Objective-C Cheat Sheet

Modern Objective-C 2.0 · Apple Clang · ARC, Clang modules, lightweight generics, nullability
Irurueta Docs — Guides & References → Programming Languages → Objective-C Reference

CORE TYPES

`id`	Any object (already a pointer)
`Class`	Class object; `[obj class]`
`SEL`	`@selector(setName:)`
`IMP`	`id (*)(id self, SEL _cmd, ...)`
`instancetype`	Instance of receiving class — use for `init`/factories
`BOOL`	YES/NO. Never == YES
`NSInteger` `NSUInteger`	Width-adaptive. Log via `(long)`
`CGFloat`	Width-adaptive float
`nil` / `Nil`	Null object / null Class
`NULL`	Null C pointer
`NSNull`	`[NSNull null]` — inside collections
`NSNotFound`	"no such index" sentinel

Messages to nil are legal → 0 / nil / zeroed struct.

CLASS SKELETON

```
// Person.h
#import <Foundation/Foundation.h>
@class Customer;  // fwd-decl in headers

NS_ASSUME_NONNULL_BEGIN
@interface Person : NSObject
@property (nonatomic, copy) NSString *name;
- (instancetype)initWithName:(NSString *)n
      NS_DESIGNATED_INITIALIZER;
- (instancetype)init NS_UNAVAILABLE;
+ (instancetype)personWithName:(NSString *)n;
@end
NS_ASSUME_NONNULL_END
// Person.m
@interface Person ()   // class extension
@property (nonatomic) BOOL dirty;
@end

@implementation Person {
    NSMutableArray *_history;  // private ivar
}
- (instancetype)initWithName:(NSString *)n {
    self = [super init];       // FIRST
    if (self) { _name = [n copy]; }
    return self;               // may be nil
}
+ (instancetype)personWithName:(NSString *)n {
    return [[self alloc] initWithName:n];  // self, not Person
}
@end
```

Ivar visibility: `@private` `@protected` (default) `@public` `@package`.

METHODS & MESSAGES

```
- (void)insertObject:(id)o atIndex:(NSUInteger)i;
//   selector: insertObject:atIndex:
+ (instancetype)shape;   // class method

[receiver message];
[receiver insertObject:o atIndex:0];
[[Person alloc] initWithName:@"Ada"];

person.name = @"Ada";  // [person setName:]
NSString *n = person.name; // [person name]

[obj respondsToSelector:@selector(draw)];
[obj isKindOfClass:[NSString class]];
[obj conformsToProtocol:@protocol(P)];
[obj performSelector:sel withObject:x];
NSStringFromSelector(sel);
NSStringFromClass([obj class]);
```

No overloading by type — the selector IS the name.

@PROPERTY ATTRIBUTES

`strong`	Owns it. Default for objects (ARC)
`weak`	No ownership; **auto-nils**. Delegates, back-refs
`copy`	**Required** for NSString/NSArray/NSDictionary/blocks
`assign`	Scalars & structs. Default for non-objects
`unsafe_unretained`	No ownership; **dangles**
`nonatomic`	No lock. Use it — atomic is default
`atomic`	Indivisible accessor. **NOT thread safety**
`readonly`	Getter only; redeclare readwrite in the ext.
`getter=is…`	Booleans: `getter=isEnabled`
`class`	Class property; write accessors yourself
`nullable` `nonnull`	Nullability annotations

```
@synthesize name = _name; // if BOTH written
@dynamic identifier;      // runtime-provided
```

Use `_ivar` only in `init`/`dealloc`; `self.prop` elsewhere.

