Objective-C 2.0 @Compiler Directives Cheat Sheet

Derived from an article by Steffen Itterheim (learn-cocos2d.com) found here and placed into a cheat sheet by Johann Dowa (maniacdev.com)

@class - Declares class as known without having to import the class' header file.

```
@class ClassName;
```

Getting class object by name:

```
// ERROR: this doesn't work!
Class c = @class(ClassName);
```

Instead use:

```
Class c = [ClassName class];
```

@end marks end of the class, protocol or interface declaration

@protocol @required @optional

Marks the start of a protocol declaration.

```
@protocol ProtocolName <aProtocol, anotherProtocol>
```

Get a protocol object by name:

```
Protocol *aProtocol = @protocol(ProtocolName);
```

@required - Methods following are required (default).

@optional - Methods following are optional.

Classes making use of the protocol must test Optional protocol methods for existence:

```
[object respondsToSelector:@selector(optionalProtocolMethod)];
```

@interface @public @package @protected @private @property

Marks the start of a class or category declaration. Objective-C classes should derive from NSObject directly or indirectly. Use @interface to declare that the class conforms to protocols.

```
@interface ClassName : SuperClassName <aProtocol, anotherProtocol> {
@public
  // instance variables
@package
  // instance variables
@protected
  // instance variables
@private
  // instance variables
}
// property declarations
@property (atomic, readwrite, assign) id aProperty;
// public instance and/or class method declarations
@end
```

Category declaration - Adjective-C category cannot add instance variables. Can to conform to (additional) protocols. CategoryName can be omitted if in implementation file making methods "private".

```
@interface ClassName (CategoryName) <aProtocol, anotherProtocol>
```

Dependent Directives:

@public – Declares instance variables after @public directive as publicly accessible. Read and modified with pointer notation:

```
someObject->aPublicVariable = 10;
```

@package – Declares instance variables after @package directive as public inside the framework that defined the class, but private outside the framework. Applies only to 64-bit systems, on 32-bit systems @package has the same meaning as @public.

@protected (default) – Declares instance variables after @protected directive as accessible only to the class and derived classes.

@private – Declares the instance variables following the @private directive as private to the class. Not even derived classes can access private instance variables.

@property – Declares a property which accessible with dot notation. @property can be followed by optional brackets within which property modifiers specify the exact behavior of the property. Property modifiers: readwrite (default), readonly – Generate both setter & getter methods (readwrite), or only the getter method (readonly).

assign (default), retain, copy – For properties that can safely cast to id. Assign assigns passed value – retain sends release to an existing instance variable, sends retain to the new object, assigns the retained object to the instance variable – copy sends release to the existing instance variable, sends copy to the new object, assigns the copied object to the instance variable. In latter two cases you are responsible for sending release (or assigning nil) to the property on dealloc.

atomic (default), nonatomic – Atomic properties are thread-safe. Nonatomic properties are not thread-safe. Nonatomic property access is faster than atomic and often used in single-threaded apps, or cases where you're absolutely sure the property will only be accessed from one thread.

weak (default), strong – Available if automatic reference counting (ARC) is enabled. The keyword strong is synonymous to retain, while weak is synonymous to assign, except a weak property is set to nil if instance is deallocated.

@selector - Returns the selector type SEL of the given Objective-C method. Generates compiler warning if the method isn't declared or doesn't exist.

```
-(void) aMethod {
    SEL aMethodSelector = @selector(aMethod);
    [self performSelector:aMethodSelector]; }
```

@implementation @synthesize @dynamic

Marks start of a class' or category implementation.

Class implementation:

```
@implementation ClassName
@synthesize aProperty, bProperty;
@synthesize cProperty=instanceVariableName;
@dynamic anotherProperty;
// method implementations
@end
```

Category implementation:

```
@implementation ClassName (CategoryName)
@synthesize aProperty, bProperty;
@synthesize cProperty=instanceVariableName;
@dynamic anotherProperty, banotherProperty;
// method implementations
@end
```

Dependent Directives:

@synthesize – Generate setter and getter methods for a comma separated propert y list according to property modifiers. If instance variable is not named exactly like @property, you can specify instance variable name after the equals sign.

@dynamic –Tells the compiler the setter and getter methods for the given (comma separated) properties are implemented manually, or dynamically at runtime. Accessing a dynamic property will not generate a compiler warning, even if the getter/setter is not implemented. You will want to use @dynamic in cases where property getter and setter methods perform custom code.

@end – Marks end of class implementation.

@synchronized - Encapsulates code in mutex lock ensuring that the block of code and locked object are only accessed by one thread at a time.

```
-(void) aMethodWithObject:(id)object {
   @synchronized(object) {
      // code that works with locked object
  }
}
```

@"string" - Declares a constant NSString object. Does not need to be retained or released.

```
NSString* str = @"This is a constant string.";
NSUInteger strLength = [@"This is legal!" length];
```

@throw @try @catch @finally - *Used for handling and throwing exceptions.*

Throwing and Handling exceptions:

```
@try {
   // code that might throw an exception
   NSException *exception =
      [NSException
exceptionWithName:@"ExampleException"
                              reason:@"In your face!"
                            userInfo:nil];
   @throw exception; }
@catch (CustomException *ce) {
   // CustomException-specific handling ... }
@catch (NSException *ne) {
   // generic NSException handling ...
   // re-throw the caught exception in a catch block:
   @throw; }
@finally  {
   // runs whether an exception occurred or not
}
```

@autoreleasepool - In an ARC (automatic reference counting) enabled about 6x faster than NSAutoreleasePool and used as a replacement. Avoid using a variable created in an @autoreleasepool after the autoreleasepool block.

```
@autoreleasepool {
      // code that creates temporary objects
 }
```

@encode - Returns the character string encoding of a type.

```
char *enc1 = @encode(int);  // enc1 = "i"
char *enc2 = @encode(id);  // enc2 = "@"
char *enc3 = @encode(@selector(aMethod));  // enc3 = ":"
// practical example:
 CGRect rect = CGRectMake(0, 0, 100, 100);
 NSValue *v = [NSValue value:&rect withObjCType:@encode(CGRect)];
```

@compatibility_alias - Sets alias name for an existing class. First parameter is the alias, second the actual class name.

```
@compatibility_alias AliasClassName ExistingClassName
```

After this you can use AliasClassName in place of ExistingClassName.

Please share this Cheat Sheet with your twitter followers by clicking here.