FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

Programação para Dispositivos Móveis II

Análise e Desenvolvimento de Sistemas - UNIFOR

Glauco Aquino
glauco.aquino@gmail.com

Fundamentos de Programação em Objective-C

- Objective-C: classes,métodos e propriedades
- Test Driven Development no XCode
- Objective-C: Gerenciamento de Memória(MRR e ARC)

Próximo conteúdo:

- DesignPatterns: Adapter,Decorator,Singleton,Memento
- Objective-C: protocolos, categories e blocos

Objective-C

- Objective-C é a linguagem de programação principal que você usa ao escrever software nativo para iOS.
- É um *superset* da linguagem de programação C e fornece capacidades orientadas a objetos e um *runtime* dinâmico.
- O Objective-C herda a sintaxe, tipos primitivos, e as instruções de controle do C, mas também adiciona sintaxe para definição de classes e métodos.
- A sintaxe de objetos herda as características do Smalltalk, sendo baseada no envio de mensagens para os objetos.

Definindo Classes em Objective-C

- Em Objective-C, as classes são definidas em blocos distintos: *interface* e *implementation*.
- A interface de classe define a maneira como você pretende que outros objetos interajam com as instâncias de sua classe. Ela representa a interface pública do objeto, descrita separadamente do comportamento da classe.
- O bloco *implementation* define o comportamento interno da classe, contendo métodos que respondem às mensagens definidas na interface, além de métodos não expostos publicamente (de uso interno).
- A implementação é feita em um arquivo de código-fonte com a extensão .m
- Para que uma classe possa estar acessível para outras classes em outros arquivos, sua interface pública deve ser separada em um arquivo de header com a extensão .h

Declarando métodos

- Declarações de método indicam as mensagens de um objeto pode receber
- A sintaxe básica para declaração de métodos é:

```
- (return_type) method_name;
```
(para métodos de instância)

```
+ (return_type) method_name;
```
(para métodos de classe)

- *Parâmetros podem ser incluídos utilizando a seguinte sintaxe:*

```
- (return_type) method_name:(param_type)param_name;
```

- Métodos com vários parâmetros são declarados como *multi-part*:

method_name(param1_name, param2_name, param3_name) :return_type

```
- (return_type)method_nameParam1:(param1_type)param1_name
param2:(param3_type)param3_name param3:(param3_type)param3_name;
```

Diretrizes para declaração de métodos

- Devem utilizar capitalização *camel-cased* (a primeira palavra em caixa baixa e as demais com a inicial maiúscula)
- *Multi-part methods* devem incluir preposições ou conjunções para formar a mensagem (exceto antes de atributos da classe)
- Não se deve utilizar o prefixo *get* antes de métodos que retornem atributos do objeto
- **Coding Guidelines for Cocoa** em http://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CodingGuidelines/Articles/NamingMethods.html

Declarando propriedades

- Objetos têm atributos destinadas para acesso público
- O acesso ao valor de um atributo é feito através de métodos chamados de *Accessor Methods*
- O Objective-C inclui suporte a membros chamados *declared properties*, que representam propriedades de uma instância, através das quais os acessor methods são definidos e utilizados.
- A sintaxe para a declaração de uma propriedade é:

```
@property (property_attributes) attribute_type property_name;
```

- *property_attributes* indicam de que forma os dados estão acessíveis (*readonly*, *readwrite*), concorrência (acesso em *multi-threading*) e armazenamento (gerenciamento de memória).

Trabalhando com objetos (1/4)

- Objetos sempre são acessados através de ponteiros para classe do objeto (ou um ponteiro para uma superclasse)
- Os métodos são invocados utilizando a notação de colchetes, com o qualificador (variável ou classe) à esquerda, e o método e parâmetros à direita:

```
[variable_name method_name];
[variable_name method_nameParam1:p1 param2:p2 param3:p3];
[class_name class_method_name];
[class_name class_method_nameParam1:p1 param2:p2 param3:p3];
```

- A notação de ponto é utilizada somente para propriedades:

```
atribute_type variable_name = object_variable_name.property_name;

object_variable_name.property_name = expression;
```

Trabalhando com objetos (2/4)