CATEGORIES · EXTENSIONS · PROTOCOLS

```
// Category — add to a class you don't own
@interface NSString (MyValidation)
- (BOOL)my_isValidEmail;  // ALWAYS prefix
@end

// Class extension — private iface, .m only
@interface Person () <NSURLSessionDelegate>
@property (nonatomic, readwrite) double progress;
@end

// Protocol
@protocol MyDelegate <NSObject>
@required - (void)didFinish:(id)sender;
@optional - (void)didProgress:(double)p;
@end

@property (nonatomic, weak) id<MyDelegate> delegate;
if ([d respondsToSelector:@selector(didProgress:)])
    { ... }                          // @optional
```

Category override order is undefined; there is no `super`. `poseAsClass:` is removed.

No ivars in a category → use associated objects.

BLOCKS

```
// returnType (^name)(paramTypes)
void (^log)(NSString *) = ^(NSString *m) {
    NSLog(@"%@", m);
};
log(@"hi");

typedef void (^Handler)(NSData * _Nullable,
                        NSError *
_Nullable);
@property (nonatomic, copy) Handler done;

__block NSInteger n = 0;  // shared, writable
^{ n++; }();              // plain capture is const

// Retain cycle → weak/strong dance
__weak typeof(self) weakSelf = self;
self.done = ^(NSData *d, NSError *e) {
    __strong typeof(weakSelf) s = weakSelf;
    if (!s) { return; }
    [s render:d];
};
```

Calling a NULL block CRASHES: `if (b) b();`

No weakSelf for dispatch_async / animation / enumeration blocks.

LITERALS & SUBSCRIPTING

```
NSString     *s = @"text";
NSNumber     *i = @42;   // @YES @3.14 @'x'
NSNumber     *e = @(x * 2);  // expression
NSArray      *a = @[ @"x", @"y" ];
NSDictionary *d = @{ @"k": @"v" };

id item = a[0];  id v = d[@"k"];
mArray[0]    = @"new";
mDict[@"k"] = @"new";  // = nil REMOVES key
```

Literals reject nil. Collections can't hold nil → `[NSNull null]`.

STRINGS, NUMBERS & COLLECTIONS

```
[NSString stringWithFormat:@"%@ %ld", o, (long)n];
[a isEqualToString:b];   // NOT ==
[s hasPrefix:@"he"]; [s containsString:@"ell"];
[s substringToIndex:5]; [s uppercaseString];
[s componentsSeparatedByString:@", "];
[s localizedStandardCompare:t]; // user-visible sort
NSRange r = [s rangeOfString:@"x"];
if (r.location != NSNotFound) { ... }
[i integerValue]; [d doubleValue]; [b boolValue];
[NSValue valueWithRange:r]; // box a struct

arr.count; arr.firstObject; arr.lastObject;
[m addObject:o]; [m removeObjectAtIndex:0];
[arr containsObject:o]; [arr indexOfObject:o];
[set unionSet:other]; [set intersectsSet:other];
[arr sortedArrayUsingComparator:
  ^NSComparisonResult(id a, id b){ return [a compare:b]; }];
[arr filteredArrayUsingPredicate:[NSPredicate
  predicateWithFormat:@"age >= %d", 18]];

// Fast enumeration — never mutate while in it
for (NSString *x in arr) { ... }
for (NSString *k in dict) { id v = dict[k]; }
[arr enumerateObjectsUsingBlock:
  ^(id o, NSUInteger i, BOOL *stop){ *stop = YES; }];
```

Declare collection properties `copy`: mutable is-a immutable.

ARC & BRIDGING

`__strong`	Default; keeps it alive
`__weak`	No ownership; zeroed on dealloc
`__unsafe_unretained`	No ownership; dangles
`__autoreleasing`	`NSError **` out-parameters

```
// ARC ↔ Core Foundation
(__bridge CFStringRef)s       // borrow, no transfer
(__bridge_retained CFStringRef)s // you CFRelease
(__bridge_transfer NSString *)cf // ARC owns it
CFBridgingRetain(s) / CFBridgingRelease(cf)
// Create/Copy in the name → __bridge_transfer

@autoreleasepool { ... } // loops, new threads

- (void)dealloc {   // no [super dealloc]!
  [[NSNotificationCenter defaultCenter]
      removeObserver:self];
  [_timer invalidate]; CFRelease(_cf);
free(_buf);
}
```

ARC never breaks retain cycles. Owner strong → owned weak.

