# Convenções de código e projeto

## Motivação

Convenções de código tem por objetivo descrever um guia de codificação a ser usado no projeto, de maneira que leitores do código possam se concentrar no funcionamento e não percam tempo tentando entender como foi escrito.

Por mais simples que pareça, uma das recomendações é consistência. Ao lidar com um projeto que não se conhece, deve-se tomar um tempo para entender qual o padrão da codificação. Assim, novas alterações podem ser escritas no mesmo padrão. Futuramente, a consistência auxilia programadores a entenderem os padrões adotados somente uma vez, ao invés de lidar com um estilo diferente em cada arquivo.

## Idioma

O idioma utilizado na codificação dos aplicativos será inglês, uma vez que os métodos e documentações usadas no Android Studio também são inglês. Dessa maneira, não haverá códigos em várias línguas.

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

## Linguagem Java

A programação Android segue as convenções da linguagem Java mostradas abaixo nos próximos tópicos.

- **Não ignore exceções**

```
void setServerPort(String value) {
    try {
        serverPort = Integer.parseInt(value);
    } catch (NumberFormatException e) { }
}
```

Embora seja tentador escrever um código desse tipo ou considerar que esse tipo de erro nunca acontecerá, esse trecho de código pode se tornar uma mina para futuras manutenções no código. Exceções devem ser tratadas independente do contexto.

- ## Nomes de métodos curtos

Na medida do possível, manter nomes de métodos curtos e usando as palavras chaves que descrevem o seu funcionamento.

- ## Métodos curtos

Manter métodos curtos e focados em sua responsabilidade. Geralmente existem métodos grandes, mas isso não deve ser tomado como uma prática. O maior problema de métodos grandes é a facilidade que o mesmo pode englobar várias regras do sistema, e assim dificultar a manutenção.

- ## Definir variáveis em locais padronizados

Definir as variáveis no começo do arquivo. Evitar que exista perda de tempo procurando variáveis declaradas aleatoriamente pelo código.

- ## Limitar escopo de variáveis

Variáveis usadas dentro de um único método devem ser declaradas dentro do mesmo. Não existe necessidade de declarar a variável com o escopo da classe quando se pode declarar localmente. Esta prática reduz que existam muitas variáveis de classe e se torne difícil entender a utilidade de cada uma.

- ## Convenções de nomes para atributos

Para a definição de variáveis:

1. Campos não públicos e não estáticos começam com m.
2. Campos estáticos começam com s.
3. Outros campos começam com a primeira letra minúscula.
4. Campos estáticos públicos são escritos MAIUSCULO_COM_UNDERLINE.

```
public class MyClass {
    public static final int SOME_CONSTANT = 42;
    public int publicField;
    private static MyClass sSingleton;
    int mPackagePrivate;
    private int mPrivate;
    protected int mProtected;
}
```

- ## Nome de arquivos - Classe

Nomes de classes são escritas em UpperCamelCase.

Para classes que extendem funcionamento de componentes Android, o nome da classe deve terminar com o nome do componente. Por exemplo, UserRegistrationActivity, LoginFragment, ChangePasswordDialog.

- ## Ordenação dentro de classes

A ordem recomendada para código dentro da classe é a seguinte:

1. Constantes
2. Variáveis
3. Construtores
4. Métodos override
5. Métodos públicos
6. Métodos privados

- ## Ordenação de parâmetros em métodos

Quando se está programando para Android, é extremamente comum declarar métodos que recebam o contexto da aplicação. Métodos que recebem o contexto devem sempre defini-lo como o primeiro parâmetro.

Parâmetros de callback devem sempre ser os últimos parâmetros.

# Convenções Android

- ## Nome de arquivos - Layout

Nomes 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 |
| AdapterView | -- | item_food.xml |

- ## XML

Quando um XML não tiver nenhum conteúdo dentro das tags, deve usar “self closing” tags.

```
<TextView
   android:id="@+id/text_name"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content" />
```

Sim

```
<TextView
   android:id="@+id/text_name"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
</TextView>
```

Não

- **Nome de elementos - Layout**

Nomes de elementos de layout devem fazer uso de prefixo do tipo de elemento seguido da responsabilidade do elemento.

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

```
<ImageView
   android:id="@+id/image_user"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content" />
```

```
<menu>
   <item
       android:id="@+id/action_settings"
       android:orderInCategory="100"
       android:title="@string/action_settings" />
</menu>
```

## Fontes


1. SourceAndroid. **AOSP Java Code Style for Contributors**. Disponível em: <https://source.android.com/setup/contribute/code-style>. Acesso em 4 de junho de 2020.
2. Ribot GitHub, **Android guidelines**. Disponível em <https://github.com/ribot/android-guidelines/blob/master/project_and_code_guidelines.md>. Acesso em 4 de junho de 2020.