

# Classes IOS

Parte 1

X-Code

Prof. Agesandro Scarpioni



# Classes

- No Objective-C, as classes possuem dois arquivos, um com a extensão .h (header file) e outro com .m (messages).



# Classes

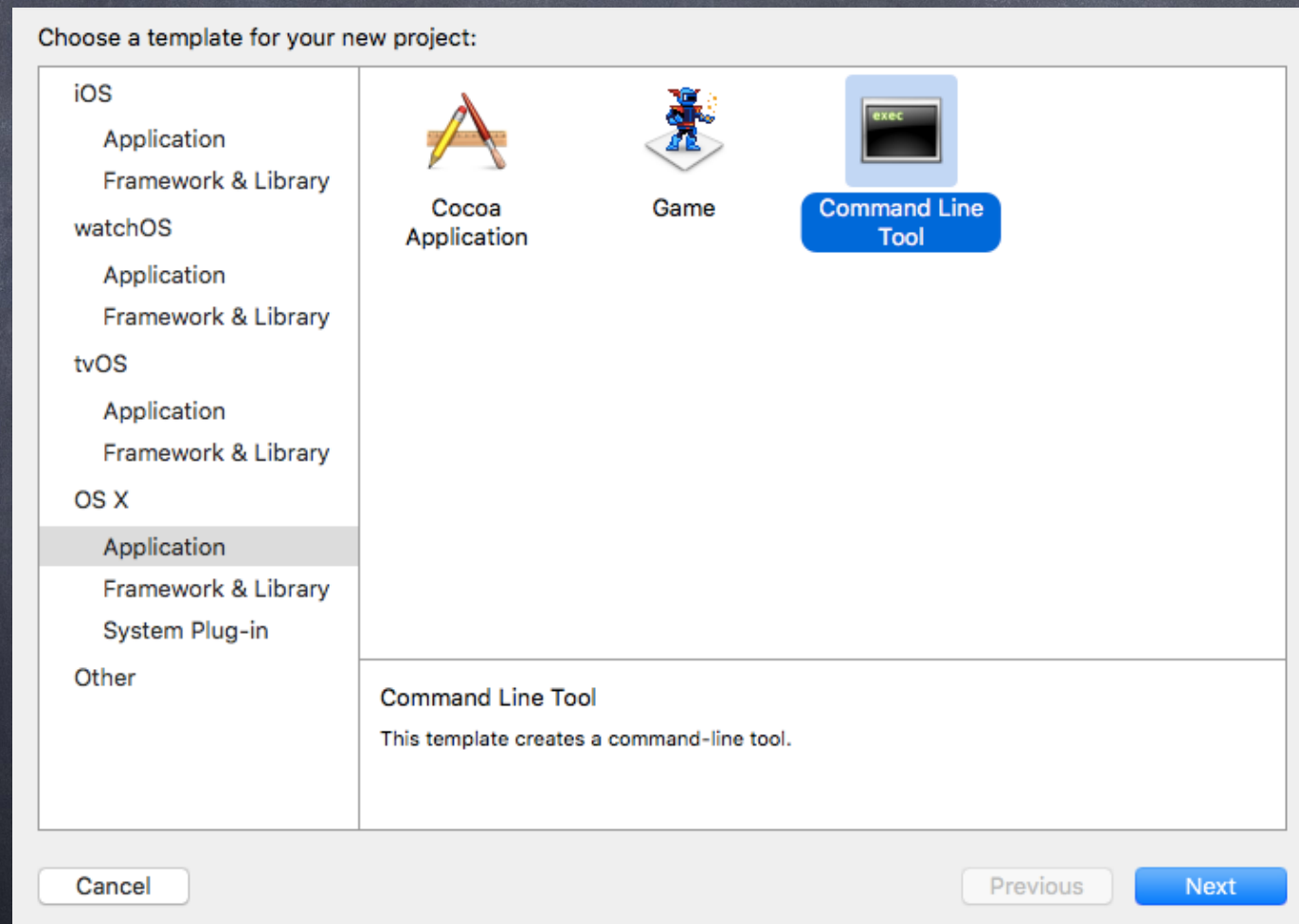
## Arquivo .h

- O primeiro .h é um arquivo de cabeçalho onde devemos declarar os atributos e métodos da classe, neste arquivo temos apenas a assinatura do método e as linhas devem terminar com ponto e vírgula (;).



