

# Desenvolvimento Mobile

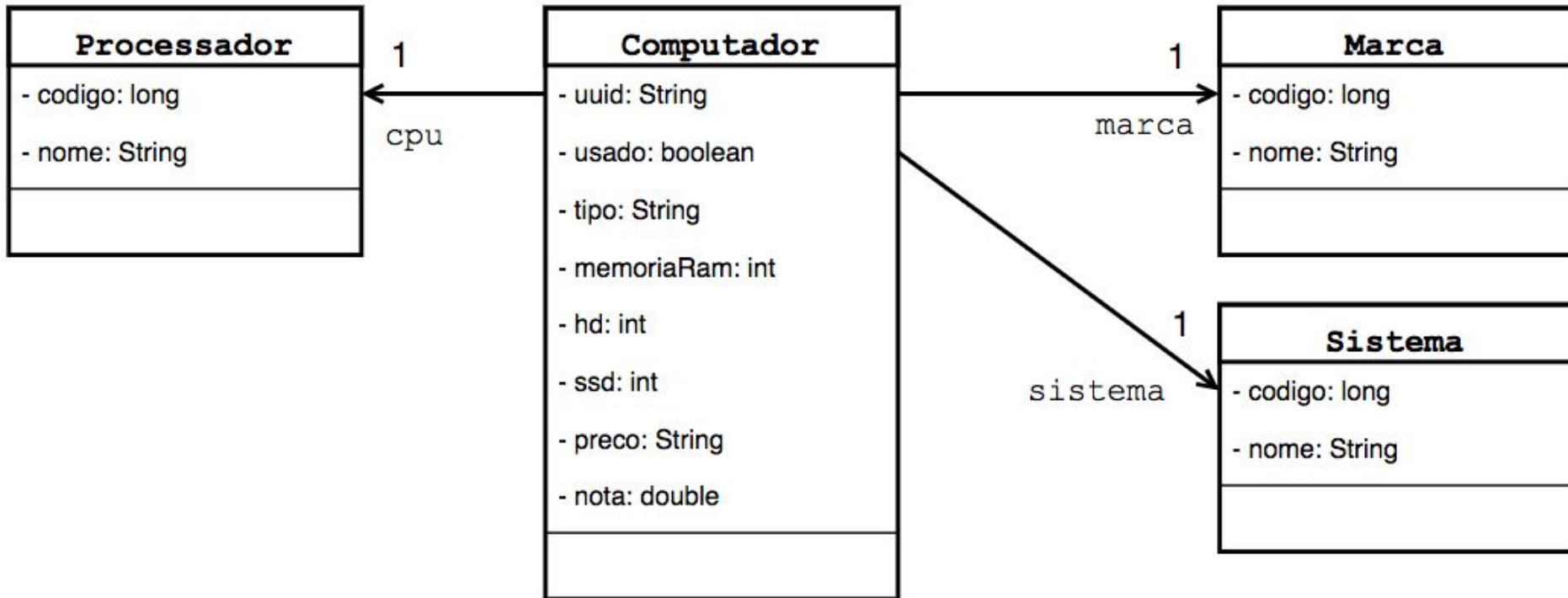
## Aula 10

# App salvando dados no *SQLite*



# App salvando dados no **SQLite**

## *Modelo de Domínio*



# Formulários Avançados

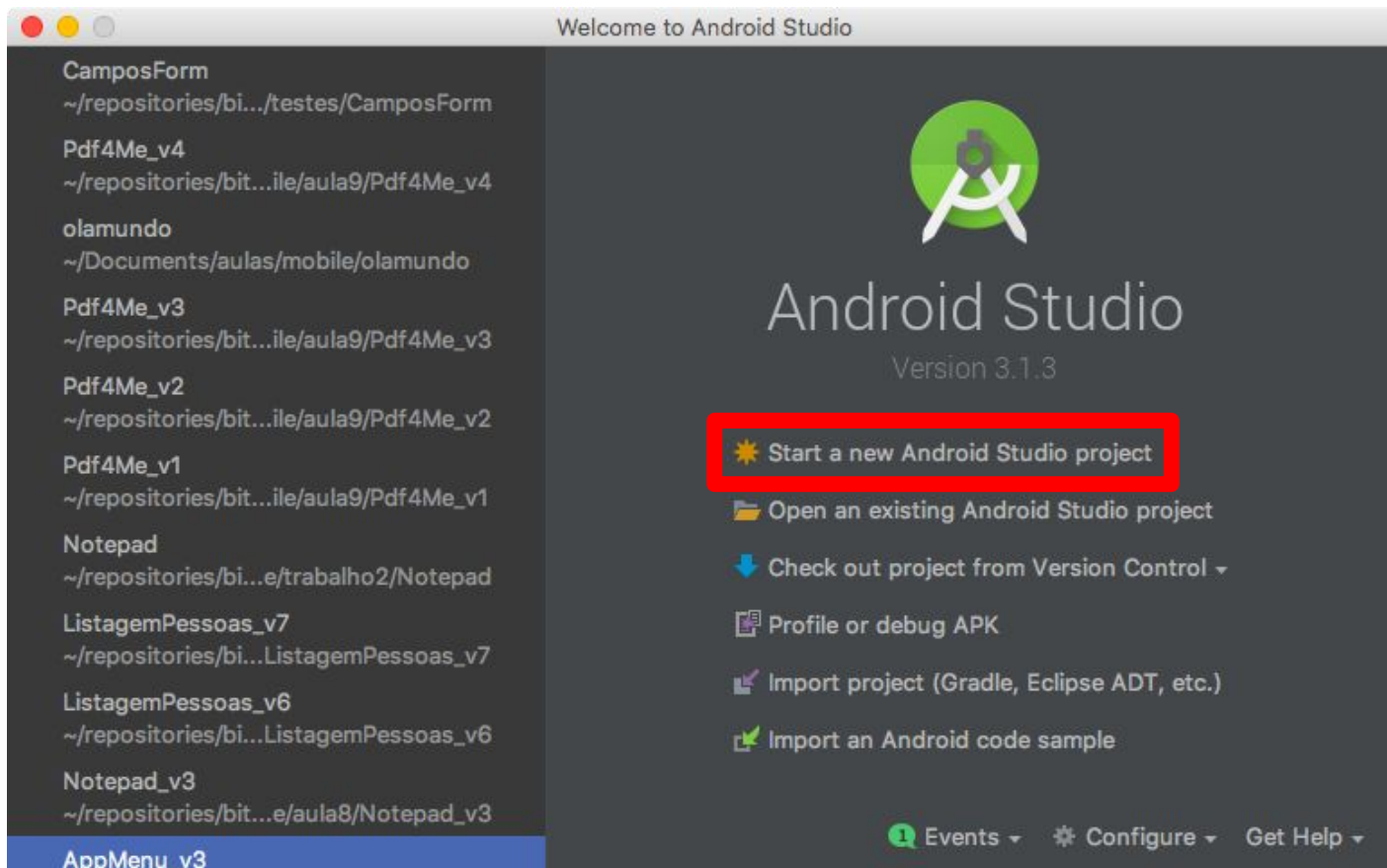
# Exercício em Sala

Crie a aplicação *DiaBoca* com os seguintes detalhes:

- API 15 e com uma *Empty Activity* chamada *CadastroActivity*
- O XML da *CadastroActivity* deve ter um *LinearLayout* vazio com orientação vertical
- Altere o tema para `Theme.AppCompat.Light` (em *styles.xml*) para que o título da *ActionBar* fique preto
- Defina as cores:

```
colorPrimary: #ffff00  
colorPrimaryDark: #7f6000  
colorAccent: #f6b26b
```

# Resolvendo o Exercício



# Resolvendo o Exercício

## Application name

DicaBoa

## Company domain

lgapontes.com

## Project location

/Users/lgapontes/repositories/bitbucket/aulas/desenvolvimento-mobile/aula10/DicaBoa\_v1

...

## Package name

com.lgapontes.dicaboas

Edit

☐ Include C++ support

☐ Include Kotlin support

# Resolvendo o Exercício

## ☒ Phone and Tablet

API 15: Android 4.0.3 (IceCreamSandwich) ▼

By targeting **API 15 and later**, your app will run on approximately **100%** of devices. [Help me choose](#)

☐ Include Android Instant App support

## ☐ Wear

API 21: Android 5.0 (Lollipop) ▼

## ☐ TV

API 21: Android 5.0 (Lollipop) ▼

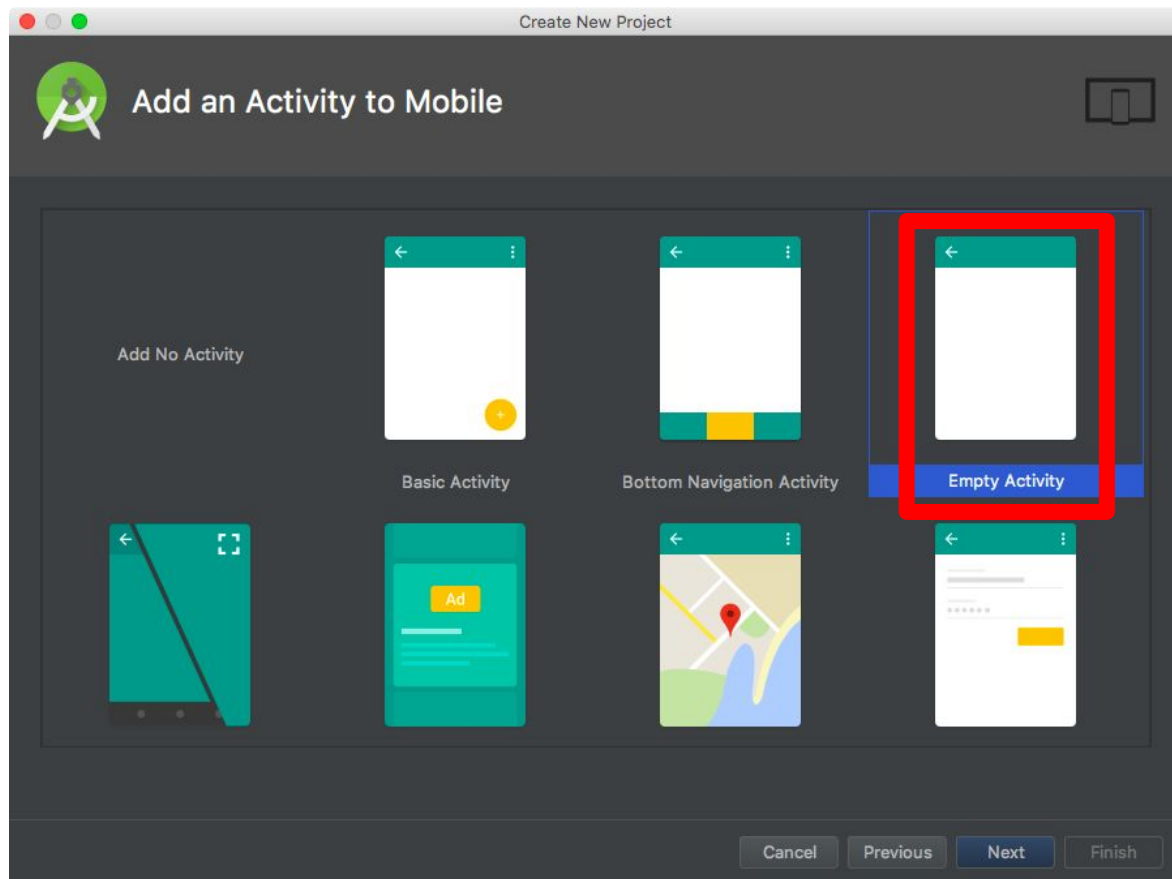
## ☐ Android Auto

## ☐ Android Things

API 24: Android 7.0 (Nougat) ▼

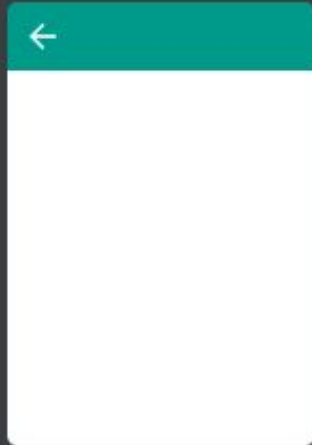


# Resolvendo o Exercício



# Resolvendo o Exercício

## Creates a new empty activity



Activity Name:

CadastroActivity

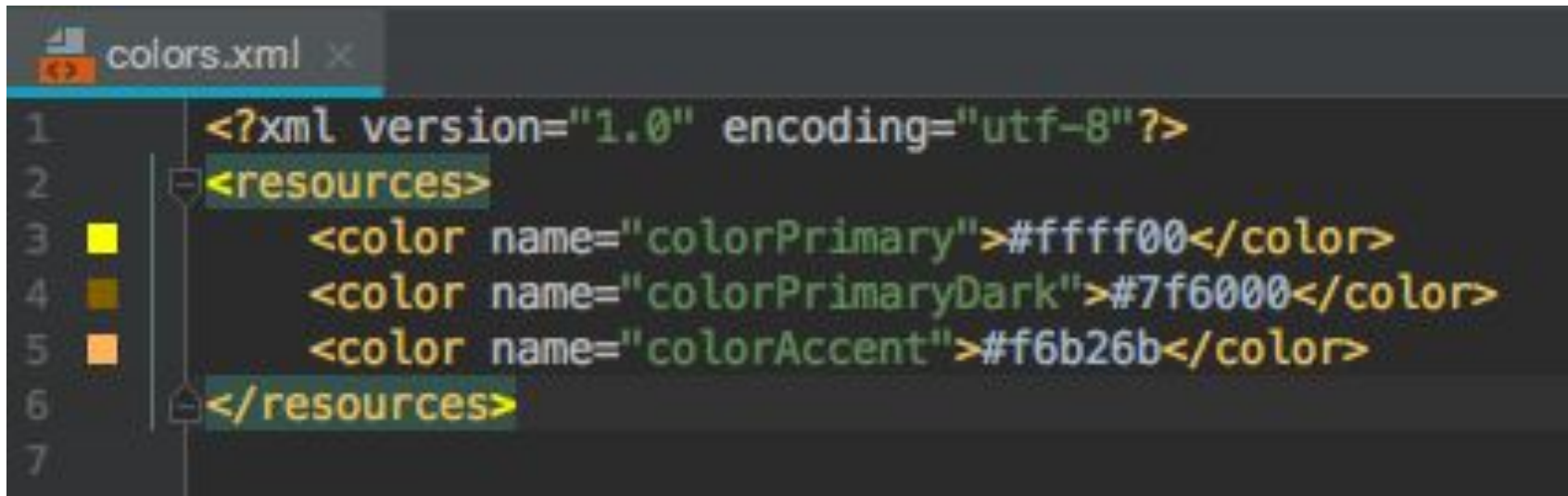
☒ Generate Layout File

Layout Name:

activity\_cadastro

☒ Backwards Compatibility (AppCompat)

# Resolvendo o Exercício



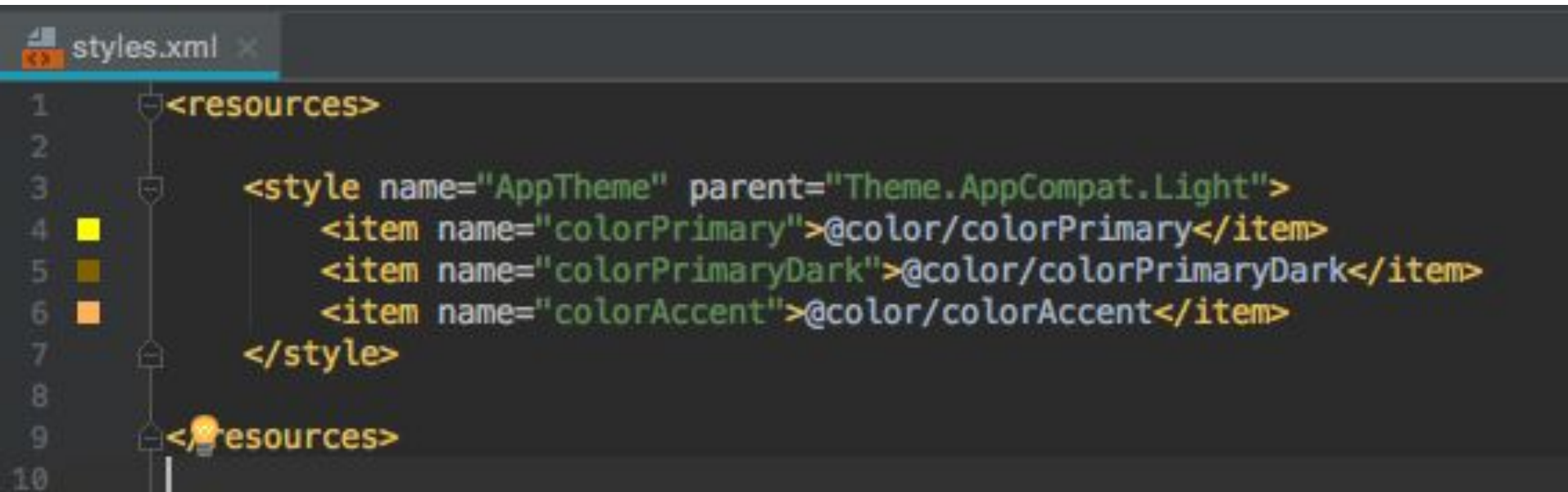
```
colors.xml x
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <color name="colorPrimary">#ffff00</color>
4     <color name="colorPrimaryDark">#7f6000</color>
5     <color name="colorAccent">#f6b26b</color>
6 </resources>
7
```

# Resolvendo o Exercício

```
activity_cadastro.xml x
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:orientation="vertical">
7
8 </LinearLayout>
```

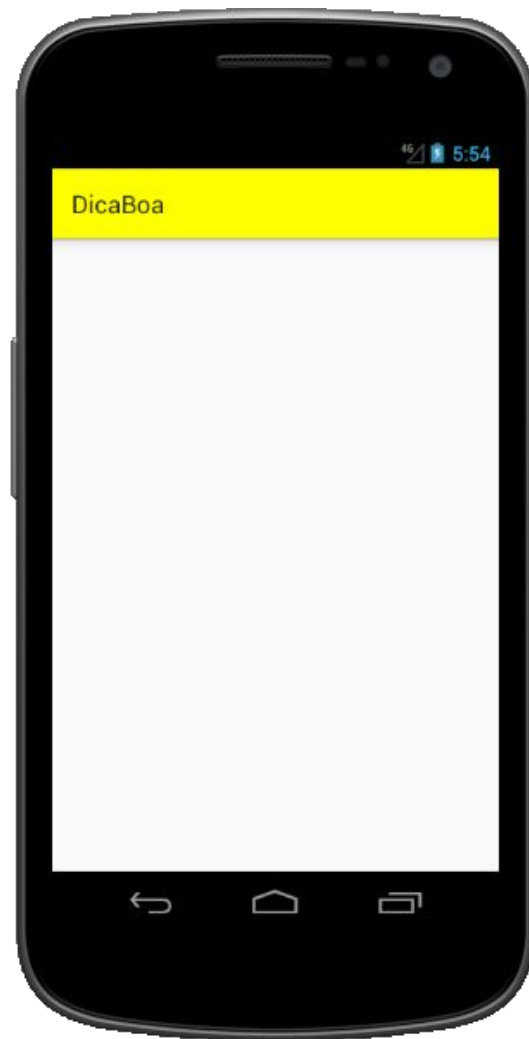
Lembre-se de definir a orientação vertical...

