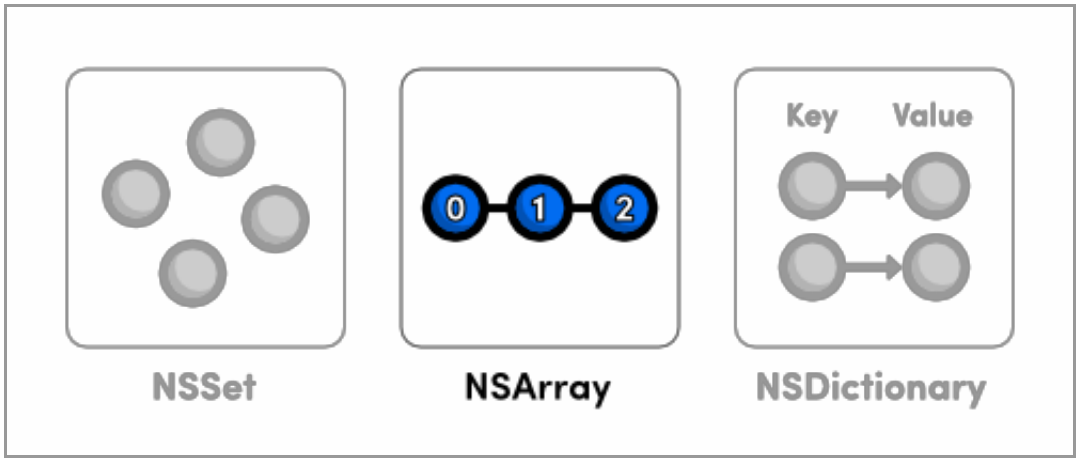
NSArray

`NSArray` is Objective-C's general-purpose array type. It represents an ordered collection of objects, and it provides a high-level interface for sorting and otherwise manipulating lists of data. Arrays aren't as efficient at membership checking as sets, but the trade-off is that they reliably record the order of their elements.



The basic collection classes of the Foundation Framework

Like `NSSet`, `NSArray` is immutable, so you cannot dynamically add or remove items. Its mutable counterpart, `NSMutableArray`, is discussed in the second part of this module.

Creating Arrays

Immutable arrays can be defined as literals using the `@[]` syntax. This was added relatively late in the evolution of the language (Xcode 4.4), so you're likely to encounter the more verbose `arrayWithObjects:` factory method at some point in your Objective-C career. Both options are included below.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
NSArray *ukMakes = [NSArray arrayWithObjects:@"Aston Martin",
                    @"Lotus", @"Jaguar", @"Bentley", nil];

NSLog(@"First german make: %@", germanMakes[0]);
NSLog(@"First U.K. make: %@", [ukMakes objectAtIndex:0]);
```

As you can see, individual items can be accessed through the square-bracket subscripting syntax (`germanMakes[0]`) or the `objectAtIndex:` method. Prior to Xcode 4.4, `objectAtIndex:` was the standard way to access array elements.

Enumerating Arrays

Fast-enumeration is the most efficient way to iterate over an `NSArray`, and its contents are guaranteed to appear in the correct order. It's also possible to use the `count` method with a traditional for-loop to step through each element in the array:

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
// With fast-enumeration
for (NSString *item in germanMakes) {
    NSLog(@"%@", item);
}
// With a traditional for loop
for (int i=0; i<[germanMakes count]; i++) {
    NSLog(@"%d: %@", i, germanMakes[i]);
}
```

If you're fond of blocks, you can use the `enumerateObjectsUsingBlock:` method. It works the same as NSSet's version, except the index of each item is also passed to the block, so its signature looks like `^(id obj, NSUInteger idx, BOOL *stop)`. And of course, the objects are passed to the block in the same order as they appear in the array.

```
[germanMakes enumerateObjectsUsingBlock:^(id obj,
```

```
                                                NSUInteger idx,
                                                BOOL *stop) {
    NSLog(@"%ld: %@", idx, obj);
}];
```

Comparing Arrays

Arrays can be compared for equality with the aptly named `isEqualToArray:` method, which returns `YES` when both arrays have the same number of elements and every pair pass an `isEqual:` comparison. `NSArray` does not offer the same subset and intersection comparisons as `NSSet`.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
NSArray *sameGermanMakes = [NSArray arrayWithObjects:@"Mercedes-Benz",
                            @"BMW", @"Porsche", @"Opel",
                            @"Volkswagen", @"Audi", nil];

if ([germanMakes isEqualToArray:sameGermanMakes]) {
    NSLog(@"Oh good, literal arrays are the same as NSArrays");
}
```

Membership Checking

`NSArary` provides similar membership checking utilities to `NSSet`. The `containsObject:` method works the exact same (it returns `YES` if the object is in the array, `NO` otherwise), but instead of `member:`, `NSArray` uses `indexOfObject:`. This either returns the index of the first occurrence of the requested object or `NSNotFound` if it's not in the array.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
// BOOL checking
if ([germanMakes containsObject:@"BMW"]) {
    NSLog(@"BMW is a German auto maker");
}
```

```
// Index checking
NSUInteger index = [germanMakes indexOfObject:@"BMW"];
if (index == NSNotFound) {
    NSLog(@"Well that's not quite right...");
} else {
    NSLog(@"BMW is a German auto maker and is at index %ld", index);
}
```

Since arrays can contain more than one reference to the same object, it's possible that the first occurrence isn't the only one. To find other occurrences, you can use the related `indexOfObject:inRange:` method.

