Introduction to Objective-C
2501ICT/7421ICT Nathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2012
Outline

1. Objective-C Classes
   - Objective-C

2. Compiling Objective-C Code
   - Compiling and Makefiles
   - Documentation with HeaderDoc
Objective-C Classes
Compiling Objective-C Code

Classes and Objects in Objective-C

So far: Pure C
- procedural, no object-oriented concepts
- difficult to write re-usable code: disadvantage for larger projects

Objective-C is a small syntax addition to C
- highly dynamic and very powerful object model
- classes are first class objects
  → most features implemented through methods
A Point Class Interface

Java: `Point.java`

```java
import java.lang.System;

class Point extends Object {
    int x;
    int y;

    public Point() { x = 0; y = 0; }

    public int getX() { return x; }

    public void setX(int newx) { x = newx; }
}
```

Objective-C: `Point.h`

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

@interface Point : NSObject {
    int x; // member variables
    int y; // protected by default
}

- init; // constructor
- (int) x; // access methods
- (void) setX: (int) newx;
@end
```
A Point Class Implementation

Java: Point.java

```java
import java.lang.System;

class Point extends Object {
    int x;
    int y;

    public Point() { x = 0; y = 0; }

    public int getX() { return x; }

    public void setX(int newx) {
        x = newx;
    }
}
```

Objective-C: Point.m

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

@implementation Point

- (instancetype)init {
    x = 0;
    y = 0;
    return self;
}

- (int)x {
    return x;
}

- (void)setX:(int)newx {
    x = newx;
}
@end
```
Objective-C Additions So Far

- **#import**
  - imports a header file only once
  - like `#include` in plain C, but does not require `#ifndef` include protection!

- **@interface / @end**
  - Class Interface
  - member variables, method declarations
    - explicitly extend root class `NSObject`

- **@implementation / @end**
  - Class Implementation
  - method definitions

- **- init**
  - the default initialiser (constructor) method
  - no parameters
Using the Point Class: invoking Methods

**Java: Main.java**

```java
import java.lang.System;

public class Main
{
    public static void main(String[] args)
    {
        Point xy = new Point();

        int x = xy.getX();
        xy.setX(x + 5);
    }
}
```

**Objective-C: Main.m**

```objc
#import "Point.h"

int main(int argc, char *argv[])
{
    Point *pt = [Point new];

    int x = [pt x]; // get x
    [pt setX: x + 5]; // set x

    return 0;
}
```
Constructors

- **What happens when `new` gets called?**
  - unlike Java, `new` is not a keyword
  - just another method!
    - invokes `alloc` to allocate memory, then `init`
- **`init` needs to return `self`**
  - `self` points to the current object
  - like this in Java
- **Additional constructors**
  - should start with `init`... by convention
  - can take parameters, e.g.:
    - `- initWithX: (int) x y: (int) y`
    - invoked as, e.g., `[point initWithX: 10 y: 5];`
- → all constructors need to return `self`!
Method Nesting

Example (original `Point` class)

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

int main(int argc, char *argv[]) {
    Point *pt = [Point new];
    int x = [pt x]; // get x
    [pt setX: x + 5]; // set x
    return 0;
}
```

Example (`alloc/init`)

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

int main(int argc, char *argv[]) {
    Point *pt = [[Point alloc] init];
    int x = [pt x];
    [pt setX: x + 5];
    return 0;
}
```
**Multiple Parameters**

- E.g., a `setXY()` method in *Java* that takes two parameters:
  - `void setXY(int x, int y)`
  - invocation, e.g.: `point.setXY(10, 5);`

- Problem: which parameter is which?
  - easy for one or two parameters – what about 10?

- Objective-C allows to split the method name, e.g.:
  - `(void) setX: (int) x  y:(int) y`
  - invocation, e.g.: `[point setX: 10  y:5];`
Objective-C types can be referenced through their class pointer
- e.g. `Point *x = [[Point alloc] init];`
  → cannot be assigned to a pointer of a different type

In Objective-C, objects are completely dynamic
- runtime method resolution

Every object is of type `id`
- `id` is completely nonrestrictive
- any object pointer can be assigned to `id` and vice versa
  → allows invoking any methods on a variable of type `id`
Dynamic Typing Example

Example (Using `id` instead of `Point *`)

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

int main(int argc, char *argv[]) {
    id point = [[Point alloc] init]; // the 'point' variable is of type id

    int x = [point x]; // 'x' method is resolved at run time

    [point setX: x + 5 y: 10]; // same for 'setX:y:'

    return 0;
}
```
### Summary (1)

- Classes are split into interface *file.h* and implementation *file.m*
  - the name of the *file* should always be the class name
- Classes should subclass **NSObject**
  - **NSObject** is the standard root class of the Foundation API
- Typed Object references are Pointers *`
  - `Point *p` (instead of `Point p` in Java)
- Generic Object references are of type **id**
  - → possible because methods are resolved at run time
    - no casting needed!
- Method invocations use `[]` instead of `.`
  - `[object method]; vs. object.method(); in Java`
- No *get* prefix for getter methods!
Method names with multiple parameters are split

- `[point setX: 3 y: 2 z: 1];` (instead of `point.setXYZ(3, 2, 1);` in Java)

Allocation versus Initialisation

- `[[anObject alloc] init]` instead of `[[anObject new]`

No dedicated Constructor

→ initialiser method names should start with `init` by convention!

- `e.g. Point *p = [[Point alloc] initWithX: 5 y: 7];`

→ initialiser methods need to return `self`

`self` refers to the current object

- `like this in Java`
Compiling Objective-C Code
Clang knows Objective-C

- clang -c -Wall -o file.o file.m

Linking is more complex, requires:

- standard Objective-C runtime: libobjc
- standard OpenStep API: libFoundation and libAppKit

Different API setups have different locations

- flags for clang vary, depending on where to find libraries

⇒ Standardised ways of accessing API

- -framework on Mac OS X
- GNUmakefile framework for GNUstep (Linux, Windows, ...)

René Hexel
Introduction to Objective-C
Example (Mac OS X Makefile for an Objective-C program)