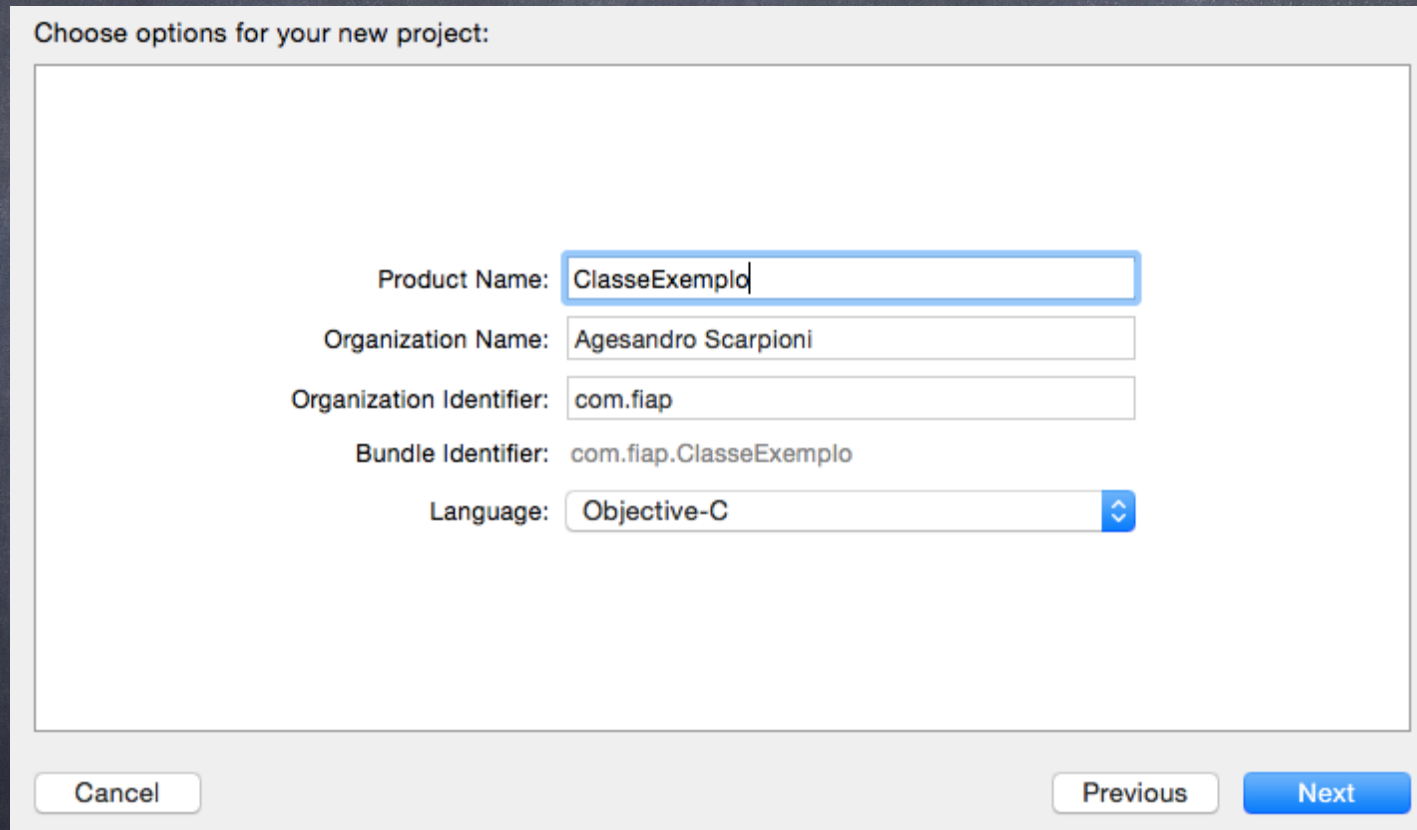
# X-Code

- Abra o Xcode e clique em: File --> New --> Project --> À esquerda no item OS X escolha Application, depois selecione Command Line Tool e clique em Next



# X-Code

- Informe o nome da aplicação em Product Name, preencha o Organization Name com seu nome, escolha Objective-C na linguagem e clique em Next.



The screenshot shows the 'Choose options for your new project' dialog box in Xcode. The dialog has a title bar with the text 'Choose options for your new project:'. Inside, there are five input fields arranged vertically. The first field is 'Product Name' with the text 'ClasseExemplo'. The second is 'Organization Name' with 'Agesandro Scarpioni'. The third is 'Organization Identifier' with 'com.fiap'. The fourth is 'Bundle Identifier' with 'com.fiap.ClasseExemplo'. The fifth is 'Language' with a dropdown menu showing 'Objective-C'. At the bottom of the dialog, there are three buttons: 'Cancel', 'Previous', and 'Next'.

Field	Value
Product Name	ClasseExemplo
Organization Name	Agesandro Scarpioni
Organization Identifier	com.fiap
Bundle Identifier	com.fiap.ClasseExemplo
Language	Objective-C

