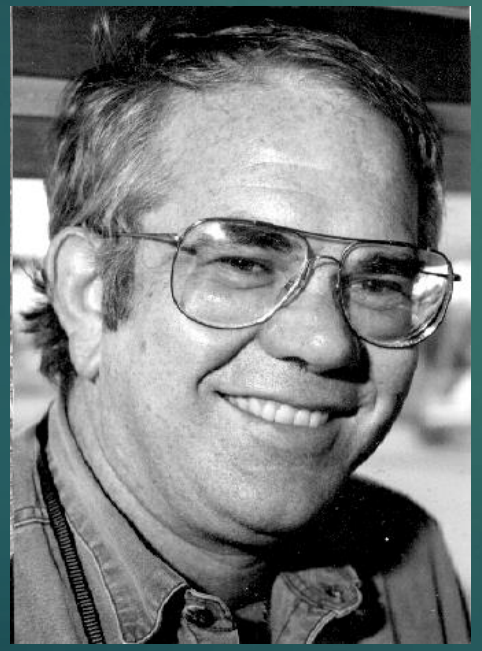
Objective-C

FERNANDO CELESTRINO RIBEIRO
FLAVIO DUARTE
IVO ZANDONADI NICCHIO

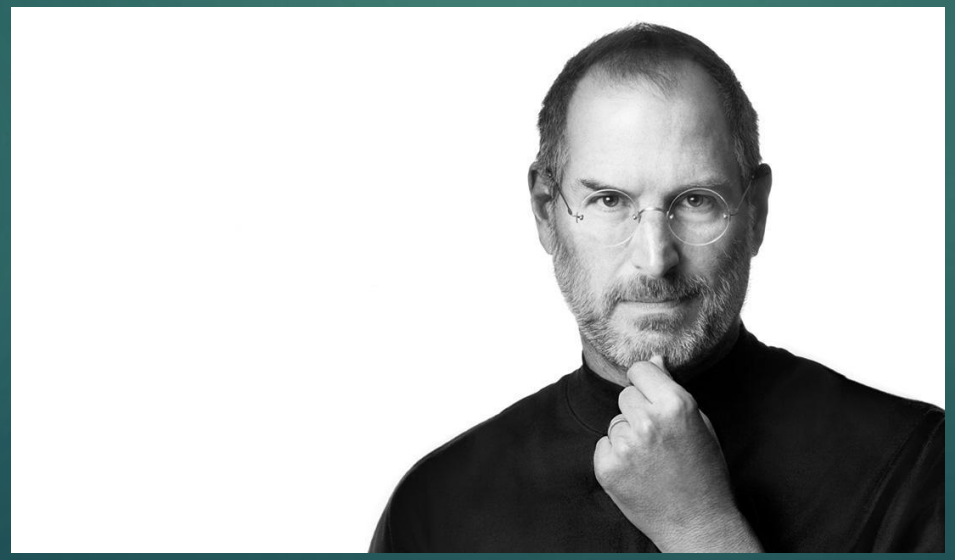
História

- O Objective-C foi criado por Brad Cox e Tom Love no início da década de 80 pela empresa Stepstone.



História

- Em 1988, a NeXT de Steve Jobs licenciou e liberou sua própria versão do compilador e das bibliotecas da linguagem.
- Em 1996 Jobs volta para a Apple e cria o Mac OS X tendo o Objective C como linguagem.



Caracteristicas

- Objective-C é apenas um conjunto de adições à linguagem C. Ela dá ao C suporte à construções orientadas a objetos como as da Smalltalk.
- Suporta polimorfismo e é uma linguagem dinâmica, com typing e binding dinâmicos.
- Possibilita adicionar classes e categorias em tempo de execução de forma fácil com uma sintaxe de mensagem simples e elegante.
- Realiza chamadas de mensagem dinâmicas rapidamente, entre 1,5 e 2,0 vezes o tempo de uma chamada de função em C.

Compilação

Para compilar um programa em Objective C é preciso ter o GCC instalado

```
gcc arquivos.m -o nome_do_programa -l objc
```

- gcc: Indica que usaremos o compilador gcc
- arquivos.m: nome do arquivo do programa
- -o : vem de output, indica qual nome o arquivo executável vai ter
- -l objc: O -l vem de library (biblioteca) -lobjc indica que usaremos as bibliotecas do objective-c para compilar o nosso programa.

Porém ao tentar compilar em um Linux ou Windows o gcc irá retornar alguns 'Warning' para resolver isto basta adicionar o comando "-Wno-import".

Sintaxe

- O Objective-C é uma linguagem simples baseada nas linguagens smalltalk em conjunto de C ,ou seja, é possível compilar qualquer programa C com um compilador Objective-C.

- A maior parte de sua sintaxe (incluindo pré-processamento, expressões, declaração e chamadas de funções) foi herdada da linguagem C, enquanto a sintaxe para os aspectos orientados a objetos foi criada para permitir passagem de mensagens no estilo Smalltalk.

