArray:japaneseMakes];
NSLog(@"%@", uniqueMakes);      // Honda, Mazda, Mitsubishi
```

NSARRAY

NSArray is Objective-C's general-purpose array type. It represents an ordered collection of objects, and it provides a high-level interface for sorting and otherwise manipulating lists of data. Arrays aren't as efficient at membership checking as sets, but the trade-off is that they reliably record the order of their elements.

NSARRAY (CONT)

```
NSArray *ukMakes = [NSArray arrayWithObjects:@"Aston Martin",
                    @"Lotus", @"Jaguar", @"Bentley", nil];
```

LITERAL SYNTAX

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
```

NSDictionary

Like an NSSet, the NSDictionary class represents an unordered collection of objects; however, they associate each value with a key, which acts like a label for the value. This is useful for modeling relationships between pairs of objects.

NSDICTIONARY (CONT)

```
NSDictionary* inventory = [NSDictionary dictionaryWithObjectsAndKeys:
                          [NSNumber numberWithInt:13], @"Mercedes-Benz SLK250",
                          [NSNumber numberWithInt:22], @"Mercedes-Benz E350",
                          [NSNumber numberWithInt:19], @"BMW M3 Coupe",
                          [NSNumber numberWithInt:16], @"BMW X6", nil];
```

LITERAL SYNTAX

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};
```

NSDATE

An NSDate object represents a specific point in time, independent of any particular calendrical system, time zone, or locale. Internally, it just records the number of seconds from an arbitrary reference point (January 1st, 2001 GMT).

NDDATE (CONT)

```
NSDate *now = [NSDate date];
```

NSDATE WITH FORMATTER

```
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
[formatter setDateFormat:@"MM/dd/yy"];
NSString *prettyDate = [formatter stringFromDate:now]; // 08/31/13
```

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November
E	Sun

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November
E	Sun
EEEE	Sunday

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November
E	Sun
EEEE	Sunday
Week' w 'of 52'	Week 45 of 52

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November
E	Sun
EEEE	Sunday
Week' w 'of 52'	Week 45 of 52
Day' D 'of 365'	Day 309 of 365

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November
E	Sun
EEEE	Sunday
Week' w 'of 52'	Week 45 of 52
Day' D 'of 365'	Day 309 of 365
QQQ	Q4

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November
E	Sun
EEEE	Sunday
Week' w 'of 52'	Week 45 of 52
Day' D 'of 365'	Day 309 of 365
QQQ	Q4
QQQQ	4th quarter

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
M/d/y	11/4/2012
MM/dd/yy	11/04/12
MMM d, ''yy	Nov 4, '12
MMMM	November
E	Sun
EEEE	Sunday
Week' w 'of 52'	Week 45 of 52
Day' D 'of 365'	Day 309 of 365
QQQ	Q4
QQQQ	4th quarter
m 'minutes past' h	9 minutes past 8

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
h:mm a	8:09 PM

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
h:mm a	8:09 PM
HH:mm:ss's'	20:09:00s

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
h:mm a	8:09 PM
HH:mm:ss's'	20:09:00s
HH:mm:ss:SS	20:09:00:00

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
h:mm a	8:09 PM
HH:mm:ss's'	20:09:00s
HH:mm:ss:SS	20:09:00:00
h:mm a zz	8:09 PM CST

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
h:mm a	8:09 PM
HH:mm:ss's'	20:09:00s
HH:mm:ss:SS	20:09:00:00
h:mm a zz	8:09 PM CST
h:mm a zzzz	8:09 PM Central Standard Time

NDDATE (CONT)

FORMAT STRING	OUTPUT STRING
h:mm a	8:09 PM
HH:mm:ss's'	20:09:00s
HH:mm:ss:SS	20:09:00:00
h:mm a zz	8:09 PM CST
h:mm a zzzz	8:09 PM Central Standard Time
yyyy-MM-dd HH:mm:ss Z	2012-11-04 20:09:00 -0600

CLASS FILES

CLASS FILES

INTERFACE FILE (.H)

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property NSString* title;

@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property NSString* title;
@property BOOL completed;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property NSString* title;
@property BOOL completed;
@property NSNumber* age;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property NSString* title;
@property BOOL completed;
@property NSNumber* age;
@property int weight;


@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property NSString* title;
@property BOOL completed;
@property NSNumber* age;
@property int weight;
@property float winningPct;

@end
```