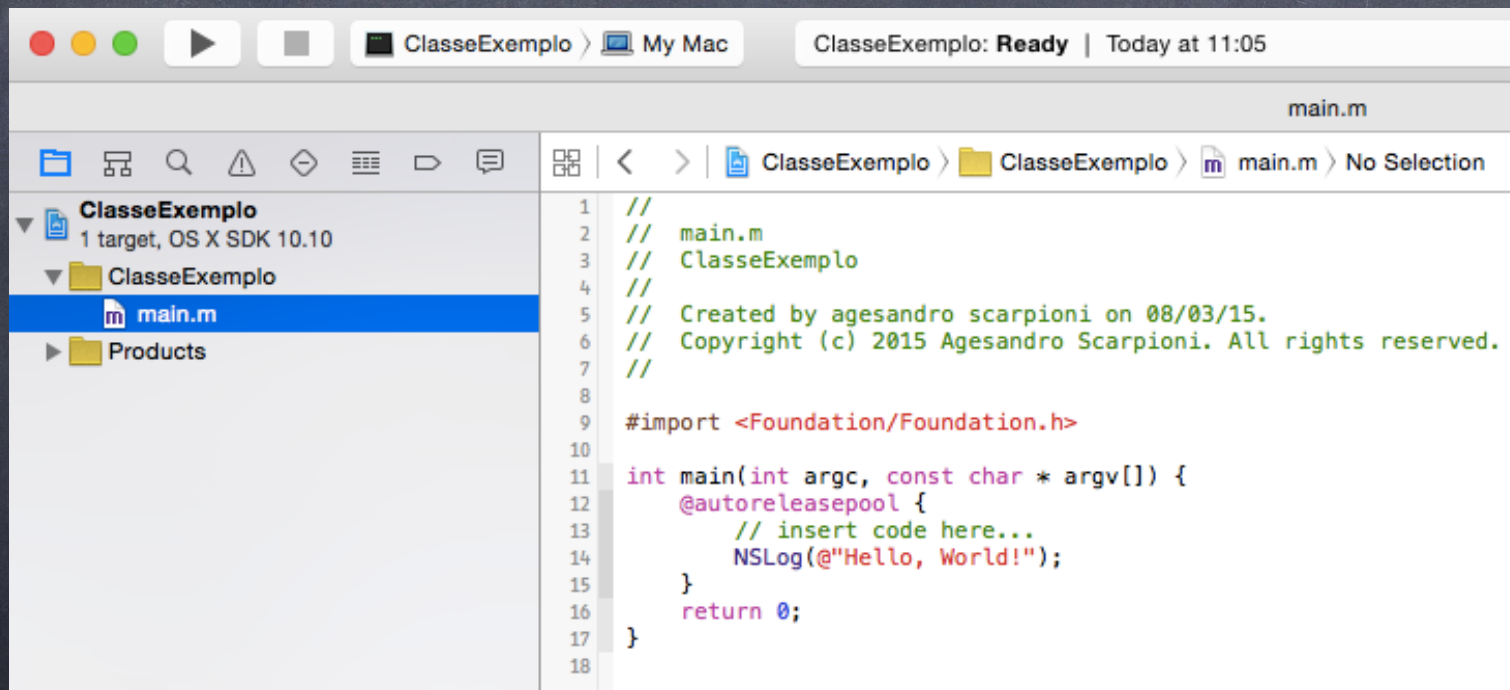
- Na tela seguinte salve na pasta padrão, clicando em Create.

**OBS:** O Organization Identifier é semelhante ao NameSpace no Visual Studio ou o Package no Java, Organization Name informe seu nome ou o nome da empresa.



# X-Code

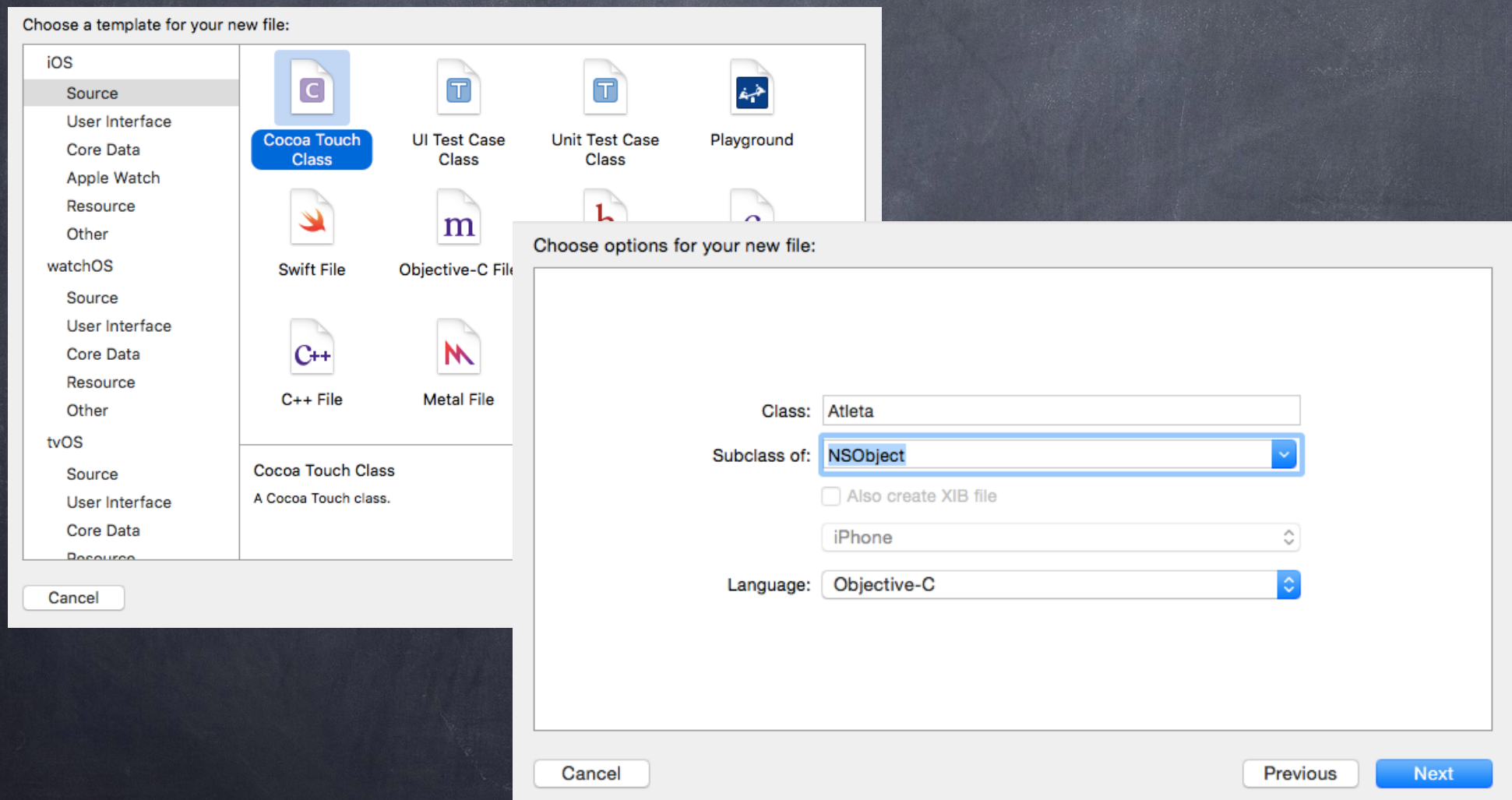
- No canto à esquerda selecione a pasta main abaixo do nome do projeto e do lado direito REMOVA a linha: NSLog(@"Hello, Word!?);
- OK, Vamos por a mão na massa e fazer nossos testes.