Pre-ARC: `retain`/`release`/`autorelease`; own what you `alloc` `new` `copy` `mutableCopy`.

ERRORS & EXCEPTIONS

```
// NSError = expected, recoverable failure
- (nullable NSData *)load:(NSURL *)u
                    error:(NSError **)error {
  if (bad) {
    if (error) {        // may be NULL!
      *error = [NSError
errorWithDomain:MyDomain
          code:MyCodeBad userInfo:@{
          NSLocalizedDescriptionKey: @"Bad URL.",
          NSUnderlyingErrorKey: inner}];
    }
    return nil;
  }
}
NSError *err = nil;
NSData *d = [self load:u error:&err];
if (!d) { ... }  // check the RETURN VALUE

typedef NS_ERROR_ENUM(MyDomain, MyCode) { ... };

// Exceptions = programmer errors. Don't catch.
@try { ... } @catch (NSException *e) { ... }
@finally { ... }
[NSException raise:NSInvalidArgumentException
            format:@"index %ld", (long)i];
[self doesNotRecognizeSelector:_cmd]; // abstract

NSParameterAssert(obj);  // out in Release
NSAssert(i < n, @"out of bounds");
os_log_error(lg, "failed: %{public}@", e);
```

GCD & CONCURRENCY

```
dispatch_async(dispatch_get_global_queue(
    QOS_CLASS_USER_INITIATED, 0), ^{
  id r = [self work];
  dispatch_async(dispatch_get_main_queue(), ^{
    [self render:r];     // ALL UI on main
  });
});

dispatch_queue_t q = dispatch_queue_create(
    "com.x.serial", DISPATCH_QUEUE_SERIAL);
dispatch_sync(q, ^{ ... }); // never on own queue
dispatch_barrier_async(cq, ^{ ... }); // excl. write

dispatch_group_t g = dispatch_group_create();
dispatch_group_enter(g); /* ... */
dispatch_group_leave(g);
dispatch_group_notify(g, main, ^{ ... });

dispatch_semaphore_t s =
dispatch_semaphore_create(4);
dispatch_semaphore_wait(s,
DISPATCH_TIME_FOREVER);
dispatch_semaphore_signal(s);

static dispatch_once_t once;   // singleton
dispatch_once(&once, ^{
    shared = [[self alloc] init]; });
@synchronized (token) { ... } // recursive mutex
```

Mutable Foundation collections are NOT thread-safe (NSCache is).

KVC / KVO

```
[o valueForKey:@"name"];
[o setValue:@"Ada" forKey:@"name"];
[o valueForKeyPath:@"address.city"];
[people valueForKey:@"name"];        // map
[people valueForKeyPath:@"@sum.age"]; // @count @avg
static void * const Ctx = (void *)&Ctx;
[o addObserver:self forKeyPath:@"progress"
  options:NSKeyValueObservingOptionNew
context:Ctx];

- (void)observeValueForKeyPath:(NSString *)kp
      ofObject:(id)o change:(NSDictionary *)ch
       context:(void *)c {
  if (c == Ctx) { id v =
ch[NSKeyValueChangeNewKey]; }
  else [super observeValueForKeyPath:kp
ofObject:o
                              change:ch
context:c];
}
[o removeObserver:self forKeyPath:@"progress"
   context:Ctx];

[self willChangeValueForKey:@"k"]; // manual
+ (NSSet *)keyPathsForValuesAffectingFullName;
```

Not removing an observer before dealloc CRASHES.

