**Challenge 3**

**by Steven Lytovka**

**Section 1 - Getting Started**

**1.3 Updates to this course 3m 31s**

**Why do you think it's important to be aware of the idiosyncracies with older versions of Objective-C and to keep up with new features as they are added?**

A: Programming is very dynamic industry, software developers constantly releases new versions, improvements, some changes that makes programming much easier and pleasant process. To be competitive in this market its important to stay updated to innovations. Also its handy to be aware of idiosyncrasies of older version, because sometimes if you need to craft some code written in older version its important to be aware of it's functionality and syntax, and interaction with newer versions.

**Section 2 - Objective-C Basics**

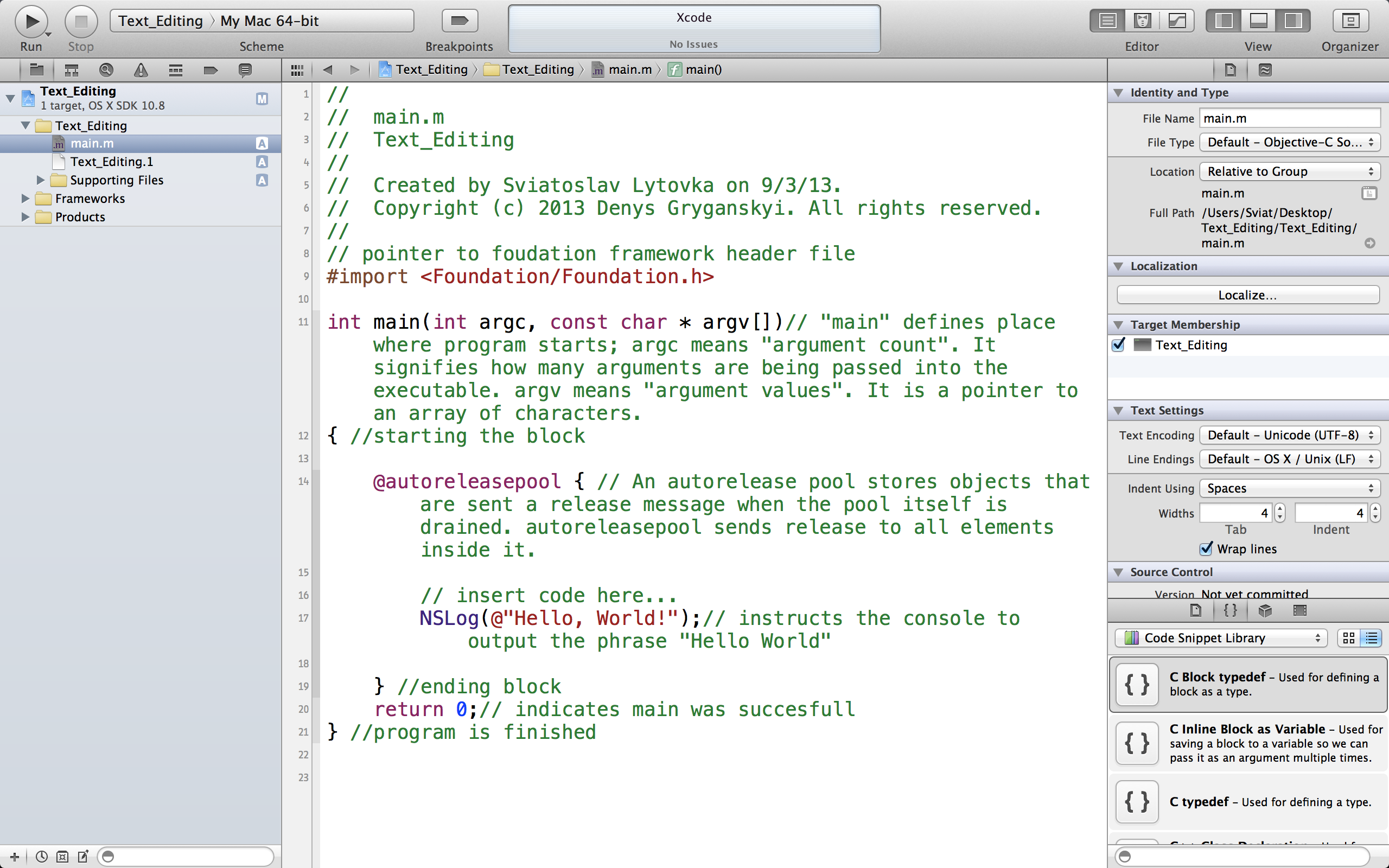
**2.1 The Objective-C language 4m 11s**

**How did Objective-C become the language to learn if you want to make apps for the iPhone** **and iPad?**

A: Objective-C became programming language for making apps for iPhone and iPad historically. Apple chose Objective-C back in the day because it was already the language of NeXTSTEP, the desktop operating system pioneered by Steve Jobs and NeXT. When Apple bought NeXT, Objective-C became the backbone of OS X and iOS years later.

**2.2 The structure of an Objective-C program**

**Create a new project. Go to the menu option Xcode/Preferences/Text\_Editing and make sure "Line Numbers" is checked in the section marked "Show." Then add comments describing the purpose of each auto-generated line in the main.m file. For example on Line 17 I would write: " NSLog(@"Hello, World!"); //instructs the console to output the phrase "Hello,World!")"**

****

**2.3 Compiling and running your code 8m 37s**

**Q;”Why might you build in one version of iOS but deploy in an older version?**

A: Sometimes we might build a program in one version of iOS but deploy in an older one, because we have to take in count who will use this app. Users of this app might use version that doesn't support some features from a new one and program won't work.

