OBJECTIVE- C
MEMORY MANAGEMENT

For a greater good!

THE RETAIN COUNT

- Every object keep tracks of it's retain count
- The methods init and retain increments the retain count
- The method release decrements the counter
- When an the retain count becomes 0 the object is deallocated
  - No matter if someone still has a pointer referencing it
  - It's dealloc method is called right before to do cleanup

```
NSString string = [[NSString alloc] init];  // 1
[string retain];                            // 2
[string release];                           // 1
```

RULES OF MEMORY MANAGEMENT

- You own any object you create
- You can take ownership of an object using retain
- When you no longer need it, you must relinquish ownership of an object you own
- You must not relinquish ownership of an object you do not own

PROPERTIES
ASSIGN, RETAIN OR COPY?

- Assign does not increment the retain counter
- Retain increments the counter
- Copy creates another object with retain count 1

```
@property (nonatomic, retain) UIWindow *window;
```

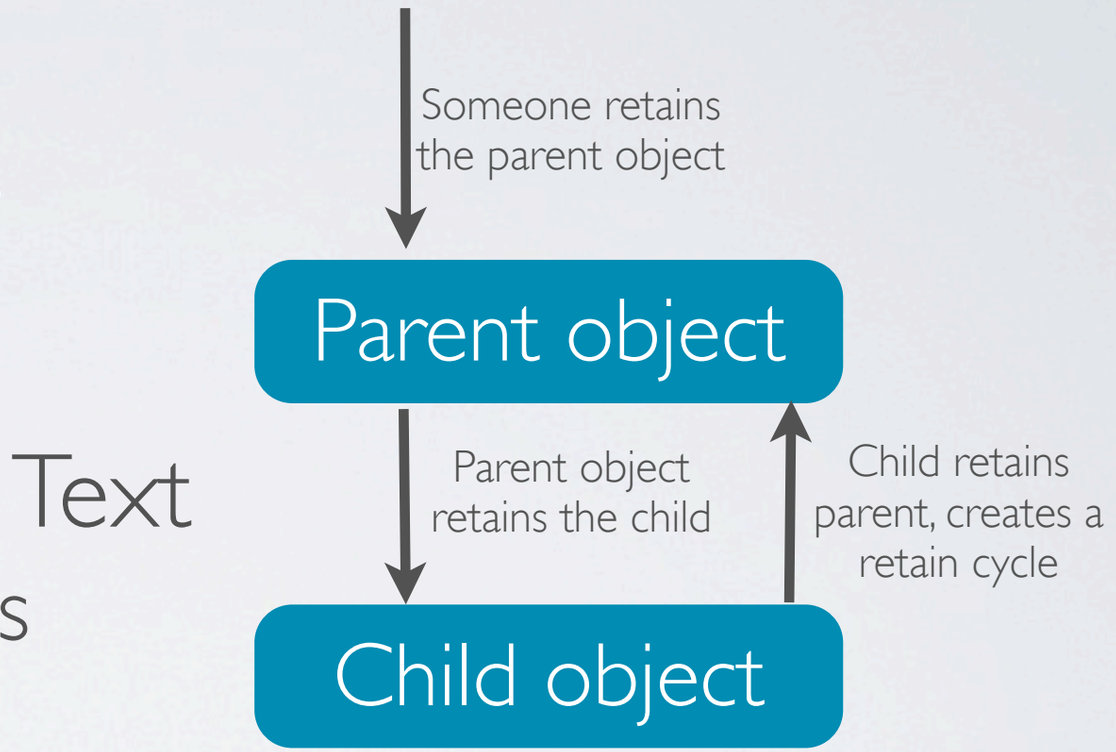
```
- (void)setWindow:(UIWindow *)window
{
    [window retain];
    _window = window;
}
```

RETAIN CYCLES

- When dealloc can never be called because of circular ownership

- Many many cases where this can happen in some form

- Fixed using the __weak keyword




```
@class Child;
@interface Parent : NSObject
{
    Child *child;
}
@end

@interface Child : NSObject
{
    __weak Parent *parent;
}
@end
```

RETAIN CYCLES IN BLOCKS

Retain cycle example:

```objc
@implementation TopScoreScene

- (id)init
{
    self = [super init];
    if (self) {
        _networkManager = [[NetworkManager alloc] init];
        [_networkManager onDataReceived: ^(NSArray *data) {
            [self displayScores:data];
        }];
    }
    return self;
}

@end
```

```objc
@implementation TopScoreScene

- (id)init
{
    self = [super init];
    if (self) {
        _networkManager = [[NetworkManager alloc] init];
        __weak TopScoreScene *weakSelf = self;
        [_networkManager onDataReceived: ^(NSArray *data) {
            [weakSelf displayScores:data];
        }];
    }
    return self;
}

@end
```

- Blocks can cause nasty retain cycles
- Define weakSelf using __weak