PROPERTIES

PROPERTIES

atomic (default) - It will lock the code which will set or get your attribute and make sure that process is completed before it allow any other code to execute. (DON'T USE)

PROPERTIES

atomic (default) - It will lock the code which will set or get your attribute and make sure that process is completed before it allow any other code to execute. (DON'T USE)

nonatomic - this property getter and setter are not thread safe
We do not need locking. (USE IT)

PROPERTIES

atomic (default) - It will lock the code which will set or get your attribute and make sure that process is completed before it allow any other code to execute. (DON'T USE)

nonatomic - this property getter and setter are not thread safe
We do not need locking. (USE IT)

strong - when using ARC strong (helps the compiler) says I want this object to stay in the heap as long as I point to it
example setting an object to 0 or nil ARC will remove it.

PROPERTIES

atomic (default) - It will lock the code which will set or get your attribute and make sure that process is completed before it allow any other code to execute. (DON'T USE)

nonatomic - this property getter and setter are not thread safe
We do not need locking. (USE IT)

strong - when using ARC strong (helps the compiler) says I want this object to stay in the heap as long as I point to it
example setting an object to 0 or nil ARC will remove it.

weak - only keep this in the heap as long as someone else is pointing to it strong (only interested in pointing this as long as someone else in interested in pointing to it)

CLASS FILES

CLASS FILES

INTERFACE FILE (.H)

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic) NSString* title;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic) NSString* title;
@property (nonatomic) BOOL completed;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic) NSString* title;
@property (nonatomic) BOOL completed;
@property (nonatomic) NSNumber* age;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic) NSString* title;
@property (nonatomic) BOOL completed;
@property (nonatomic) NSNumber* age;
@property (nonatomic) int weight;


@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic) NSString* title;
@property (nonatomic) BOOL completed;
@property (nonatomic) NSNumber* age;
@property (nonatomic) int weight;
@property (nonatomic) float winningPct;

@end
```

CLASS FILES

CLASS FILES

INTERFACE FILE (.H)

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic, strong) NSString* title;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic, strong) NSString* title;
@property (nonatomic, isGetter=isCompleted) BOOL completed;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic, strong) NSString* title;
@property (nonatomic, isGetter=isCompleted) BOOL completed;
@property (nonatomic, strong) NSNumber* age;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic, strong) NSString* title;
@property (nonatomic, isGetter=isCompleted) BOOL completed;
@property (nonatomic, strong) NSNumber* age;
@property (nonatomic) int weight;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic, strong) NSString* title;
@property (nonatomic, isGetter=isCompleted) BOOL completed;
@property (nonatomic, strong) NSNumber* age;
@property (nonatomic) int weight;
@property (nonatomic) float winningPct;



@end
```

CLASS FILES

INTERFACE FILE (.H)

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

@interface MyClass : NSObject

@property (nonatomic, strong) NSString* title;
@property (nonatomic, isGetter=isCompleted) BOOL completed;
@property (nonatomic, strong) NSNumber* age;
@property (nonatomic) int weight;
@property (nonatomic) float winningPct;

- (void) createData;

@end
```

METHOD

METHOD

```
- (void) insertObject:(id)anObject atIndex:(NSUInteger)index;
```

METHOD

method type

(-) instance method

(+) class method