# Resolvendo o Exercício



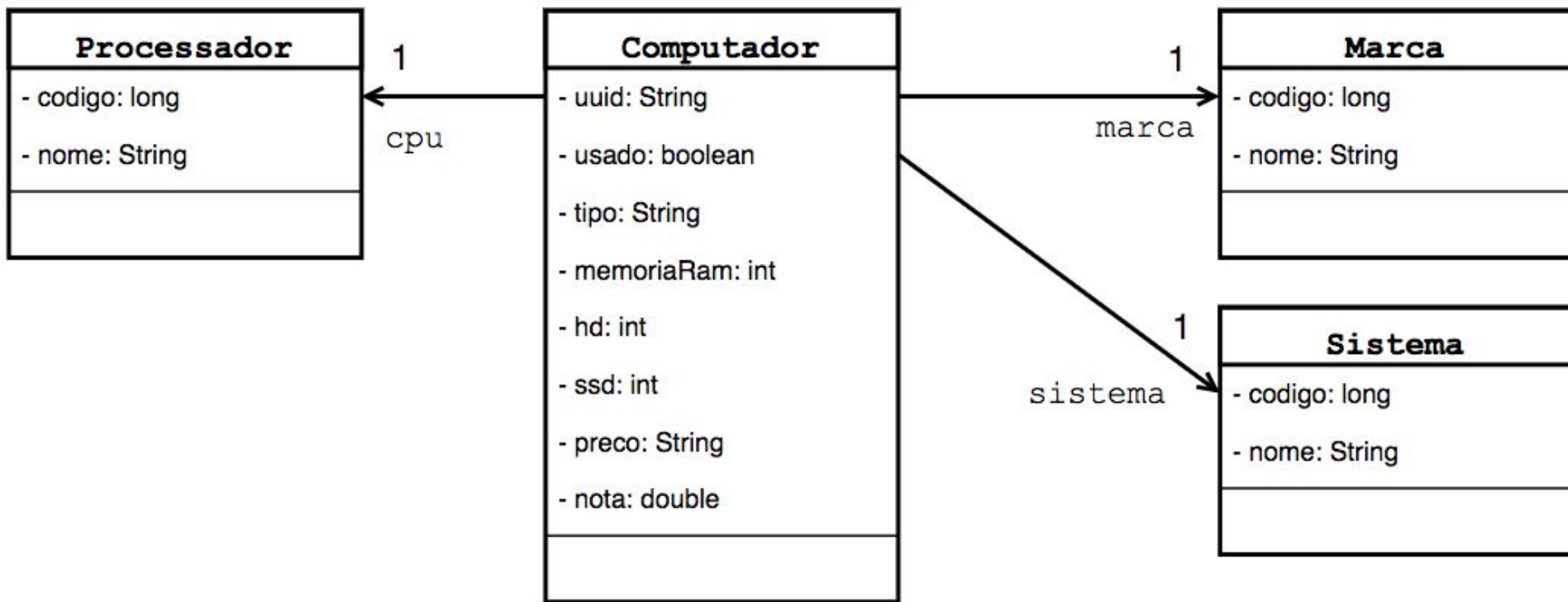
```
1 <resources>
2
3     <style name="AppTheme" parent="Theme.AppCompat.Light">
4         <item name="colorPrimary">@color/colorPrimary</item>
5         <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
6         <item name="colorAccent">@color/colorAccent</item>
7     </style>
8
9 </resources>
```

*DicaBoa\_v1*

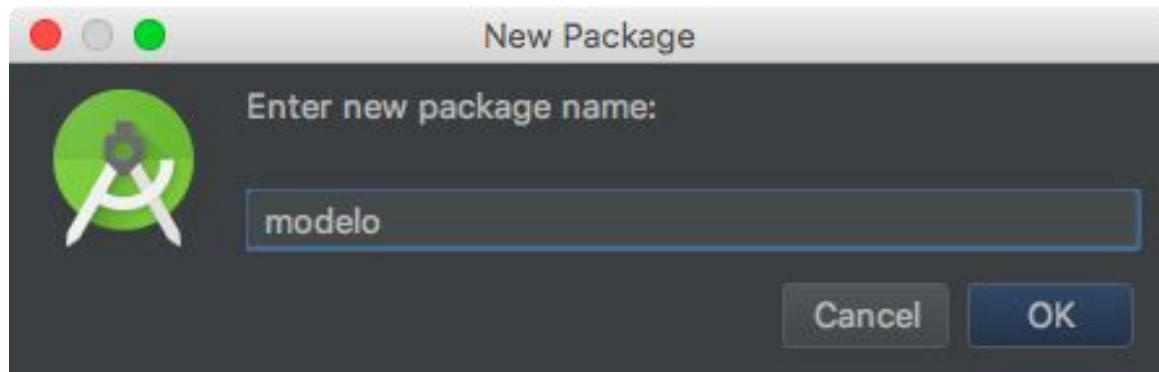
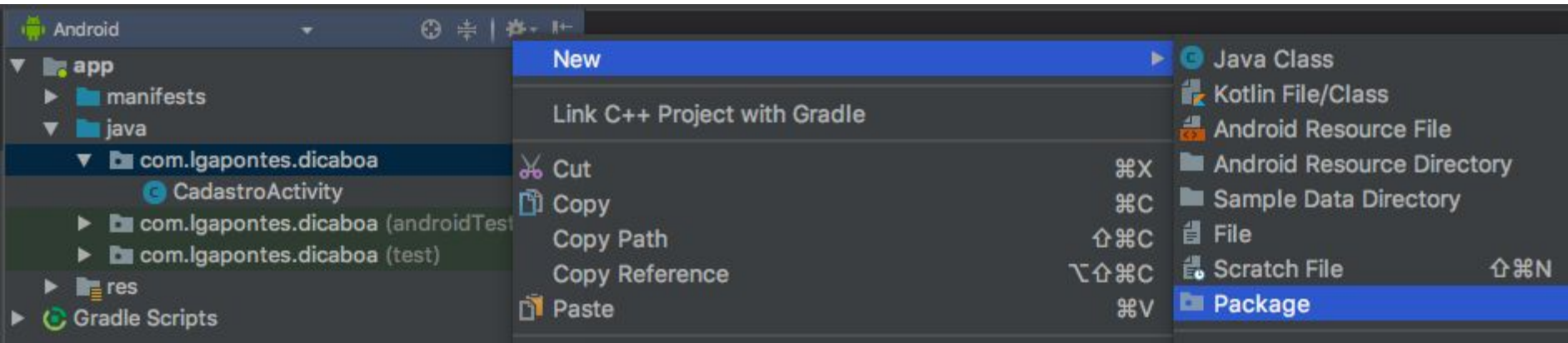


# Exercício em Sala

Crie as classes a seguir dentro de um pacote chamado `modelo`. As classes *Marca*, *Sistema* e *Processador* devem implementar o método `toString()`.

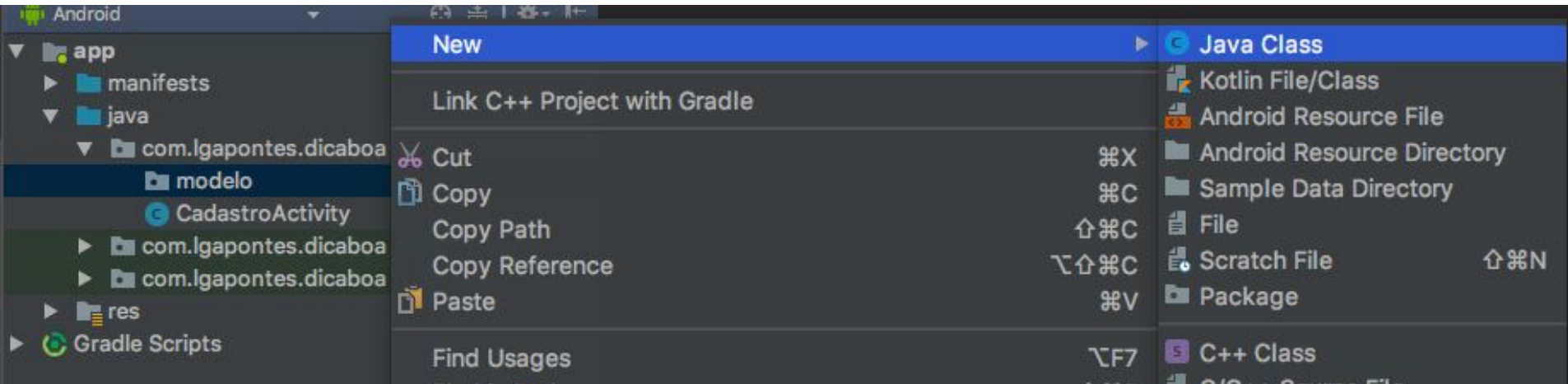


# Resolvendo o Exercício

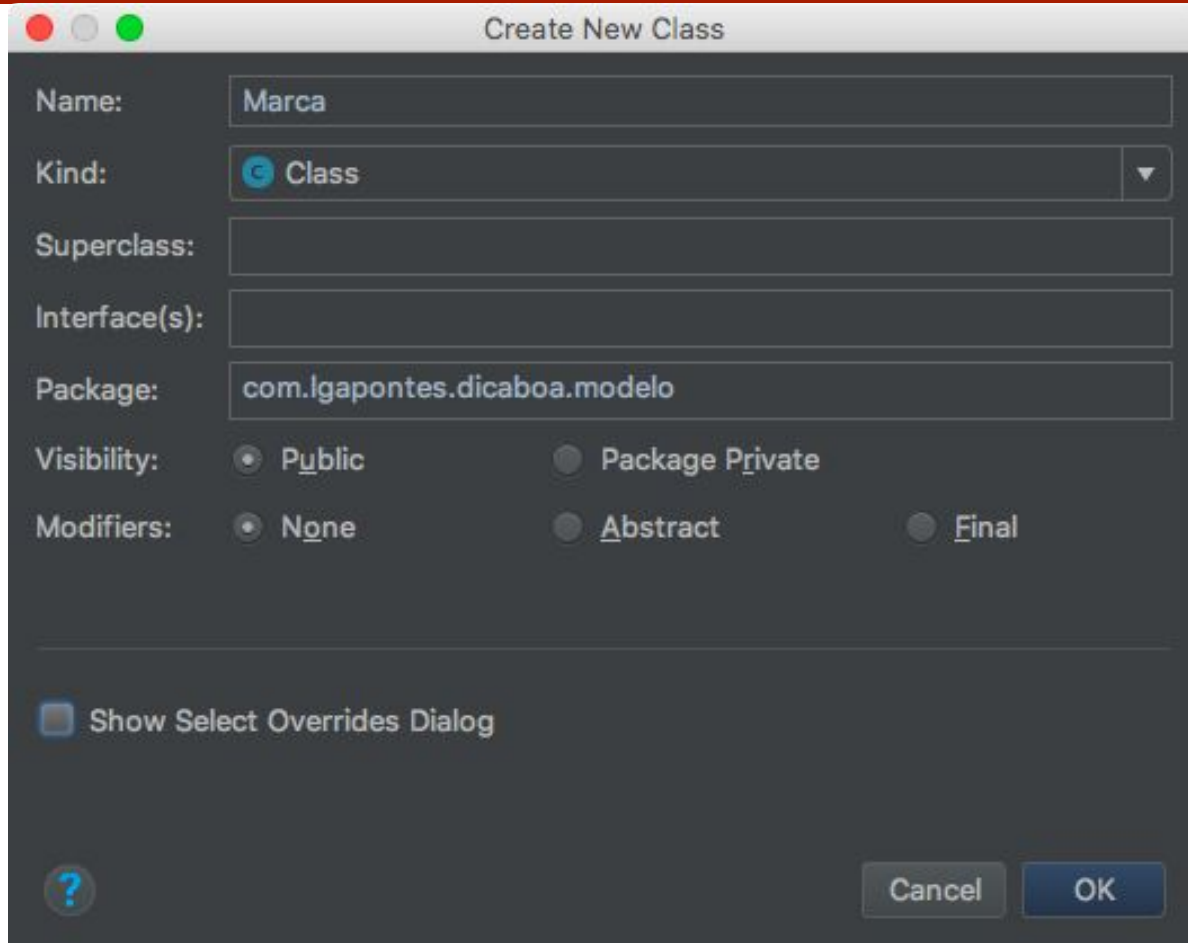




# Resolvendo o Exercício



# Resolvendo o Exercício

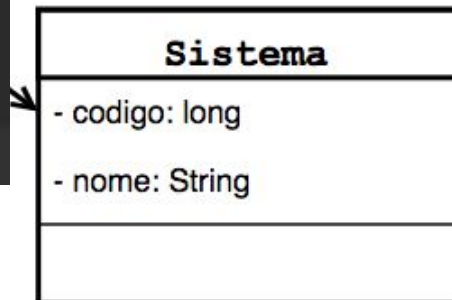
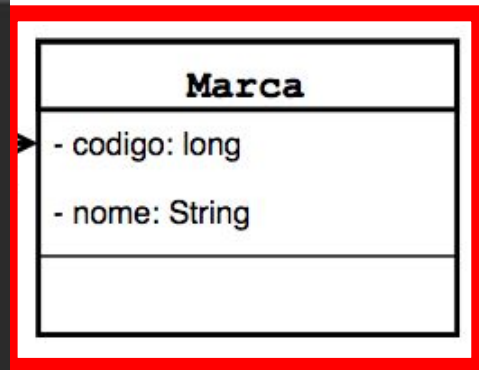
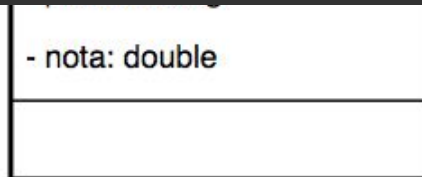


A screenshot of the 'Create New Class' dialog box in an IDE. The dialog has a title bar with standard macOS window controls (red, yellow, green buttons) and the title 'Create New Class'. The main area contains several fields and options:

- Name:** A text field containing 'Marca'.
- Kind:** A dropdown menu with 'Class' selected, indicated by a blue icon and a downward arrow.
- Superclass:** An empty text field.
- Interface(s):** An empty text field.
- Package:** A text field containing 'com.lgapontes.dicaboa.modelo'.
- Visibility:** Two radio buttons: 'Public' (selected) and 'Package Private'.
- Modifiers:** Three radio buttons: 'None' (selected), 'Abstract', and 'Final'.
- Show Select Overrides Dialog:** A checkbox that is currently unchecked.
- Buttons:** At the bottom right, there are 'Cancel' and 'OK' buttons. At the bottom left, there is a help icon (a question mark inside a circle).

