Aprendendo a programar para iOS

Entendendo o Código

Parte 2 - Framework

Framework

- NSNumber
- NSString
- NSDate
- NSArray
- NSDictionary
- Exercícios

NSNumber

- Classe utilizada para armazenar tipos primitivos: int, float, bool, e muitas outras.
- Classes de rede e banco a utilizam, pois por serem classes, podem ter o valor nil.
- An object wrapper for primitive scalar numeric values.

NSNumber

```
NSNumber *theLetterZ = @'Z'; // [NSNumber numberWithChar:'Z’]
// integral literals.
NSNumber *fTwo = @42; // [NSNumber numberWithInt:42]
NSNumber *fTwoUnsigned = @42U; // [NSNumber numberWithUnsignedInt:42U]
NSNumber *fTwoLong = @42L; // [NSNumber numberWithLong:42L]
NSNumber *fTwoLongLong = @42LL; // [NSNumber numberWithLongLong:42LL]
// floating point literals.
NSNumber *piFloat = @3.1415F; // [NSNumber numberWithFloat:3.1415F]
NSNumber *piDouble = @3.1415; // [NSNumber numberWithDouble:3.1415]
// BOOL literals.
NSNumber *yesNumber = @YES; // [NSNumber numberWithBool:YES]
NSNumber *noNumber = @NO; // [NSNumber numberWithBool:NO]


char z = theLetterZ.charValue; // [theLetter charValue];
int f = fortyTwo.intValue; // [fortyTwo intValue];
```

NSString

- Em objective-c, utilizamos duas classes para representar strings: NSString e NSMutableString
- NSString armazena uma string Unicode, e possui métodos auxiliares para comparar, buscar e modificar strings.
- NSStrings uma vez criadas, sempre precisam ser reinicializadas em qualquer modificação, se você precisa construir uma string ou modificar uma, utilize NSMutableString

NSString

- Em objective-c, utilizamos duas classes para representar strings: NSString e NSMutableString
- NSString armazena uma string Unicode, e possui métodos auxiliares para comparar, buscar e modificar strings.
- NSStrings uma vez criadas, sempre precisam ser reinicializadas em qualquer modificação, se você precisa construir uma string ou modificar uma, utilize NSMutableString

NSString

```
// Creating and printing a string
NSString *greeting = @"Hello";
NSLog(@"Greeting message: %@\n", greeting );
NSError * error;
greeting = [NSString stringWithContentsOfFile: @”file.txt”
                                     encoding: NSUTF8StringEncoding
                                        error: &error];
NSURL *url = [NSURL URLWithString:@"http://google.com"];
greeting = [NSString stringWithContentsOfURL: url
                                    encoding:NSUTF8StringEncoding
                                       error:&error];
greeting = [NSString stringWithFormat:@”formato %i", 4];
```

NSString

```
// Converting
long l = [greeting length];
NSString *number = @"2";
int n = [number intValue];
float f = [number floatValue];
NSLog(@"length: %li, number values:
%i and %f", l, n, f);
```

NSString

```
// String manipulation
greeting = [greeting substringFromIndex:2];
NSLog(@"Substring: %@", greeting);
// Comparação de strings
bool equal = [greeting isEqualToString:s2];
// Append de strings só rola assim
greeting = [greeting stringByAppendingFormat: @”valores %i
%f %@", 1, 2.3, (equal ? @"a" : @"b")];
g = [g stringByReplacingOccurrencesOfString: @"abc"
                                 withString:@"cde"];
// Split e Join
NSArray *p = [greeting componentsSeparatedByString:@" "];
[parts componentsJoinedByString:@", "];
```

NSString

```
// Leitura de strings do console, para estas aulas de sintaxe:
char buffer[500] = {0}; // init all to 0
scanf(” %[^\n]s", buffer);
NSString *name = [NSString stringWithUTF8String:buffer];
```

NSDate

- A representation of a specific point in time
- Representa uma data independente do calendário (usamos o gregoriano)
- É armazenado como uma data relativa a 01/01/2001 00:00:00 UTC
- A classe NSDate possui método pra comparar, calcular intervalos, criar datas baseadas em uma data e um intervalo
- NSDateFormatter pra formatar, NSCalendar pra operações de calendário

NSDate

```
// Data criada é a data atual
NSDate *date = [NSDate date];
NSLog(@"%@", date);

// Data calculada com valor em segundos
NSDate *date1 = [NSDate
dateWithTimeIntervalSinceNow:100000000];
NSDate *date2 = [NSDate
dateWithTimeInterval:3600 sinceDate:d2];
NSLog(@"%@ - %@", date1, date2);
```

NSDate

```objc
// Compara duas datas
NSDate *date = [NSDate distantFuture];
NSLog(@"%d", [date isEqualToDate:[NSDate date]]);

// Retorna a data mais antiga/futura
NSDate *early = [d1 earlierDate:d2];
NSDate *later = [d1 laterDate:d2];

if ([date1 compare:date2] == NSOrderedAscending) {
    // NSOrderedSame
    // NSOrderedDescending
}
```

