Fundation(framework)

three fundamental categories of classes and protocols
1. the root class and related protocols;
2. value classes, wrapper for a data type (string, number, date, binary data);
3. collection classes, the instance of collection class manages a group of objects.

App & Object

app is a network of cooperating objects at runtime.
these objects communicate with each other to get the work of app done.
each object has at least one responsibility and is connected to at least one other object.

Allocate & Initialize

When you allocate an object, the Objective-C runtime allocates enough memory for the object from application virtual memory. In addition to allocating memory, the runtime does a few other things during allocation, such as setting all instance variables to zero.

Initialization sets an object's initial state—that is, its instance variables and properties—to reasonable values and then returns the object. The purpose of initialization is to return a usable object.

Create an object

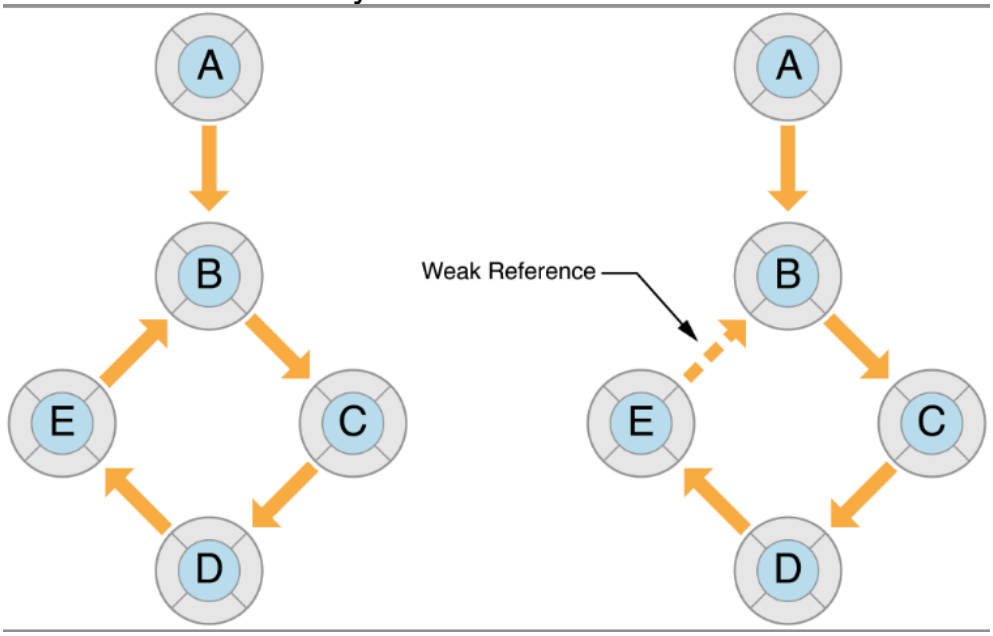
1. by allocating and initializing;
2. by calling a class factory method(a class method whose purpose is to allocate, initialize, and return an instance of itself.).

Reference

the reference an object has are either one-to-one or (via collection) one-to-many.
reference between objects are either strong or weak.
1. a strong reference
it indicates ownership, the referring object owns the referred object.
2. a weak reference
it implies that the referring object does not own the referred object.
The lifetime of an object is determined by how many strong references there are to it.
An object is not freed as long as there is a strong reference to it.
For variables, use `__weak` qualifier before the variable name;

For properties, use the weak option.

Memory Leaks
Each of objects are connected with strong reference => No objects will freed => Leak memory.



Inmutable & Mutable
1. the instances of immutable classes usually are ones that encapsulate collections of discrete or buffered values.
for example, arrays and strings.
These classes usually have mutable variants with “Mutable” in their names. For example, there is the NSString class (immutable) and the NSMutableString class.
Note that some immutable objects that encapsulate discrete values, such as NSNumber or NSDate, do not have mutable class variants.

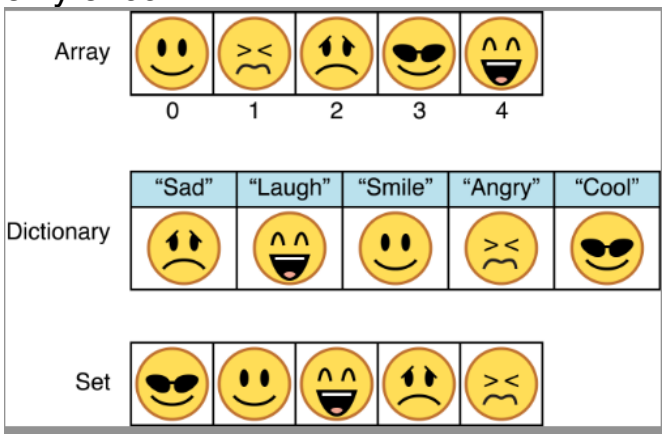
Value Object
When you declare value classes as properties, you should use `copy` option.