# Resolvendo o Exercício

```
Marca.java x
1 package com.lgapontes.dicaboa.modelo;
2
3 public class Marca {
4
5     private long codigo;
6     private String nome;
7
8 }
9
```



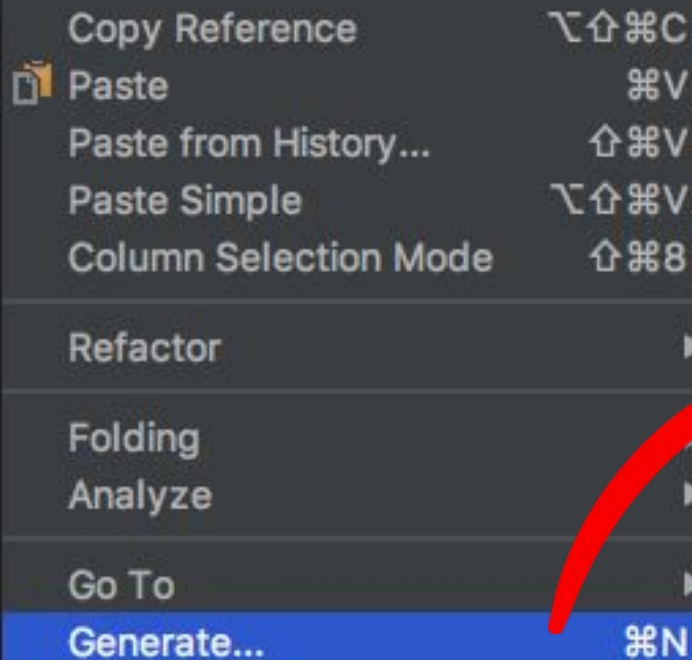
```
1 package com.lgapontes.dicaboa.modelo;
```

```
2  
3 public class Marca {
```

```
4  
5     private long codigo;
```

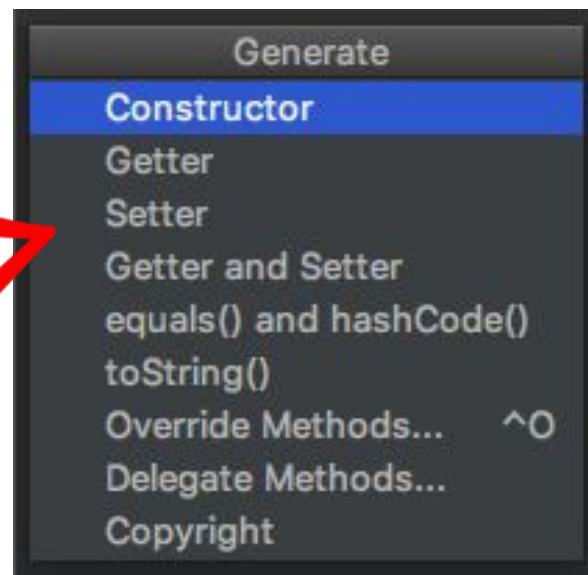
```
6     private String nome;
```

```
7  
8  
9  
10 }  
11
```



## Resolvendo o Exercício

Clique com o botão direito do mouse sobre uma área vazia abaixo dos atributos e selecione a opção "Generate...". Em seguida, selecione a opção "Constructor"



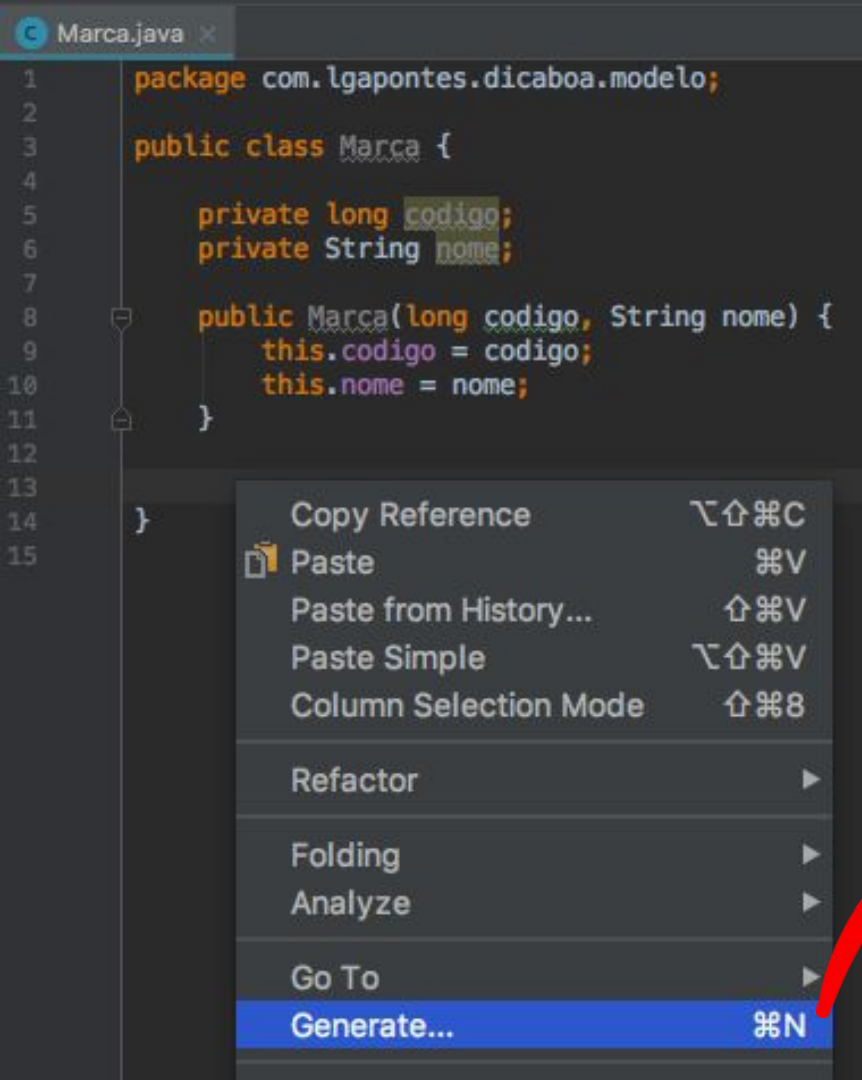
# Resolvendo o Exercício



Selecione os dois atributos e clique em OK. O código do constructor será gerado automaticamente.

```
public class Marca {  
  
    private long codigo;  
    private String nome;  
  
    public Marca(long codigo, String nome) {  
        this.codigo = codigo;  
        this.nome = nome;  
    }  
}
```

```
1 package com.lgapontes.dicaboa.modelo;
2
3 public class Marca {
4
5     private long codigo;
6     private String nome;
7
8     public Marca(long codigo, String nome) {
9         this.codigo = codigo;
10        this.nome = nome;
11    }
12
13 }
14
15 }
```



Copy Reference  $\text{Ctrl}+\text{Alt}+\text{C}$

Paste  $\text{Ctrl}+\text{V}$

Paste from History...  $\text{Alt}+\text{Ctrl}+\text{V}$

Paste Simple  $\text{Ctrl}+\text{Alt}+\text{V}$

Column Selection Mode  $\text{Alt}+\text{Ctrl}+\text{8}$

Refactor ▶

Folding ▶

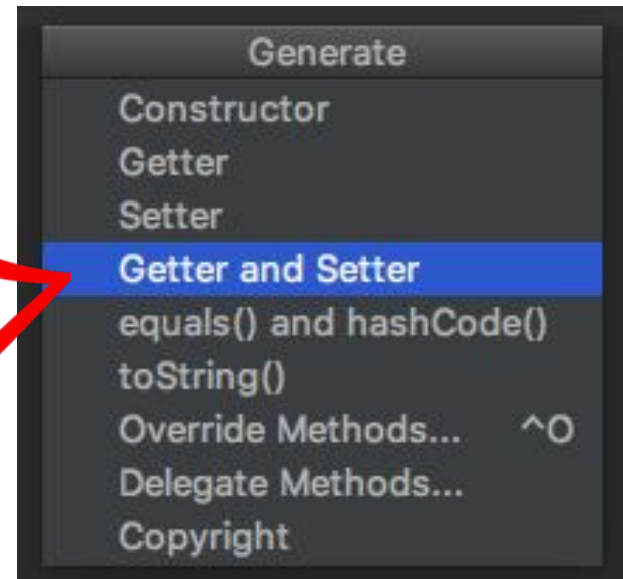
Analyze ▶

Go To ▶

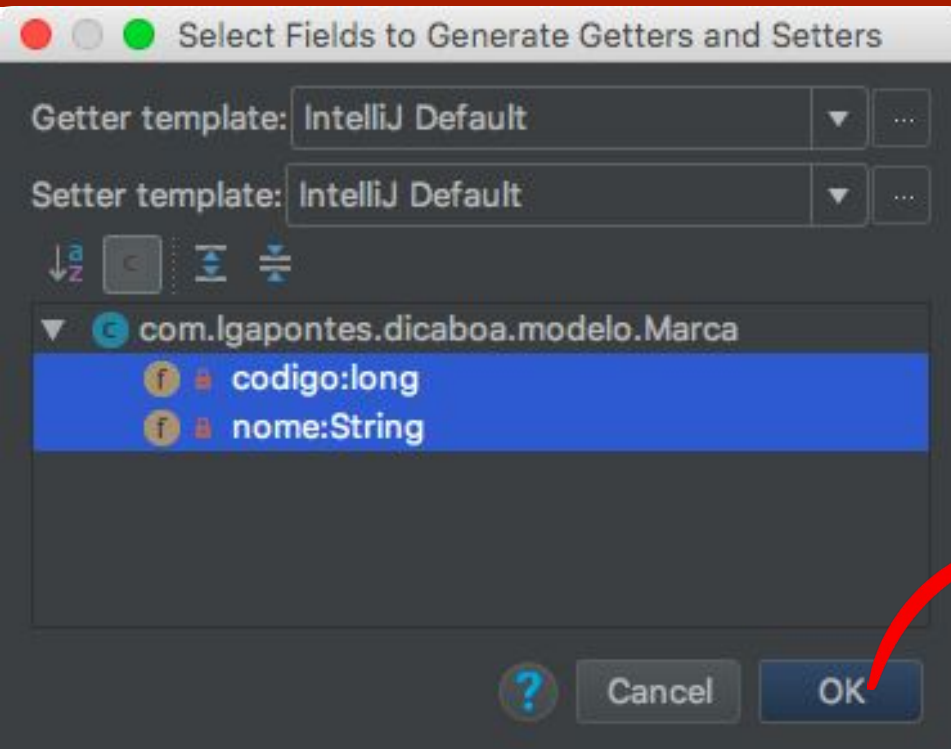
**Generate...  $\text{Ctrl}+\text{N}$**

# Resolvendo o Exercício

Agora clique com o botão direito do mouse sobre uma área vazia abaixo do constructor e selecione a opção "Generate...". Desta vez, selecione a opção "Getter and Setter"



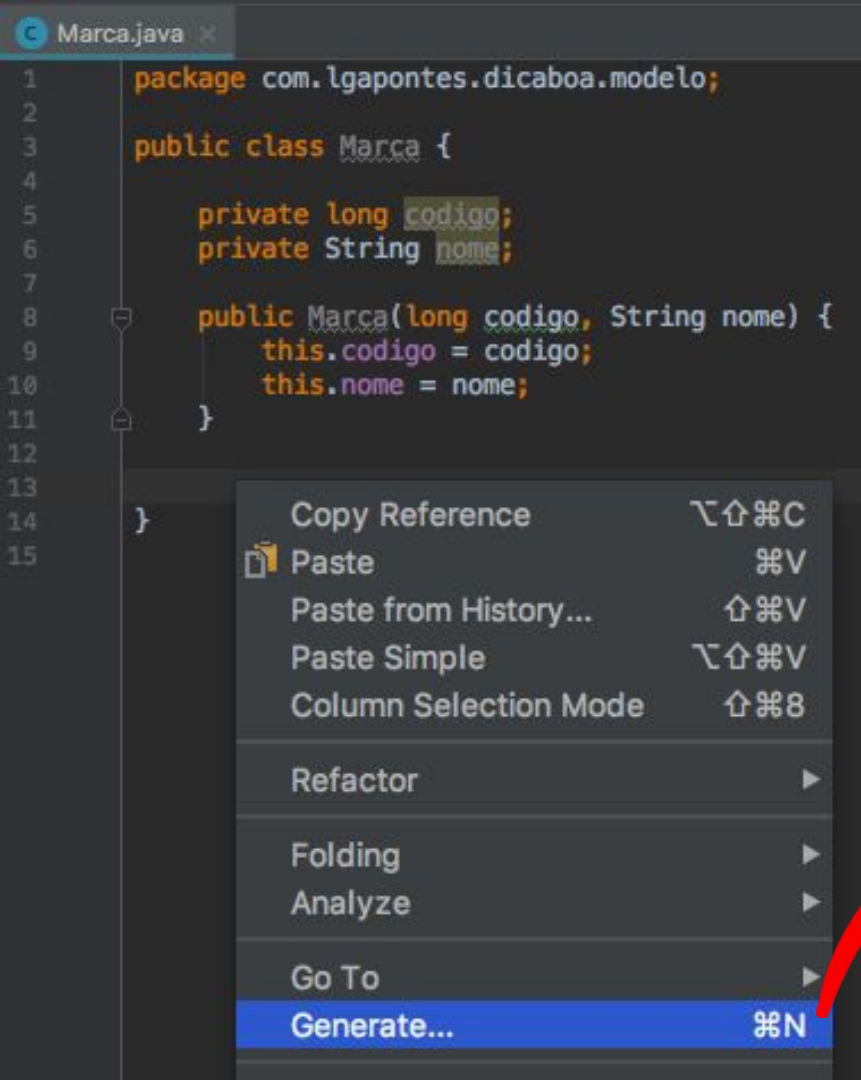
# Resolvendo o Exercício



```
public class Marca {  
  
    private long codigo;  
    private String nome;  
  
    public Marca(long codigo, String nome) {  
        this.codigo = codigo;  
        this.nome = nome;  
    }  
  
    public long getCodigo() {  
        return codigo;  
    }  
  
    public void setCodigo(long codigo) {  
        this.codigo = codigo;  
    }  
  
    public String getName() {  
        return nome;  
    }  
  
    public void setName(String nome) {  
        this.nome = nome;  
    }  
}
```



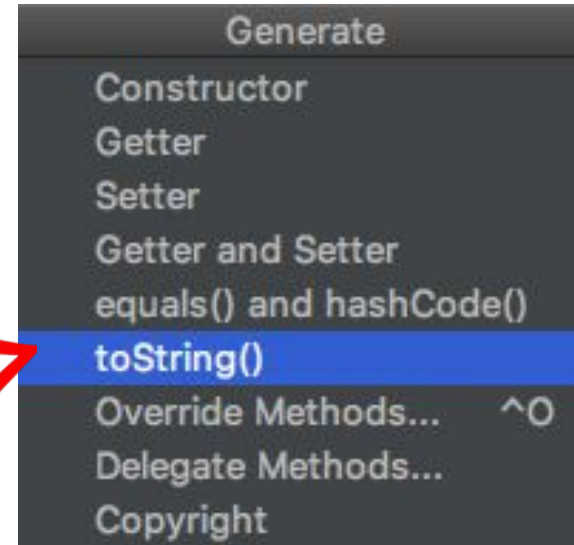
```
1 package com.lgapontes.dicaboa.modelo;
2
3 public class Marca {
4
5     private long codigo;
6     private String nome;
7
8     public Marca(long codigo, String nome) {
9         this.codigo = codigo;
10        this.nome = nome;
11    }
12
13 }
14
15
```



The screenshot shows an IDE window titled 'Marca.java'. The code defines a 'Marca' class with two private attributes: 'codigo' (long) and 'nome' (String). It also has a constructor that initializes these attributes. A context menu is open over the code, with 'Generate...' highlighted at the bottom. Other options in the menu include 'Copy Reference', 'Paste', 'Paste from History...', 'Paste Simple', 'Column Selection Mode', 'Refactor', 'Folding', 'Analyze', 'Go To', and 'Generate...'.

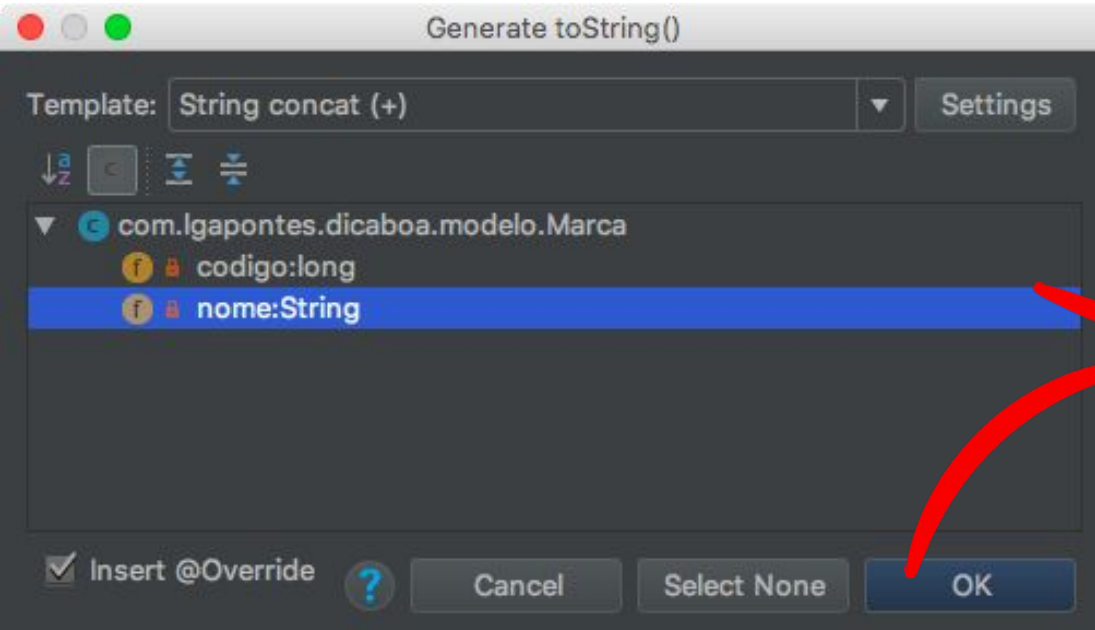
## Resolvendo o Exercício

Por fim clique com o botão direito do mouse sobre uma área vazia abaixo dos métodos e selecione a opção "Generate...". Desta vez, selecione a opção "toString()".





