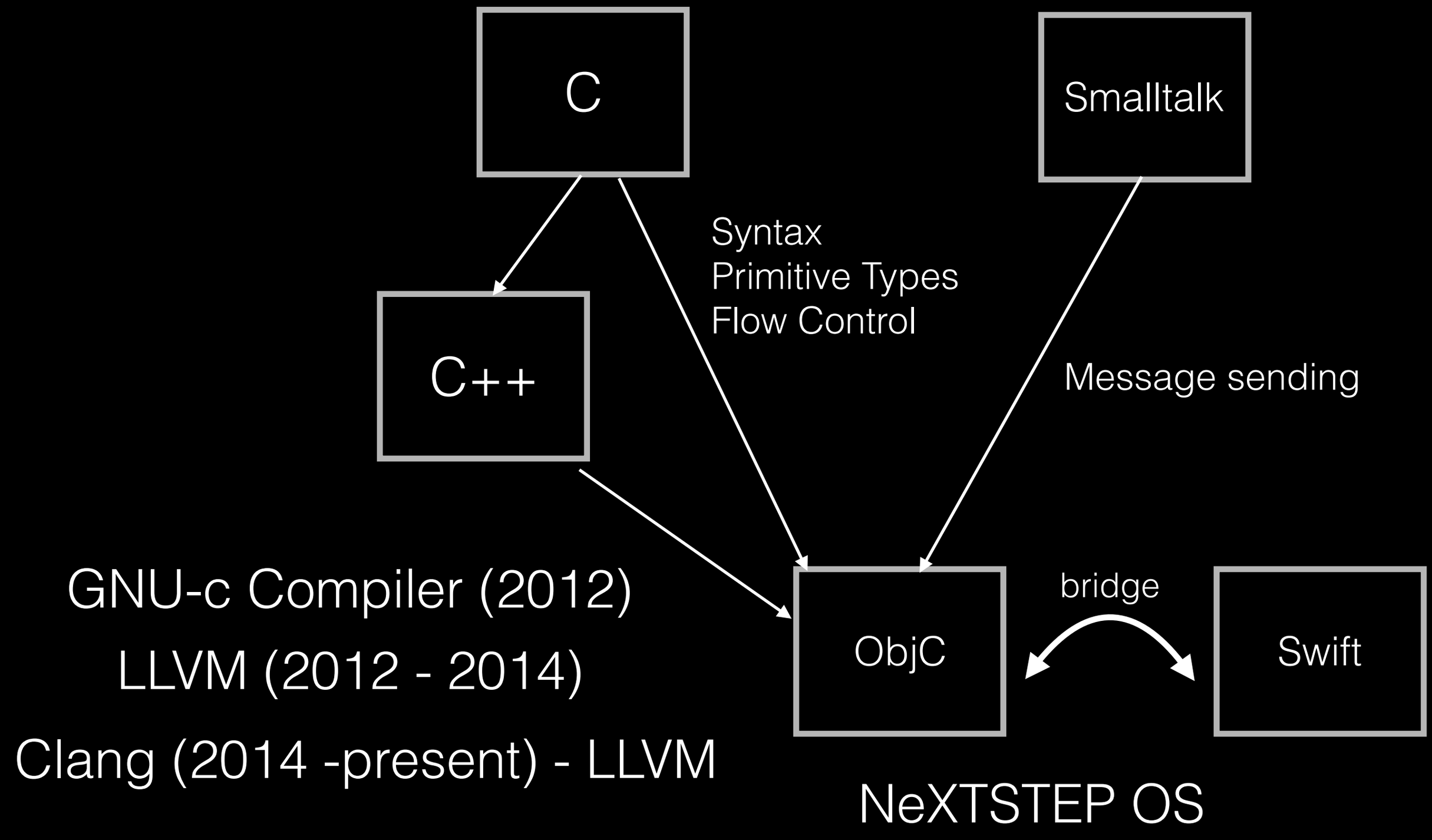
Day 1 - ObjC Syntax, Semantics and Caveats

1. ObjC
2. Cocoa
3. Namespaces
4. .h and .m files
5. Primitive Types
6. Messaging
7. Constructors
8. Reference Counting
9. NSObject
10. Nil and Null
11. Classes
12. Properties
13. Class Extensions and Categories
14. Protocols
15. Class Collections

Useful Sites

- Ray Wenderlich - http://www.raywenderlich.com
- NSHipster - http://nshipster.com
- objc.io - http://www.objc.io
- Apple Docs - https://developer.apple.com/library/ios/navigation/
- Apple Member Center - Need to be registered developer

Day 1 - The Basics



GNU-c Compiler (2012)

LLVM (2012 - 2014)

Clang (2014 -present) - LLVM

NeXTSTEP OS

Code Example (1.0)

Cocoa and Cocoa Touch

Cocoa is the native API for the OS X operating system

Cocoa Touch is the native API for the iOS operating system (Cocoa light)

It includes different graphical controls, support for gestures and a rich set of animation API’s

Namespaces achieved with Prefixes

Apple reserves the right to two letter namespaces

E.g.