Primeiro programa

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

int main (int argc, const char * argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    NSLog (@"Hello, World!");
    [pool drain];
    return 0;
}
```

Tipagem

- Objective-C suporta tanto tipificação fraca quanto forte para as variáveis do objeto. Variáveis com tipificação forte incluem o nome da classe na declaração do tipo da variável e as com tipificação fraca usam o tipo id.
- Variáveis com tipificação fraca são usadas normalmente para coleção de classes, onde o tipo exato do objeto em uma coleção pode ser desconhecido.

```
MyClass *myObject1;  // Strong typing
id       myObject2;  // Weak typing
```

- Id é um tipo de objeto genérico, que pode ser utilizado para guardar objetos pertencentes a qualquer classe.

Tipos

Os tipos em Objetive-C podem ser divididos em 4 categorias:

- Os tipos básicos;
- Os tipos enumerados;
- O tipo void;
- Os tipos derivados;

Tipos

- Os básicos consistem em dois tipos, os inteiros e os de ponto flutuante.

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Tipos

- Os tipos enumerados permitem ao programador predefinir a faixa de valores que podem ser atribuídos a uma variável e usar nomes autoexplicativos ao definir esses valores.

```
enum temperatura {cold = 5, warm = 50, hot = 95};
enum temperatura currentTemp;
int currentTemp = hot;
NSLog (@“A temperatura atual é %i", currentTemp);
```

Tipos

- O tipo void especifica que nenhum valor esta disponível.

Temos nesse tipo as funções que não retornam nenhum valor, então podemos dizer que elas retornam void.

```
void exit(int status)
```

E as funções que não precisam de parâmetros para funcionarem, nesse caso elas também são do tipo void.

```
int rand(void)
```

Tipos

Nos tipos derivados nós temos:

- Ponteiros
- Matrizes
- Estruturas
- Funções
- Uniões

Operadores

Primeiro nós temos os operadores aritméticos. No exemplo considere: A = 10 e B = 20.

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

Operadores

A seguir temos os operadores relacionais:

Operator	Description	Example
==	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

Operadores

- Os operadores lógicos:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

- E o operador de atribuição:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C

Strings

- Como um subconjunto de C, o Objective-C suporta as mesmas convenções para especificação de string que o C. Todavia, o framework do Objective-C tipicamente não usa string no estilo C. Ao invés disso, ele passa strings através de objetos NSString.

- Já que string são muito usadas, o Objective-C fornece uma notação curta para criar objetos NSString a partir de valores constantes. Para usar esse atalho, tudo que se tem que fazer é preceder uma string normal com aspas duplas com o simbolo @.

Strings

- Exemplo do uso do NSString:

```
NSString *make = @"Porsche";
NSString *model = @"911";
int year = 1968;
NSString *message = [NSString stringWithFormat:@"Esse é um %@ %@ de %d!", make, model, year];
NSLog(@"%@", message);
```

Strings

- Para as comparações de strings, nós não usamos o operador “==“, então devemos utilizar os seguintes métodos:

  “isEqualToString”, “hasPrefix” e hasSuffix”

```
NSString *car = @"Porsche Boxster";
if ([car isEqualToString:@"Porsche Boxster"]) {
        NSLog(@"That car is a Porsche Boxster");
}
if ([car hasPrefix:@"Porsche"]) {
        NSLog(@"That car is a Porsche of some sort");
}
if ([car hasSuffix:@"Carrera"]) {
        // This won't execute
        NSLog(@"That car is a Carrera");
}
```

Strings

Outros métodos interessantes:

- Concatenar duas strings:

```
NSString *string3 = [string1 stringByAppendingString:string2];
```

- Fazer uma busca na string:

```
NSRange searchResult = [string rangeOfString:@“fulano"];
```

Caso não encontre, teremos searchResult.location == NSNotFound, mas se for encontrado, searchResult.location irá retornar a posição em que se encontra a palavra.

Strings

- Alternar entre letras maiúsculas e minúsculas:

```
NSString *car = @"lotUs beSpoKE";