# Resolvendo o Exercício



O código gerado retorna uma String em um formato que não ficará adequado ao nosso caso

Retorne apenas o atributo *nome*

```
@Override
public String toString() {
    return "Marca{" +
        "nome=" + nome + '\n' +
    }
}
```

```
@Override
public String toString() {
    return nome;
}
```

Desta vez selecione apenas o campo *nome*

```
public class Sistema {
```

```
    private long codigo;  
    private String nome;
```

```
    public Sistema(long codigo, String nome) {  
        this.codigo = codigo;  
        this.nome = nome;  
    }
```

```
    public long getCodigo() { return codigo; }
```

```
    public void setCodigo(long codigo) { this.codigo = codigo; }
```

```
    public String getNome() { return nome; }
```

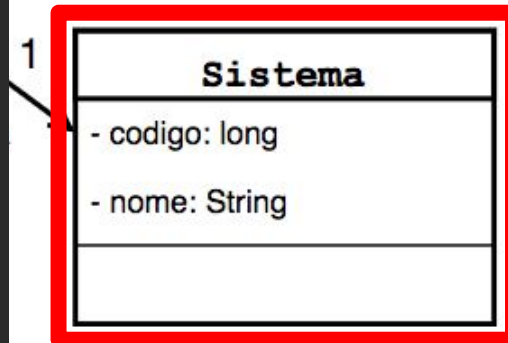
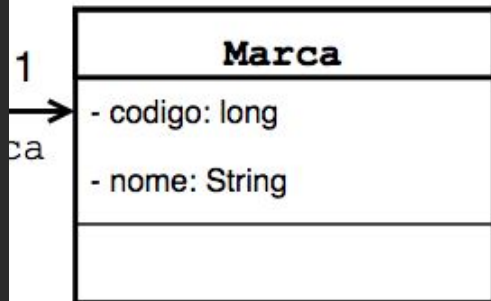
```
    public void setNome(String nome) { this.nome = nome; }
```

```
    @Override
```

```
    public String toString() {  
        return nome;  
    }
```

```
}
```

## Resolvendo o Exercício



# Resolvendo o Exercício

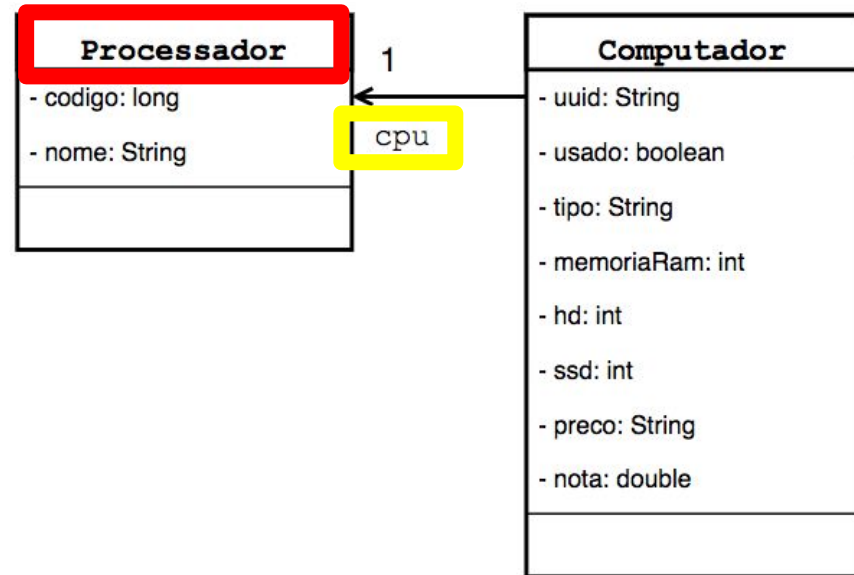
## Processador

- codigo: long
- nome: String

```
public class Processador {  
  
    private long codigo;  
    private String nome;  
  
    public Processador(long codigo, String nome) {  
        this.codigo = codigo;  
        this.nome = nome;  
    }  
  
    public long getCodigo() { return codigo; }  
  
    public void setCodigo(long codigo) { this.codigo = codigo; }  
  
    public String getNome() { return nome; }  
  
    public void setNome(String nome) { this.nome = nome; }  
  
    @Override  
    public String toString() {  
        return nome;  
    }  
}
```

# Resolvendo o Exercício

```
Computador.java x
1 package com.lgapontes.dicaboa.modelo;
2
3 public class Computador {
4
5     private String uuid;
6
7     private Marca marca;
8     private Sistema sistema;
9     private Processador cpu;
10
11     private boolean usado;
12     private String tipo;
13     private int memoriaRam;
14     private int hd;
15     private int ssd;
16     private String preco;
17     private double nota;
18
19 }
20
```



Veja que as associações representadas no UML são codificadas colocando as próprias classes como tipo de dado dos atributos *marca*, *sistema* e *cpu*.

# Resolvendo o Exercício

```
Computador.java x
1 package com.lgapontes.dicaboa.modelo;
2
3 public class Computador {
4
5     private String uuid;
6
7     private Marca marca;
8     private Sistema sistema;
9     private Processador cpu;
10
11     private boolean usado;
12     private String tipo;
13     private int memoriaRam;
14     private int hd;
15     private int ssd;
16     private String preco;
17     private double nota;
18
19 }
20
```

Neste caso, como existem muitos atributos, um constructor com todos eles seria uma má prática.

Crie apenas os métodos *Getters and Setters*.

## Dica Extra

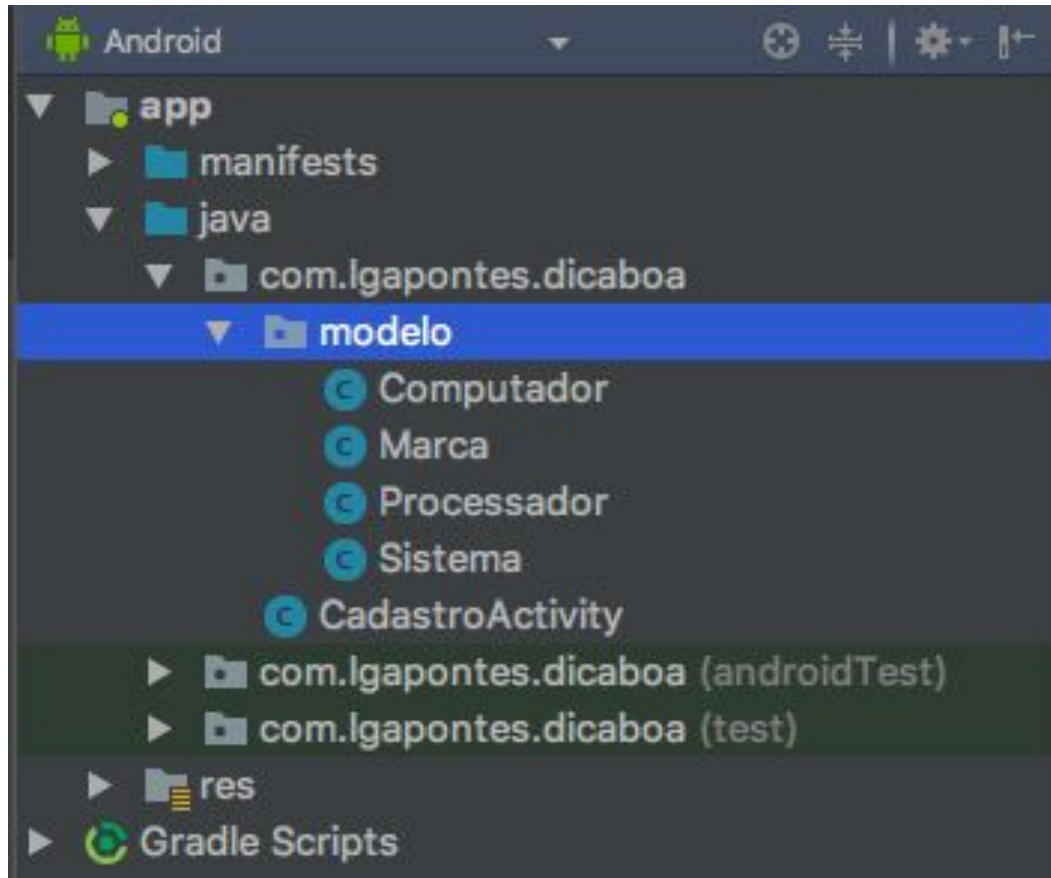
Existe um padrão de projeto chamado *Fluent Interface Pattern* bastante utilizado nas linguagens de programação no geral. Veja a seguir um exemplo da biblioteca jOOQ.

```
Author author = AUTHOR.as("author");  
create.selectFrom(author)  
    .where(exists(selectOne()  
        .from(BOOK)  
        .where(BOOK.STATUS.eq(BOOK_STATUS.SOLD_OUT))  
        .and(BOOK.AUTHOR_ID.eq(author.ID)))));
```

Caso você queira criar algo semelhante para a classe *Computador*, veja mais detalhes em: <https://martinfowler.com/bliki/FluentInterface.html>



# Resolvendo o Exercício



# App salvando dados no **SQLite**

Computador
- uuid: String
- usado: boolean
- tipo: String
- memoriaRam: int
- hd: int
- ssd: int
- preco: String
- nota: double

*Ok, mas para que serve esse **UUID**?*





*Explicar no quadro:*

- App local com SQLite
- App distribuído com SQLite e ID sequence
- App distribuído com ID no servidor
- IDs aleatórios
- App distribuído com UUID

- 32 hexadecimais e 4 hífens

123e4567-e89b-12d3-a456-426655440000



# App salvando dados no **SQLite**

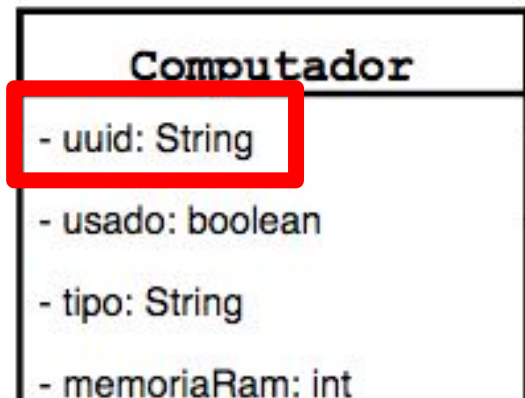
*Ok, mas como vamos  
implementar **UUID** no Java?*



Computador
- uuid: String
- usado: boolean
- tipo: String
- memoriaRam: int
- hd: int
- ssd: int
- preco: String
- nota: double

# App salvando dados no *SQLite*

```
public Computador() {  
    this.uuid = UUID.randomUUID().toString();  
}  
  
public Computador(String uuid) {  
    this.uuid = uuid;  
}
```



## PASSO 1

Crie 2 constructors:

- Um deles sem parâmetros utilizando a classe `java.util.UUID` disponível no Java. Sempre que criarmos um *Computador* por este constructor ele automaticamente terá um UUID definido.
- O outro deve receber o UUID por parâmetro. Este caso será necessário para quando quisermos criar objetos para os registros que estiverem salvos no banco de dados.

# App salvando dados no *SQLite*

## PASSO 2

Vamos organizar os *Getters and Setters*:

- Apague o método *setUuid()*. Ele não será mais necessário.
- Renomeie o *getUuid()* para *getUUID()*. É uma boa prática manter acrônimos em letras maiúsculas.

```
public String getUUID() {  
    return uuid;  
}
```

```
public void setUuid(String uuid) {  
    this.uuid = uuid;  
}
```

- hd: int  
- ssd: int  
- preco: String  
- nota: double

Rode o App só para testarmos se está tudo funcionando.

Vamos agora implementar os campos do formulário para preencher o objeto *Computador*.



# Utilizando um *Spinner* (*DicaBoa\_v2*)

*Spinner* é um campo de entrada através do qual os usuários podem selecionar um valor entre uma lista de valores.



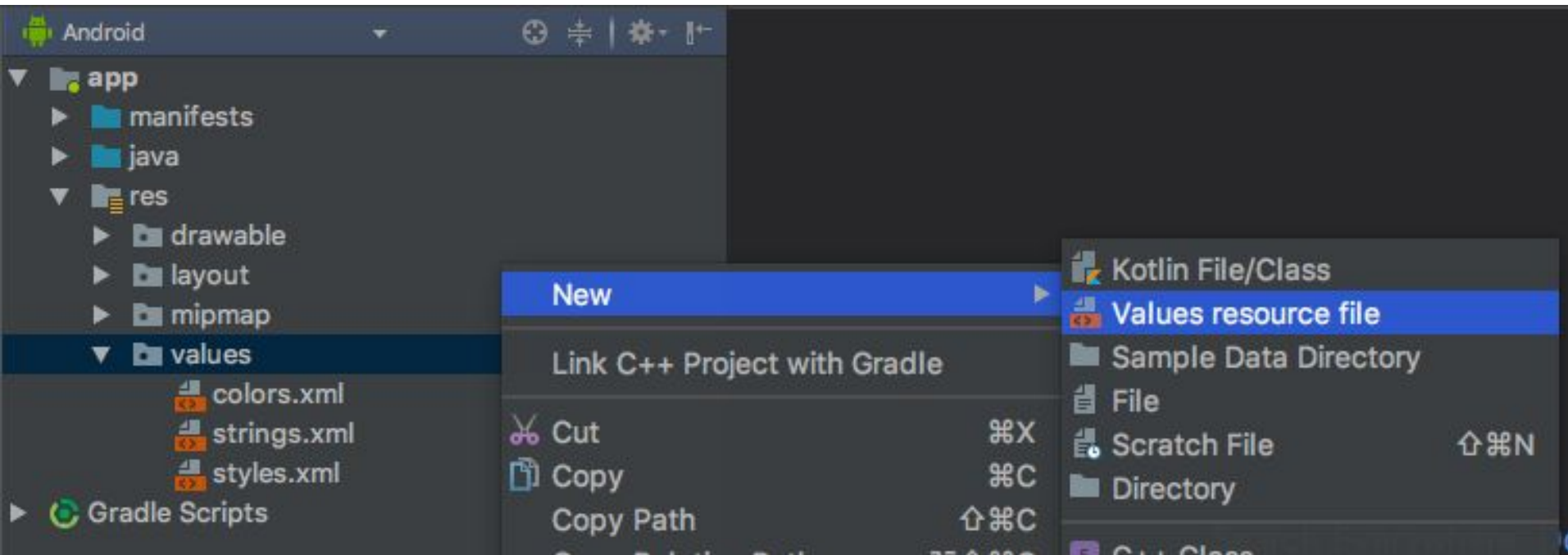
## Dica Extra



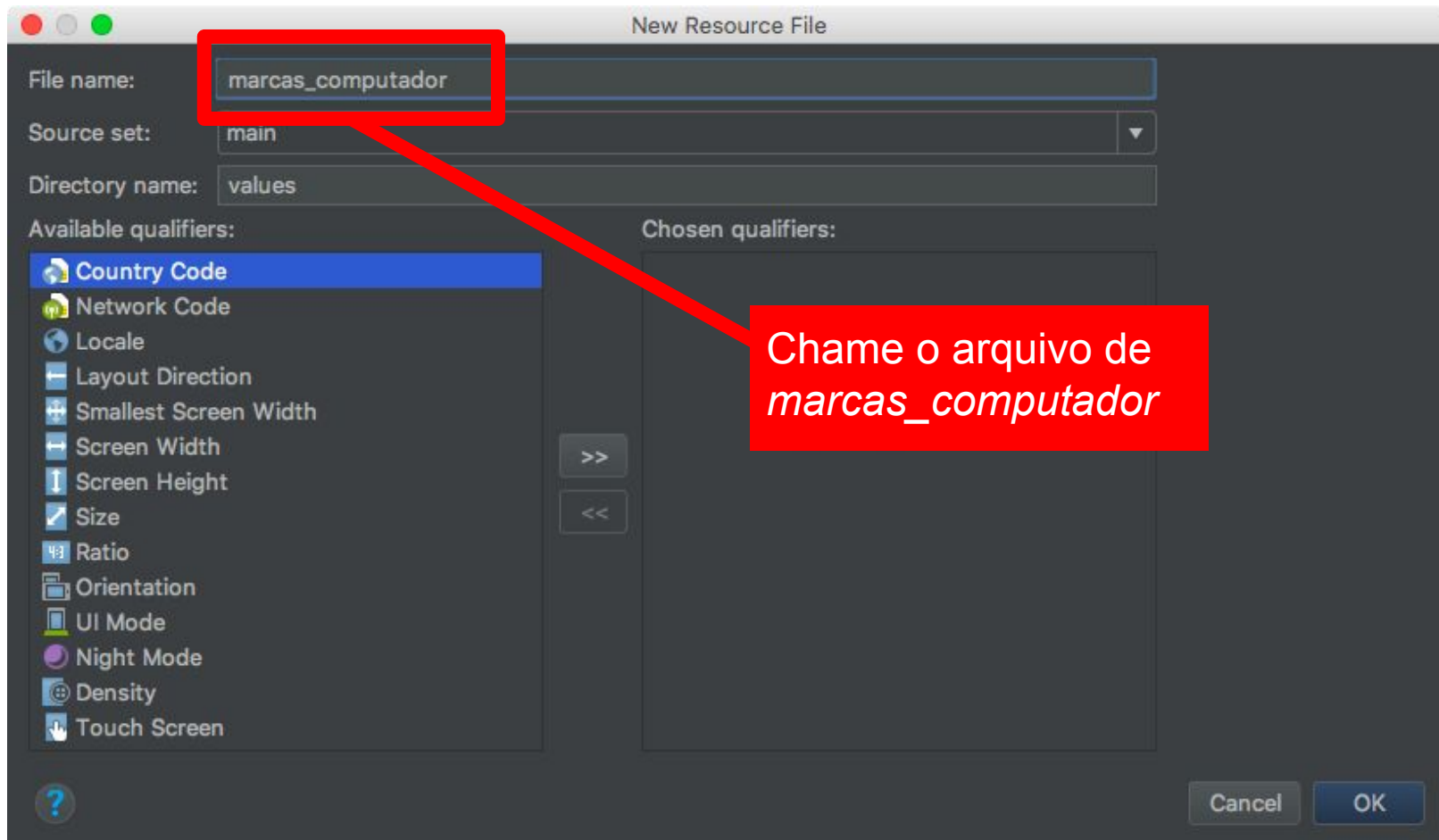
No mundo desktop, ele seria o equivalente aos componentes conhecidos como *Drop-down*, `<select>`, *Combobox*, entre outros.