# X-Code

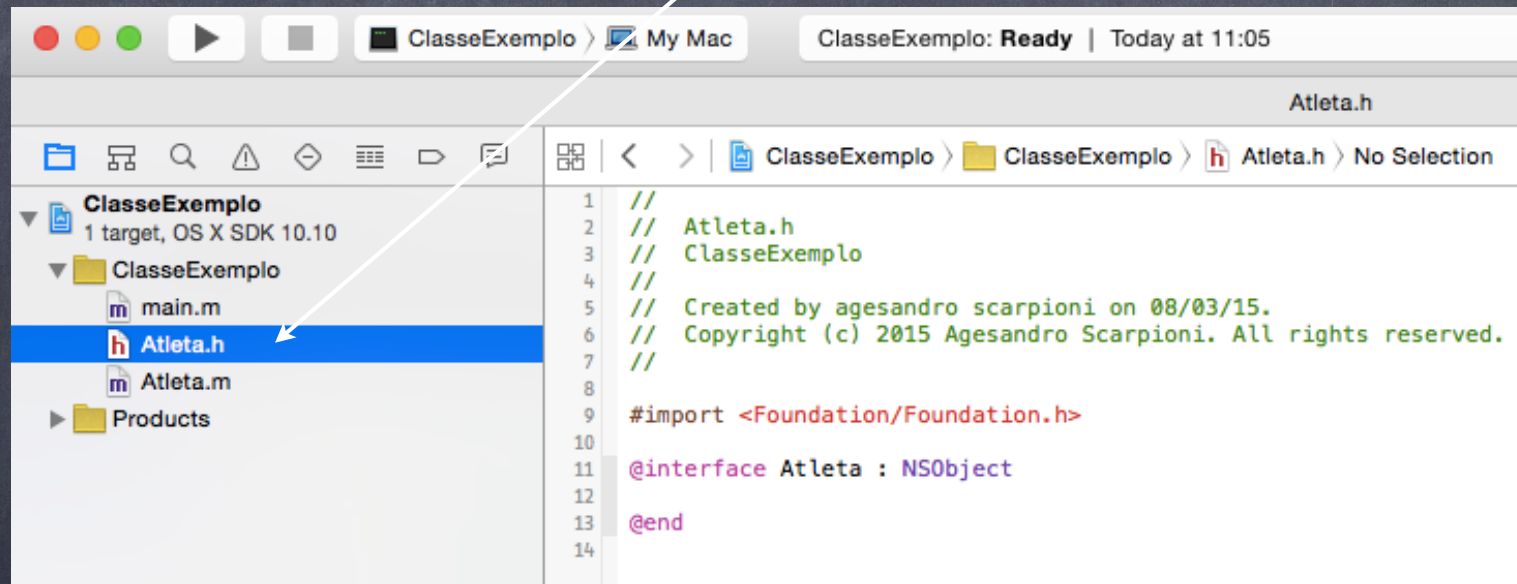
- Os passos abaixo ilustram como devemos criar a classe dentro de nosso projeto. Clicando em File → New File → Cocoa Touch Class → Sub classe de NSObject.