NSNumber
create NSNumber objects representing unsigned integers, long integers, long long integers, and float values by appending the letters “U”, “L”, “LL”, and “F”.

```
NSNumber *myFloatValue = @3.2F
```

Collection

an object that contained otter objects in a certain way and allow clients to access those objects.
1. NSArray & NSMutableArray
an array is an ordered collection of objects.
the object is accessed by specifying its position(index).
the first element in an array is at index 0(zero).
2. NSDictionary & NSMutableDictionary
key-value pairs
key: a unique identifier, usually a string, access by key
value: an object you want to store.
3. NSSet & NSMutableSet
a set is an unordered collection of objects, with each object occurring only once.



Array
create an array

```
// Compose a static array of string objects
NSString *objs[3] = {@"One", @"Two", @"Three"};
// Create an array object with the static array
NSArray *arrayOne = [NSArray arrayWithObjects:objs count:3];
// Create an array with a nil-terminated list of objects
NSArray *arrayTwo = [[NSArray alloc] initWithObjects:@"One", @"Two", @"Three", nil];
```

```
NSArray *myArray = @[ @"Hello World", @67, [NSDate date] ];
```

access objects in an array(`objectAtIndex:`)

```
NSString *theString = [arrayTwo objectAtIndex:1]; // returns second object in array
```

do something(compare) with each objects in an array(enumeration)
1. fast enumeration

```
for (type variable in array){ /* inspect variable, do something with it */ }
```

```
NSArray *myArray = // get array
for (NSString *cityName in myArray) {
    if ([cityName isEqualToString:@"Cupertino"]) {
        NSLog(@"We're near the mothership!");
        break;
    }
}
```

2. enumeration with a block

`enumerateObjectsUsingBlock:`

```
NSArray *myArray = // get array
[myArray enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    if ([obj isEqual:@"Cupertino"]) {
        NSLog(@"We're near the mothership!");
        *stop = YES;
    }
}];
```

3. use of an NSEnumerator object

Managing mutable array

`addObject:`
`insertObject:atIndex:`
`removeObject:`
`removeObject:atIndex:`

Dictionary

create an dictionary

`dictionaryWithObjects:forKeys:`

pass in an array of objects and an array of keys, the keys are positionally matched their values.

(`dictionaryWithObjectsAndKeys:`)

specify the first object value and then its key, the second object value and then its key, and so on; you signal the end of this series of objects with nil.

```
// First create an array of keys and a complementary array of values
NSArray *keyArray = [NSArray arrayWithObjects:@"IssueDate", @"IssueName", @"IssueIcon", nil];
```

```
NSArray *valueArray = [NSArray arrayWithObjects:[NSDate date],
@"Numerology Today",
    self.currentIssueIcon, nil];
// Create a dictionary, passing in the key array and value array
NSDictionary *dictionaryOne = [NSDictionary
dictionaryWithObjects:valueArray forKeys:keyArray];
// Create a dictionary by alternating value and key and terminating with
nil
NSDictionary *dictionaryTwo = [[NSDictionary alloc]
initWithObjectsAndKeys:[NSDate date],
    @"IssueDate", @"Numerology Today", @"IssueName",
self.currentIssueIcon, @"IssueIcon", nil];
```

```
NSDictionary *myDictionary = @{
   @"name" : NSUserName(),
   @"date" : [NSDate date],
   @"processInfo" : [NSProcessInfo processInfo]
};
```

access objects in a dictionary

`objectForKey:`

```
NSDate *date = [dictionaryTwo objectForKey:@"IssueDate"];
```

```
NSString *theName = myDictionary[@"name"];
```

Managing mutable dictionary

`setObject:forKey:`

replaces any existing value for the given key.

`removeObjectForKey:`

```
NSMutableDictionary *mutableDict = [[NSMutableDictionary alloc] init];
mutableDict[@"name"] = @"John Doe";
```

Set

strore unordered objects

In Core Data,

when you declare a property for a to-many relationship, the property type should be NSSet or NSOrderedSet.

Sets are also important in native touch-event handling in the UIKit framework

Verify object Capabilities
1. whether it's an instance of a particular class or subclass
`isKindOfClass:`

```
static int sum = 0;
for (id item in myArray) {
    if ([item isKindOfClass:[NSNumber class]]) {
        int i = (int)[item intValue];
        sum += i;
    }
}
```

2. whether it's responds to a message
(`respondsToSelector:`)

```
if ([item respondsToSelector:@selector(setState:)]) {
    [item setState:[self.arcView.font isBold] ? NSOnState : NSOffState];
}
```

3. whether it' conforms to a protocol
(`conformsToProtocol:`)

```
- (void) setDelegate:(id __weak) obj {
    NSParameterAssert([obj conformsToProtocol:
        @protocol(SubviewTableViewControllerDataSourceProtocol)]);
    delegate = obj;
}
```

Compare Objects
Note that object equality is different from object identity. For the latter, use the equality operator == to test whether two variables point to the same instance.

What is compared when you compare two objects of the same class? That depends on the class. The root class, NSObject, uses pointer equality as the basis of comparison. Subclasses at any level can override their superclass's implementation to base the comparison on class-specific criteria, such as object state. For example, a hypothetical Person object might equal another Person object if the first-name, last-name, and birth-date attributes of both objects match.

Copy Objects
1. Most classes implement deep copying, which makes a duplicate of all instance variables and properties;

2. Some classes (for example, the collection classes) implement shallow copying, which only duplicates the references to those instance variables and properties.