# Utilizando um *Spinner* (*DicaBoa\_v2*)

Vamos começar criando um *Spinner* estático, cujos valores serão estaticamente definidos em um arquivo XML.



# Utilizando um *Spinner* (*DicaBoa\_v2*)





# Utilizando um *Spinner* (DicaBoa\_v2)

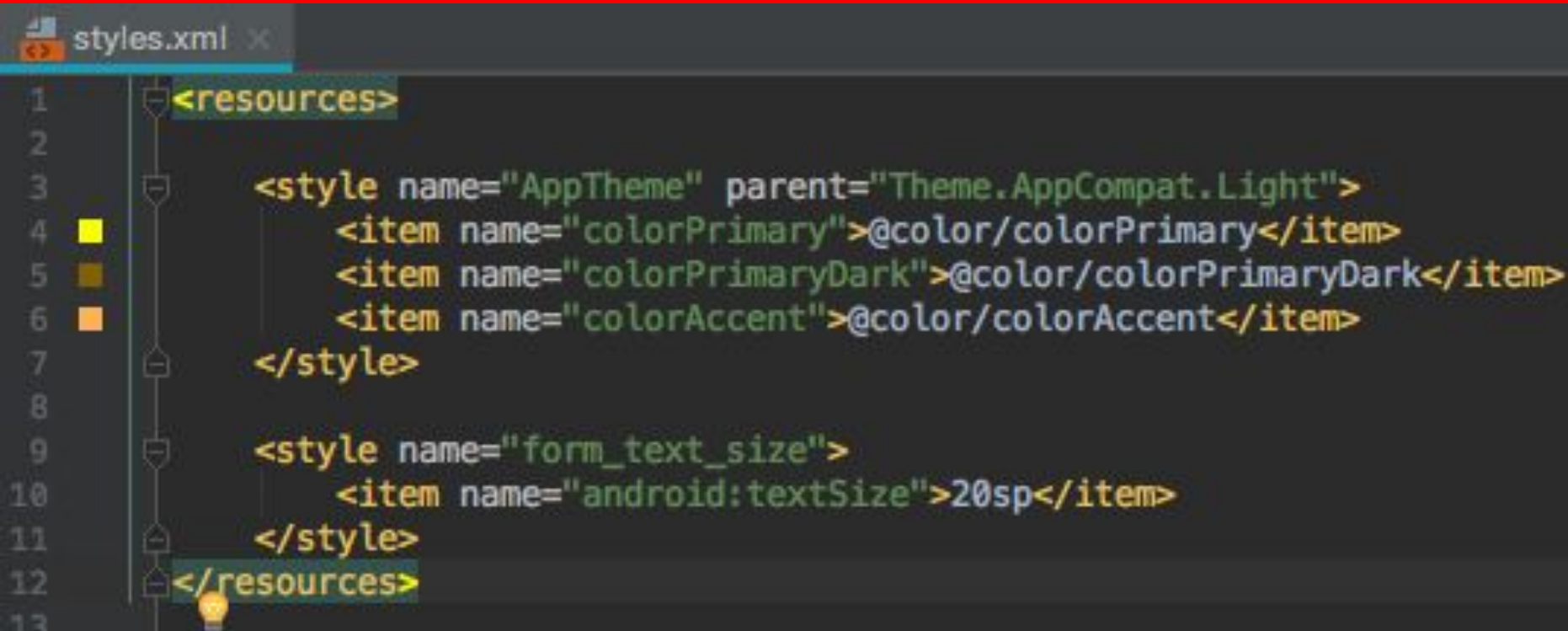
```
marcas_computador.xml x
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string-array name="marcas_computador">
4          <item>Acer</item>
5          <item>Dell</item>
6          <item>Lenovo</item>
7          <item>Samsung</item>
8          <item>HP</item>
9          <item>Apple</item>
10         <item>LG</item>
11         <item>AOC</item>
12         <item>Asus</item>
13         <item>Avell</item>
14     </string-array>
15 </resources>
```

Dentro da tag *resources*, inclua uma tag *string-array* e para cada uma das marcas, acrescente uma tag *item*.

O nome indicado no atributo *name* do *string-array* será utilizado para referenciar esta listagem no *Spinner*.

## Utilizando um *Spinner* (DicaBoa\_v2)

Vamos criar um estilo chamado *form\_text\_size* para padronizar o tamanho do texto de todos os elementos do formulário. Coloque o *item textSize* igual a 20sp.



```
1 <resources>
2
3     <style name="AppTheme" parent="Theme.AppCompat.Light">
4         <item name="colorPrimary">@color/colorPrimary</item>
5         <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
6         <item name="colorAccent">@color/colorAccent</item>
7     </style>
8
9     <style name="form_text_size">
10         <item name="android:textSize">20sp</item>
11     </style>
12 </resources>
13
```

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:stretchColumns="0,1">
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:text="Marca"
                android:theme="@style/form_text_size" />
            <Spinner
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:theme="@style/form_text_size"
                android:entries="@array/marcas_computador" />
        </TableRow>
    </TableLayout>
</LinearLayout>

```

Vamos agora organizar o layout do arquivo *activity\_cadastro.xml*

## PASSO 1

Crie uma *TableLayout* ocupando toda a área do pai e definindo duas colunas

## PASSO 2

Crie uma *TableRow* conforme o código ao lado. Coloque uma *marginTop* de 10dp para criar um espaçamento adequado.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:stretchColumns="0,1">
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:text="Marca"
                android:theme="@style/form_text_size" />
            <Spinner
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:theme="@style/form_text_size"
                android:entries="@array/marcas_computador" />
        </TableRow>
    </TableLayout>
</LinearLayout>

```

## Dentro do *TableRow*:

### PASSO 3

Crie um *TextView* com o texto "Marca" (preferencialmente extraído do *strings.xml*), com *marginLeft* igual a 10dp e utilizando o tema *form\_text\_size*

### PASSO 4

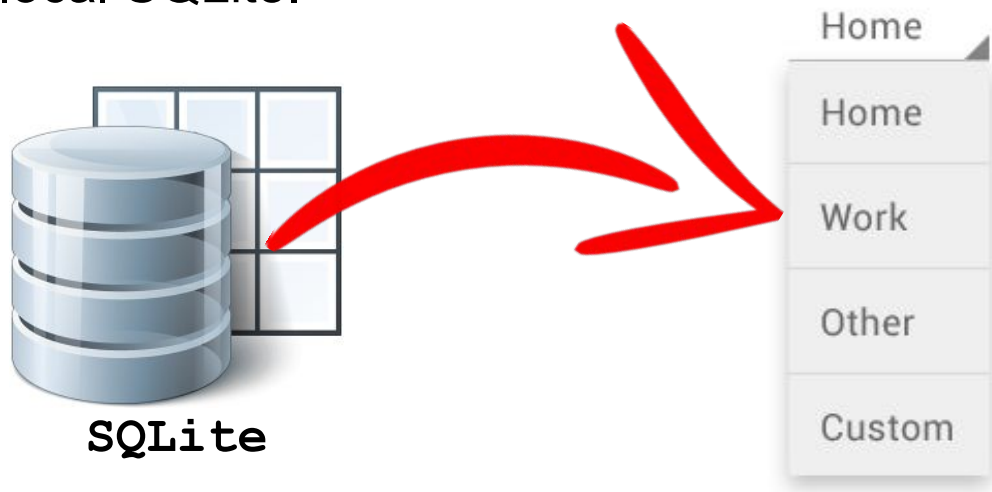
Crie um *Spinner*. O atributo *entries* serve para apontar para os valores definidos no XML *marcas\_computador*. Utilize também o tema *form\_text\_size*





## Utilizando um *Spinner* (*DicaBoa\_v3*)

É claro que a abordagem anterior só poderá ser utilizada quando tivermos uma listagem fixa. No nosso caso, pretendemos obter todos os valores a partir do banco local SQLite.



Como ainda não criamos o acesso ao SQLite, vamos converter a listagem do *resources* para valores obtidos programaticamente da classe *CadastroActivity.java*

## Utilizando um *Spinner* (DicaBoa\_v3)

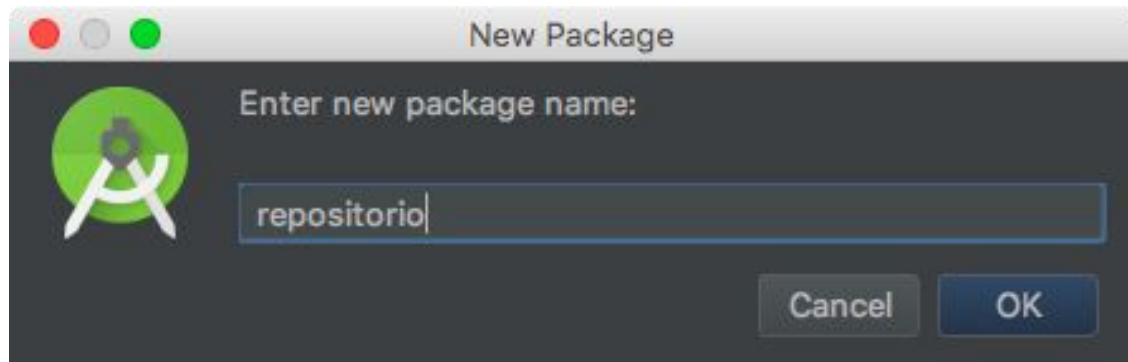
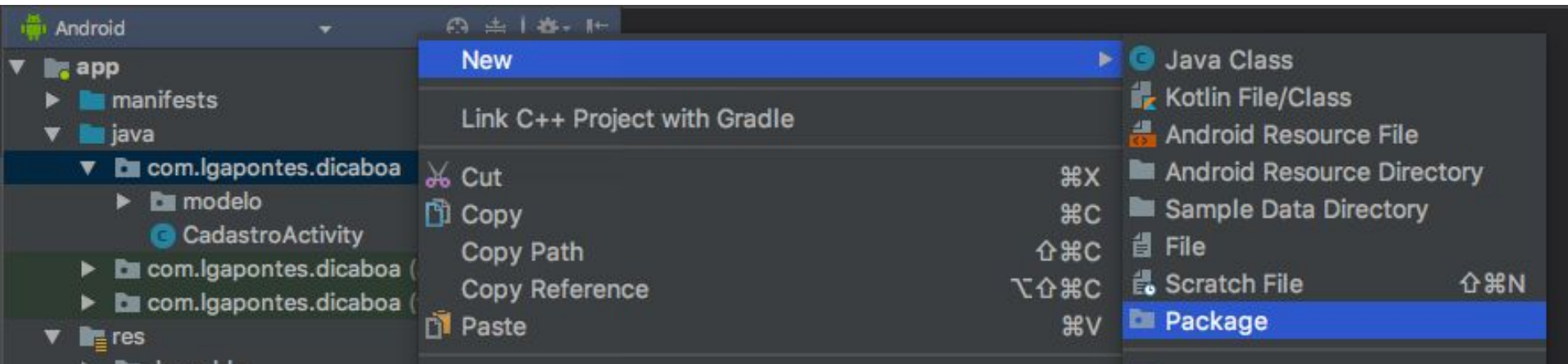
```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:text="Marca"
        android:theme="@style/form_text_size" />
    <Spinner
        android:id="@+id/form_spinner_marcas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:theme="@style/form_text_size" />
</TableRow>
```

Vamos alterar o código do *Spinner* do arquivo *activity\_cadastro.xml* para retirar o atributo *entries* e para acrescentar um ID.

Exclua também o arquivo *marcas\_computador.xml*. Ele não será necessário.

# Utilizando um *Spinner* (DicaBoa\_v3)

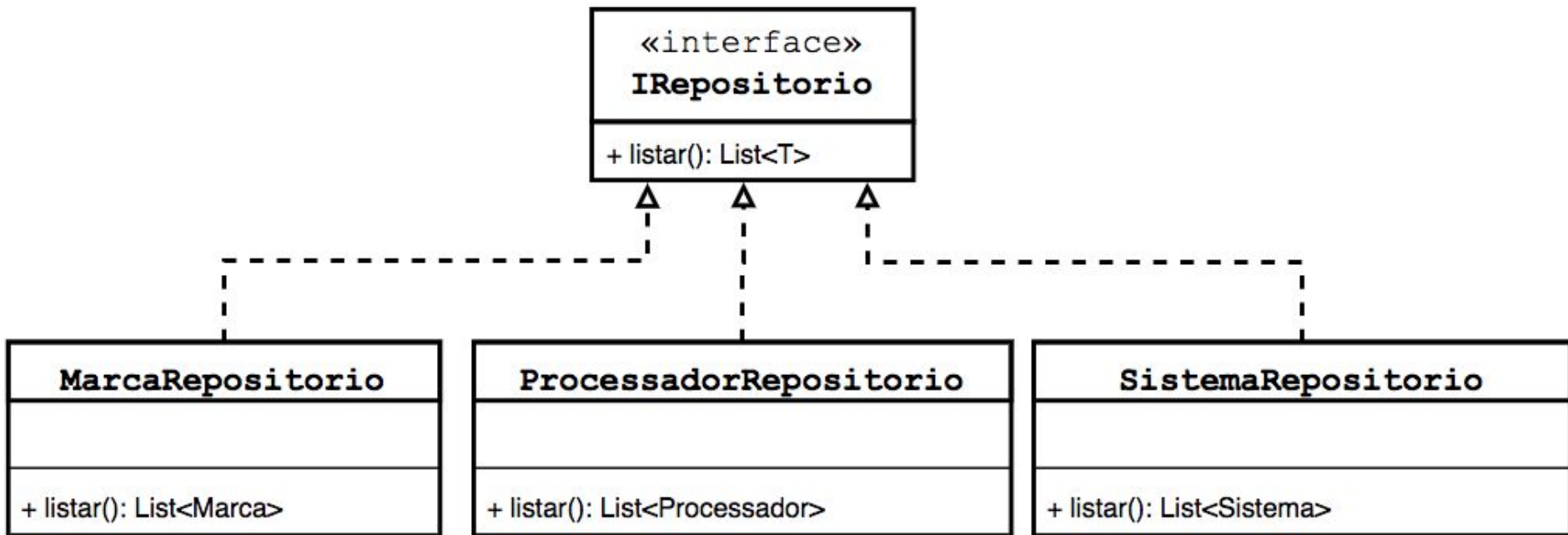
Dentro do pacote principal da aplicação, crie um pacote chamado *repositorio*





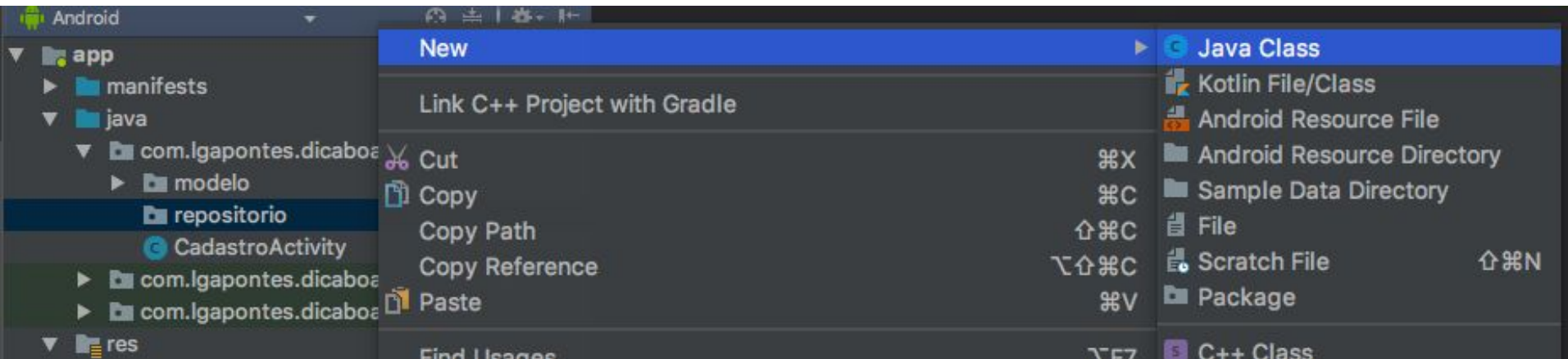
# Utilizando um *Spinner* (DicaBoa\_v3)

Quando temos vários objetos com interfaces (métodos) iguais e desconhecemos a implementação, é uma boa prática utilizar **interfaces** Java.