NSDate

```objc
// Retorna intervalo em segundos entre datas
float f = [date1 timeIntervalSinceDate:date2];

// Formatando datas
NSDateFormatter *dateFormatter = [[NSDateFormatter
alloc]init];
[dateFormatter setDateFormat:@"yyyy-MM-dd"];
[dateFormatter setTimeZone:[NSTimeZone
timeZoneWithAbbreviation:@"UTC"]];
NSDate *dd = [dateFormatter dateFromString:@"2017-07-22"];
NSLog(@"%@", [dateFormatter stringFromDate:dd]);
```

NSArray

- Classe que armazena arrays em Objective-C
- NSArray representa um array estático
- NSMutableArray representa um array dinâmico

NSArray

```
// Inicializando arrays
NSArray *a = @[@12, @YES, @"abc"];
NSArray *b = [NSArray arrayWithObjects:@12, @YES, @"abc",
nil];
NSLog(@"%@ - %@", a, b);

// Acessando elentos de um array
NSString *c = a[2];

// Manipular um NSArray sempre resulta em retornar o array
recriado
NSSortDescriptor *descriptor =[[NSSortDescriptor alloc]
initWithKey:@"attribute1" ascending:YES];
NSArray *d = [a sortedArrayUsingDescriptors: descriptor];
```

NSMutableArray

```
// Inicializando arrays
NSMutableArray *a = [NSMutableArray array];
NSMutableArray *b = [NSMutableArray arrayWithArray:@[@"a",
@"b", @"c"]];
[a addObject:@"abc"];
[a addObject:@"abc"];
```

NSArray

```
// Iterando arrays
for (NSString *element in a) {
    NSLog(@"%@", element);
}

for (int i = 0; i < [a count]; i++) {
    NSLog(@"%@", a[i]);
}
```

NSDictionary

- Classe que armazena Dicionários, que são duplas de chave/valor.
- An object representing a static collection of key-value pairs
- NSDictionary guarda uma coleção estática. Para modificar o mesmo, crie um NSMutableDicionary

NSDictionary

```
// initialization
NSDictionary *dict = @{@"key1":@”Teste",@"key2": @”Easy"};
// access value for key1
NSString *value = dict[@"key1"];
NSLog(@"%@",value);

// Iterando elementos do dictionary
for (NSString* key in dict) {
    NSLog(@"%@", dict[key]);
}
```

NSMutableDictionary

```
// Inicializando um vazio, e adicionando um elemento
NSMutableDictionary *mdic = [NSMutableDictionary
dictionary];
mdic[@"teste"] = @"Lalala";

// Inicializando um com um NSDictionary
NSMutableDictionary *mdic = [NSMutableDictionary
dictionaryWithDictionary: dic];
```

Exercícios

- Link para material desta aula: http://bit.ly/ios-2
- Link para material de sintaxe da aula passada: http://bit.ly/ios-1-p2
- Link para compilador online de objective-c: http://bit.ly/objective-c-online

Exercícios

- No projeto da aula anterior, das contas correntes, faça as seguintes modificações:
  - Adicione os atributos data_de_nascimento: Date e history: NSMutableArray
  - Adicione um método description à classe para imprimir todas as informações da conta.
  - A cada transação, adicione um NSDictionary no NSArray contendo as informações da transação realizada.

Exercícios

- No projeto da aula anterior, das contas correntes, faça as seguintes modificações:
  - No Main, Crie um NSMutableArray de contas corrente para armazenar as contas criadas.
  - Na criação das duas contas iniciais, faça a criação com dados preenchidos pelo usuário (correntista, número, data de nascimento)
  - Forneça uma opção adicional de adicionar uma nova conta no array, no menu inicial

Exercícios

- No projeto da aula anterior, das contas correntes, faça as seguintes modificações:
  - Quando o usuário digitar o número da conta, no menu, faça uma busca no array de contas
  - Adicione a opção **extrato**, que imprime o array de histórico da conta

NSError

- Classe que encapsula e armazena informações sobre erros que possam ter ocorrido em algumas operações
- Classes que utilizam NSError geralmente recebem uma variável numa passagem por referência pra retorná-la preenchida em caso de erro

NSError

- Criando um NSError:

```
NSString *domain = @"com.MyCompany.MyApplication.ErrorDomain";
NSString *desc = NSLocalizedString(@"Unable to process", @"");
NSDictionary *userInfo = @{ NSLocalizedDescriptionKey : desc };
NSError *error = [NSError errorWithDomain:domain code:-101
userInfo:userInfo];
```

- Tratando erro:

```
NSError * error;
greeting = [NSString stringWithContentsOfFile: @”file.txt”
                                     encoding: NSUTF8StringEncoding
                                        error: &error];

if (error != nil) {
    // Do something
}
```

NSException

- A Classe NSException armazena informações sobre uma exceção ocorrida.
- Para o tratamento de exeções se usa: @try, @catch, @finally