- Objetos são instanciados em dois passos: alloc e init
- O método alloc é chamado a partir da classe e é responsável por alocar memória para o objeto e inicializá-lo com os valores padrão
- O método init (ou alguma variação) é responsável por inicializar o objeto e torná-lo pronto para uso

```
object_type * variable_name = [object_type alloc];
variable_name = [variable_name init];
```

- Métodos podem fazer chamadas aninhadas, usando o retorno de um método ou propriedade, como qualificador para próxima chamada

```
object_type * variable_name = [[object_type alloc] init];
[variable_name.property_name method_name];
```

Trabalhando com objetos (3/4)

- No escopo de um método (no bloco de implementação) o objeto que está executando a ação (respondendo a mensagem) é acessado pela variável de contexto *self*.

```
@implementation class_name
-(void) method_name {
    [self other_method_name];
    self.property_name= value ;
}
@end;
```

Trabalhando com objetos (4/4)

- Em Objective-C, todos os métodos são polimórficos, e para acessar a implementação da superclasse, ele deve ser chamado a partir de *super*:

```
-(void) method_name {
   [super other_method_name];
  ...
}
-(return_type) other_method_name {
   return_type variable_name = [super other_method_name];
  ...
}
```

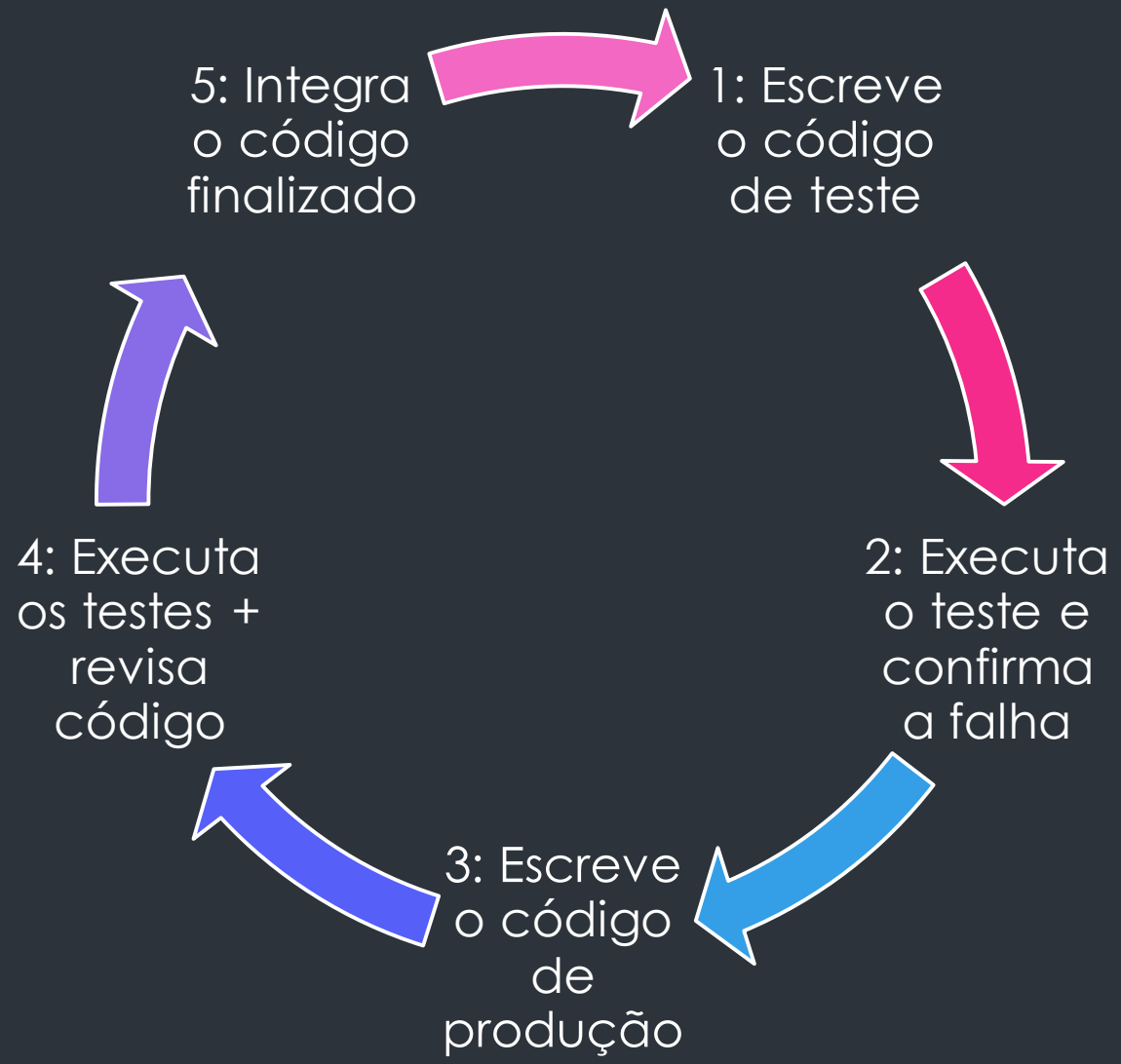
Test Driven Development

[Astels 2003] define o TDD como sendo um estilo de desenvolvimento onde:

- Uma suíte exaustiva de testes de programadores é mantida;
- Nenhum código entra em produção a não ser que tenha testes associados;
- Os testes são escritos antes;
- Os testes determinam que código precise ser escrito.

- A orientação a testes implica que os programadores deverão criar testes de unidade que definam requisitos de uso do objeto antes de escrever o código.
- As classes de testes baseiam-se na verificação de um comportamento esperado da classe através de asserções (verdadeiras ou falsas) e são executados sobre um *engine* automatizada.

O processo geral do TDD



Test Driven Development no XCode

- Para realizar testes de unidade no XCode, o projeto deve incluir pacotes de teste de unidade (targets separados para o código de teste).
- Classes de teste (Test Suits) são classes que implementam um ou mais métodos de testes para diferentes situações sobre uma classe específica. No XCode, as classes de testes devem ser criadas usando SenTestKit.
- Os testes de unidade são executados no próprio ambiente do XCode e os resultados são acompanhados na IDE.
- **A sample iPhone application with complete uni test** em http://www.cocoawithlove.com/2009/12/sample-iphone-application-with-complete.html