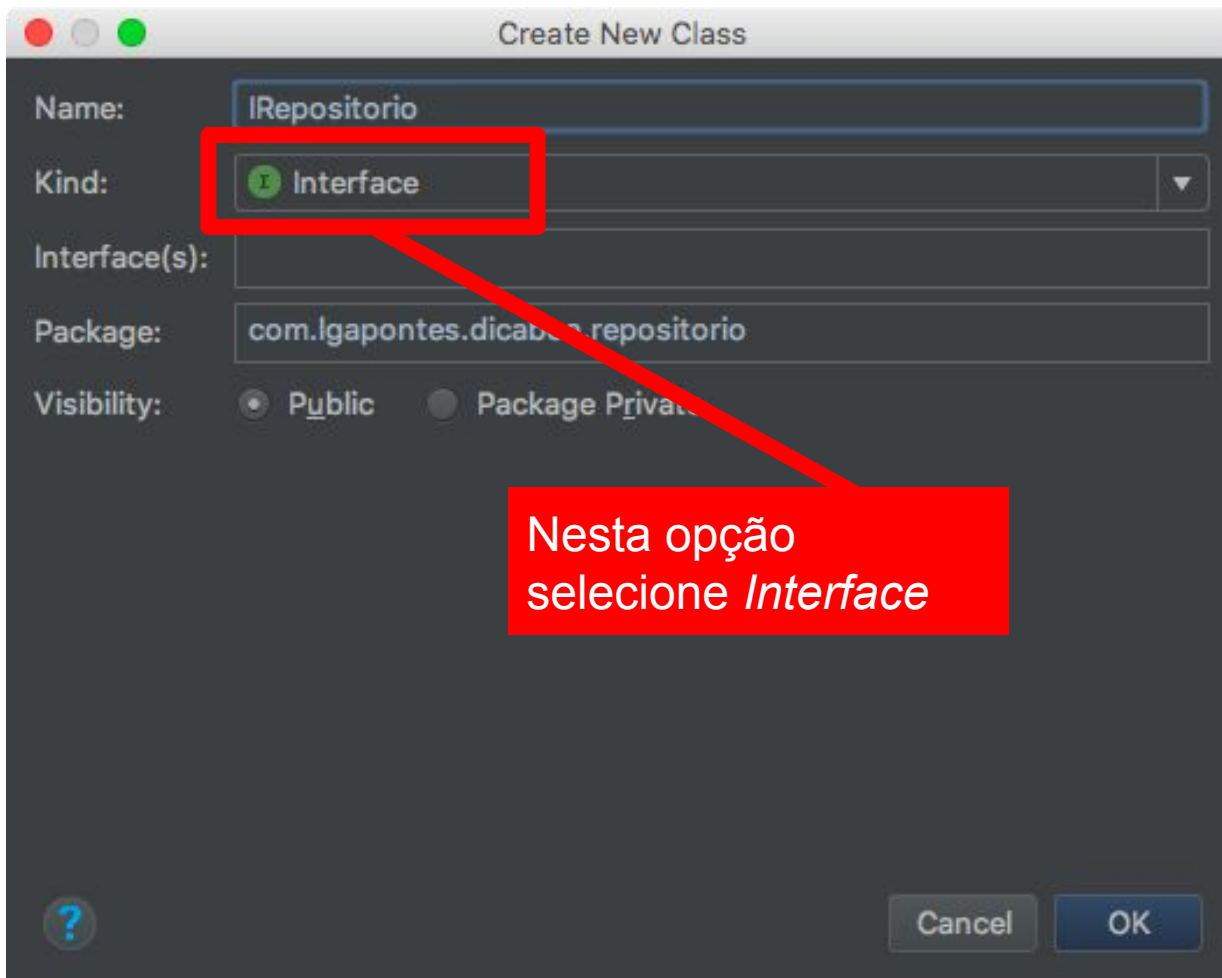


# Utilizando um *Spinner* (DicaBoa\_v3)

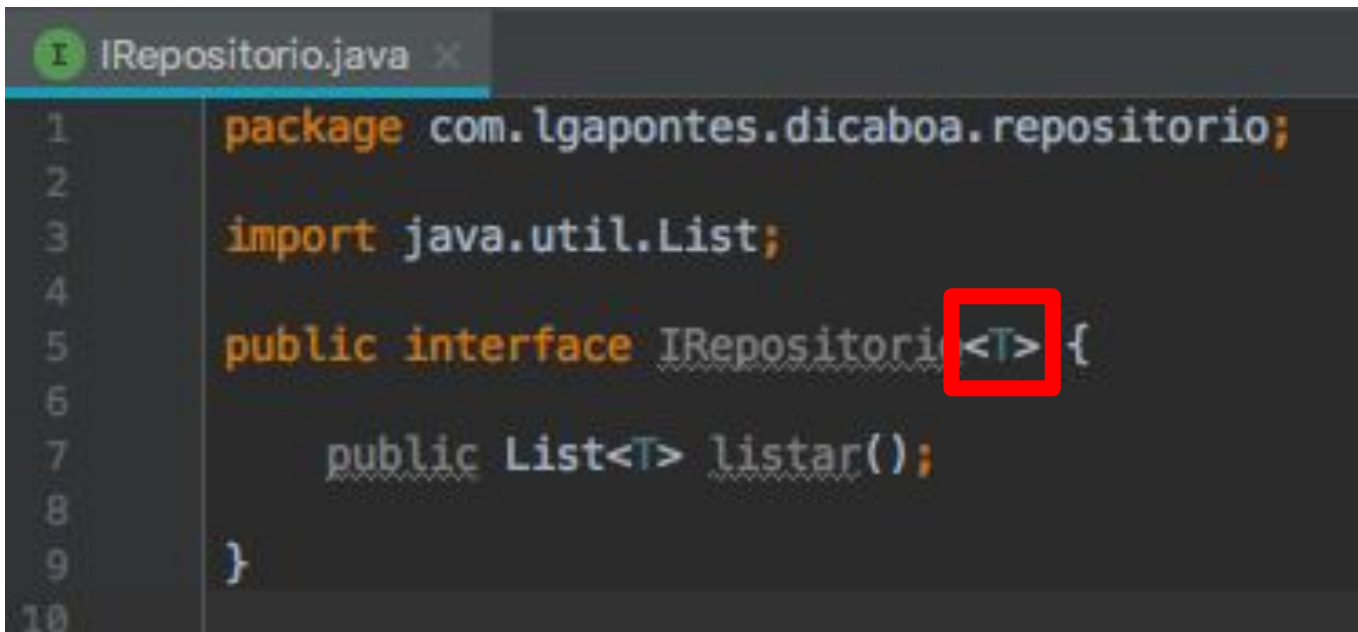
Vamos criar a interface **IRepositorio** dentro do pacote *repositorio*. **Note que o menu é o mesmo para criar classes.**



# Utilizando um *Spinner* (DicaBoa\_v3)



## Utilizando um *Spinner* (DicaBoa\_v3)



```
IRepositorio.java x
1 package com.lgapontes.dicaboa.repositorio;
2
3 import java.util.List;
4
5 public interface IRepositorio<T> {
6
7     public List<T> listar();
8
9 }
10
```

O uso de *generics* na interface não é obrigatório, mas também é uma boa prática. Ele traz problemas que poderiam ocorrer em tempo de execução para tempo de compilação.

## Utilizando um *Spinner* (DicaBoa\_v3)

Agora crie a classe *MarcaRepositorio* implementado a interface *IRepositorio*

```
public class MarcaRepositorio implements IRepositorio<Marca> {  
  
    @Override  
    public List<Marca> listar() {  
        String[] itens = new String[] {  
            "Acer", "Dell", "Lenovo", "Samsung", "HP",  
            "Apple", "LG", "AOC", "Asus", "Avell"  
        };  
  
        List<Marca> marcas = new ArrayList<Marca>();  
        for (int i=0; i<itens.length; i++) {  
            marcas.add(new Marca(    codigo: i+1, itens[i]));  
        }  
  
        return marcas;  
    }  
}
```

## Utilizando um *Spinner* (DicaBoa\_v3)

Na classe *CadastroActivity*, vamos criar um método *popularSpinner()*

```
private void popularSpinner(Spinner spinner, IRepository repositorio) {  
    ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(  
        context: this,  
        android.R.layout.simple_spinner_item,  
        repositorio.listar()  
    );  
    spinnerArrayAdapter.setDropDownViewResource(  
        android.R.layout.simple_spinner_dropdown_item);  
    spinner.setAdapter(spinnerArrayAdapter);  
}
```

## Utilizando um *Spinner* (DicaBoa\_v3)

Na classe *CadastroActivity*, vamos criar um método *popularSpinner()*

```
private void popularSpinner(Spinner spinner, IRepository repositorio) {  
    ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(  
        context: this,  
        android.R.layout.simple_spinner_item,  
        repositorio.listar()  
    );  
    spinnerArrayAdapter.setDropDownViewResource(  
        android.R.layout.simple_spinner_dropdown_item);  
    spinner.setAdapter(spinnerArrayAdapter);  
}
```

Este trecho está criando um adapter. O layout *simple\_spinner\_item* fornecido pelo Android define como o item selecionado será exibido. O método *listar()* retorna a lista com os dados para serem exibidos.



## Utilizando um *Spinner* (DicaBoa\_v3)

Na classe *CadastroActivity*, vamos criar um método *popularSpinner()*

```
private void popularSpinner(Spinner spinner, IRepository repositorio) {  
    ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(  
        context: this,  
        android.R.layout.simple_spinner_item,  
        repositorio.listar()  
    );  
    spinnerArrayAdapter.setDropDownViewResource(  
        android.R.layout.simple_spinner_dropdown_item);  
    spinner.setAdapter(spinnerArrayAdapter);  
}
```

Este trecho define o layout dos itens quando o *Spinner* é aberto (*simple\_spinner\_dropdown\_item*). Posteriormente ele define o adapter do view *Spinner* passado como parâmetro para o método.



```
public class CadastroActivity extends AppCompatActivity {
```

```
    private Spinner spinnerMarcas;
```



```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_cadastro);
```

```
        spinnerMarcas = (Spinner) findViewById(R.id.form_spinner_marcas);
```

```
        popularSpinner(spinnerMarcas, new MarcaRepositorio());
```

```
    }
```

```
    private void popularSpinner(Spinner spinner, IRepositorio repositorio) {
```

```
        ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(
```

```
            context: this,
```

```
            android.R.layout.simple_spinner_item,
```

```
            repositorio.listar()
```

```
        );
```

```
        spinnerArrayAdapter.setDropDownViewResource(
```

```
            android.R.layout.simple_spinner_dropdown_item);
```

```
        spinner.setAdapter(spinnerArrayAdapter);
```

```
    }
```

```
}
```

Por fim, no método *onCreate()*, vamos recuperar o *Spinner* pelo ID e chamar o método *popularSpinner()* passando um novo *MarcaRepositorio()*.



*Mas eu ainda não entendi  
a vantagem de utilizar a  
interface...*



# Vantagens de trabalhar com interfaces

Veja que `popularSpinner()` recebe a interface `IRepositorio` no lugar de `MarcaRepositorio`. Com ajuda do polimorfismo, a invocação do método `popularSpinner()` passando `MarcaRepositorio` como parâmetro fará com que o método carregue os dados do `Spinner` corretamente.



```
private void popularSpinner(Spinner spinner, IRepositorio repositorio) {  
    ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(  
        context: this,  
        android.R.layout.simple_spinner_item,  
        repositorio.listar()  
    );  
    spinnerArrayAdapter.setDropDownViewResource(  
        android.R.layout.simple_spinner_dropdown_item);  
    spinner.setAdapter(spinnerArrayAdapter);  
}
```

# Vantagem de trabalhar com interfaces



Marca  
Repositorio



Sistema  
Repositorio



Processador  
Repositorio



```
private void popularSpinner(Spinner spinner, IRepository repositorio) {  
    ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(  
        context, this,  

```

Isso significa que qualquer objeto que implemente a interface *IRepositorio* poderá ser passado como parâmetro para *popularSpinner()*.

Ou seja, as futuras classes *SistemaRepositorio* e *ProcessadorRepositorio* poderão reutilizar este mesmo método porque elas obedecem o **contrato** estabelecido pela interface *IRepositorio*.

# Exercício em Sala

Crie as classes *SistemaRepositorio* e *ProcessadorRepositorio* com os dados apresentados no próximo slide. Eles devem ser utilizados para popular mais dois *Spinners*, conforme imagem a seguir.



# Exercício em Sala

## Sistema Repositorio:

- Windows 8
- Windows 10
- Ubuntu 18
- Debian
- macOS X

## Processador Repositorio:

- Intel i3 2GHz
- Intel i5 2,5GHz
- Intel i5 3GHz
- Intel i7 3GHz
- Intel i9 3,5GHz



## Resolvendo o Exercício

```
1 package com.lgapontes.dicaboa.repositorio;
2
3 import com.lgapontes.dicaboa.modelo.Sistema;
4
5 import java.util.ArrayList;
6 import java.util.List;
7
8 public class SistemaRepositorio implements IRepositorio<Sistema> {
9
10     @Override
11     public List<Sistema> listar() {
12         String[] itens = new String[] {
13             "Windows 8", "Windows 10",
14             "Ubuntu 18", "Debian",
15             "macOS X"
16         };
17
18         List<Sistema> sistemas = new ArrayList<Sistema>();
19         for (int i=0; i<itens.length; i++) {
20             sistemas.add(new Sistema( codigo: i+1, itens[i]));
21         }
22
23         return sistemas;
24     }
25 }
```



## Resolvendo o Exercício

```
1 package com.lgapontes.dicaboa.repositorio;
2
3 import com.lgapontes.dicaboa.modelo.Processador;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class RepositorioProcessador implements IRepositorio<Processador> {
8
9     @Override
10    public List<Processador> listar() {
11        String[] itens = new String[] {
12            "Intel i3 2GHz",
13            "Intel i5 2,5GHz",
14            "Intel i5 3GHz",
15            "Intel i7 3GHz",
16            "Intel i9 3,5GHz"
17        };
18
19        List<Processador> cpus = new ArrayList<Processador>();
20        for (int i=0; i<itens.length; i++) {
21            cpus.add(new Processador( i+1, itens[i]));
22        }
23
24        return cpus;
25    }
26 }
27
```

# Resolvendo o Exercício

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:text="Sistema"
        android:theme="@style/form_text_size" />
    <Spinner
        android:id="@+id/form_spinner_sistemas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:theme="@style/form_text_size" />
</TableRow>
```

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:text="CPU"
        android:theme="@style/form_text_size" />
    <Spinner
        android:id="@+id/form_spinner_processadores"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:theme="@style/form_text_size" />
</TableRow>
```

Adicione mais duas *TableRow*'s na *TableView*. Uma deve ter um *Spinner* para exibir os sistemas e a outra para os processadores.

# Resolvendo o Exercício

Por fim, altere a *CadastroActivity.java* para obter os *Spinners* do XML e populá-los através do método *popularSpinner()*.

```
private Spinner spinnerMarcas;  
private Spinner spinnerSistemas;  
private Spinner spinnerProcessadores;
```

```
@Override
```

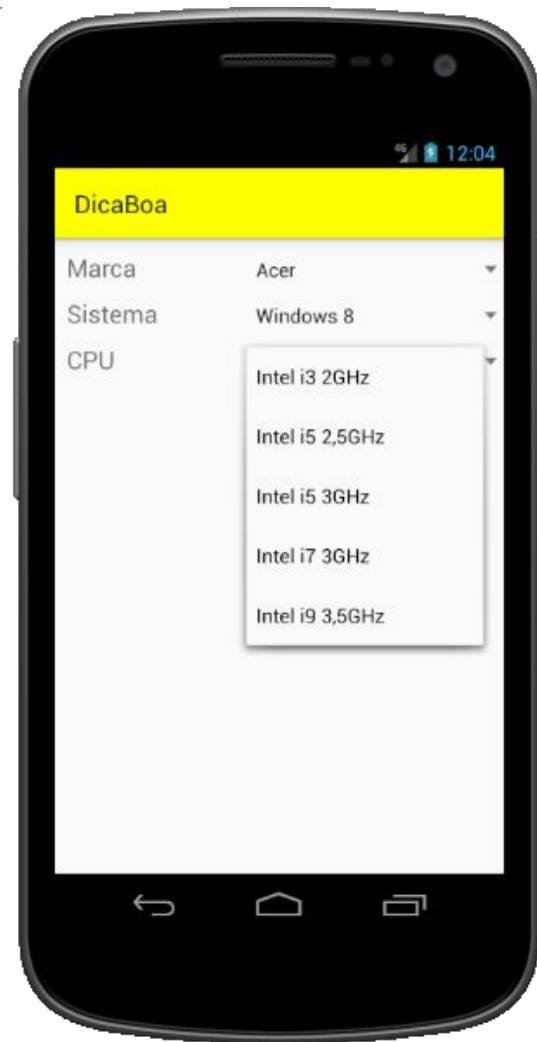
```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_cadastro);
```

```
    spinnerMarcas = (Spinner) findViewById(R.id.form_spinner_marcas);  
    popularSpinner(spinnerMarcas, new MarcaRepositorio());
```

```
    spinnerSistemas = (Spinner) findViewById(R.id.form_spinner_sistemas);  
    popularSpinner(spinnerSistemas, new SistemaRepositorio());
```

```
    spinnerProcessadores = (Spinner) findViewById(R.id.form_spinner_processadores);  
    popularSpinner(spinnerProcessadores, new ProcessadorRepositorio());
```

```
}
```



# Ajuste fino na aparência do *Spinner* (*DicaBoa\_v4*)



Veja que há uma pequena diferença de tamanho de texto entre a label e o conteúdo dos *Spinners*.