NSException

```
NSString *test = @"test";
unichar a;
int index = 5;

@try {
    a = [test characterAtIndex:index];
}
@catch (NSException *exception) {
    NSLog(@"%@", exception.reason);
}
@finally {
    NSLog(@"Char at index %d cannot be found", index);
    NSLog(@"Max index is: %d", [test length]-1);
}
```

Blocos

- Blocos são funções sem nome definido, que podem ser armazenadas em variáveis e passadas como parâmetros de outras funções e métodos.
- São como funções anônimas de Javascript.
- A sintaxe de blocos utiliza o carácter ^

Blocos

```
// Bloco sem parâmetros
void (^simpleBlock)(void) = ^{
    NSLog(@"This is a block");
};
simpleBlock();

// Bloco com parâmetros
double (^multiplyTwoValues)(double, double) =
^(double firstValue, double secondValue) {
    return firstValue * secondValue;
};
double result = multiplyTwoValues(2,4);
NSLog(@"The result is %f", result);
```

Blocos

- Blocos têm acesso ao escopo do local onde foram definidos:

```
- (void)testMethod {
    int anInteger = 42;

    void (^testBlock)(void) = ^{
        NSLog(@"Integer is: %i", anInteger);
    };

    testBlock();
}
```

Blocos

- Mas apesar disto, variáveis passadas para blocos mantém o valor de quando o bloco foi definido.

```
- (void)testMethod {
    int anInteger = 42;
    void (^testBlock)(void) = ^{
        NSLog(@"Integer is: %i", anInteger);
    };
    anInteger = 84;
    testBlock(); // Ainda imprime 42
}
```

Blocos

- Para compartilhar armazenamento, declare variáveis com __block.

```
- (void)testMethod {
    __block int anInteger = 42;

    void (^testBlock)(void) = ^{
        NSLog(@"Integer is: %i", anInteger);
        anInteger = 100;
    };

    testBlock();
    NSLog(@"%i", anInteger);
}
```

Blocos

```
// Bloco sem parâmetros
void (^simpleBlock)(void) = ^{
    NSLog(@"This is a block");
};
simpleBlock();

// Bloco com parâmetros
double (^multiplyTwoValues)(double, double) =
^(double firstValue, double secondValue) {
    return firstValue * secondValue;
};
double result = multiplyTwoValues(2,4);
NSLog(@"The result is %f", result);
```

Blocos

- Uso prático: ordenar um array de objetos!

```
contas_ordenadas = [contas
sortedArrayUsingComparator:^NSComparisonResult(id a, id b)
{
    NSString *first = [(ContaCorrente*)a correntista];
    NSString *second = [(ContaCorrente*)b correntista];
    return [first compare:second];
}];
```

NSTimer

- Outro exemplo do uso de blocos: NSTimer é uma classe que faz chamadas temporizadas de blocos.

```
NSTimer *timer = [NSTimer
   scheduledTimerWithTimeInterval:2
                          repeats:YES
                          block:^(NSTimer *timer) {
    NSLog(@"Executado %@", [NSDate date]);
}];
```

Selector

- Em objective C, Selector é como chamamos o nome usado pra selecionar um método a ser executado por um objeto, é o identificador único do método quando o código é compilado.
- @selector(methodName:secondPart:third:)
- Podemos usar para verificar se um objeto possui um método

```
ContaCorrente *c = [[ContaCorrente alloc] init];
if ([c respondsToSelector:@selector(transfere:para:)]) {
    [c transfere:30 para:c2];
}
```

Class checking

- Em objective C, todo objeto possui o método isKindOfClass:
- Podemos usar em aplicações com polimorfismo para verificar qual a classe que uma variável está armazenando.

```
[myObject isKindOfClass:[NSString class]]
```

Exercícios

- No projeto da conta corrente, faça as seguintes modificações:
  - Adicione um método que ordene as contas baseado em um critério escolhido, que pode ser: número, nome e data de nascimento.
  - Adicione um método de listagem, que lista todas as contas na ordem que estiverem armazenadas dentro do array.
  - Crie uma subclasse da classe ContaCorrente chamada ContaPoupanca, que possua o método calculaRendimento

Exercícios

- No projeto da conta corrente, faça as seguintes modificações:
  - No menu da aplicação, adicione a opção de calcular rendimento da conta. Este método deve verificar se a classe armazenada é do tipo conta corrente ou poupança, e deve verificar ainda se o objeto em questão possui o método **calculaRendimento**.
  - Na criação de contas, perguntar se o usuário deseja criar uma conta do tipo corrente ou poupança.

Gabarito

- Exercício solicitado durante a aula (ainda sem a parte de herança):
  - Versão xcode: http://bit.ly/2uLDjqA
    - Código implementado na última versão do xcode
  - Versão compilador online: http://bit.ly/2eEOJpP
    - Mudanças necessárias para funcionar no compilador online foram:
      - Inicialização de NSNumber, NSArray e NSDictionary no formato antigo
      - Acesso de elementos de NSDictionary em formato antigo
      - Ordenação de arrays usando NSSortDescriptors ao invés de blocks
      - Definição das properties em formato antigo, com synthesize