# Classes

Arquivo .h





# Classes

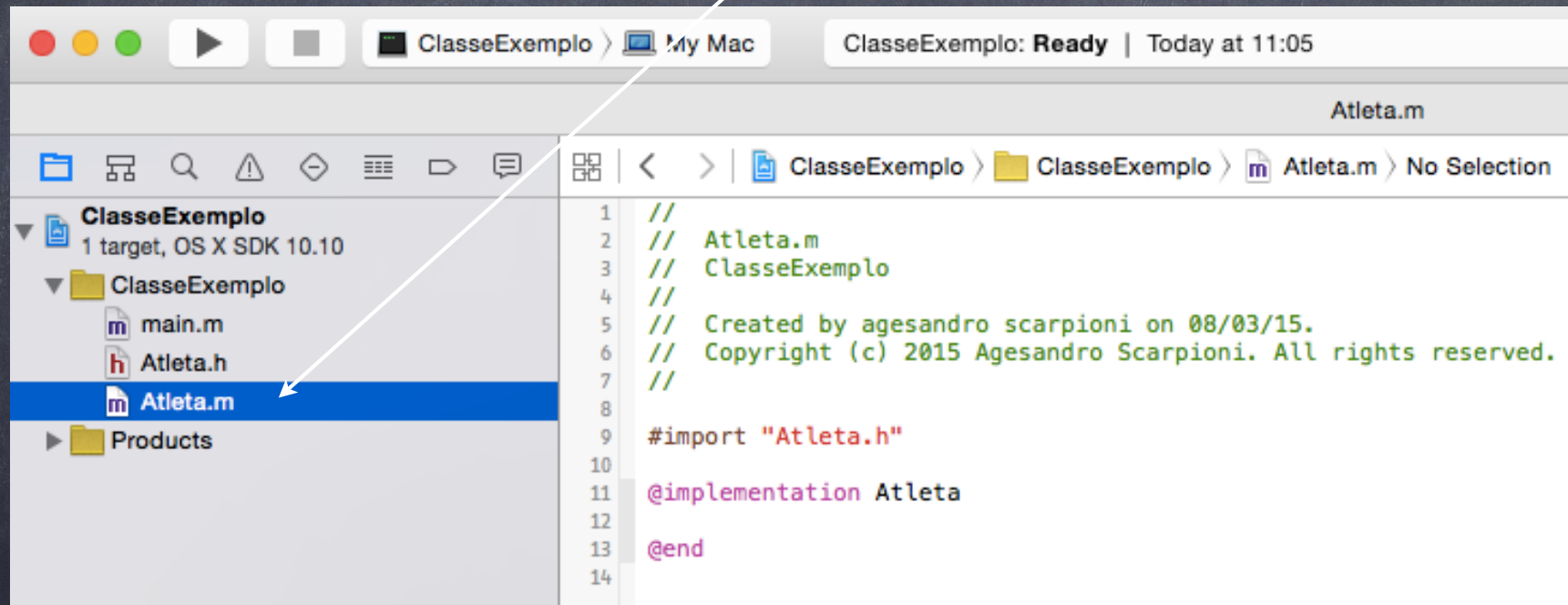
Arquivo .m

- O segundo .m é o local onde fazemos a implementação, ou seja, onde é digitado o código fonte.



# Classes

Arquivo .m





# Classes – Atributos

- No header file vamos declarar dois atributos entre chaves.
- Usamos o asterisco indicando um ponteiro para a área de memória, como todos os objetos são referenciados como ponteiros o atributo nome que é do tipo (NSString) precisa do asterisco, a idade é um tipo primitivo e não precisa do asterisco.



# Classes – Atributos

## Atleta.h

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

@interface Atleta : NSObject{
    NSString *nome;
    int idade;
}

@end
```

**OBS:** Veja como é declarada uma herança no Obj-C, os ":" após o nome da classe "Atleta" indica que Atleta é filha de NSObject e como no Java, só é possível herdar de uma única classe.



# Classes – Atributos

- Todos os métodos que serão públicos devem ser declarados no header file, devem iniciar com “-”, o hífen serve para iniciar a declaração de métodos.
- Para métodos da instância do objeto usamos “-”.
- Para métodos de classe usamos “+”, semelhante a métodos estáticos do Java ou C#.
- Por default os atributos de uma classe são protected, para termos acesso a estes atributos devemos declarar o get e set para cada um.



# Classes – Atributos

## Atleta.h