NSLog(@"%@", [car lowercaseString]);      // lotus bespoke

NSLog(@"%@", [car uppercaseString]);      // LOTUS BESPOKE

NSLog(@"%@", [car capitalizedString]);    // Lotus Bespoke
```

Estruturas de repetição

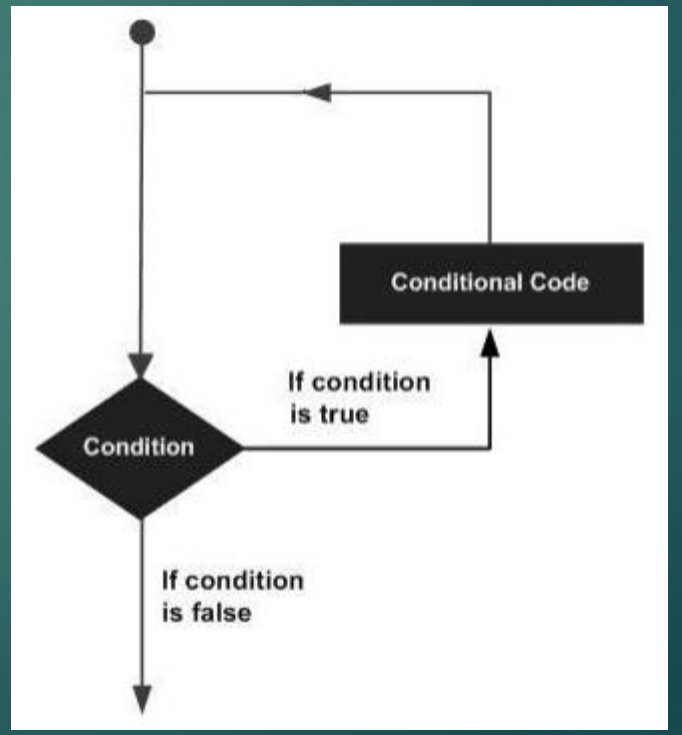
- Essas estruturas realizam ou repetem diferentes algoritmos dependendo se uma condição é verdadeira ou falsa

- Em Objective C são utilizadas:

  While

  For

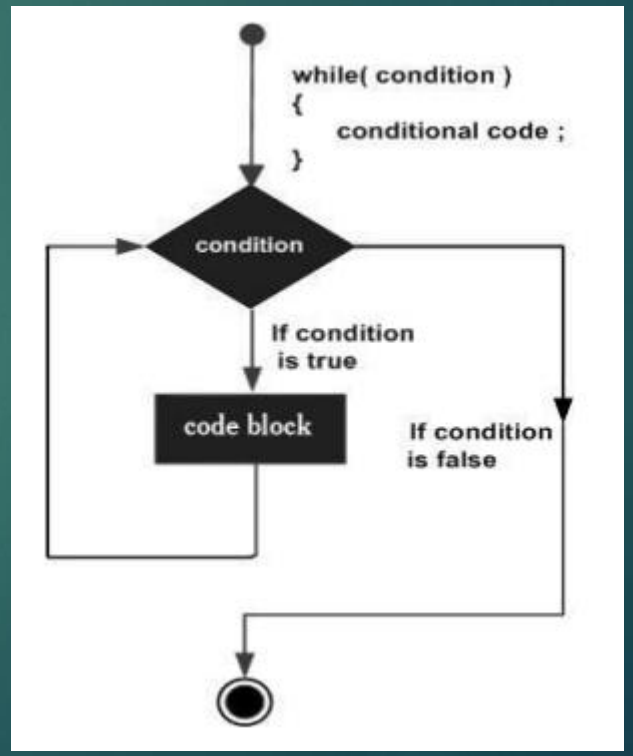
  Do...While



Estruturas de repetição

- A estrutura “while” irá executar repetidamente as declarações enquanto sua condição for verdadeira.

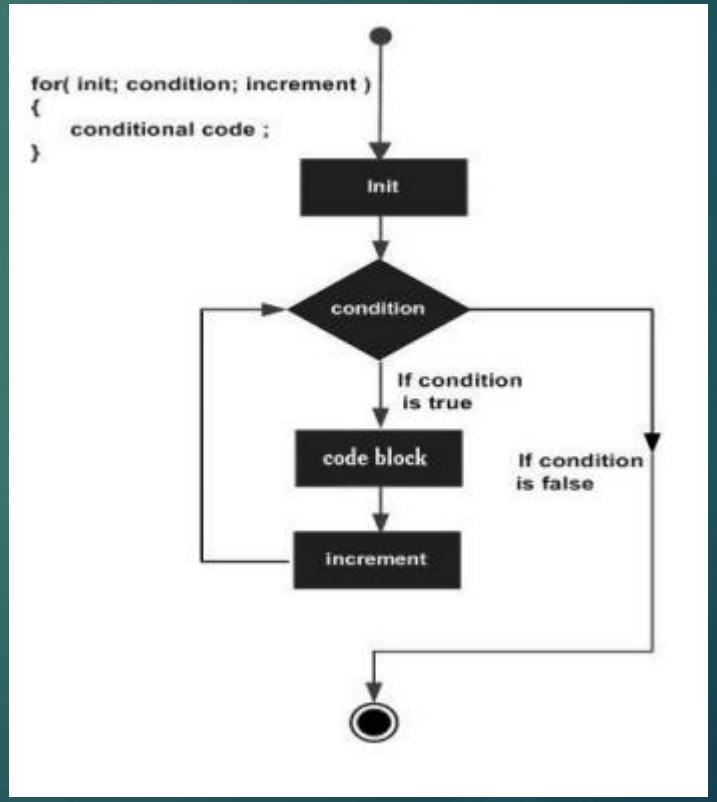
```
while(condition)
{
 statement(s);
}
```



Estruturas de repetição

- A estrutura “for” permite que você execute o código desejado um número especifico de vezes.