NS (Next Step), UI (User Interface), CA (Core Animation), CG (Core Graphics), CF (Core Foundation) ……

USE Three Letter Prefixes

Interface files and implementation files

- A class' interface is defined in a '.h' file. This is what is publicly accessible.
- The class' actual implementation of that interface is done in a .m file

ELDPerson.h

```
@interface ELDPerson : NSObject

 @property (assign)  CGFloat weight;

 @property (assign)  CGFloat height;

 - (void) run;
- (void) walk;


@end
```

ELDPerson.m

```
#import

@implementation ELDPerson
    {
    BOOL _isRunning;
    }

- (void) run
    {
    self.weight - -;
     _isRunning = YES;
    }

@end
```

Primitive Types

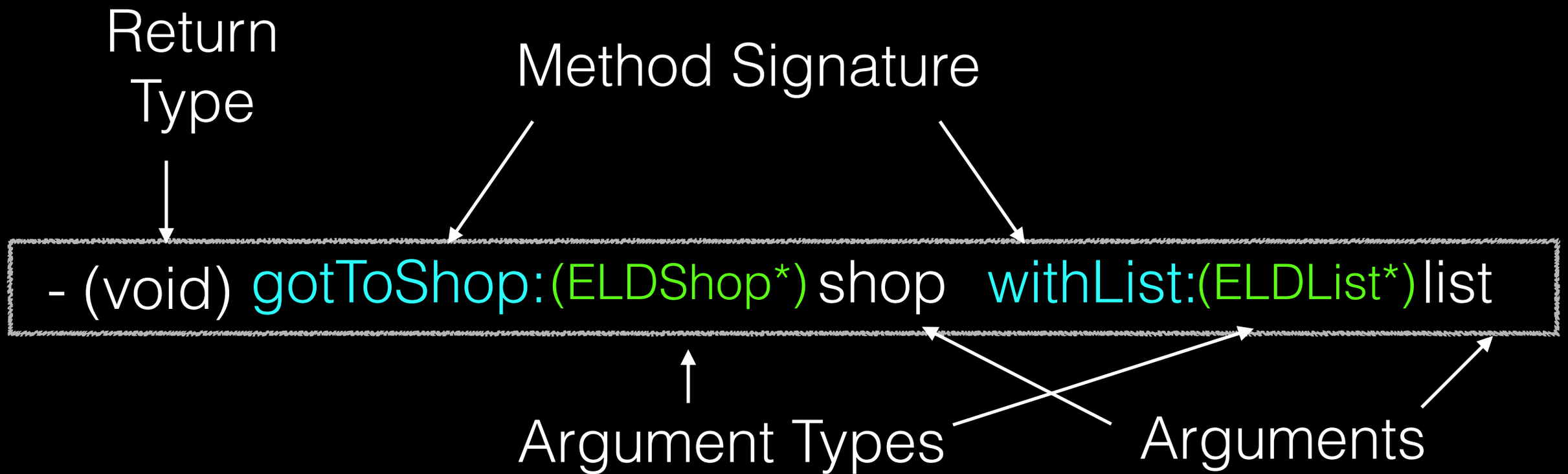
C primitives

- void (empty data type) -> not void*
- BOOL (YES/NO)
- int
- char
- unsigned char
- long
- float
- int8_t
- uint8_t
- int16_t
- uint16_t
- float
- double

Obj-C primitives

- NSInteger
- NSUInteger
- CGFloat

Method Signatures



The Actual Method: gotToShop: withList:

JAVA: public void goToShopWithList(ELDShop* shop,ELDList* list)

Different Method: - (void) gotToShop: (ELDShop*) shop

Invoking Methods

```
ELDPerson* person;
ELDShop* shop;
ELDList* list;
```

```
....          (constructors)
.....
[person   gotToShop: shop   withList: list];
```

this is not the usual method of invocation used

or

```
[person performSelector:@selector(gotToShop: withList: )
                withObjects:@[shop, list] ]
```

Messages

The object-orientated model is based on message passing

You DON’T call a
method on an object

You send a message
to an object

The object is dynamically resolved at runtime.
i.e. we can tell anything to do anything.

