Objective-C 2.0

2501ICT/7421ICTNathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2012
Outline

1. Fast Enumeration and Properties
   - For Each Loops
   - Properties

2. Exception Handling
   - Native Exception Handling

3. Memory Management and Blocks
   - Using Blocks
   - Garbage Collection and ARC
Objective-C 2.0
Remember Enumerators?

- **NSEnumerator**
  - handles enumeration in an object-oriented way

---

**Example (Enumerators using NSEnumerator)**

```c
#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;
}
```
New Language feature: `for (object in collection)`
- uses fast enumeration

Example (Fast Enumeration)

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

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

    for (NSString *s in list) { // fast enumeration
        printf("%s ", [s UTF8String]); // print each element
    }

    printf("\n");

    [pool release];

    return EXIT_SUCCESS;
}
```
Every object property should have two access methods → Setters and Getters

Example (Point2D.h with access methods)

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

@interface Point2D: NSObject
{
    int x;
    int y;
}

-(int) x; // access methods
-(void) setX: (int) newX;

-(int) y;
-(void) setY: (int) newY;
@end
```
Access Methods Reviewed

- Every object property should have two access methods → Setters and Getters

Example (Point2D.m with access methods)

```objective-c
#import "Point2D.h"

@implementation Point2D
- (int) x {
    return x;
}

- (void) setX: (int) newX {
    x = newX;
}

- (int) y {
    return y;
}

- (void) setY: (int) newY {
    y = newY;
}
@end
```

René Hexel  Objective-C 2.0
Objective-C 2.0 Properties

- @property
  - declares class properties
Objective-C 2.0 Properties

- `@synthesize` creates code for setters and getters automatically!
Normal set and get methods can be used
→ properties stick to standard naming conventions!

Point2DMain

```objective-c
#import "Point2D.h"

int main(int argc, char *argv[]) {
    Point2D *pt = [Point2D new];
    [pt setX: 5];
    printf("x = %d\n", [pt x]);
    return EXIT_SUCCESS;
}
```

with dot notation

```objective-c
#import "Point2D.h"

int main(int argc, char *argv[]) {
    Point2D *pt = [Point2D new];
    pt.x = 5;  // invokes setX:
    printf("x = %d\n", pt.x);
    return EXIT_SUCCESS;
}
```
Properties Summarised

- `@property` defines a property → needs a storage type!
  - `assign` for a simple assignment (e.g. `int`)
  - `retain` for retaining Objects (release and retain)
  - `copy` for copying Objects (e.g. `NSString`)

- Other qualifiers: `readonly`, `readwrite`, `nonatomic`

- `@synthesize` creates getter and setter code in `.m` file

- Dot notation: syntactic sugar for invoking setters and getters, e.g.:
  - `int x = point.x;` for `int x = [point x];`
  - `point.x = x;` for `[point setX: x];`
  - `point.x++;` for `[point setX: [point x] + 1];`
@try
- starts an exception handling domain
  - like try in Java
  - replaces older NS_DURING macro
  - exceptions that occur will be caught

@catch (NSEXception *localException)
- the actual exception handler
- catches exceptions that occur in the handling domain
  - replaces older NS_HANDLER macro

@finally
- follows both normal and abnormal termination
  - replaces older NS_ENDHANDLER macro
Objective-C 2.0 Exception Handling Example

Example (NSRangeException)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool *pool = [NSAutoreleasePool new];
    NSArray *array = [NSArray array]; // an empty array

    @try
    {
        id object = [array objectAtIndex: 0]; // will this work?
        printf("%s", [object UTF8String]); // never reached
    }
    @catch (NSException *localException)
    {
        printf("%s: %s", [[localException name] UTF8String],
                        [[localException reason] UTF8String]); // print exception
    }
    @finally
    {
        printf("%, count = %lu\n", [array count]);
    }
    [pool release];

    return EXIT_SUCCESS;
}
```

René Hexel
Objective-C 2.0
# Objective-C 2.0 Exception Throwing Example

```objective-c
#import <Foundation/Foundation.h>

void some_function(void)
{
    [NSError raise: @"MyException" // raise 'MyException'
       format: @"reason %d", 42];  // a not very readable reason!
}

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

    @try {
        some_function(); // call some function
    }

    @catch (NSError *localException) {
        printf("%s: %s\n", 
               [[localException name] UTF8String],
               [[localException reason] UTF8String]);
    }

    [pool release];

    return EXIT_SUCCESS;
}
```
C Functions can be useful, but have drawbacks
- global namespace (e.g. only one `main()` function
- module-specific `static` functions
  - still requires separate definition and naming

Blocks
- allow defining functions on the fly
- syntax addition at the C Language level
- can vastly simplify code
- very powerful lambda expressions
Blocks are defined using `^ { ... }`

Example (Passing a Block to a Method)

```objective-c
MyClass *myObject = [[MyClass alloc] init];

[myObject doSomethingUsingBlock: ^{ /* ... some code */ } ];
```
Blocks can take parameters (like functions)

Example (Block Enumeration inNSArray)

```objective-c
NSArray *array = [NSArray arrayWithObjects:@"1", @"2", @"3", nil];

[array enumerateObjectsUsingBlock: ^(id obj, NSUInteger i, BOOL *stop) {
    NSLog(@"String at Index %d is %@", i, obj);
    if (i == 1) *stop = YES; // stop enumeration
}
];
```
Passing local Variables to Blocks

- Local Variables declared “outside” can be used within blocks
- Inside the block, these variables are *read only*!
- But: *instance variables* can be modified!
Local Variables can be declared as __block to be writable

Example (Read/Write variable used inside block)

```objective-c
NSArray *array = [NSArray arrayWithObjects: @"1", @"2", @"3", nil];
__block int count = 0;

[array enumerateObjectsUsingBlock: ^(id obj, NSUInteger i, BOOL *stop)
{
    count++;
    // increment count
}]

printf("Count is %d\n", count);
```
Garbage Collection

- Normally Objective-C uses reference counting, i.e. reference count gets:
  - set to 1 by `new`, `alloc`, and `copy`
  - incremented by `retain`
  - decremented by `release` (calls `dealloc` if 0)
- Garbage Collection is available for Objective-C 2.0:
  - Mac OS X 10.5 (or above) only!
  - not on GNUstep
  - not on the iPhone!
- Optimising Code for Garbage Collection: Autorelease Pools
  - use `[pool drain]` instead of `[pool release]`
Automatic Reference Counting

- The latest clang Objective-C compiler knows Memory Management Rules
- Programmer must still make explicit what kind of reference each pointer is!
  - → strong properties are retained and released
  - → weak properties are zeroed out on dealloc
- Normal pointers also need to indicate their ownership
  - → Language extension introduced by clang
  - → Must be specified if the compiler cannot determine ownership
  - → __strong
  - → __weak
  - → __unsafe_unretained (not handled by compiler)
  - → __autoreleasing (put on autorelease pool)
Autorelease Pools Revisited

- @autoreleasepool { ... }
- Compiler Support for Autorelease Pools
  → faster than NSAutoreleasePool
  → works for ARC and non-ARC code as well as Garbage Collection

Example (Using @autoreleasepool)

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

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

        for (NSString *s in list) // fast enumeration
            printf("%s ", [s UTF8String]); // print each element

        printf("\n");
    }

    return EXIT_SUCCESS;
}
```

René Hexel  Objective-C 2.0