RUNTIME

```
#import <objc/runtime.h>
Class c = objc_getClass("NSString");
object_getClass(c);   // the METAclass
Method m = class_getInstanceMethod(c, sel);
class_addMethod(c, sel, (IMP)fn, "v@:");
method_exchangeImplementations(a, b); // swizzle
Method *l = class_copyMethodList(c, &n);
free(l);
IMP imp = [obj methodForSelector:sel];
// @encode: v void  @ id  : SEL  i int  d double  B BOOL
// - (void)setAge:(NSInteger)a   →   "v@:q"

// Forwarding chain, in order:
+ (BOOL)resolveInstanceMethod:(SEL)sel;
- (id)forwardingTargetForSelector:(SEL)sel;
- (NSMethodSignature
*)methodSignatureForSelector:;
- (void)forwardInvocation:(NSInvocation *)inv;
- (void)doesNotRecognizeSelector:(SEL)s; // raises
```

Never ship a library that swizzles.

GENERICS, NULLABILITY, SWIFT

```
NS_ASSUME_NONNULL_BEGIN  // wrap EVERY header
NSArray<NSString *> *names;
NSDictionary<NSString *, NSNumber *> *ages;
@interface Box<__covariant ObjectType> : NSObject
- (__kindof UIView *)viewWithTag:(NSInteger)t;
- (nullable NSString *)find:(NSString *)k;
NS_ASSUME_NONNULL_END

if (@available(iOS 15.0, macOS 12.0, *)) { ... }
- (void)m API_AVAILABLE(ios(15.0));
- (void)o API_DEPRECATED("Use -m", ios(9.0,15.0));
+ (instancetype)with:(id)x
NS_SWIFT_NAME(init(_:));
- (void)raw NS_REFINED_FOR_SWIFT; // → __raw
- (void)old NS_SWIFT_UNAVAILABLE("Use m()");
```

`nonnull T *`	Swift T
`nullable T *`	Swift T?
`unaudited`	Swift T! — avoid
`BOOL+NSError**`	Swift throws
`NS_ENUM`	enum
`NS_OPTIONS`	OptionSet
`instancetype`	Self

ObjC→Swift: bridging header. Swift→ObjC: `#import "App-Swift.h"` + `@objc`.

ENUMS, MODULES, CONVENTIONS

```
typedef NS_ENUM(NSInteger, MyState) {
    MyStateIdle, MyStateRunning };
typedef NS_OPTIONS(NSUInteger, MyOpts) {
    MyOptsNone = 0, MyOptsWrap = 1 << 0 };
// switch with NO default: → -Wswitch warns

extern NSString * const MyDefaultAgent; // not const NSString*

#import "X.h"   // never twice; no guards needed
@import Foundation;  // Clang module
#pragma mark - Lifecycle
```

`Prefix`	3+ letters (2 are Apple's); my_ on category methods
`Getter`	-name, never -getName
`Delegate`	-obj:didThing:, should/will/did
`Mutate`	sort in place vs sortedArray copy
`Notif.`	MyClassDidFinishNotification

BUILD, TEST & DEBUG

```
# macOS / Apple Clang
clang -fobjc-arc -fmodules -Wall -Wextra \
      -framework Foundation main.m -o app

# Linux / GNUstep
clang `gnustep-config --objc-flags` main.m \
      -o app `gnustep-config --base-libs`
# Key flags
-fobjc-arc / -fno-objc-arc  # ARC, per file
-fmodules                   # enables @import
-ObjC                       # static libs w/ categories!
-Werror  -fsanitize=address|thread|undefined

xcodebuild -workspace A.xcworkspace -scheme A build
xcodebuild test -scheme A \
  -only-testing:ATests/PersonTests
xcodebuild analyze -scheme A
```

```
// XCTest
- (void)testThing {
  XCTAssertEqualObjects(a, b);  // OBJECTS
  XCTAssertEqual(i, 36);        // scalars
  XCTAssertNil(x); XCTAssertThrows(e);
}
XCTestExpectation *ex =
  [self expectationWithDescription:@"d"];
[ex fulfill];
[self waitForExpectationsWithTimeout:5.0
handler:nil];
```

lldb: `po` `obj` · `bt` · `breakpoint set -E objc` (break where thrown).