Objective-C Arrays and Language Features

2501ICT Nathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2011

Outline

Lists and Arrays

Linear Collections in Objective-C

Objective-C Arrays

- `NSArray`
  - basic linear collection (list or array) class
  - class cluster with concrete classes optimized for different representation
- `NSMutableArray`
  - dynamic subclass of `NSArray`
    - → allows adding and removing of elements
- `NSEnumerator`
  - enumerates all elements
  - similar to Iterator in Java

Objective-C Array Examples

Example (prints: `a5 is:  Hello, Hello with 2 elements`)

```
NSArray *a1 = [NSArray new];                                  // empty array
NSArray *a2 = [NSArray arrayWithObject: @"Hello"];            // array w/ single object
NSArray *a3 = [NSArray arrayWithArray: a2];                   // copy a2 into a3
NSArray *a4 = [[NSArray alloc]                                // create a new array
                initWithArray: a2                             // by copying a2 doing a
                copyItems: YES];                              // deep copy of items
NSArray *a5 = [a4 arrayByAddingObjectsFromArray: a3];         // a5 = a4 + a3
int count5 = [a5 count];                                      // number of elements

if ([a1 isEqualToArray: a2])                                  // same content?
        printf("a1 is equal to a2 -- how come?\n");

NSString *s = [a5 componentsJoinedByString: @", "];           // convert a5 to string
printf("a5 is: %s ", [s UTF8String]);                         // print the string
printf("with %d elements\n", count5);                         // number of elements

[a1 release];   [a4 release];           // don't forget proper memory management!
```

Other Useful Methods

+ `arrayWithObjects:` ...
  - convenience method
  - creates array from `nil` terminated list of objects
  - e.g.: `[NSArray arrayWithObjects: @"1", @"2", nil];`
- `indexOfObject:`
  - searches for an object within the array
- `subarrayWithRange:`
  - returns a sub-array within a given range
- `mutableCopy`
  - returns a mutable copy of an array

→ See `NSArray` and `NSMutableArray` in the Foundation API

Enumerating Array Example

Example (prints: 1 2 3)

```
#import <Foundation/Foundation.h>

int main(int argc, char *argv[])
{
        NSAutoreleasePool *pool = [NSAutoreleasePool new];
        NSArray *list = [NSArray arrayWithObjects: @"1", @"2", @"3", nil];

        NSEnumerator *enumerator = [list objectEnumerator];

        NSString *s;
        while (s = [enumerator nextObject])             // loop through array
                printf("%s ", [s UTF8String]);          // print each element

        printf("\n");

        [pool release];

        return EXIT_SUCCESS;
}
```

Lists and Arrays

Container Classes

Container Classes

- The Problem: lists, arrays, and other linear collection classes can only store objects (not primitive types)!
- Primitive types need to be encapsulated within objects
  - e.g. `Integer` vs. `int` in Java
- `NSNumber`
  - generic Objective-C container class for numbers
  - holds `int`, `long`, `float`, `double`, etc.
- `NSValue`
  - superclass of `NSNumber` holding any kind of value
  - e.g. pointers, sizes, ranges, points, rectangles, etc.
  - e.g. can be extended to hold any value you want
- `NSNull`
  - place holder object for collections

  → `nil` (or `NULL`) pointers cannot be stored within collections!
- `NSData`
  - container class for binary data (byte arrays)
  - efficient storage for large amounts of data

Objective-C Container Example

Example (prints: `1 2.5 <00104a30> <null>`)

```
#import <Foundation/Foundation.h>

int main(int argc, char *argv[])
{
        NSAutoreleasePool *pool = [NSAutoreleasePool new];

        // create some Container objects
        NSNumber *n1 = [NSNumber numberWithInt: 1];
        NSNumber *n2 = [NSNumber numberWithDouble: 2.5];
        NSValue *val = [NSValue valueWithPointer: pool];
        NSNull *null = [NSNull null];

        NSArray *arr = [NSArray arrayWithObjects: n1, n2, val, null, nil];

        NSEnumerator *enumerator = [arr objectEnumerator];

        id obj;
        while (obj = [enumerator nextObject])           // loop through array
                printf("%s ",  [[obj description] UTF8String]);
        printf("\n");

        [pool release];

        return EXIT_SUCCESS;
}
```