**Section 3 - Program Flow**

**3.1 Logging messages to the command line 6m 7s**

**Following the example in the video, write a program that calculates and outputs to the**

**console the number of seconds in ten years. Copy and paste your code here.**

A: Here is my program:

#import <Foundation/Foundation.h>

int main(int argc, const char \* argv[])

{

@autoreleasepool {

// insert code here...

int minutes=60;

int hours=24;

int days=365;

int minutesInYear=10\*minutes\*hours\*days;

NSLog(@"There are %i minutes in 10 years.", minutesInYear);

}

return 0;

}

Outcome: There are 5256000 minutes in 10 years.

**3.2 Writing conditional code 7m 1s**

**Using Objective-C, create an integer variable called "day" that represents the days of the**

**week. Write an if statement that checks whether "day" is a weekend day. If the day is a**

**weekend day then have your program print a message saying "Have a nice weekend!" and if it's not, print a message saying "I hope you're having a good week!"**

A: Program for days of the week:

// main.m

// Text\_Editing

//

// Created by Sviatoslav Lytovka on 9/3/13.

// Copyright (c) 2013 Denys Gryganskyi. All rights reserved.

//

#import <Foundation/Foundation.h>

int main(int argc, const char \* argv[])

{

@autoreleasepool {

// insert code here...

int day=7;

NSLog(@"In this programm integers 1 to 7 represent days of week Monday trough Sunday. Change day value to get different message");

if (day<=5 && day>=1) {

NSLog(@"I hope you are having a good week");

} else if (day==6 || day==7) {

NSLog(@"Have a nice weekend!");

}

else NSLog(@"Wrong number was entered");

}

**return 0;**

**}**

**3.3 The switch statement**

**Create a variable called "hurricaneCategory" and a switch statement that prints out a message describing a hurricane's category from 1-5.**

A: Program:

#import <Foundation/Foundation.h>

int main(int argc, const char \* argv[])

{

@autoreleasepool {

int hurricaneCategory=3;

switch (hurricaneCategory) {

case 1:

NSLog(@"Cat 1. Wind 74-95 mph");

break;

case 2:

NSLog(@"Cat 2. Wind 96-110 mph");

break;

case 3:

NSLog(@"Cat 3. Wind 111-130 mph");

break;

case 4:

NSLog(@"Cat 4. Wind 131-155 mph");

break;

case 5:

NSLog(@"Cat 5. Wind 155+ mph");

break;

default: NSLog(@"That was not hurricane!");

break;