```
# A Mac OS X Makefile example for Objective-C and the Foundation framework
#
# -- this assumes a main() module ObjcMain.m and a class ObjcModule.m
# -- (the class comes with a corresponding ObjcModule.h)
#
CC=clang

.SUFFIXES: .o .m

.m.o:
    $(CC) -c -std=c99 -Wall -o $*.o $*.m

Program:  ObjcMain.o ObjcModule.o
    $(CC) -o Program ObjcMain.o ObjcModule.o -framework Foundation

ObjcModule.o:  ObjcModule.m ObjcModule.h
```
GNUstep Makefiles have all the rules already pre-defined

- GNUmakefile
  - the name of the main makefile (rather than Makefile)
- common.make
  - common rules to be included in all GNUmakefiles
- tool.make
  - pre-defined rules for command line utilities
  - set TOOL_NAME to be the command name
- program_OBJC_FILES
  - the Objective-C files needed to compile program
- ADDITIONAL_CPPFLAGS
  - set to -Wall -Wno-import
Example (GNUmakefile)

# A simple GNUmakefile example for an Objective-C command line utility
#
include $(GNUSTEP_MAKEFILES)/common.make

# Build a simple Objective-C program, called Example
TOOL_NAME = Example

# The Objective-C Implementation files to compile
Example_OBJC_FILES = Main.m Some_Class.m Other_Class.m

# Class Header (Interface) files
Example_HEADER_FILES = Some_Class.h Other_Class.h

# Define the compiler flags
ADDITIONAL_CPPFLAGS = -Wall -Wno-import

# Include the rules for making Objective-C command line tools
include $(GNUSTEP_MAKEFILES)/tool.make
Example (GNUmakefile after removing the Comments)

```makefile
include $(GNUSTEP_MAKEFILES)/common.make

TOOL_NAME = Example

Example_OBJC_FILES = Main.m Some_Class.m Other_Class.m
Example_HEADER_FILES = Some_Class.h Other_Class.h

ADDITIONAL_CPPFLAGS = -Wall -Wno-import

include $(GNUSTEP_MAKEFILES)/tool.make
```
AutoGSDoc in GNUmakefiles

- autogsdoc extracts comments starting with /** *
- Can be automated in a GNUmakefile
  - document.make
  - pre-defined rules for autogsdoc
  - DOCUMENT_NAME
    - variable containing the name of the documentation
  - Document_AGSDOC_FILES
    - lists the source files to scan for documentation

→ Only works for C and Objective-C (not C++)
Example (GNUmakefile plus autogsdoc)

```makefile
include $(GNUSTEP_MAKEFILES)/common.make

TOOL_NAME = Example
Example_OBJC_FILES = Main.m Some_Class.m Other_Class.m
Example_HEADER_FILES = Some_Class.h Other_Class.h

DOCUMENT_NAME = Documentation
Documentation_AGSDOC_FILES = Some_Class.h Other_Class.m

ADDITIONAL_CPPFLAGS = -Wall -Wno-import

include $(GNUSTEP_MAKEFILES)/tool.make
include $(GNUSTEP_MAKEFILES)/documentation.make
```
HeaderDoc in Makefiles

- **headerdoc** extracts comments starting with /* !
- Can be automated in a Makefile
  - add a `doc` target
- Needs to run once for each header file
  
  → `headerdoc2html -o Documentation MyHeader.h`
- **Table of Contents generated by** `gatherheaderdoc`
  
  → `gatherheaderdoc Documentation`
Example Makefile with HeaderDoc

Example (Makefile with HeaderDoc)

```bash
# # An example Mac OS X Makefile with a 'doc' target #

CC=clang

.SUFFIXES: .o .m

.m.o:
    $(CC) -std=c99 -c -Wall -o $*.o $*.m

all: Program doc

Program: ObjcMain.o ObjcModule.o
    $(CC) -o Program ObjcMain.o ObjcModule.o -framework Foundation

ObjcModule.o: ObjcModule.m ObjcModule1.h Header2.h

doc: ObjcMain.m ObjcModule.m ObjcModule.h
    headerdoc2html -o Documentation ObjcModule.h
    headerdoc2html -o Documentation Header2.h
    gatherheaderdoc Documention
```
Doxygen in Makefiles

- `doxygen` extracts comments starting with `/**`
- Can be automated in a Makefile
  - add a `doc` target
- Needs a configuration file (`Doxyfile`)
  - manually run `doxygen -g`
  - `cvs add Doxyfile`
- The default `Doxyfile` is not very useful!
  - edit `Doxyfile`
    - fill in `PROJECT_NAME`
    - set `JAVADOC_AUTOBRIEF` to `YES`
    - set `EXTRACT_ALL` to `YES`
Example Makefile with Doxygen

Example (C++ example Makefile)

```bash
# An example Makefile for C++ with a 'doc' target
#
# CPLUS=g++

.SUFFIXES: .o .cc

.cc.o:
    $(CPLUS) -c -Wall -o $*.o $*.cc

all: Program doc

Program: CppMain.o CppModule.o
    $(CPLUS) -o Program CppMain.o CppModule.o

CppModule.o: CppModule.cc CppModule.h

doc: CppMain.cc CppModule.cc CppModule.h
doxygen Doxyfile
```