NSData Example

Example (an efficient file copy program)

```
#import <Foundation/Foundation.h>

int main(int argc, char *argv[])
{
        NSAutoreleasePool *pool = [NSAutoreleasePool new];

        // get the file names (needs error checking in the real world!)
        NSString *from = [NSString stringWithUTF8String: argv[1]];
        NSString *to   = [NSString stringWithUTF8String: argv[2]];

        // read the file data into memory
        NSData *data = [NSData dataWithContentsOfMappedFile: from];

        // write the data into the destination file
        [data writeToFile: to  atomically: YES];

        [pool release];

        return EXIT_SUCCESS;
}
```

NSData Example (2)

Example (efficient file copy using NSProcessInfo)

```
#import <Foundation/Foundation.h>

int main(int argc, char *argv[])
{
        NSAutoreleasePool *pool = [NSAutoreleasePool new];
        NSArray *args = [[NSProcessInfo processInfo] arguments];

        [[NSData dataWithContentsOfMappedFile: [args objectAtIndex: 1]]
                                  writeToFile: [args objectAtIndex: 2]
                                   atomically: YES];

        [pool release];

        return EXIT_SUCCESS;
}
```

Protocols in Objective-C

Objective-C Protocols

Protocols

- Protocols work like interfaces in Java
  - they specify a number of methods a class must implement

Example (protocol example)

```
@protocol Printing
- (void) print;          // conforming classes must have a 'print' method
@end


@interface MyClass: NSObject <Printing>          // MyClass conforms to Printing
{
        int a, b;
}
- init;
- setA: (int) newA  b: (int) newB;
// - (void) print;                                // must exist, but not in interface!
@end
```

Example for using Protocols

Example (NSCopying and Printing protocols)

```
/*
 * statically indicate that an object conforms to a protocol
 */
id<Printing> aPrintingObject = [obj someMethod];

id<Printing, NSCopying> other = [obj someOtherMethod];

[aPrintingObject print];                // we know this conforms to Printing

aPrintingObject = [other copy];         // 'other' conforms to NSCopying as well

/*
 * we can also test conformance dynamically via conformsToProtocol:
 */
id obj = other;

if ([obj conformsToProtocol: @protocol(Printing)])
        [obj print];                    // only invoke print if obj conforms
```

Introspection

Introspection

Checking for individual Methods

- Objective-C allows to check for individual Methods
  - does not require a full protocol
  - useful if only one method needs to be checked dynamically

Example (`-respondsToSelector:` example)

```
id obj = [anArray objectAtIndex: 5];    // whatever object is found in the array

/*
 * check if "obj" has a "print" method before invoking it:
 */
if ([obj respondsToSelector: @selector(print)])
        [obj print];                    // only invoke print if method exists
```

Determining an Object's type

- Objective-C also allows to check which class an object belongs to
  - → `isMemberOfClass:` tests for a specific class only
  - → `isKindOfClass:` tests for a class or any of its subclasses

Example (dynamically determining class membership)

```
id obj = [anArray objectAtIndex: 6];    // whatever object is found in the array

/*
 * check if "obj" is a mutable string
 */
if ([obj isMemberOfClass: [NSMutableString class]])
       [obj appendString: @","];        // append a comma

/*
 * check if "obj" is any kind of string (including NSMutableString) or number
 */
if ([obj isKindOfClass: [NSString class]])
       printf("%s", [obj UTF8String]);          // print as a string
else if ([obj isKindOfClass: [NSNumber class]])
       printf("%lg", [obj doubleValue]);        // print as a double
```

Categories in Objective-C

Categories

Using and Extending Classes

- When should a class be subclassed?
  - if you just want to use a class, make it a member variable of your class
    - → a `Zoo` class should just have `Animal` members
  - → for more specific concepts, use a subclass
    - → a `Cat` class should be derived from an `Animal` class
- Objective-C offers a third option: Categories
  - a category allows you to add methods to an existing class
    - → these methods become available immediately to any code using the existing class!
  - useful if you believe a method is missing from a class!

Category Example: extending NSArray

Example (a `firstObject` method for NSArray)

```
#import <Foundation/Foundation.h>

@interface NSArray (AddFirstObject)             // a category for NSArray
- firstObject;                                  // adds a firstObject method
@end

@implementation NSArray (AddFirstObject)        // category implementation
- firstObject                                   // firstObject implementation
{
        return [self objectAtIndex: 0];         // get first object
}
@end

int main(int argc, char *argv[])
{
        NSAutoreleasePool *pool = [NSAutoreleasePool new];
        NSArray *list = [NSArray arrayWithObjects: @"one", @"two", nil];

        printf("%s", [[list firstObject] UTF8String]);  // print first object

        [pool release];

        return EXIT_SUCCESS;
}
```