}

}

return 0;

}

**3.5 Operators and expressions**

**List the 6 types of operators described in this video. Provide their name, a description of their meaning, and the syntax you would use to execute them. What code snippet does the ternary operator replace?**

A: Types of operators:

1) Arithmetic: + addition, - subtraction, \*multiplying, / division, = assignment operator,

2) Comparison operators:

== exact equal to;

!= not equal to;

>, <, >=, <= - more than, less than, more than or equal to, less than or equal to

3) Logical operators AND/OR

&& - AND;

|| - OR

4)Modulus

% - gives the reminder from dividing, can use only for integers;

5) Increment/Decrement operators

a++ or ++a is equal to a=a+1

a-- or --a is equal to a=a-1

a+=b, a-=b, a\*=b, a/=b is equal to a=a+b, a=a-b, a=a\*b, a=a/b

6) Ternary operator. This operator replaces if statement:

(condition)?true:false

**3.6 Loops**

**CHALLENGE: Think of a scenario while using a mobile app that might require you to use a "continue" statement in the middle of a loop.**

A: Operator “continue” is used when we need to skip some statement inside the loop. For example, program counts average age of app's users. If user entered 0 or left space blank we can use "continue" to sort this data out, so we will count only right entries.

**3.7 Functions**

**What is a function? What is a function prototype? What are the purposes of each? What are the rules for when and how you can call a function?**

A: Function is simply chunk of code wrapped up and given name. Function definition is writing the code within the function. If we want to call the function before defining it then we need to prototype it. Function prototyping is to tell the compiler that we are using the function name "some function" before defining it then it is defined somewhere later in the program.

Function prototype is written before the main() function. To call the function we just need to display its name and parameters within parentheses, for example, myFunction().

It is necessary to do function prototyping if we are calling it before the function definition.

Functions allow a programmer to wrap up some piece of code, so it doesn't get mixed up with other parts of app. Also by using function we can reuse it in many different contexts without repeating parts of the program text.

**Section 4 - Variables**

**4.1 Data types**

**What are the primitive data types in Objective-C? Why did Apple add a set of classes to handle other data types?**

A: Primitive data types in Objective-C are:

int - integer, doesnt require decimal point;

float - floating-point variable;

double - floating-point variable for large numbers;

char - single characters;

BOOL - Boolean type, return YES or NO

Apple took a core of the Objective-C language and they wrote code to make it easy to work with strings, dates, images, video, buttons etc.

**4.2 Working with numbers**

**Make a table of Objective-C primitive data types. Add numeric data types and their**

**properties to this table. Add char and BOOL (the character data types) to your table created above.**

|  |  |  |
| --- | --- | --- |
| short int | -32,767 | 32,768 |
| int | -2147483648 | unsigned  4294967295 |
| long | -9223372036854775808 | unsigned  18446744073709551615 |
| float | 32 bit |  |
| double | 64 bit |  |
| char | 1 byte integer |  |
| BOOL | YES or NO |  |

**4.4 Variable scope**

**Describe in your own words what the scope of a variable is in Objective-C**

A: There are local and global variables. Local are variables that exist only inside the block/statement they are defined. If variable declared inside {} it only exist inside of that block and we can't redefine it in other blocks of program. Global variables are accessible for any part of code, it can be used or changed anywhere in program.

**4.5 Enumerations**

**What does "enum" allow you to do?**

**Enum method allows us to define some kind of data type with a restricted range of possible values. It's a user-defined type consisting of a set of named constants called enumerators. Features of enumeration:**

**a) restricts values that enum variable can take;**

**b) force developer to think about all possible values that the enum can take;**

**c) increase readability of code.**

**4.6 Using typedef**

**When would you define your own data type versus use an enum?**

Typedef method allows us to assign a new name to an existing data type. It mostly used just to shorten some code, to redefine some data type that are very cumbersome to write. So using typedef make it easier to create variables of a particular type.

**4.7 Preprocessor directives**

**Describe the three common preprocessor directives, #import, #define, and #if DEBUG. Come up with one example where you would use each.**

A: Compiling a file of Objective-C is done in two stages, where on the first stage preprocessors runs through the file; and output from preprocessor then goes into real compiler.