```
- (void) insertObject:(id)anObject atIndex:(NSUInteger)index;
```

METHOD

method type

(-) instance method

(+) class method

↓

- (void) insertObject:(id)anObject atIndex:(NSUInteger)index;

↑

return type

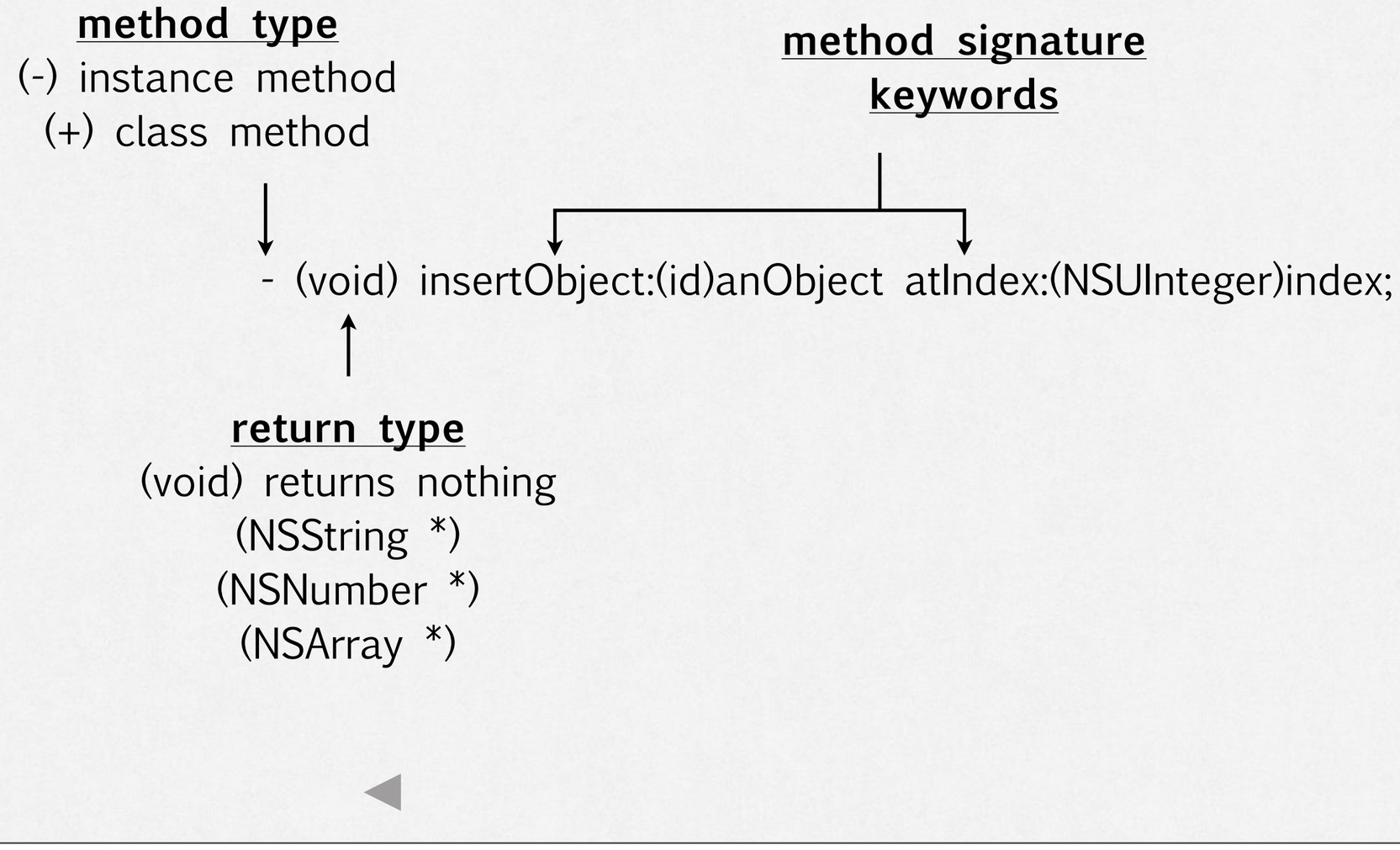
(void) returns nothing

(NSString *)

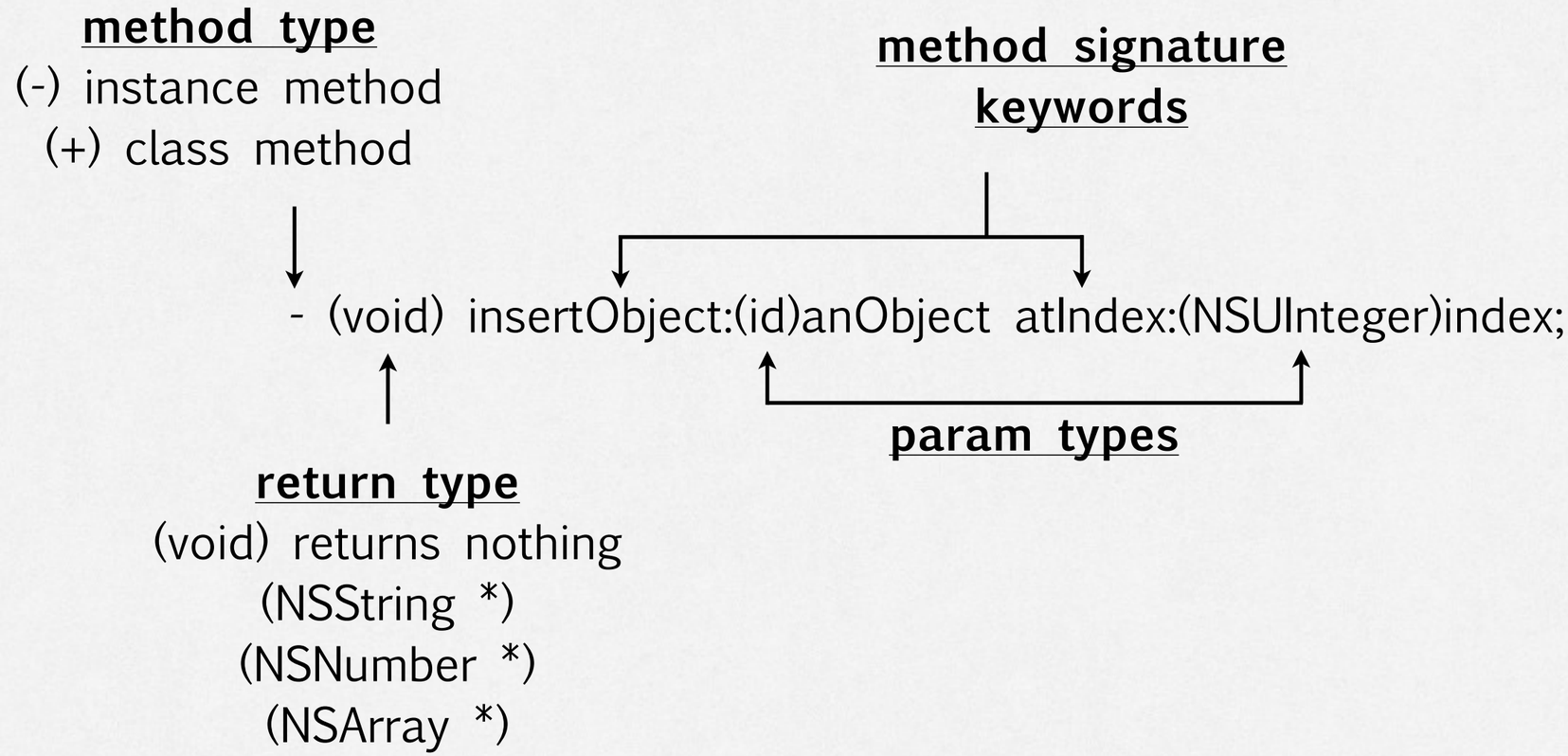
(NSNumber *)

(NSArray *)

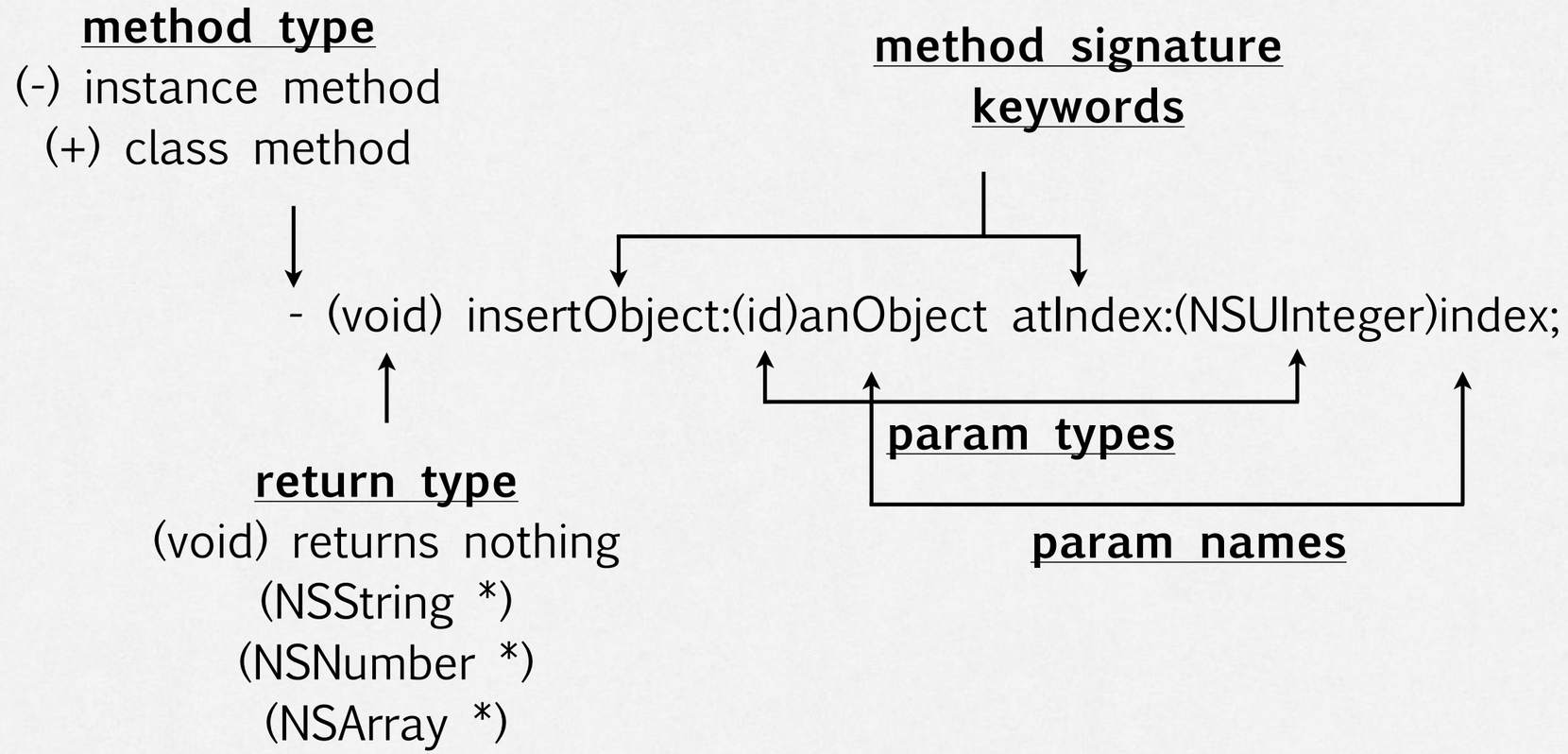
METHOD


method type
(-) instance method
(+) class method
method signature
keywords
- (void) insertObject:(id)anObject atIndex:(NSUInteger)index;
return type
(void) returns nothing
(NSString *)
(NSNumber *)
(NSArray *)

METHOD


method type
(-) instance method