```
1 //
2 // Atleta.h
3 // ClassesExemplo
4 //
5 // Created by agesandro scarpioni on 20/04/13.
6 // Copyright (c) 2013 Agesandro Scarpioni. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10
11 @interface Atleta : NSObject{
12     NSString *nome;
13     int idade;
14 }
15
16
17 - (void) setName:(NSString *)_nome;
18 - (NSString *) getName;
19 - (void) setIdade: (int)_idade;
20 - (int) getIdade;
21
22 @end
23
```

**OBS:** No arquivo .h temos apenas as assinaturas dos métodos, precisamos implementá-los no arquivo .m



# Classes – Atributos

Implementando getters e setters no arquivo .m

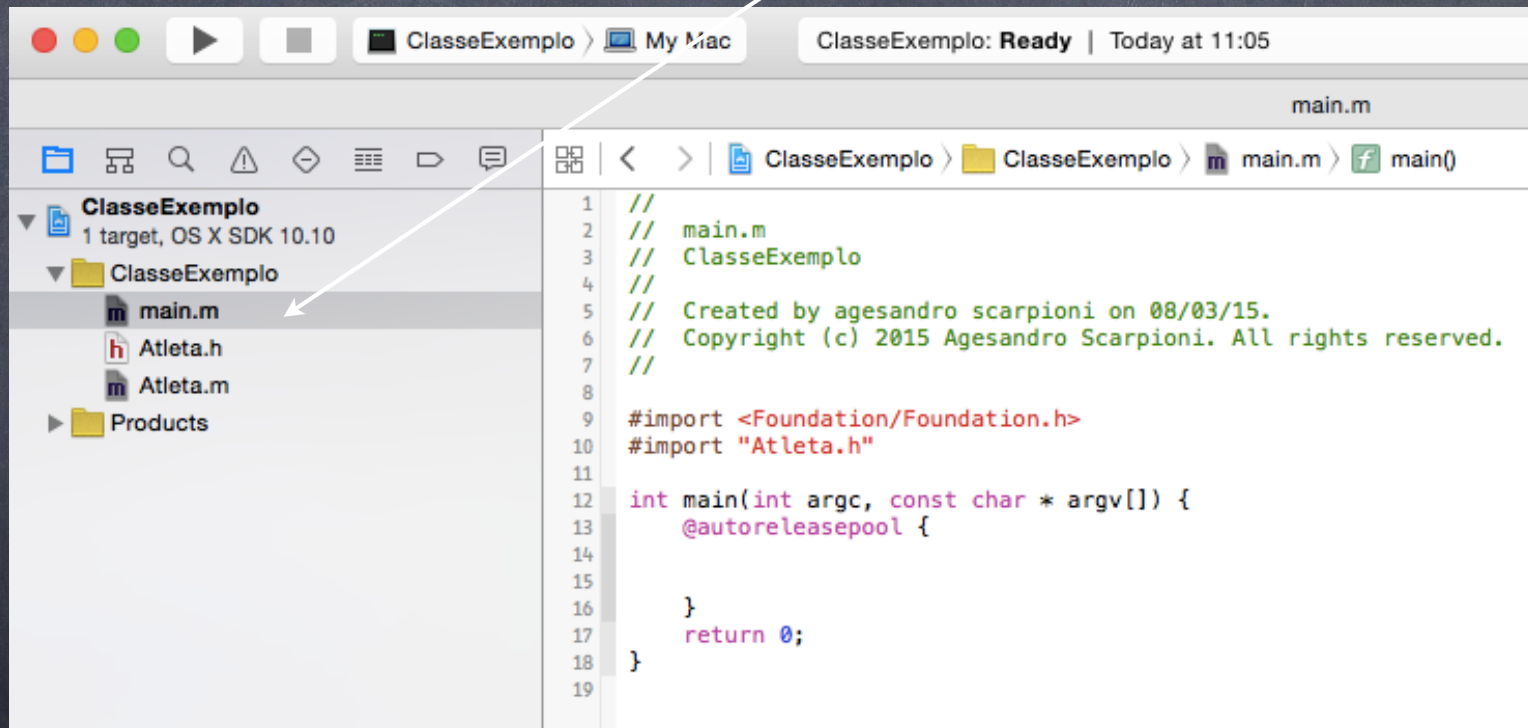
```
1 //  
2 // Atleta.m  
3 // ClassesExemplo  
4 //  
5 // Created by agesandro scarpioni on 20/04/13.  
6 // Copyright (c) 2013 Agesandro Scarpioni. All rights reserved.  
7 //  
8  
9 #import "Atleta.h"  
10  
11 @implementation Atleta  
12 - (void) setName:(NSString *)_nome{  
13     nome=_nome;  
14  
15 }  
16 - (NSString *) getName{  
17     return nome;  
18  
19 }  
20 - (void) setIdade: (int)_idade{  
21     idade =_idade;  
22  
23 }  
24 - (int) getIdade{  
25     return idade;  
26  
27 }  
28 @end  
29
```

**DICA:** Para ganhar tempo copie as assinaturas dos métodos no .h, cole no .m retire o ";" e coloque as chaves { }; depois basta fazer as atribuições e os return`s.



# Classes – Atributos

## Implementando o Main



**DICA:** Antes de iniciar, devemos importar o arquivo Atleta.h



# Classes – Atributos

## Implementando o Main