Objective-C e Gerenciamento de Memória

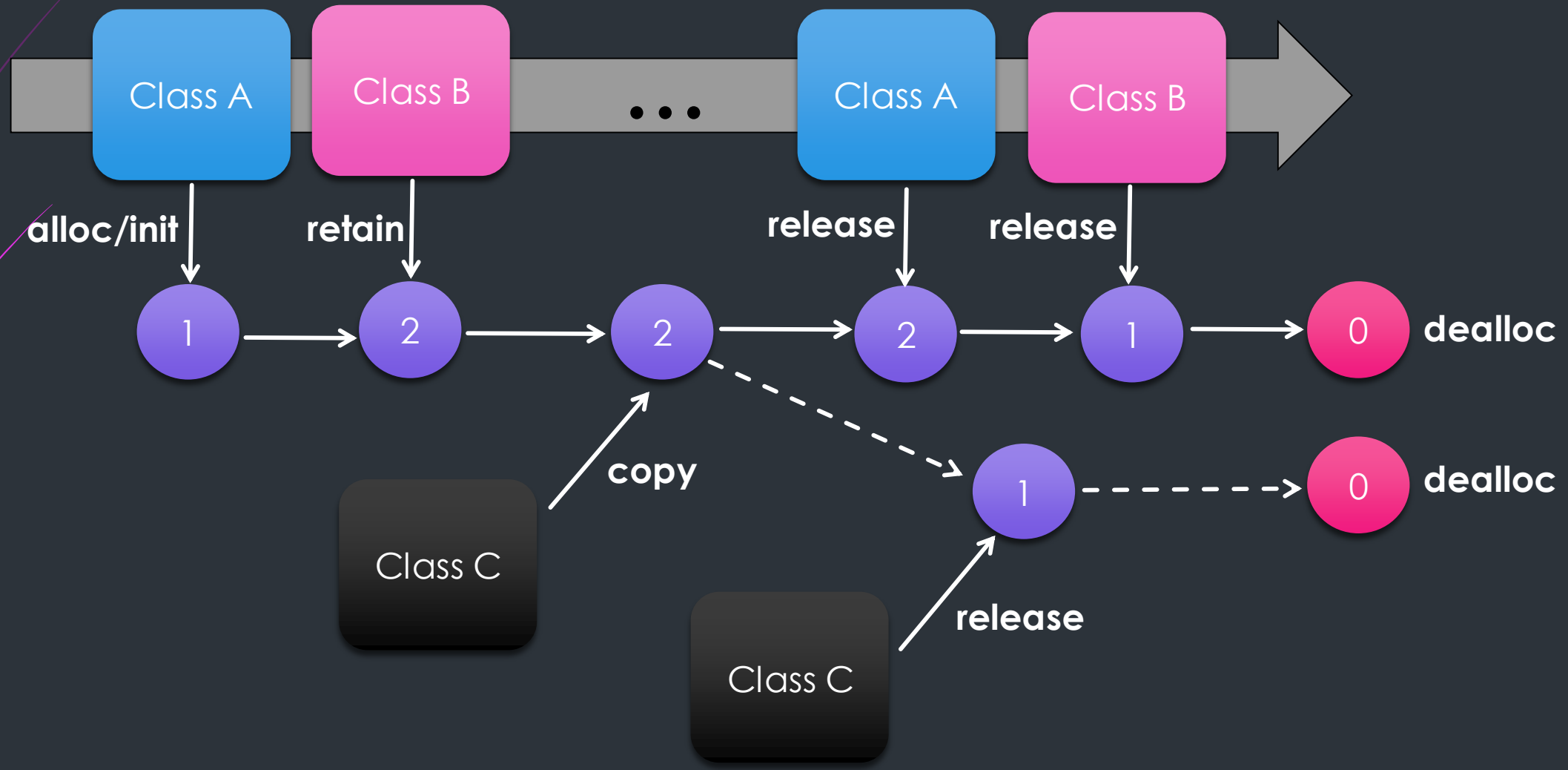
Em iOS, não há gerenciamento de memória através de coleta de lixo (*garbage collection*). Ao invés disso, é utilizado um modelo de gerenciamento de memória chamado de "Contagem de Referências".

Objective-C oferece dois métodos de gerenciamento de memória do aplicativo:

- Manual Retain-Release (MRR), em que você gerencia explicitamente a memória por balanceamento entre retenções e liberações.
- Contagem de referência automática (ARC), em que o sistema usa o mesmo mecanismo do MRR, mas insere o código apropriado em tempo de compilação

Você é fortemente encorajado a usar ARC para novos projetos!

Contagem de Referências



Regras para Manual Retain-Release

- Você é o proprietário de qualquer objeto que você crie através de alocação ou cópia
- Se você não é o criador de um objeto, mas quer garantir que ele permanece na memória para que você possa usar, você deverá retê-lo
- Se você cria um objeto ou retém, você é responsável por liberá-lo (obrigatoriamente) quando você não precisar mais dele
- Se você não criou ou fez a retenção de um objeto, você nunca deverá liberá-lo
- Ao criar um objeto (por alocação ou cópia), você poderá incluí-lo no Autorelase Pool. Assim, você não precisará fazer a liberação explicitamente.

Automatic Reference Count (ARC)

- Contagem de Referências Automática (ARC) é um recurso do compilador que fornece gerenciamento de objetos Objective-C automático.
- O ARC funciona através da adição de código em tempo de compilação para garantir que os objetos viver tanto tempo quanto for necessário, mas não mais. Ele segue as mesmas convenções do MRR, mas adicionando o gerenciamento de memória apropriado chama para você.

Com o ARC, não é possível utilizar o gerenciamento explícito (*retain*, *release* e *autorelease*) e deve-se utilizar dois novos qualificadores de tempo de vida:

- **strong**: o objeto permanece "vivo" enquanto houver referência forte para ele;
- **weak**: não mantém o objeto referenciado "vivo". Além disso, ela é definida automaticamente como **nil**, quando não há referências *strong* para o objeto

A seguir...

- DesignPatterns: Adapter, Decorator, Singleton e Memento
- Objective-C: protocolos, categories e blocos