(+) class method
method signature
keywords
- (void) insertObject:(id)anObject atIndex:(NSUInteger)index;
return type
(void) returns nothing
(NSString *)
(NSNumber *)
(NSArray *)
param types

METHOD


method type
(-) instance method
(+) class method
method signature
keywords
- (void) insertObject:(id)anObject atIndex:(NSUInteger)index;
return type
(void) returns nothing
(NSString *)
(NSNumber *)
(NSArray *)
param types
param names

CLASS FILES

CLASS FILES

IMPLEMENTATION FILE (.H)

CLASS FILES

IMPLEMENTATION FILE (.H)

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

@implementation MyClass

@end
```

CLASS FILES

IMPLEMENTATION FILE (.H)

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

@implementation MyClass
@synthesize title, age, weight ...

@end
```

CLASS FILES

IMPLEMENTATION FILE (.H)

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

@implementation MyClass
@synthesize title, age, weight ...

- (void) createData
{
  //code here
}

@end
```

CLASS FILES

IMPLEMENTATION FILE (.H)

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

@implementation MyClass
@synthesize title, age, weight ...

- (void) createData
{
  //code here
}

- (void) anotherMethod:(NSString *)title
{
  //code here
}

@end
```

03 CODE TIME

FILES

bit.ly/18e5tW9

NEXT WEEK'S AGENDA

1. MVC
2. PROTOCOLS & DELEGATES
3. OBJECTS