```
for ( init; condition; increment )
{
 statement(s);
}
```

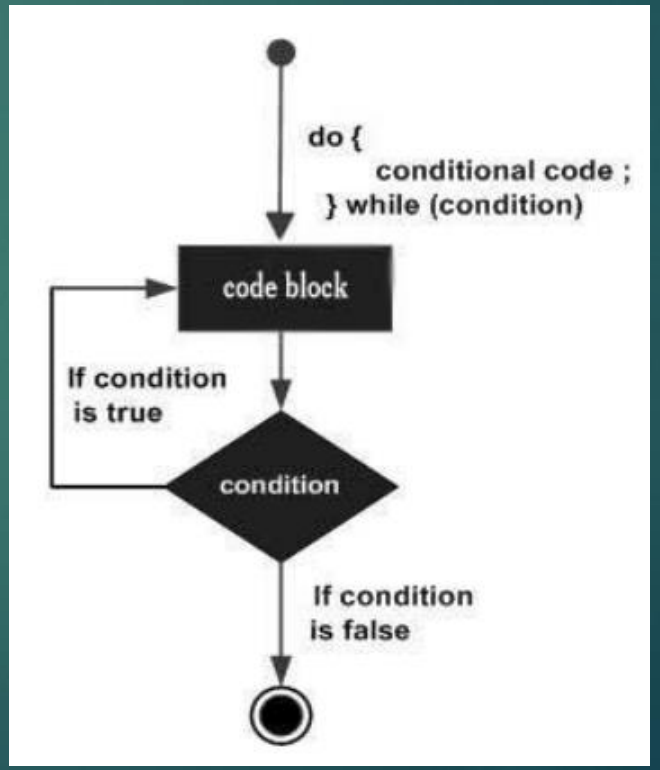


Estruturas de repetições

- Por ultimo nós temos o "do...while". Diferente dos outros dois que testam a condição no inicio, essa estrutura executa o código para depois checar as condições.

```
do
{
 statement(s);

}while( condition );
```



Estruturas de repetições

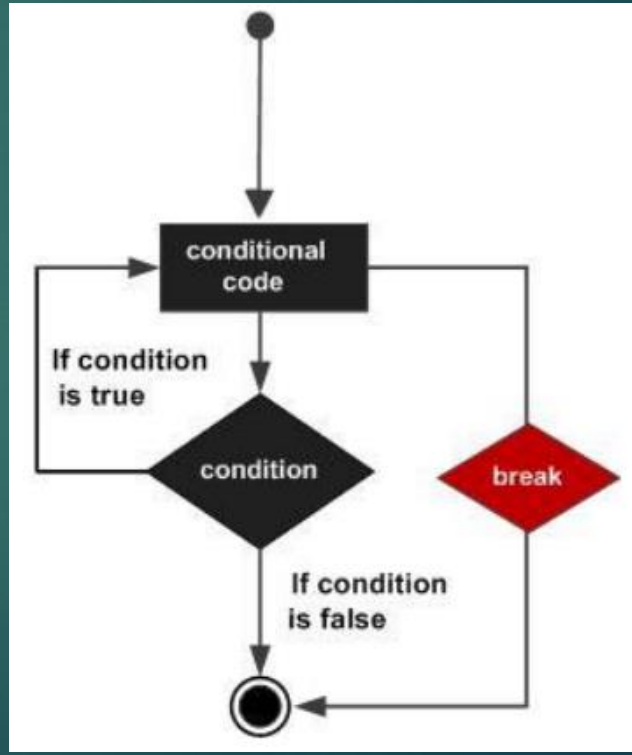
- Também é permitido o uso de "nested loop" ou loops aninhados.
- Consiste em utilizar estruturas de repetições dentro de outras estruturas.

```
for ( init; condition; increment ) {
      for ( init; condition; increment ) {
            statement(s);
      }
      statement(s);
}
```

Estruturas de repetições

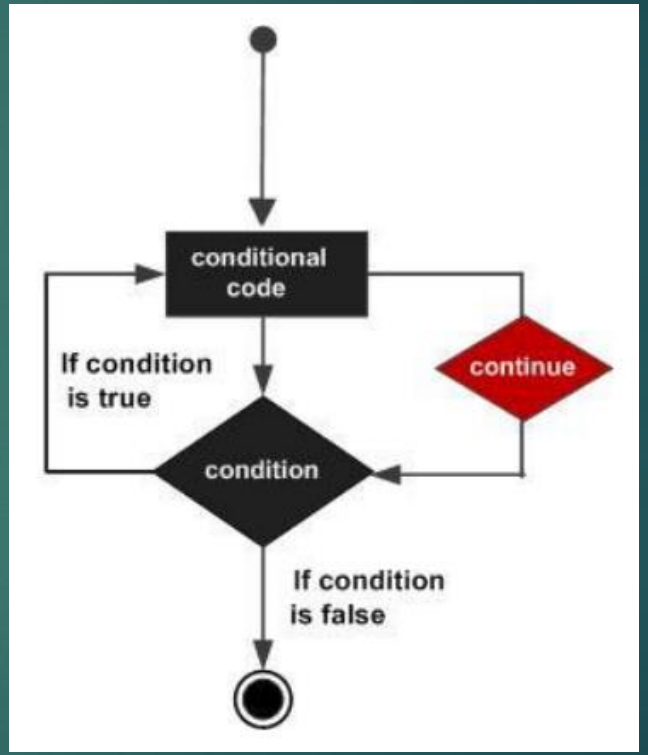
- Duas declarações muito importantes nesses loops são o “break” e o “continue”.