Ability is the name of the game (type don’t matter)

Obj-C is highly reflective. Can send any method to anything. Can ask anything if it can parse any method.

```
if ([cat  respondsToSelector:@selector(bark)])
   {
   [ cat performSelector:@selector(bark)];
                  OR
   [cat bark];
   }
```

Exercise

Create an interface for a person class. You just need to create the method signatures.
A method for earningMoney (returns nothing).
A method for going to the shop and spending a certain amount of money at the shop. The method will return nothing and will take as parameters: a shop object and an amount to spend.
A method to check if the person is broke that returns a boolean and takes no parameters.

Object Construction

Alloc

Allocates the required memory for your new object

Init

Initializes the memory that has been allocated to your object

```
ELDPerson* person = [[ELDPerson alloc] init];
```

new = alloc & init

```
ELDPerson* person = [ELDPerson new];
```

Constructors = init + …….

```
- (id) init
- (instanceType) initWithName:(NSString*) name
- (instanceType) initWithFirstName: (NSString*) firstName
                           lastName: (NSString*) lastName
- (instanceType) initWithAge: (NSUInteger) age
```

implementation

```
- (instanceType) init
{
self = [super init];
if (self)
      {
      ..
      ..
      }
   return (self);
}
```

```
- (instanceType) initWithName: (NSString*)name
{
self = [self init];
if (self)
      {
        name = name
      _name = name
      }
   return (self);
}
```

Convenience Constructors

Factory, Class Methods

[NSArray array]

[NSString stringWithFormat: …..]

[UIImage imageNamed: …..]

[NSNumber numberWithBool: …..]

Exercise

Create two initializers for the person, with implementation. One takes as a parameter a salary. One that takes as parameters a salary and initial balance.

Reference Counting

	Batman's reference count
batman= [ELDPerson new]	0 —> 1
robin.bestFriend = batman strong reference	1 —> 2
littleJapaneseGirl.favouriteHero = batman weak reference	2 —> 2
[superHeros addObject:batman]	2 —> 3
robin.bestFriend = nil	3 —> 2
[superHeros removeObject:batman]	2 —> 1

NSObject (gaia)

- NSObject is the mother of all objects
- The secret weapon of Objective-C
- All objects inherit from it (defined by its .h file and the one implemented by its .m) and as a result inherit very powerful characteristics.

https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSObject_Class/index.html

Exercise

Convert the person class into an actual class with a .m and .h file. Create a method that prints the amount of money the person has. Comment out the go to shop method. (you can create a shop class if you prefer)

clang -framework Foundation -fobjc-arc main.m ELDPerson.m

./a.out

Homework Exercise 1

Part 1

Create a pizza restaurant. There should be a restaurant object, a pizza object and a couple of pizza toppings. The pizza should have several constructors. A pizza should be able to calculate its own price based on the toppings it has and display it when the price method is called.

Part 2

Create a proxy to the pizza object. This Proxy will add an extra R10 onto the price.

nil, NSNull and NULL

In C, 0 is equivalent to nothing for primitives and NULL is a nothing pointer (0 in a pointer context)

nil is a pointer to nothing (ObjC)

NSNull is an actual object

[NSNull null] -> Singleton

nil will respond to any message sent to it with nil

Code Example

People and their cars

BOB	JENNY	MO
Ford Fiesta	-	Honda Civic

```
bob.car = @“Ford Fiesta”
jenny.car = nil;
mo.car = @“Honda Civic”

[jenny.car drive];
cars = [NSMutableArray array]
if (bob.car != nil)
  {
  [cars addObject:bob.car];
  }
  else
  {
  [cars addObject:[NSNull null]];
  }
```

```
.......


for (id car in cars)
  {
  if (car == [NSNull null])
    {
    ....
    }
  else
    {
    .....
    }
  }
```

Classes are Actual Objects

- Each class exists as a singleton and is a first class object
- You can treat classes likes objects.
- [NSString class]
- You can add instance variables through statics

Properties

ELPerson.h

```
@property (nonatomic, strong) NSString* name;
@property (nonatomic, weak) ELDPerson*favouriteSuperHero;
@property (assign, readwrite) NSUInteger age;
@property (assign, readonly) CGFloat bmi;
@property (getter=isFinished) BOOL finished;
```

Getters and setters are automatically synthesized for you.