```
1 //
2 // main.m
3 // ClassesExemplo
4 //
5 // Created by agesandro scarpioni on 20/04/13.
6 // Copyright (c) 2013 Agesandro Scarpioni. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10 #import "Atleta.h"
11
12 int main(int argc, const char * argv[])
13 {
14
15     @autoreleasepool {
16         Atleta *a = [[Atleta alloc] init];
17         [a setName:@"José da Silva"];
18         [a setIdade:25];
19         NSLog(@"Iron Man %@ %d anos", [a getName],[a getIdade]);
20
21
22     }
23
24     return 0;
25 }
26
27 |
```



# Classes – Atributos

## Informações importantes

- O símbolo # indica um pré-processamento, o utilizamos para importar o arquivo Atleta.h pois o mesmo contém todas as definições da classe. Este símbolo também é utilizado em outras situações como: #define, #warning, #pragma mark, etc.
- Utilizamos um método estático chamado alloc que foi herdado da classe NSObject, ele retorna uma referência para uma área de memória, desta forma obtemos uma nova instância.



# Classes – Atributos

## Informações importantes

- O init que é o construtor do objeto, termina a inicialização do objeto.
- O init é algo bem próximo aos construtores, ele é um método normal, não existe em Obj-C os tradicionais construtores como no Java que utilizamos para o construtor o mesmo nome da classe, ou no VB que utilizamos o Public Sub New.



# Classes – Atributos

## Informações importantes

- A sintaxe para chamar métodos em Objective-C é [objeto método] utilizando colchetes, como nos exemplos: [a setIdade:25]; [a getIdade]; .
- Para passar parâmetro para o método utilizamos dois pontos, exemplo: [a setIdade:25]; .
- Parâmetros do tipo String precisam ser passados para o método com "@" antes do texto. Exemplo: [a setNome:@"José"];
- Os parâmetros do NSLog %@ %d representam respectivamente uma NSString e um inteiro.