- # import - requests that preprocessor read a file and add it to its output. For example, to use some functions defined in other file we would use #import <new.h> preprocessor.

- #define - allows us to create some value that will be replaced everywhere in program. For example, we can set value of Pi=3,14159265358979 and whenever program encounter Pi it will replace it for that value;

- #if DEBUG - scan through the code and ask if we are compiling in DEBUG mode, if yes - it will leave this code, otherwise it will treat code like that line never existed. It's useful for debugging tasks.

**4.8 Working with strings**

**Define the same string using both NSString and C-style string syntax. Describe the purpose behind each part of your definition.**

**A: To define string with NSString style we would write:**

**NSString \*message=@"Hello"**

**where NSString indicates type of object, \* - pointer, which points to area of memory where the content of the string are stored, message - name of object, and @"Hello" - value of object, @ - indicates that this is an object. In C-Style we would have to declare array of characters:**

**char message[6]="Hello"**

**here we declare an array with 5 letters and null terminator '\0'.**

**Section 5 - Classes**

**5.1 Introduction to object orientation 7m 36s**

**Create an encapsulated (including generalized attributes and behavior) description of a "mobileMakersParticipant" class. Instantiate a single object representing yourself as a member of this class.**

**We have class mobileMakersParticipant, which consists attributes and methods:**

|  |
| --- |
| **mobileMakersParticipant** |
| **name**  **age**  **gender** |
| **study**  **programming**  **experience** |

**Single object representing me:**

|  |
| --- |
| **mobileMakersParticipant** |
| **Steven**  **24**  **male** |
| **study**  **programming**  **experience** |

**mobileMakersParticipant \*steven=@”some data”**

**5.2 Using objects and pointers 6m 38s**

**What is the pointer's role in instantiating an object from a class? How is a pointer different than a primitive?**

A: To create an object from a class we need to declare pointer to it. Primitive variables hold the value of it directly. In case with object variable is still claimed in memory, but it doesn't hold the object itself, instead it holds an address (pointer) that points to different area of memory where that object itself exists. With primitive we know how much space in memory it holds, but with object we don't know it. That’s why we need more flexible way to manage objects and pointers allow us to do that.

**5.3 Messages and methods 6m 44s**

**What is the main difference between Objective-C's messages and method calls in other languages? How can this difference be seen as an advantage while programming?**

A: In Objective-C we call method sending a message:

[myObject someMethod] - here we are sending message someMethod to the object myObject. While programming using message method code becomes more readable instead of a method call, where we just passing multiple parameters. With message method we know our parameters, thus creating more readable and maintainable code.

**5.4 Using existing classes in the foundation framework 8m 40s**

**What's the difference between a class method and an instance method? EXPLORE: Try typing "NSD…" into your code window. Use the autofill feature and select a single class name that starts with those three letters. Once the name has been auto-completed, use the handy shortcut (Option + click) and investigate the class whose name just got printed to the screen. Examine the task list for this class. Do this a few more times until you're familiar with the process, or until you've exhausted your curiosity, whichever comes last.**

A: Class methods are designed to work on the entire class itself, whereas instance method designed to work with an instance. + indicates that these are class method, - indicates that these are instance class.

**Section 6 - Memory Management**

**6.1 What's new with memory management? 1m 45s**

**Let it soak in. No questions for this one.**

**Thank you!**

**6.2 Memory management in Objective-C 6m 58s**

**What is the relationship between a pointer to an object, a block of memory, and the owning and releasing process? Can you come up with an analogy for this relationship?**

Once object is created some block of memory is cleaned for that object and we have a pointer to that area of memory. That object gets retainCounter, which indicates who is responsible for this object. Once object is created the retainCount is equal 1, when we release object retainCount gets 0. Once its zero, Objective-C erase object from block of memory and it becomes available to use.