```
int a = 10;
while( a < 20 ){
      NSLog(@"value of a: %d\n", a);
      a++;
      if( a > 15){
            break;
       }
 }
```



Estruturas de repetições

```
int a = 10;
 do{
      if( a == 15){
          /* skip the iteration */
          a = a + 1;
          continue;
      }
      NSLog(@"value of a: %d\n", a);
      a++;
 }while( a < 20 );
```



Estruturas de controle

- Essas estruturas exigem que o programador especifique uma ou mais condições a serem avaliados ou testadas pelo programa, juntamente com uma declaração ou instruções a serem executadas se a condição for considerada verdade, e, opcionalmente, outras instruções a serem executadas se a condição for considerada falsa.

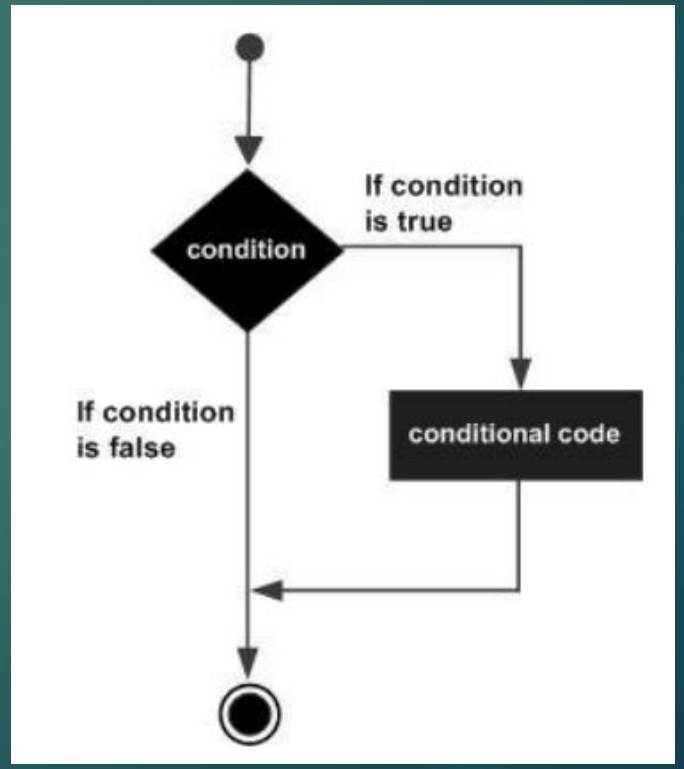
- Dentre essas estruturas, nós temos:

  "if", "if...else" e a "switch"

Estruturas de controle

- O “if” consiste em checar se a expressão boleana é verdadeira, caso ela seja, o código dentro do if será executado.

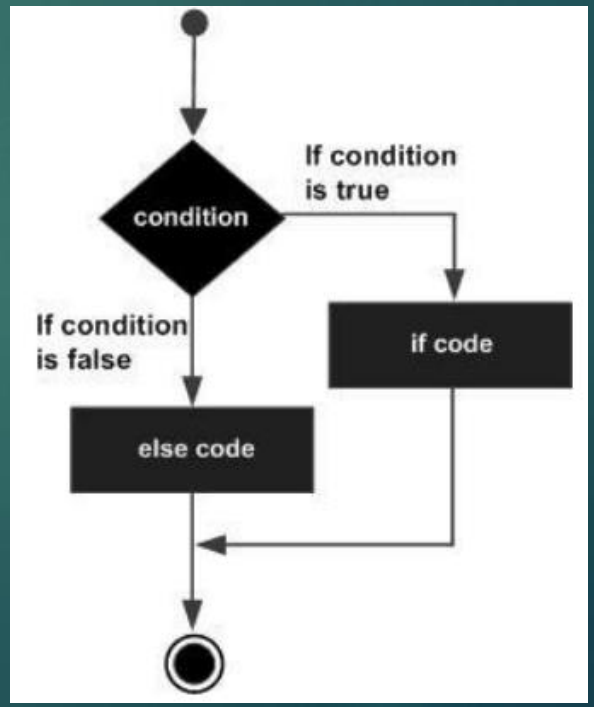
```
if(boolean_expression){
    /* statement(s) */
}
```



Estruturas de controle

- O “if...else” é bem parecido com o “if”, a diferença é que no anterior, caso a expressão fosse falsa, nada iria acontecer, agora teremos um código que será executado caso isso ocorra.