Remember that sets are more efficient for membership checking, so if you're querying against a large collection of objects, you should probably be using a set instead of an array.

Sorting Arrays

Sorting is one of the main advantages of arrays. One of the most flexible ways to sort an array is with the `sortedArrayUsingComparator:` method. This accepts an `^NSComparisonResult(id obj1, id obj2)` block, which should return one of the following enumerators depending on the relationship between `obj1` and `obj2`:

Return Value	Description
`NSOrderedAscending`	`obj1` comes before `obj2`
`NSOrderedSame`	`obj1` and `obj2` have no order
`NSOrderedDescending`	`obj1` comes after `obj2`

The following example sorts a list of car manufacturers based on how long their name is, from shortest to longest.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
NSArray *sortedMakes = [germanMakes sortedArrayUsingComparator:
    ^NSComparisonResult(id obj1, id obj2) {
        if ([obj1 length] < [obj2 length]) {
            return NSOrderedAscending;
        } else if ([obj1 length] > [obj2 length]) {
```

```
            return NSOrderedDescending;
        } else {
            return NSOrderedSame;
        }
}];
NSLog(@"%@", sortedMakes);
```

Like NSSet, NSArray is immutable, so the sorted array is actually a *new* array, though it still references the same elements as the original array (this is the same behavior as NSSet).

Filtering Arrays

You can filter an array with the `filteredArrayUsingPredicate:` method. A short introduction to predicates can be found in the NSSet module, and a minimal example is included below. Just as with the sort method discussed above, this generates a brand new array.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];

NSPredicate *beforeL = [NSPredicate predicateWithBlock:
    ^BOOL(id evaluatedObject, NSDictionary *bindings) {
        NSComparisonResult result = [@"L" compare:evaluatedObject];
        if (result == NSOrderedDescending) {
            return YES;
        } else {
            return NO;
        }
    }];
NSArray *makesBeforeL = [germanMakes
                         filteredArrayUsingPredicate:beforeL];
NSLog(@"%@", makesBeforeL);    // BMW, Audi
```

Subdividing Arrays

Subdividing an array is essentially the same as extracting substrings from an `NSString`, but instead of `substringWithRange:`, you use `subarrayWithRange:`, as shown below.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];

NSArray *lastTwo = [germanMakes subarrayWithRange:NSMakeRange(4, 2)];
NSLog(@"%@", lastTwo);    // Volkswagen, Audi
```

Combining Arrays

Arrays can be combined via `arrayByAddingObjectsFromArray:`. As with all of the other immutable methods discussed above, this returns a *new* array containing all of the elements in the original array, along with the contents of the parameter.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
NSArray *ukMakes = @[@"Aston Martin", @"Lotus", @"Jaguar", @"Bentley"];

NSArray *allMakes = [germanMakes arrayByAddingObjectsFromArray:ukMakes];
NSLog(@"%@", allMakes);
```

String Conversion

The `componentsJoinedByString:` method concatenates each element of the array into a string, separating them by the specified symbol(s).

```
NSArray *ukMakes = @[@"Aston Martin", @"Lotus", @"Jaguar", @"Bentley"];
NSLog(@"%@", [ukMakes componentsJoinedByString:@", "]);
```

This can be useful for regular expression generation, file path manipulation, and rudimentary CSV processing; however, if you're doing serious work with file paths/data, you'll probably want to look for a dedicated library.

NSMutableArray

The `NSMutableArray` class lets you dynamically add or remove items from arbitrary locations in the collection. Keep in mind that it's slower to insert or delete elements from a mutable array than a set or a dictionary.

Like mutable sets, mutable arrays are often used to represent the state of a system, but the fact that `NSMutableArray` records the order of its elements opens up new modeling opportunities. For instance, consider the auto repair shop we talked about in the NSSet module. Whereas a set can only represent the status of a collection of automobiles, an array can record the order in which they should be fixed.