**6.3 Object creation 7m 31s**

**What does the new method do when used to create an object instance of a class? Why do**

**we avoid using this method? How long is an object's lifetime?**

A: When we use the New method it creates an object instance for us. It does it in two stages: allocation (we get some block of memory to hold an object) and initialization (initializing variables). Eventually, New method returns the address of memory, where object will be stored. But if we need to define those 2 stages more explicitly we use [alloc [init]] method. Often its necessary to have several inits methods, so we can do allocation and then implement slightly different initializers. The lifetime of object is time between its creation (instantiation) until its destruction, when we release object or its released automatically.

**6.4 Using autorelease pools 5m 14s**

**How does the autorelease pool work? How and when can you use it deliberately?**

A: When we call autorelease pool, instead of immediate reducing retainCount, it adds entry to list of objects that need to have release called on them at some point later. When program execute [pool drain] line it goes through that list and call release on every single object in the list. Usually, in applications by default autorelease pool is drained repeatedly at the end of every event loop. If we own object we have to release it deliberately.

**6.5 Apple autoreleased objects 3m 39s**

**What does NARC stand for? Why is it important to remember this?**

A: NARC stand for new, alloc, retain and copy. If we use any of this word that means we own that object, once we own object we are responsible for releasing it.

**6.6 Introduction to Automatic Reference Counting (ARC) 4m 43s**

**What does ARC save us from having to do? How does it keep us from having to make this**

**extra effort?**

A: ARC (Automatic Reference Counter) is great tool for Memory Management Code. With ARC we don't have to write retain and release calls anymore. ARC is able to scan through the application and determine all the possible paths through, figure out where objects have being used and synthesize retain and release calls at the best point in the code. So it prevents from dangling points and memory leeks, which can occur if we release object too soon or too late.

**6.7 What ARC manages 2m 42s**

**What are the differences between ARC and garbage collection? What makes these**

**differences advantageous?**

A: ARC is done at compile time, while garbage collectors need to be constantly run port of the runtime that monitors memory usage of the program. Also with garbage collection is non-deterministic, and ARC is completely predictable and it doesn't make any change to runtime mode.

**6.8 The rules of ARC 4m 20s**

**Why can you not release or dealloc memory when working with ARC?**

A: ARC is doing retaining and releasing of objects. While using ARC we can’t do memory deallocation or releasing objects, because it does everything automatically.

**Section 7 - Custom Classes**

**7.1 Creating your own classes 14m 1s**

**What are the two different sections used to create a class? What do they hold and what files are they placed in? CHALLENGE: Create a Tweet class for a twitter style app.**

A: There are 2 different sections:

a) the interface (.h file)

b) the implementation (.m file)

The interface part declares what this class is going to do, what methods and properties are available. And in implementation part we write the real code, how we are going to implement methods we declared in a header file.

Tweet.h--------------------

#import <Foundation/Foundation.h>

@interface Tweeter : NSObject

@property NSString \*message;

@property NSDate \*postDate;

@property NSString \*authorTweet;

-(void) postTweet;

@end

Tweet.m--------------------------

#import "Tweeter.h"

@implementation Tweeter

@synthesize message, postDate, authorTweet;

-(void) postTweet {

NSLog(@"%@, posted by %@ on %@", message, authorTweet, postDate);

}

@end

main---------------------------

#import <Foundation/Foundation.h>

#import "Tweeter.h"

int main(int argc, const char \* argv[])

{

@autoreleasepool {

Tweeter \*steven=[[Tweeter alloc]init];

...

[steven postTweet];

}

return 0;

}

**7.2 Defining methods 8m 36s**

**CHALLENGE: Define what should get passed in and what should get returned by each of your methods in your Tweet class above.**

**In the class above I just used one method:**

-(void) postTweet {

NSLog(@"%@, posted by %@ on %@", message, authorTweet, postDate);

}

Here minus sign before method means that it is instance type class, then in parentheses we specify return type (in this case void means that this method doesn’t return anything). Here postTweet is name of method. Then inside the curly braces {} we write what this method is actually going to do.