```
if(boolean_expression){
      /* statement(s) will execute if true */
}
else{
 /* statement(s) will execute if false */
}
```

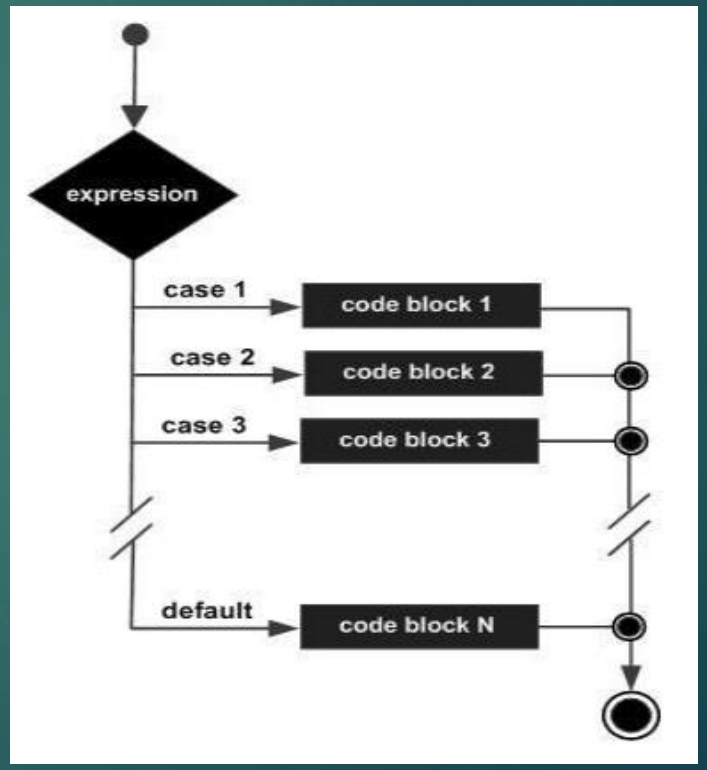


Estruturas de controle

- O “switch” permite que uma variável seja testada em uma lista de valores.

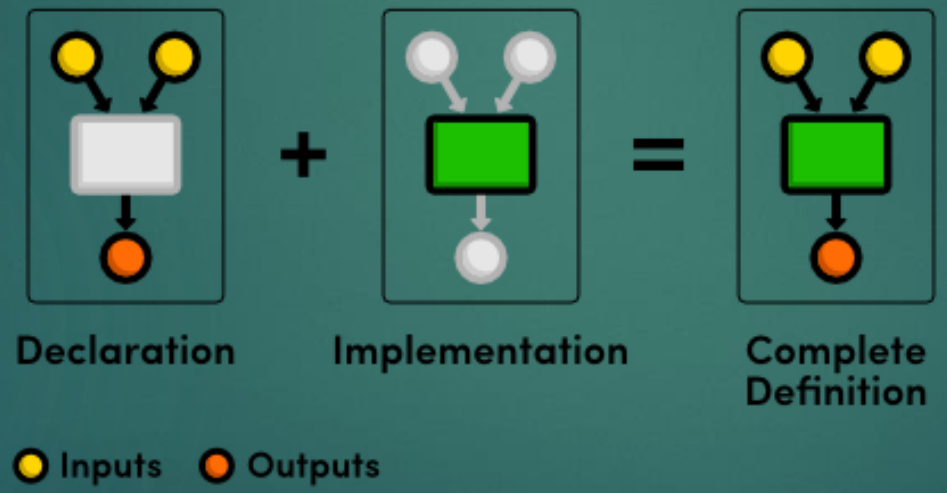
```
switch(expression){
    case constant-expression :
        statement(s);
        break; /* optional */
    case constant-expression :
        statement(s);
        break; /* optional */

    default : /* Optional */
        statement(s);
}
```



Funções

- Uma função é um grupo de declarações e implementações, que juntas executam uma tarefa.



Funções

- Forma geral da definição das funções em Objective C:

```
- (return_type) method_name:( argumentType1 )argumentName1
joiningArgument2:( argumentType2 )argumentName2 ...
joiningArgumentn:( argumentTypen )argumentNamen
{
 body of the function
}
```

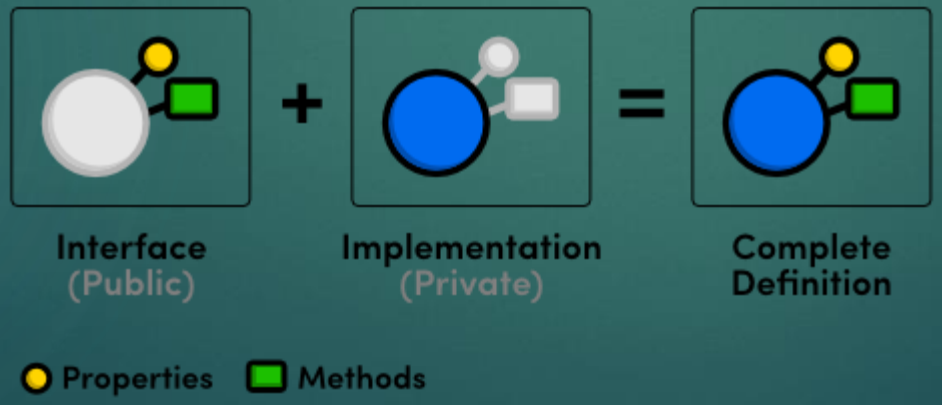
Funções

Exemplo de uma função que retorna o maior número entre os dois:

```
- (int) max:(int) num1 secondNumber:(int) num2
{
 int result;
 if (num1 > num2){
        result = num1;
 }
 else{
        result = num2;
 }
 return result;
}
```

Classes

- Para especificar uma classe é necessário uma interface e uma implementação.
- A interface é a parte que contém a declaração e define as variáveis que serão instanciadas e os métodos associados com a classe.
- A implementação é a parte que contém o código dos métodos da classe.



Classes

A interface deve ser colocada num arquivo .h

```
//Exemplo do formato

@interface NovaClasse: SuperClasse
{
    //Declaração_de_variáveis;
}

//Declaração_de_métodos;

@end
```

```
//Gato.h
@interface Gato: NSObject
{
        double peso;
}
-(void) miau;
-(void) setPeso: (double) p;
-(double) getPeso;
@end
```

Classes

- A implementação deve ser colocada em um arquivo .m.

