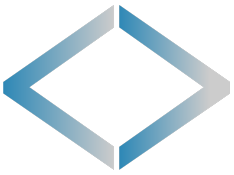

Padrões de projeto e de código

Motivação

Convenções de código são um conjunto de recomendações para definir o estilo de codificação usado em métodos, variáveis, nome de classes, organização e nomenclatura de arquivos, comentários e outros recursos da linguagem. Convenciona-se para que a leitura seja mais fácil ao se deparar com um código desconhecido e também para que não haja diferentes estilos de programação dentro de um projeto, uma vez que frequentemente projetos são executados por diversas pessoas.

As convenções descritas aqui estão no site oficial da linguagem Kotlin [1].

Idioma

O idioma utilizado no curso será inglês, uma vez que os métodos por padrão compartilhados em repositórios e também documentações oficiais da linguagem são inglês. Dessa maneira, não existirão métodos e documentações em diferentes línguas. No entanto, a linguagem dos métodos não interferem no entendimento dos conceitos. A qualquer momento, é possível trocar o nome dos métodos caso necessário.

Os comentários serão português para compreensão mais clara já que se trata de um curso destinado para falantes de português.

Ponto e vírgula

A linguagem deixa como opcional o uso do ponto e vírgula e dessa maneira, como Kotlin preza pela limpeza do código, o uso do ponto e vírgula é desencorajado. Caso seja declarado na IDE, não haverá problema de compilação, mas haverá o aviso constante de que não é necessário.

Nomenclaturas de código

- Uso de **camelCase** começando com letra minúscula e evitando *underline* para*:*
    - variáveis
    - métodos
    - atributos de classe
- Identação com espaços com o auxílio da própria IntelliJ IDEA ou Android Studio.
- Variáveis privadas da classe se iniciam com **m** minúsculo seguido do nome da variável usando **camelCase**.

```
private val mCourseName: String = “Curso Kotlin”

fun main () {
   val isAwesome = true
   isThisCourseAwesome(isAwesome)
}

// Verifica se o curso é demais!
fun isThisCourseAwesome (isAwesome: Boolean) {
   if (isAwesome) {
      println(“Com certeza! Estamos falando do $mCourseName”)
   }
}
```

Para constantes, marcados com a palavra **const** ou declarações **val** fora de classes e funções, devem usar todas as letras maiúsculas separados com underline.

```
const val COURSE_NAME: String = “Curso Kotlin”
val MAXIMO_ITEMS: Int = 8
```

Métodos que façam sobrecarga são próximos uns dos outros.

Classes

Se um arquivo Kotlin possuir uma classe somente, o nome do arquivo deve ser o mesmo nome da classe com a extensão .kt. Se um arquivo contiver múltiplas classes, o ideal é que o nome do arquivo seja algo que descreva a responsabilidade da classe da melhor maneira e mais sucinta possível.

Para a nomenclatura, usar o padrão **CammelCase** com a primeira letra sempre maiúscula.

Já para os parâmetros definidos dentro das classes, caso sejam poucos, podem ser escritas em uma única linha.

Classes com construtores mais longos devem ser formatadas para que cada propriedade fique em uma linha. Além disso, o parênteses para fechar o construtor deve ficar em uma nova linha.

```
// Classe com pouco parâmetros
class Pessoa (val nome: String, val anoNascimento: Int)

// Muitos parâmetros para somente uma linha
class Curso (
        val titulo: String,
        val descricao: String,
        val alunos: Int,
        val nota: Float
)
```

Objects, companion objects, métodos que façam a sobrescrita da classe pai e construtores secundários ficam no início da classe.

Chaves - {}

Em if/else, while, for, do..while, try..catch, funções e classes sempre utilizar as chaves para abertura e fechamento do corpo das instruções.

```
if (condicao) {
   // Lógica
} else {
   // Lógica
}
```

```
while(condicao) {
   // Lógica
}
```

```
try {
   // Lógica
} catch (e: Exception) {
   // Lógica
} finally {
}
```

```
for(l in listOf(1,2)) {
   // Lógica
}
```

Unit (Void)

O tipo Unit corresponde a nulo. Em funções que não há retorno, não há necessidade de declarar o retorno como Unit, basta deixar em branco. Caso a função possua retorno, então é necessário declarar o tipo.

```
fun semRetorno () {
   // Lógica
}

fun digaMeuNome (nome: String) : String {
   return “Hey $nome”
)
```

Activities e Layout

Nomes de arquivos de layout devem ser prefixados com os elementos que representam:

Componente	Classe	Nome do Layout
Activity	UserProfileActivity	activity_user_profile.xml
Fragment	SignUpFragment	fragment_sign_up.xml
Dialog	ChagePasswordDialog	dialog_change_password.xml

Arquivos XML

Quando um elemento não tiver nenhum conteúdo dentro das tags, deve-se usar "self closing" tags.

Ruim :(

```
<EditText
   android:id="@+id/edit_name" >
</EditText>
```

Bom!

```
<EditText
   android:id="@+id/edit_name" />
```

Identificadores de elementos

Identificadores de elementos são escritos em minuscula_com_underline.

Elemento	Prefixo
TextView	text_
EditText	edit_
Button	button_
ImageView	image_
Menu	menu_

```
<EditText
   android:id="@+id/edit_name" />

<Button
   android:id="@+id/button_login" />

<TextView
   android:id="@+id/text_welcome" />
```

Referências

[1] https://kotlinlang.org/docs/reference/coding-conventions.html, disponível em 08/03/2020.