**7.3 Defining properties 7m 21s**

**How did Objective-C programmers handle instance variables before 2012? How are they**

**handled now? What got easier and what got obscured?**

A: Before 2012 to define the data for the class, it was necessary to define instance variable inside header file enclosed in curly braces {...}. Instance variable couldn't be accessed directly from outside the class they were defined it. So developers had to write code inside the class to manipulate it, but it couldn't be accessed or changed from a different class. Thus it was necessary to use Accessor Methods, which added two methods: setter and getter.

After 2012 @property and @synthesizying were added to Objective-C. Those methods automatically generate or synthesize accessor methods. But using Xcode 4.4+ we even don't need to declare @synthesizying method, compiler do all work.

**7.4 Defining initializers 12m 30s**

**What are initializers and why do we need to use them? Describe a situations when you can**

**rely on the standard init method and when you have to create your own custom initializer.**

A: Initializers set the instance variables of an object to some values. Initialization can allocate and prepare other global resources needed by the object. Every object that declares instance variables should implement an initializing method, unless we want to set variables to zero initial parameters. Standard method Init initialize its instance variables to usable values, it makes object ready to work. But Init has no argument, so if we need to pass some information from the method we are calling, we need to build custom initializer, thus passing initial information to argument.

**7.5 Using dealloc 5m 33s**

**Why can we have a dealloc method in a class when using ARC, but we can't call dealloc**

**manually oursevles when using ARC?**

A: Dealloc is method, which automatically called when object is deallocated (it doesn't have any owner). With ARC we are not allowed to call dealloc method of any object ourselves. But we can write it and let Objective-C runtime call this method when it's suitable. In a complex system when object is holding onto some kind of resource, we might need to add dealloc method to make sure that connection is properly closed before object is destroyed. So, we can write dealloc method but we don't call it ourselves and let program decide when to use it.

**Section 8 - Collections**

**8.1 Working with C-style arrays 7m 12s**

**What are the three constraints when using C-style arrays? Create a C-style array that holds the days of the week.**

A: There are 3 constains when using C-style arrays:

- no bounds checking (we need to claim memory explicitly);

- fixed size (once we created array, we can't change it);

- we can't mix different types of variables.

C-Style Array:

#include <stdio.h>

int main(int argc, const char \* argv[])

{

char \*week[63];

week[0]="Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday";

printf("%s\n", week[0]);

**8.2 Working with Objective-C array objects 8m 0s**

**What is the difference between a mutable and an immutable array? CHALLENGE: Create an immutable array containing the days of the week. Create a mutable array that contains the days of the week that you will be at Mobile Makers. Add the days of the week from the immutable array to the mutable array.**

A: Immutable array is unchangeable, once we created it, we can’t add or remove new objects from it. In Mutable array this problem is fixed. NSMutableArrays is subclass of NSArray, which is immutable.

#import <Foundation/Foundation.h>

int main(int argc, const char \* argv[])

{

@autoreleasepool {

NSArray \*days=[[NSArray alloc] initWithObjects:@"Monday", @"Tuesday", @"Wednesday", @"Thursday", @"Friday", @"Saturday", @"Sunday", nil];

NSMutableArray \*daysMm=[NSMutableArray arrayWithObjects: @"Monday", @"Tuesday", @"Wednesday", @"Thursday", nil];

NSLog(@"The days at Mobile Makers are: %@", daysMm);

[daysMm addObjectsFromArray:(NSArray\*)days];

NSLog(@"Combined days are %@", daysMm);

}

return 0;

**8.3 Using dictionaries 5m 55s**

**Create a dictionary that lists five or more events in your life and the accompanying year (or date if you want to get fancy) of the event.**

A:

#import <Foundation/Foundation.h>

int main(int argc, const char \* argv[])

{

@autoreleasepool {

NSMutableDictionary \*events=

[NSMutableDictionary dictionaryWithObjectsAndKeys:

@"was Born", @"1987",

@"went to School", @"1994",

@"went to University", @"2004",

@"moved to USA", @"2010",

@"start Mobile Makers Project", @"2013",

nil];

NSString \*eventName=@"2010";

NSLog(@"In %@ I %@", eventName, [events objectForKey:eventName]);