Syntax:
getter: name of property
setter: set+NameOfProperty

```
- (NSString*) name;
- (void) setName:(NSString*)name
```

Writing your own Getters and Setters

Underscored variable is created for you when using the synthesized getters and setters. When writing your own you need to create the private variable

```
- (NSString*) name
{
return (_name)
}
```

```
- (void) setName: (NSString*)name
{
_name = name;
}
```

Can use @synthesize to have the generated methods use a different variable

```
@synthesize name = _firstName
```

Properties

The Dot Notation

The dot notation is their for the purpose of helping developers to differentiate between methods and properties (under the covers it is just a method invocation NOTHING MORE).

is the same as

theName = person.name; → theName = [person name];

person.name = @“Bob”; → [person setName:@“Bob”];

Privacy and Definition

1.

ELDPerson.h

```
@interface ELDPerson : NSObject

@property (assign) CGFloat weight;

@end
```

2.

ELDPerson.h

```
@interface ELDPerson : NSObject
    {
    BOOL _isRunning;
    }

@end
```

3.

ELDPerson.m

```
#import ELDPerson.h

@interface ELDPerson ()

@property (assign) CGFloat weight;

@end

@implementation ELDPerson

@end
```

4.

ELDPerson.m

```
#import ELDPErson.h

@implementation ELDPerson
    {
    BOOL _isRunning;
    }

@end
```

Spot the mistake

```
@implementation ELDPerson
   {
   CGFloat _weight;
   CGFloat _height;
   CGFloat _bmi;
   }

-  (instanceType) init
   {
   self = [super init];
   if (self)
      {
      _height = 180;
      _weight = 80
      _bmi = 180 / (80*80);
      }
   return (self);
   }

-  (void) setHeight: (CGFloat)height
   {
   _height = height;
   _bmi = _height / (self.weight*_weight);
   }
```

```
- (CGFloat) height
      {
      return (_height)
      }

-  (void) setWeight: (CGFloat)weight
   {
   _weight = weight;
   _bmi = self.height /(_weight*_weight);
   }

-  (CGFloat) bmi
   {
   return (_bmi)
   }

- (void) eatCake
      {
      _weight++;
      }
```

Exercise

Add the properties of salary and balance to the person object. Create a custom setter for the balance that prevents the balance from ever being below the salary amount. (Will you then need a custom setter for the salary?)

Class Extensions and Categories

Categories are for extending the behavior of existing classes

```
ELDPerson (MyCategory)
```

An Extension is a special case of a category. (An empty category).You can add properties and instance variables. Use extensions for private methods

```
ELDPerson (MyCategory)
```

Protocols

Protocols define messaging Contracts (similar to interfaces in Java)