Isso acontece porque na criação do *ArrayAdapter* (na classe *CadastroActiviy.java*) nós utilizamos o padrão

Spinner fechado

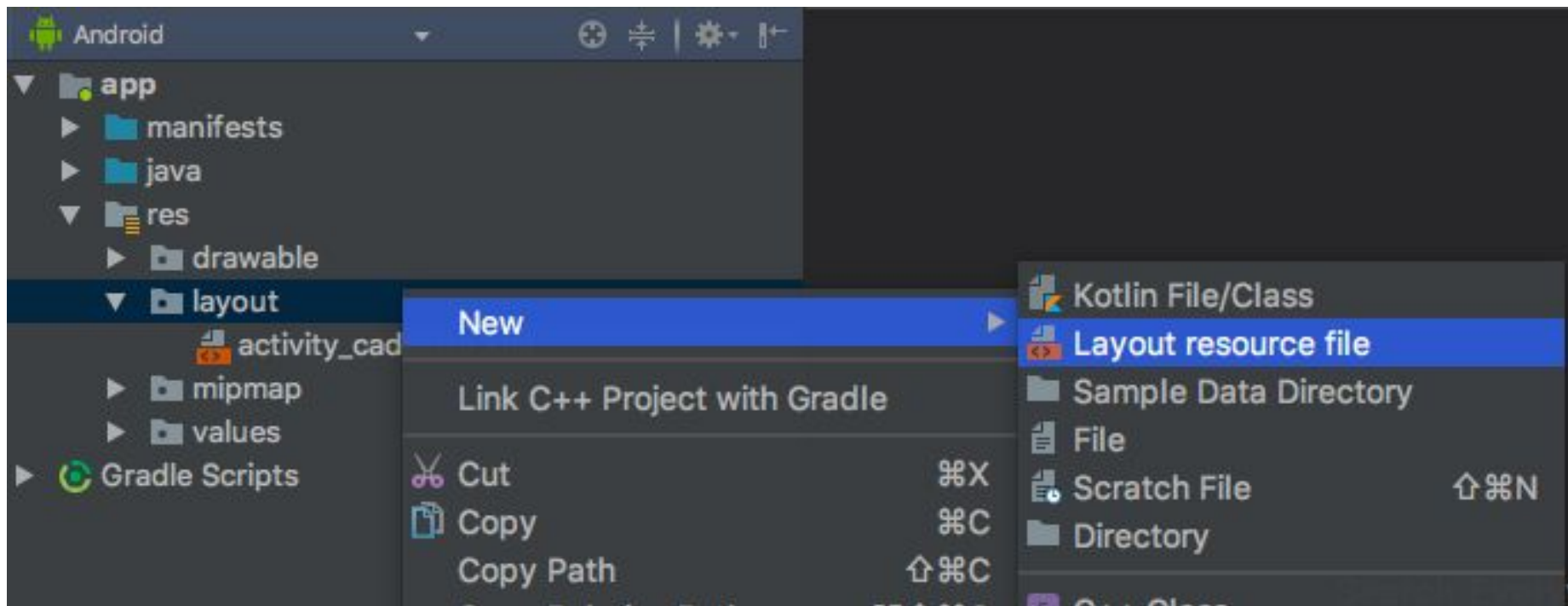
Spinner aberto

```
private void popularSpinner(Spinner spinner, IRepositorio repositorio) {  
    ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(  
        Context.this,  
        android.R.layout.simple_spinner_item,  
        repositorio.getSpinnerItems());  
    spinner.setAdapter(spinnerArrayAdapter);  
}
```

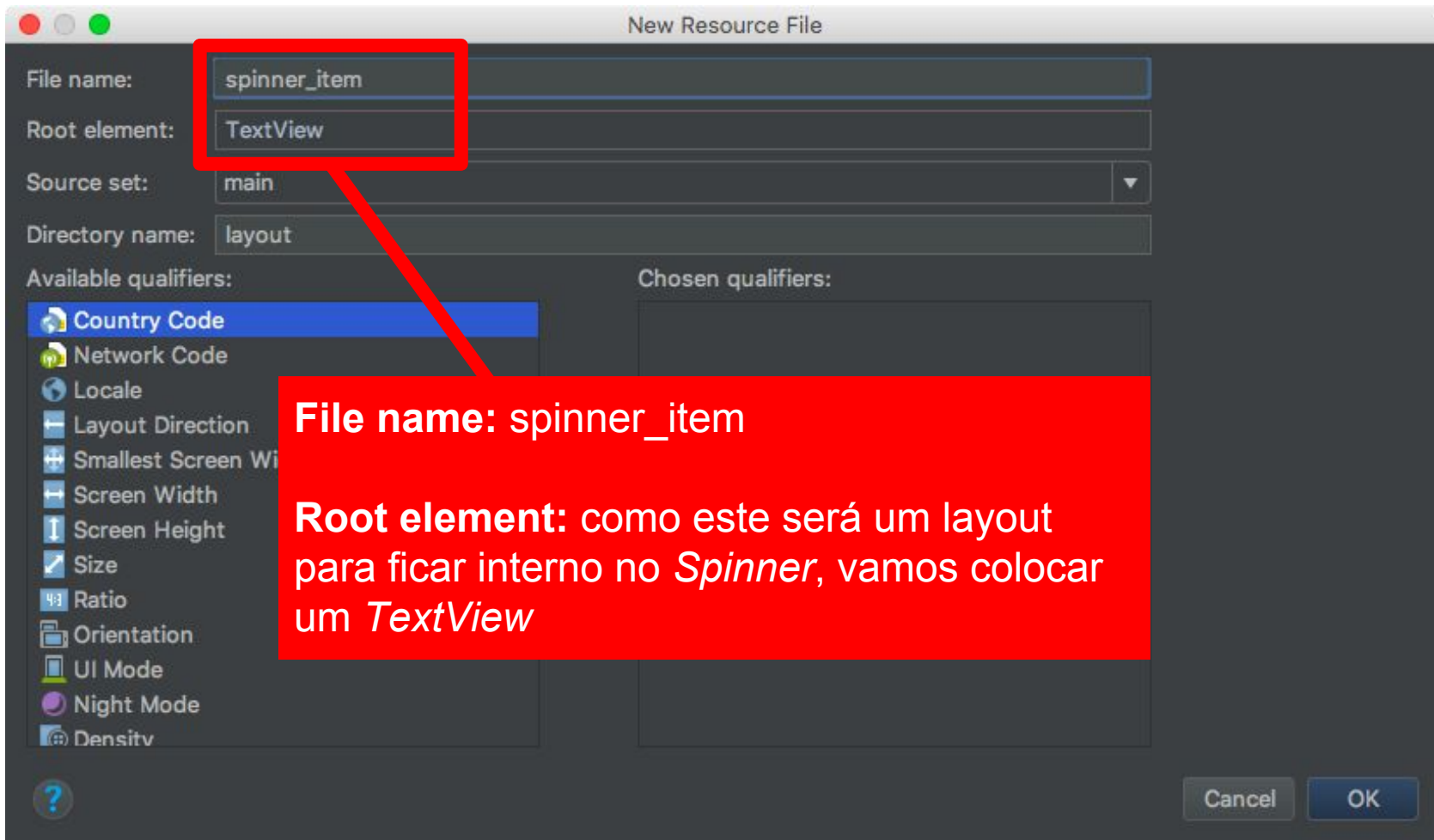


# Ajuste fino na aparência do *Spinner* (DicaBoa\_v4)

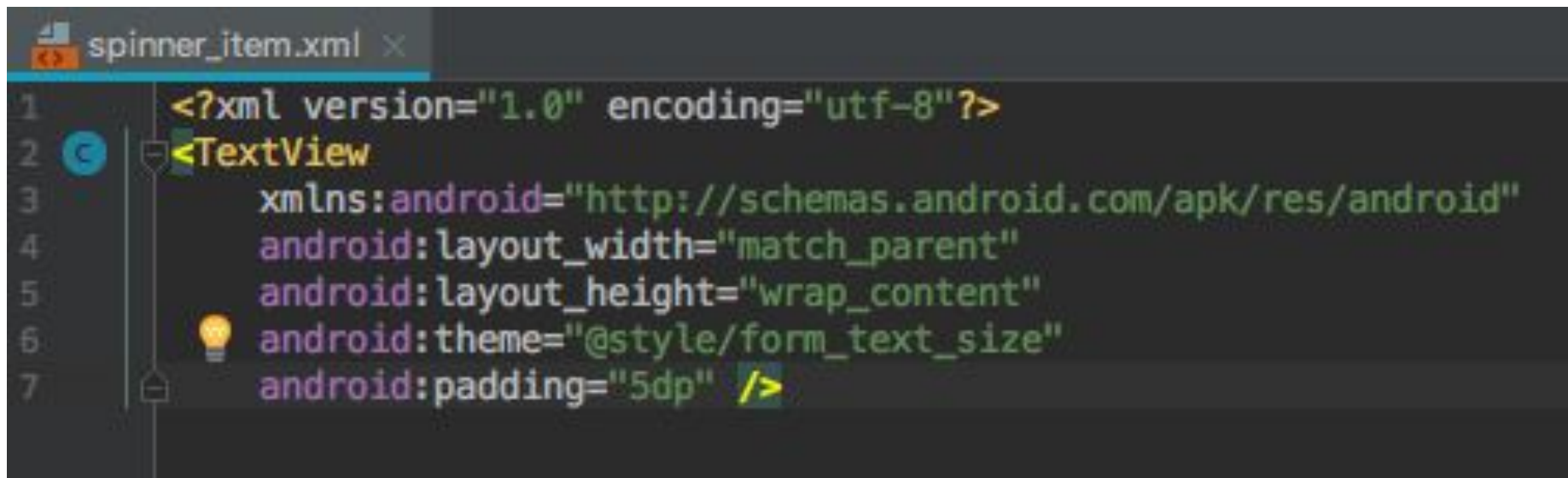
Vamos resolver isso criando nosso próprio Layout de itens para o *Spinner*. Clique com o botão direito sobre a pasta *layout*, menu *New*, submenu *Layout Resource file*



# Ajuste fino na aparência do *Spinner* (DicaBoa\_v4)



## Ajuste fino na aparência do *Spinner* (DicaBoa\_v4)



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TextView
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_width="match_parent"
5      android:layout_height="wrap_content"
6      android:theme="@style/form_text_size"
7      android:padding="5dp" />
```

Este *TextView* pode seguir o mesmo padrão dos demais (utilizando o tema *form\_text\_size*). Acrescente também um *padding* para melhorar a visualização dos itens quando o *Spinner* estiver aberto.



## Ajuste fino na aparência do *Spinner* (DicaBoa\_v4)

Por fim, altere o método *popularSpinner()* para utilizar o novo layout *spinner\_item* nas duas configurações: criação do novo *ArrayAdapter<String>()* e no método *setDropDownViewResource()*

```
private void popularSpinner(Spinner spinner, IRepositorio repositorio) {  
    ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<String>(  
        context: this,  
        R.layout.spinner_item,  
        repositorio.listar()  
    );  
    spinnerArrayAdapter.setDropDownViewResource(R.layout.spinner_item);  
    spinner.setAdapter(spinnerArrayAdapter);  
}
```

# Ajuste fino na aparência do *Spinner* (*DicaBoa\_v4*)



Pronto! Fontes com o mesmo tamanho!

O *Spinner* padrão não oferece opção para desabilitar uma das opções (a primeira opção, por exemplo, poderia ser "Selecione...").

Entretanto, nós podemos customizar um *Adapter* capaz de ter esse comportamento. Para maiores detalhes, vide:

<https://www.android-examples.com/disable-single-item-inside-spinner-in-android/>

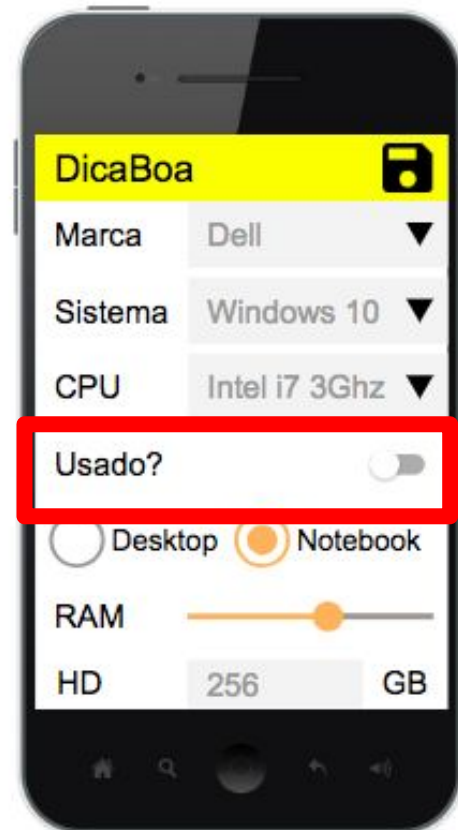
**Faremos *Adapters* próprios em breve!!**

# Utilizando um *Switch* (DicaBoa\_v4)

Continuando, vamos relembrar agora como utilizar um *Switch*

```
strings.xml
1 <resources>
2   <string name="app_name">DicaBoa</string>
3
4   <string name="form_label_marca">Marca</string>
5   <string name="form_label_sistema">Sistema</string>
6   <string name="form_label_processador">CPU</string>
7   <string name="form_label_usado">Usado?</string>
8   <string name="form_label_usado_on">Sim</string>
9   <string name="form_label_usado_off">Não</string>
10
11 </resources>
```

O primeiro passo é definir as labels do texto que fica à esquerda, do status ligado e do status desligado. Vamos fazer isso no arquivo *strings.xml*



<Switch

```
android:id="@+id/form_switch_usado"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:layout_marginLeft="10dp"  
android:layout_marginTop="10dp"  
android:theme="@style/form_text_size"  
android:text="@string/form_label_usado"  
android:textOn="@string/form_label_usado_on"  
android:textOff="@string/form_label_usado_off" />
```

Em seguida vamos acrescentá-lo no Layout XML.

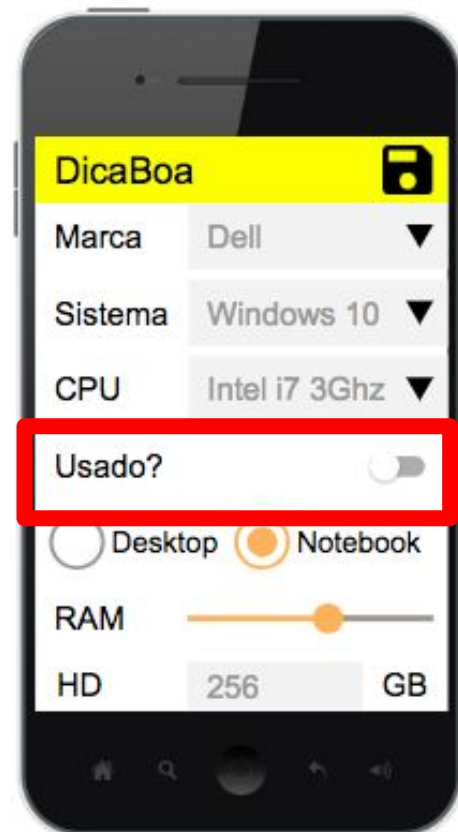
Crie uma propriedade privada *switchUsado*.

```
private Switch switchUsado;
```

E por fim, recupere o *Switch* por ID no método onCreate()

```
switchUsado = (Switch) findViewById(R.id.form_switch_usado);
```

## Utilizando um *Switch*



# Utilizando um *Switch* (*DicaBoa\_v4*)

Switch pronto!



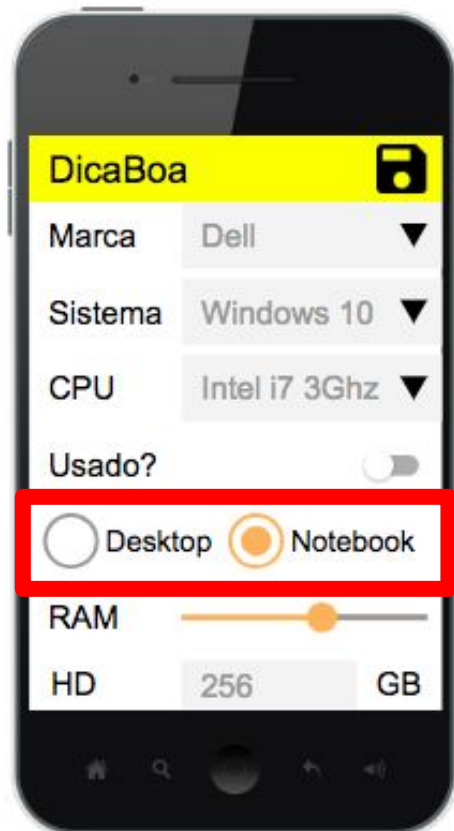
## Utilizando um **RadioButton** (*DicaBoa\_v4*)

Os Radio Buttons devem ser utilizados quando queremos exibir opções lado a lado para o usuário escolher. Ele é aconselhado para poucas opções.

Como os Radio Buttons são mutuamente exclusivos, devemos agrupá-los em um **RadioGroup** para que o Android controle a marcação de apenas um deles.

Vamos começar criando suas labels no arquivo *strings.xml*

```
<string name="form_radio_desktop">Desktop</string>  
<string name="form_radio_notebook">Notebook</string>
```





# Utilizando um *RadioButton*

Agora vamos criar o *RadioGroup* e seus *RadioButton*'s no arquivo *activity\_cadastro.xml*.