# Classes – Atributos

## Informações importantes

- O gerenciamento de memória dos objetos no Obj-C é chamado de contador de referências (Reference Counting), o método `release` é o destrutor do objeto e sempre é chamado quando o contador de referência do objeto chega em zero. Nós não o chamamos porque iniciamos o projeto com este gerenciamento automático (Desde o Xcode5 isto é padrão), não precisamos mais nos preocupar em liberar o objeto da memória. Caso contrário seria necessário chamar o método destrutor para o objeto criado. Em nossa classe `Atleta` seria: `[a release];`.



# Classes – Atributos

## Linha do Tempo

- Se você desabilitar o gerenciamento automático de memória, sempre deverá chamar o método `release` quando chamar o `alloc` para criar um objeto, não vamos nos preocupar com isso no Xcode 6 vamos deixar esse gerenciamento automático, hoje ele é Default para a forma automática.
- No Xcode 4 você tinha que selecionar o gerenciamento automático de memória marcando um checkbox com a frase "Use Automatic Reference Counting)", já no Xcode 5 o ARC já vinha habilitado.



# Classes – Atributos

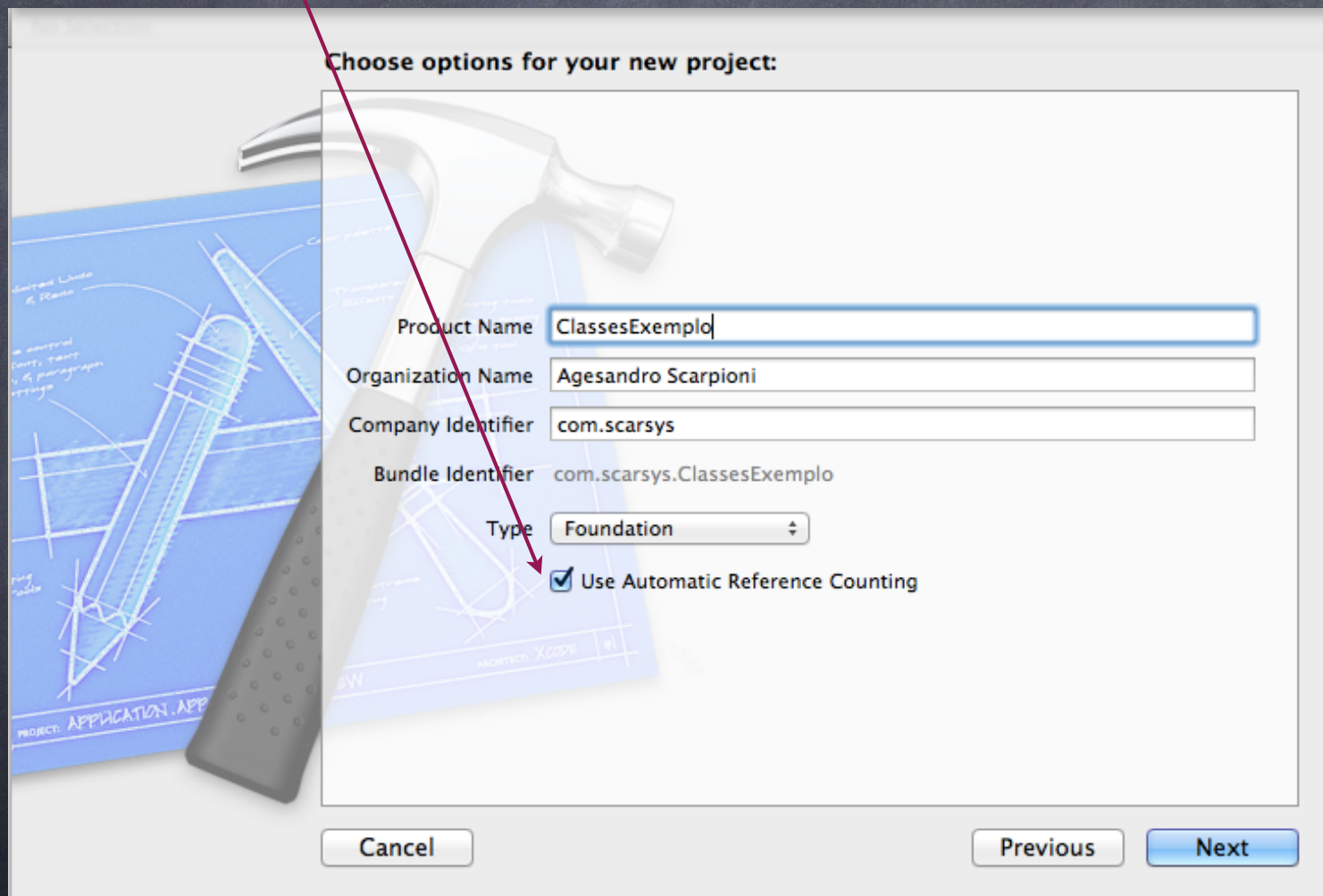
## Linha do Tempo

- Nas versões mais velhas, ou seja anteriores ao Xcode 4, nós tínhamos que gerenciar a memória de forma manual.
- Na versão 4 do Xcode nós tínhamos que definir que iríamos trabalhar com o gerenciamento de memória automática (veja o checkbox marcado no próximo slide).
- Desde a versão 5 do Xcode o gerenciamento automático de memória está habilitado por padrão.



# Classes

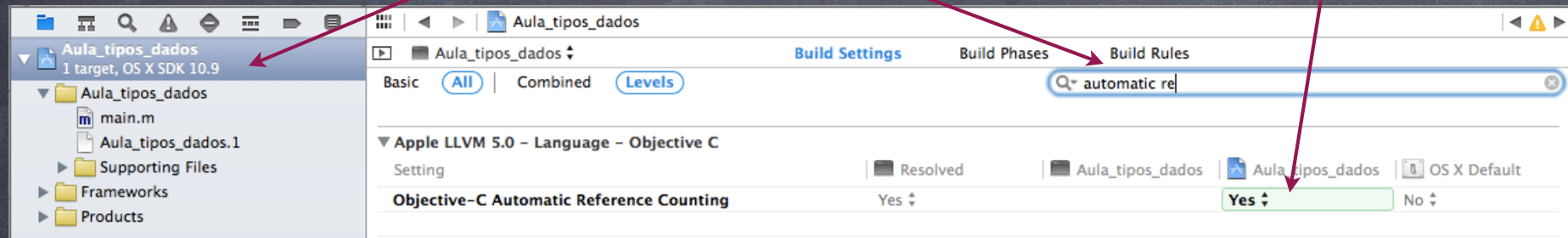
Como era o check box para utilizar o ARC no Xcode 4.



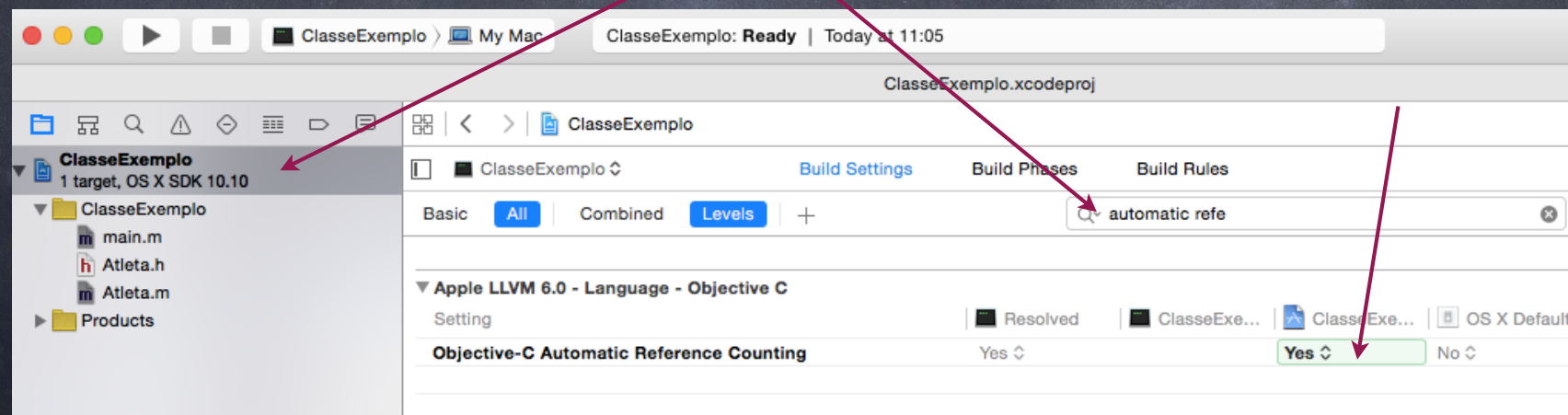


# Classes

Onde ficava o ARC no Xcode 5



Onde fica o ARC no Xcode 6





# Prática

Criação de um programa para testarmos todos os conceitos deste tópico.

- Crie um projeto novo e crie uma classe chamada Enfermeira.
- Esta classe deve possuir atributos do tipo NSString, float, bool e int.
- No Main instancie a classe, passe algumas informações para os atributos do objeto e exiba o resultado com NSLog, se você quiser ao invés de chamar os getters na linha do NSLog você pode criar novas variáveis, passar os getters para as variáveis e usá-las no NSLog.



# Próxima aula

- Métodos do tipo void e function com parâmetros e construtores