Independence of any specific class. They allow anonymity

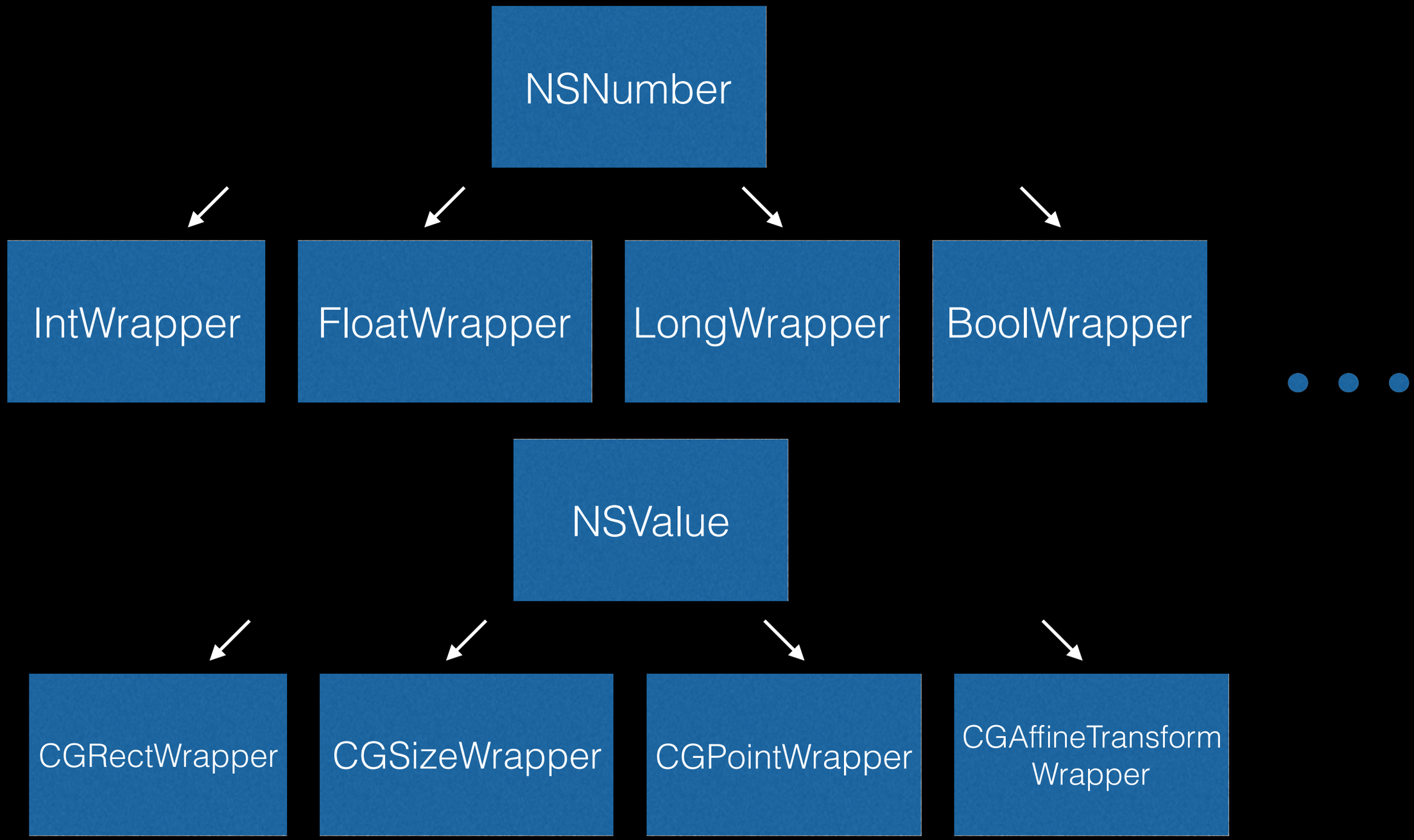
```
@protocol ELDTableCellDisplaying

@required
-  (NSString*) displayName;

@optional
- (NSString*) detailText;

@end
```

```
@interface ELDTableCell: UITableViewCell

@property (nonatomic, weak) id <ELDTableViewCellDisplaying> contentItem

@end
```

```
@interface ELDPerson: NSObject <ELDTableCellDisplaying>
```

```
- (void) setContentItem: (id <ELDTableViewCellDisplaying>) contentItem
   {
   _contentItem = contentItem
   self.textLabel = [_contentItem displayName];
   if ([_contentItem respondsToSelector: @selector(detailText)])
       {
       self.detailTextLabel = [_contentItem detailText];
       }
   }
```

NSNumber & NSValue,



Homework 2

Part 1

Create a subclass for the pizza class, called healthy pizza, this pizza can calculate its nutritional output. Move the price calculation into a class extension so that its private but can be used by the subclass.

Part 2

Create a category on NSObject allowing you to query an NSObject whether or not its a pizza. NSObject should return NO while a pizza should return YES. Also create a method on pizza that adds a method called ‘cookInOven’. Create a private class extension that allows pizza subclasses to see this method

Part 3

Create a …Cooking protocol. Make the restaurant class adhere to this protocol. Have a property linking the restaurant to this pizza. Use the method on pizza called cookInOven to internally call the methods declared by the protocol on the restaurant. (check that the restaurant has those methods before invoking them)

Protocol Methods:

- (void) willStartCookingPizza:(..Pizza*)pizza;
- (void) didStartCookingPizza:(..Pizza*)pizza;
- (void) willFinishCookingPizza:(..Pizza*)pizza;
- (void) didFinishCookingPizza:(..Pizza*)pizza;

References

- About Objective-C, https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011210-CH1-SW1
- NSObject Class Reference, https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSObject_Class/index.html
- Concepts in Objective-C programming, https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/Introduction/Introduction.html#//apple_ref/doc/uid/TP40010810-CH1-SW1
- Programming with Objective-C, https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/ErrorHandling/ErrorHandling.html
- Objective-C Succinctly: Categories and Extensions, http://code.tutsplus.com/tutorials/objective-c-succinctly-categories-and-extensions--mobile-22016
- NSValue, http://nshipster.com/nsvalue/
- Objective C, https://en.wikipedia.org/wiki/Objective-C
- NeXTSTEP, https://en.wikipedia.org/wiki/NeXTSTEP
- Equality, http://nshipster.com/equality/