Creating Mutable Arrays

Literal arrays are always immutable, so the easiest way to create mutable arrays is still through the `arrayWithObjects:` method. Be careful to use `NSMutableArray`'s version of the method, not `NSArray`'s. For example:

```
NSMutableArray *brokenCars = [NSMutableArray arrayWithObjects:
                              @"Audi A6", @"BMW Z3",
                              @"Audi Quattro", @"Audi TT", nil];
```

You can create empty mutable arrays using the `array` or `arrayWithCapacity:` class methods. Or, if you already have an immutable array that you want to convert to a mutable one, you can pass it to the `arrayWithArray:` class method.

Adding and Removing Objects

The two basic methods for manipulating the contents of an array are the `addObject:` and `removeLastObject` methods. The former adds an object to the end of the array, and the latter is pretty self-documenting. Note that these are also useful methods for treating an `NSArray` as a stack.

```
NSMutableArray *brokenCars = [NSMutableArray arrayWithObjects:
                              @"Audi A6", @"BMW Z3",
```

```
                                @"Audi Quattro", @"Audi TT", nil];
[brokenCars addObject:@"BMW F25"];
NSLog(@"%@", brokenCars);       // BMW F25 added to end
[brokenCars removeLastObject];
NSLog(@"%@", brokenCars);       // BMW F25 removed from end
```

It's most efficient to add or remove items at the *end* of an array, but you can also insert or delete objects at arbitrary locations using `insertObject:atIndex:` and `removeObjectAtIndex:`. Or, if you don't know the index of a particular object, you can use the `removeObject:` method, which is really just a convenience method for `indexOfObject:` followed by `removeObjectAtIndex:`.

```
// Add BMW F25 to front
[brokenCars insertObject:@"BMW F25" atIndex:0];
// Remove BMW F25 from front
[brokenCars removeObjectAtIndex:0];
// Remove Audi Quattro
[brokenCars removeObject:@"Audi Quattro"];
```

It's also possible to replace the contents of an index with the `replaceObjectAtIndex:withObject:` method, as shown below.

```
// Change second item to Audi Q5
[brokenCars replaceObjectAtIndex:1 withObject:@"Audi Q5"];
```

These are the basic methods for manipulating mutable arrays, but be sure to check out the official documentation if you need more advanced functionality.

Sorting With Descriptors

Inline sorts can be accomplished through `sortUsingComparator:`, which works just like the immutable version discussed in Sorting Arrays. However, this section discusses an alternative method for sorting arrays called `NSSortDescriptor`. Sort descriptors typically result in code that is more semantic and less redundant than block-based sorts.

The `NSSortDescriptor` class encapsulates all of the information required to sort an array of dictionaries or custom objects. This includes the property to be compared, the comparison method, and whether the sort is ascending or descending. Once you configure a descriptor(s), you can sort an array by passing it to the `sortUsingDescriptors:` method. For example, the following snippet sorts an array of cars by price and then by model.

```
NSDictionary *car1 = @{
    @"make": @"Volkswagen",
    @"model": @"Golf",
    @"price": [NSDecimalNumber decimalNumberWithString:@"18750.00"]
};
NSDictionary *car2 = @{
    @"make": @"Volkswagen",
    @"model": @"Eos",
    @"price": [NSDecimalNumber decimalNumberWithString:@"35820.00"]
};
NSDictionary *car3 = @{
    @"make": @"Volkswagen",
    @"model": @"Jetta A5",
    @"price": [NSDecimalNumber decimalNumberWithString:@"16675.00"]
};
NSDictionary *car4 = @{
    @"make": @"Volkswagen",
    @"model": @"Jetta A4",
    @"price": [NSDecimalNumber decimalNumberWithString:@"16675.00"]
};
NSMutableArray *cars = [NSMutableArray arrayWithObjects:
                        car1, car2, car3, car4, nil];

NSSortDescriptor *priceDescriptor = [NSSortDescriptor
                                     sortDescriptorWithKey:@"price"
                                                 ascending:YES
                                                  selector:@selector(compare:)];
NSSortDescriptor *modelDescriptor = [NSSortDescriptor
                                     sortDescriptorWithKey:@"model"
                                     ascending:YES
                                     selector:@selector(caseInsensitiveCompare:)];
```

```
NSArray *descriptors = @[priceDescriptor, modelDescriptor];
[cars sortUsingDescriptors:descriptors];
NSLog(@"%@", cars);    // car4, car3, car1, car2
```

The descriptor's `selector` is called on each key's value, so in the above code, we're calling `compare:` on `item[@"price"]` and `caseInsensitiveCompare:` on `item[@"model"]` for each pair of items in the array.

Filtering Mutable Arrays

Filtering works the same as it does with immutable arrays, except items are removed from the existing array instead of generating a new one, and you use the `filterUsingPredicate:` method.

Enumeration Considerations

As with all mutable collections, you're not allowed to alter it in the middle of an enumeration. This is covered in the Enumeration Considerations section of the `NSSet` module, but instead of using `allObjects` for the snapshot, you can create a temporary copy of the original array by passing it to the `arrayWithArray:` class method.

Continue to *NSDictionary* ›