}

return 0;

}

**8.4 Fast enumeration 3m 27s**

**Use fast enumeration to log the timeline of the life events you described above to the**

**console.**

A:

#import <Foundation/Foundation.h>

int main(int argc, const char \* argv[])

{

@autoreleasepool {

NSMutableDictionary \*events=

[NSMutableDictionary dictionaryWithObjectsAndKeys:

@"was Born", @"1987",

@"went to School", @"1994",

@"went to University", @"2004",

@"moved to USA", @"2010",

@"start Mobile Makers Project", @"2013",

nil];

for (NSString \*eventList in events) {

NSLog(@"In %@ I %@", eventList, [events objectForKey:eventList]);

}

}

return 0;

}

**Section 9 - File Management**

**9.1 Introduction to file management in Objective-C 6m 44s**

**What can you do with files using the methods you are aware of that are available in**

**Objective C's Foundation class.**

A: In Objective-C there are a lot of classes that allow us to read, copy, save, remove files or some content, read through directory. As an example, NSFileManager method which called by init-alloc method. Also there are methods returning Boolean by which we can check if file exist or not, we can return attributes of a particular path, change file's path etc.

**9.2 Working with paths and URLs 7m 17s**

**What are the three parts of a URL? What are the advantages to using NSURL?**

A: URL (Universal Resource Locator) consists of 3 parts: Scheme (as http, https, ftp or file), Domain, which shows location of it and Path.

Advantages of using NSURL:

1) Faster than using string path, more efficient

2) Can trap errors

3) Used by more classes (there are a lot of new classes that will work using NSURL only but not path)

**9.3 Reading and writing strings 4m 38s**

**What would be a reason you would want to write a string to disk instead of just keeping it**

**memory?**

A: To make sure that all necessary information is saved. Sometimes we just need to save information without loading object. In a complex system when we work with large amount of data we might need to store information in some files, which might be used with another program later.

**9.4 Archiving objects 12m 41s**

**Why would you want to archive an object instead of writing the data to disk using the**

**techniques discussed previously?**

A: Archiving allows developer to break apart different pieces of objects, give them a key and store them to separate file. Thus, archiving helps create searchable, classified repository of information that includes type and value of each separate component. And if you need to pull up some information its very easy to access this information on hard disk.

**Section 10 - More Complex Classes**

**10.1 Inheritance and NSObject 8m 13s**

**How can you determine what methods you're inheriting from a super class? How do you**

**overide a method inherited from a super class?**

A: Basically subclass inherits all methods of superclass and then we add our modifications. We can check what methods are available for a particular class at "Class Reference" tab. To override method inherited in Superclass we need to write a new method in implementation body with the same signature as one in Superclass and define description method. Once we overrode the method it used in preference to the superclass method.

**10.2 Extending classes with categories 6m 31s**

**What is the difference between a category and an inheritance? What are the limitations of**

**using a category?**

A: The difference is that categories allow us to add our own methods to an existing class without subclassing it (unlike inheritance when subclass inherits all functionality from superclass). But using Category developer can't add a new instance variable, we just add functionality to class. Also with Category we can't provide safely override methods that already defined by the class itself or another category.

**10.3 Defining protocols 5m 14s**

**How are protocols useful?**

A: Protocols declare the methods expected to be used for a particular situation, by using it we can standardize interaction between objects simply providing list of methods we want object to perform. Protocol in Objective-C called by NSCodingProtocol method and it provides: initializing with a Coder (InitWithCoder) and encoding with a Coder (EncodeWithCoder). Protocols also useful when the class of an object is not known or it needs to be hidden.

**10.4 Dynamic typing 11m 33s**

**What are the advantages and disadvantages to dynamic typing?**

A: Dynamic typing occurs when program does not associate objects strictly with a specific type, but it is designed to identify object at the runtime. Advantages of dynamic typing that we can specify in advance methods for objects without knowing it's type, so dynamic typing is perfect for systems with changing or unknown requirements. Disadvantage of dynamic typing is that we can't check errors as early as in static systems, also it decreases runtime efficiency.