```
// Gato.m
#import “Gato.h"
@implementation Gato
-(void) miau{
        NSlog(@"Miaau!");
}
-(void) setPeso: (double) p{
        peso=p;
}
-(double) getPeso{
        return peso;
}
@end
```

Métodos

- Nós métodos anteriores nós tínhamos o sinal de – no inicio da definição indicando que eles são métodos de instância da classe.
- Utilizando o sinal de + o método é de classe, ou seja, não precisamos instanciar um objeto para utilizar o método.

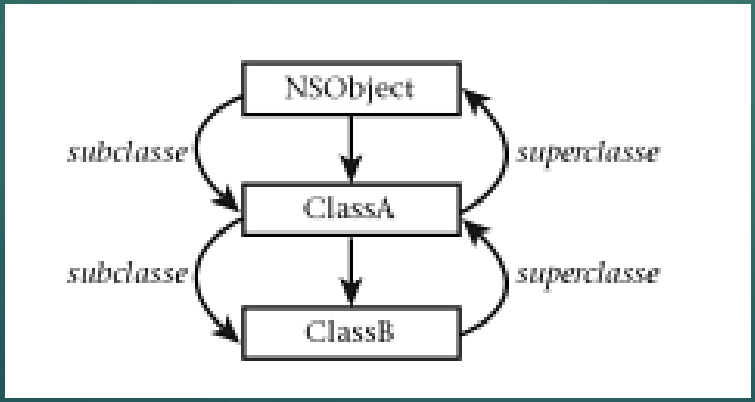
Atributos

- Também é possível declarar atributos privados em Objective C.

```
@interface MyClass : NSObject {
@private
    int someVar;     // Só é acessado por instancias de MyClass
@public
    int aPublicVar; // Pode ser acessado por qualquer objeto
}
@end
```

Heranças

- Quando uma nova classe é definida, ela herda certos atributos da classe Pai ou superclasse como as variáveis de instância (não privadas) e os métodos do pai torna-se implicitamente parte da nova definição da nova classe, pois a subclasse pode acessar esses métodos e variáveis de instância como se estivesse dentro da própria classe.



Heranças

```
@interface ClassA: NSObject
{
   int x;
}

-(void) initVar;
@end

@implementation ClassA
-(void) initVar
{
   x = 100;
}
@end

// Declaração e definição de ClassB

@interface ClassB: ClassA
-(void) printVar;
@end

@implementation ClassB
-(void) printVar
{
   NSLog (@"x = %i", x);
}
@end

int main (int argc, char * argv[])
{
   @autoreleasepool {
      ClassB *b = [[ClassB alloc] init];

      [b initVar];   // usará método herdado
      [b printVar];  // revela o valor de x;
   }
   return 0;
}
```

Polimorfismo

- Possui coerção.

- Não possui sobrecarga.

Solução:

-(void) writeToFile:(NSString *)path fromInt:(int)anInt;

-(void) writeToFile:(NSString *)path fromString:(NSString *)aString

Typing e Binding dinâmicos

- Typing dinâmico é a característica que a linguagem tem de poder adiar a determinação da classe de um objeto até o tempo de execução do programa, enquanto que Biding dinâmico é poder postergar a determinação do método que será invocado em um objeto até o tempo de execução.

- Para realizar o typing e o binding dinâmicos utiliza-se o tipo **id**. Este é um tipo de objeto genérico, que pode ser utilizado para guardar objetos pertencentes a qualquer classe.

Exemplo

```
-(id) reproduzir: (id) Animal2{
        id Filho=[[[self class] alloc] init];
        if ([self class] == [Animal2 class]){
                printf("O filho nasceu sadio!\n");
                [Filho setPeso: (peso+[Animal2 getPeso])/2];
        }
                else{
                printf("O filho nasceu morto, pois os pais são de classes diferentes!\n");
                [Filho setPeso: 0];
        }
        return Filho;
}
```

Palavras reservadas

- auto
- else
- long
- switch
- break
- enum
- register
- typedef
- case
- extern
- return
- union
- char
- float
- short
- unsigned
- const
- for
- signed
- void
- continue
- goto
- sizeof
- volatile
- default
- if
- Static
- while
- do
- int
- struct
- _Packed
- double
- protocol
- interface
- implementation
- NSObject
- NSInteger
- NSNumber
- CGFloat
- property
- nonatomic;
- retain
- strong
- weak
- unsafe_unretained;
- readwrite
- readonly

Arquivos

As operações básicas feitas em arquivos são fornecidas pelo NSFileManeger que permite:

- Criar um novo arquivo;
- Ler um arquivo já existente ;
- Escrever dados em um arquivo ;
- Remover um arquivo;
- Determinar o tamanho de um arquivo ;
- Fazer uma copia de um arquivo;

Arquivos

Para executar várias operações em um arquivo os métodos são fornecidos por NSFileHandle que são:

- Abrir um arquivo para leitura escrita ou atualização;
- Buscar uma posição especifica dentro do arquivo;
- Ler ou escrever um numero especificado de bytes;

Arquivos

Métodos de arquivo comuns de `NSFileManager`

Método	Descrição
`-(NSData *) contentsAtPath: caminho`	Lê dados de um arquivo
`-(BOOL) createFileAtPath: caminho contents: (NSData *) dados attributes: atrb`	Escreve dados em um arquivo
`-(BOOL) removeItemAtPath: caminho error: err`	Remove um arquivo
`-(BOOL) moveItemAtPath: de toPath: para error: err`	Muda o nome ou move um arquivo (`para` não pode existir)
`-(BOOL) copyItemAtPath: de toPath: para error: err`	Copia um arquivo (`para` não pode existir)
`-(BOOL) contentsEqualAtPath: caminho1 andPath: caminho2`	Compara o conteúdo de dois arquivos
`-(BOOL) fileExistsAtPath: caminho`	Testa a existência do arquivo
`-(BOOL) isReadableFileAtPath: caminho`	Testa se o arquivo existe e pode ser lido
`-(BOOL) isWritableFileAtPath: caminho`	Testa se o arquivo existe e pode ser escrito
`-(NSDictionary *) attributesOfItemAtPath: caminho error: err`	Obtém atributos do arquivo
`-(BOOL) setAttributesOfItemAtPath: atrb error: err`	Altera atributos do arquivo

Arquivos

- Cada um dos métodos de arquivo é chamado em um objeto NSFileManager que é criado pelo envio de uma mensagem defaultManager para a classe. Exemplo:

```
NSFileManager *FM;
FM= [NSFileManager defaultManager];
```

Arquivos

- Por exemplo, para excluir um arquivo chamado todolist do diretório atual, teríamos que criar um objeto NSFileManager para depois chamar o método removeItemAtPath, como segue abaixo:

```
[FM removeItemAtPath: @"todolist" error: NULL];
```

Arquivos

```
// Operações de arquivo básicas
// Presume a existência de um arquivo chamado "testfile"
// no diretório atual

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
      @autoreleasepool {
      NSString              *fName = @"testfile";
      NSFileManager         *fm;
      NSDictionary          *attr;

      // Precisa criar uma instância do gerenciador de arquivos

      fm = [NSFileManager defaultManager];

      // Permite garantir primeiro que nosso arquivo de teste exista

   if ([fm fileExistsAtPath: fName] == NO) {
      NSLog(@"File doesn't exist!");
      return 1;
   }

   //agora, vamos fazer uma cópia

   if ([fm copyItemAtPath: fName toPath: @"newfile" error: NULL] == NO) {
      NSLog(@"File Copy failed!");
      return 2;
   }

      // Agora, vamos testar para ver se os dois arquivos são iguais

   if ([fm contentsEqualAtPath: fName andPath: @"newfile"] == NO) {
      NSLog(@"Files are Not Equal!");
      return 3;
   }
```

Arquivos

Métodos comuns de `NSFileHandle`

Método	Descrição
`+(NSFileHandle *) fileHandleForReading-AtPath: caminho`	Abre um arquivo para leitura
`+(NSFileHandle *) fileHandleForWriting-AtPath: caminho`	Abre um arquivo para escrita
`+(NSFileHandle *) fileHandleForUpdating-AtPath: caminho`	Abre um arquivo para atualização (leitura e escrita)
`-(NSData *) availableData`	Retorna os dados disponíveis para leitura de um dispositivo ou canal
`-(NSData *) readDataToEndOfFile`	Lê os dados restantes, até o final do arquivo (`UINT_MAX`) bytes max
`-(NSData *) readDataOfLength: (NSU-Integer) bytes`	Lê um número especificado de *bytes* do arquivo
`-(void) writeData: dados`	Escreve dados no arquivo
`-(unsigned long long) offsetInFile`	Obtém o deslocamento do arquivo atual
`-(void) seekToFileOffset: deslocamento`	Configura o deslocamento do arquivo atual
`-(unsigned long long) seekToEndOfFile`	Posiciona o deslocamento do arquivo atual no final do arquivo
`-(void) truncateFileAtOffset: deslocamento`	Configura o tamanho do arquivo como *deslocamento* bytes (preenche, se necessário)
`-(void) closeFile`	Fecha o arquivo

Exceções

- O tratamento de exceções em Objective-C é igual ao de Java a única diferença em termos é que em Objective-C o uso de throws é opcional.

```
1     NSMutableArray *pessoas = [NSMutableArray arrayWithObjects:@"Joao", @"Jose", @"Pedro
2     @try {
3         NSLog(@"Executando try...");
4         [pessoas removeObjectAtIndex:100];
5     }
6     @catch (NSException *exception) {
7         NSLog(@"Caiu dentro de catch!");
8         NSLog(@"Excessao: %@", exception);
9     }
10    @finally {
11
12        NSLog(@"Dentro de finally");
13    }
14
15    NSLog(@"Continuando...");
```

Avaliação da LP

- Algumas das vantagens do Objective-C incluem a possibilidade de se carregar as definições das classes e métodos em tempo de execução, o fato de os objetos serem tipados dinamicamente, a possibilidade de se utilizar objetos remotos, a persistência e a existência de protocolos de delegação e meta-ação.
- Alguns de seus principais problemas são a inexistência de herança múltipla e a inexistência de variáveis de classe.