```
<RadioGroup
```

```
    android:id="@+id/form_radiogroup_tipo"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="10dp"  
    android:orientation="horizontal"  
    android:gravity="center">
```

```
<RadioButton
```

```
    android:id="@+id/form_radiobutton_desktop"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/form_radio_desktop"  
    android:theme="@style/form_text_size" />
```

```
<RadioButton
```

```
    android:id="@+id/form_radiobutton_notebook"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/form_radio_notebook"  
    android:theme="@style/form_text_size" />
```

```
</RadioGroup>
```



# Utilizando um *RadioButton*

Agora vamos criar o *RadioGroup* e seus *RadioButton*'s no arquivo *activity\_cadastro.xml*.

Defina a orientação do *RadioGroup* como horizontal (o padrão é vertical).

O atributo *gravity* é utilizado para centralizar o conteúdo do *RadioGroup*, o que na prática alinhará os *RadioButton*'s no centro.

```
<RadioGroup
    android:id="@+id/form_radiogroup_tipo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:orientation="horizontal"
    android:gravity="center">

    <RadioButton
        android:id="@+id/form_radiobutton_desktop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/form_radio_desktop"
        android:theme="@style/form_text_size" />

    <RadioButton
        android:id="@+id/form_radiobutton_notebook"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/form_radio_notebook"
        android:theme="@style/form_text_size" />

</RadioGroup>
```

## Utilizando um *RadioButton*

Outro ponto interessante do *RadioGroup* é que através dele podemos obter qual *RadioButton* foi selecionado.

**Faremos isso em breve!**

```
<RadioGroup
```

```
    android:id="@+id/form_radiogroup_tipo"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_marginTop="10dp"
```

```
    android:orientation="horizontal"
```

```
    android:gravity="center">
```

```
<RadioButton
```

```
    android:id="@+id/form_radiobutton_desktop"
```

```
    android:layout_width="wrap_content"
```

Por enquanto, crie um atributo privado para o *RadioGroup*

```
private RadioGroup radioGroupTipo;
```

E obtenha-o através do ID dentro do método *onCreate()*

```
radioGroupTipo = (RadioGroup) findViewById(R.id.form_radiogroup_tipo);
```

# Utilizando um *RadioButton*

*RadioGroup* e *RadioButton*'s prontos!

## Dica Extra

Apesar de não ser tão comum, também é possível definir os valores dos *RadioButton*'s programaticamente.



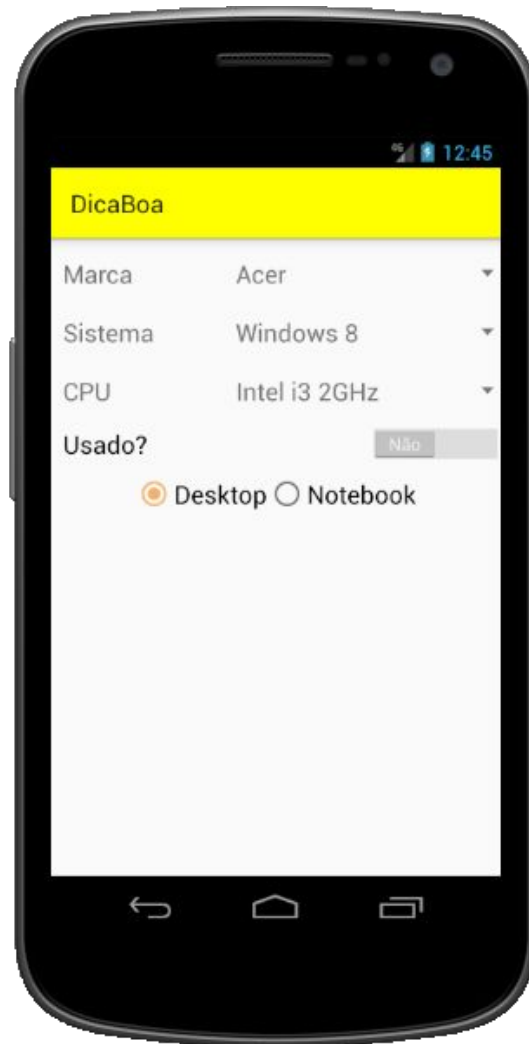
Código Java



- ☐ RadioButton 1
- ☒ RadioButton 2
- ☐ RadioButton 3

Se isso for necessário, veja:

<https://tutorialwing.com/create-an-android-radio-button-programmatically-in-android/>



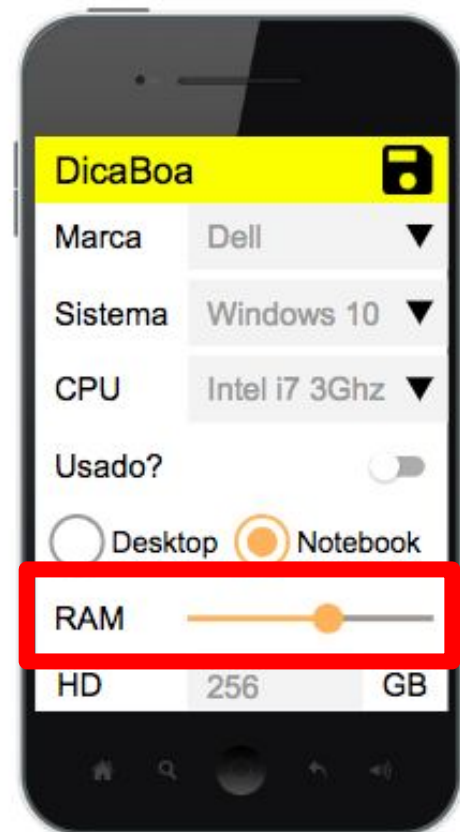
# Utilizando um **SeekBar** (*DicaBoa\_v5*)

A primeira observação a respeito do **SeekBar** é que no Material Design ele é chamado de **Slider**

<https://material.io/design/components/sliders.html#usage>



No Android SDK, todavia, ele é conhecido como **SeekBar**. Como no geral ele é utilizado para definir um valor percentual (volume, luminosidade, etc), não é necessário exibir um valor ao usuário. Para o nosso exemplo vamos criar um *TextView* auxiliar para melhorar a usabilidade.



strings.xml

```
1 <resources>
2   <string name="app_name">DicaBoa</string>
3
4   <string name="form_label_marca">Marca</string>
5   <string name="form_label_sistema">Sistema</string>
6   <string name="form_label_processador">CPU</string>
7   <string name="form_label_usado">Usado?</string>
8   <string name="form_label_usado_on">Sim</string>
9   <string name="form_label_usado_off">Não</string>
10  <string name="form_radio_desktop">Desktop</string>
11  <string name="form_radio_notebook">Notebook</string>
12  <string name="form_seekbar_memoria">Memória RAM</string>
13 </resources>
```

## Utilizando um *SeekBar*

Vamos começar definindo o texto "Memória RAM" para servir de label e a cor #dddddd para servir de fundo para a exibição do valor.



colors.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <color name="colorPrimary">#ffff00</color>
4   <color name="colorPrimaryDark">#7f6000</color>
5   <color name="colorAccent">#f0b200</color>
6   <color name="cinza">#dddddd</color>
7 </resources>
```



### <TableRow

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="10dp">
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:theme="@style/form_text_size"
    android:text="@string/form_seekbar_memoria" />
```

### <TextView

```
android:id="@+id/form_textview_memoria"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:theme="@style/form_text_size"
android:background="@color/cinza"
android:text="1 GB" />
```

### </TableRow>

```
<!-- android:min a partir da API Level 26 -->
```

### <SeekBar

```
android:id="@+id/form_seekbar_memoria"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:max="32"
android:progress="1" />
```

## Utilizando um *SeekBar*

Vamos agora alterar o layout XML para acrescentar os itens:

**TableRow:** para agrupar os dois *TextView's* dispostos sobre o *SeekBar*.

**SeekBar:** o componente *SeekBar*.



```
<TableRow
```

```
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="10dp">
```

```
    <TextView
```

```
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_marginLeft="10dp"  
        android:theme="@style/form_text_size"
```

```
        android:text="@string/form_textview_memoria" />
```

```
    <TextView
```

```
        android:id="@+id/form_textview_memoria"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:gravity="center"  
        android:theme="@style/form_text_size"  
        android:background="@color/cinza"  
        android:text="1 GB" />
```

```
</TableRow>
```

```
<!-- android:min a partir da API Level 26 -->
```

```
<SeekBar
```

```
    android:id="@+id/form_seekbar_memoria"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:max="32"  
    android:progress="1" />
```

## Utilizando um *SeekBar*

Vejamos alguns detalhes do *TextView* que vai exibir o valor selecionado.

**ID:** ele deve ter um ID para que possamos obtê-lo no código Java.

**Alinhar conteúdo:** ele deve centralizar o conteúdo no centro através da propriedade *gravity*.

**Background:** utiliza a cor cinza criada.

**Valor default:** através da propriedade *text*, defina um valor default de "1 GB".



```
<TableRow
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:theme="@style/form_text_size"
        android:text="@string/form_seekbar_memoria" />
    <TextView
        android:id="@+id/form_textview_memoria"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:theme="@style/form_text_size"
        android:background="@color/cinza"
        android:text="1 GB" />
</TableRow>
```

```
<!-- android:min a partir da API Level 26 -->
```

```
<SeekBar
```

```
    android:id="@+id/form_seekbar_memoria"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="32"
    android:progress="1" />
```

## Utilizando um *SeekBar*

Agora os detalhes do *SeekBar*:

**ID:** através do seu ID que vamos obter o valor selecionado no código Java.

**Valor máximo:** podemos definir o valor máximo através da propriedade *max*.

**Valor default:** podemos definir o valor inicial através da propriedade *progress*.

**Valor mínimo:** existe a propriedade *min*, porém ela só está disponível a partir da API Level 26. No nosso caso, quando o valor for 0 (zero), vamos exibir um texto "Sem memória RAM"

## Utilizando um **SeekBar** (*DicaBoa\_v5*)

No código *CadastroActivity.java*, vamos definir um método chamado *calcularValorSeekBar()* conforme trecho abaixo.

```
private String calcularValorSeekBar(int valor) {  
    if (valor == 0) {  
        return "Sem memória RAM";  
    } else {  
        return Integer.toString(valor) + " GB";  
    }  
}
```

Em seguida, vamos definir dois atributos: um para o *TextView* que vai exibir o valor selecionado e outro para o próprio *SeekBar*.

```
private TextView textViewMemoria;  
private SeekBar seekBarMemoria;
```

## Utilizando um **SeekBar** (*DicaBoa\_v5*)

Por fim, no método `onCreate()`, vamos obter o `TextView` e o `SeekBar` e guardar nos atributos. Para controlarmos a mudança de valor do `SeekBar`, devemos definir um listener através do método `setOnSeekBarChangeListener()`.

```
textViewMemoria = (TextView) findViewById(R.id.form_textview_memoria);
seekBarMemoria = (SeekBar) findViewById(R.id.form_seekbar_memoria);
seekBarMemoria.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int value, boolean fromUser) {
        textViewMemoria.setText(calcularValorSeekBar(value));
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {}

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {}
});
```

## Utilizando um **SeekBar** (DicaBoa\_v5)

Por fim, no método `onCreate()`, vamos obter o `TextView` e o `SeekBar` e guardar nos atributos. Para controlarmos a mudança de valor do `SeekBar`, devemos definir um listener através do método `setOnSeekBarChangeListener()`.

```
textViewMemoria = (TextView) findViewById(R.id.form_textview_memoria);
seekBarMemoria = (SeekBar) findViewById(R.id.form_seekbar_memoria);
seekBarMemoria.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SearchBar seekBar, int value, boolean fromUser) {
        textViewMemoria.setText(calcularValorSeekBar(value));
    }
});
```

O listener `OnSeekBarChangeListener` deve implementar 3 métodos. O método **`onProgressChanged()`** que gerencia a mudança do valor. Ele recebe três parâmetros: **`seekBar`**: uma referência ao próprio `SeekBar` alterado.

**`value`**: valor alterado. Neste caso, simplesmente vamos definir o texto do `TextView` com o retorno da função `calcularValorSeekBar()`

**`fromUser`**: `true`, se tiver sido alterado pelo usuário. `false`, se foi programaticamente.

# Utilizando um *SeekBar*

SeekBar  
pronto!





# Utilizando *EditText* (*DicaBoa\_v5*)

Vamos relembrar o uso do *EditText*, porém, desta vez, com uma definição do tipo de entrada para *number*.

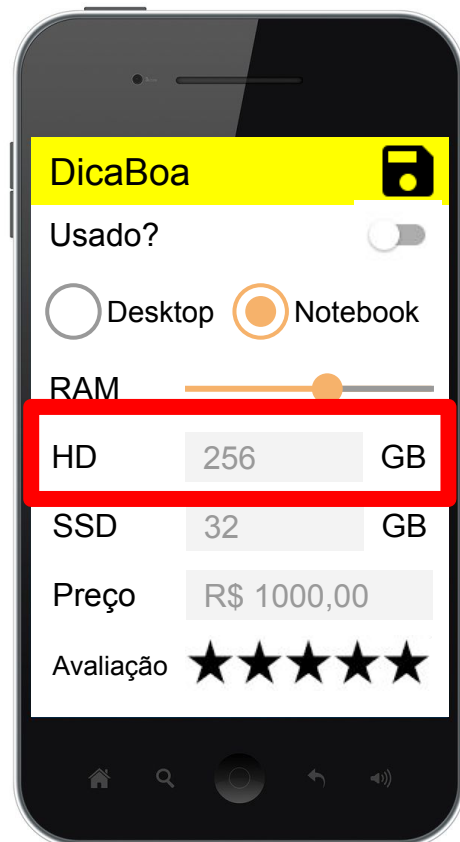
```
<EditText
    android:id="@+id/form_edittext_hd"
    android:inputType="number"
    android:gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/form_text_size"
    android:drawableRight="@drawable/ic_gigabyte"
    android:maxLength="4" />
```

Outro detalhe interessante é que através das propriedades *drawable* disponíveis no *EditText*, nós podemos posicionar imagens.

HD

512

GB



# Exercício em Sala

Altere a App *DicaBoa* com os seguintes detalhes:

- Como no código do último slide, crie dois *EditText* conforme o exemplo ao lado.
- A imagem `ic_gigabyte.png` está disponível na pasta imagens da Aula 10 do EAD
- Ambos *EditText*'s devem ter limite de 4 números (propriedade *maxLength*)
- Crie atributos e recupere-os pelo ID na classe *CadastroActivity.java*



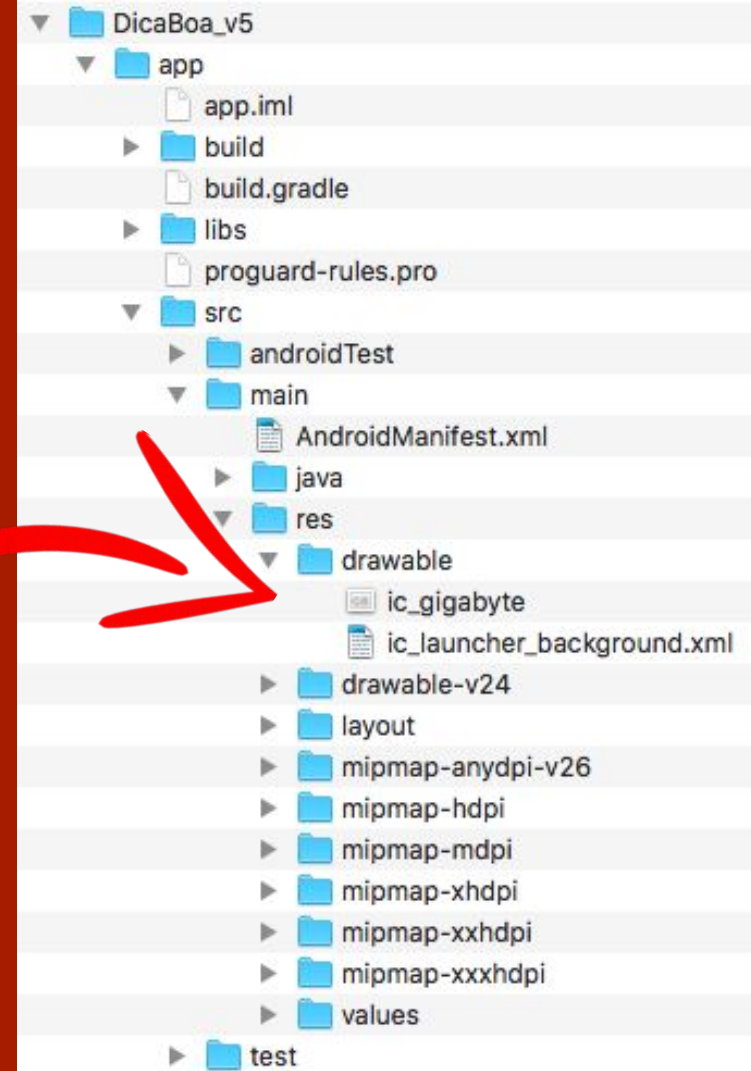
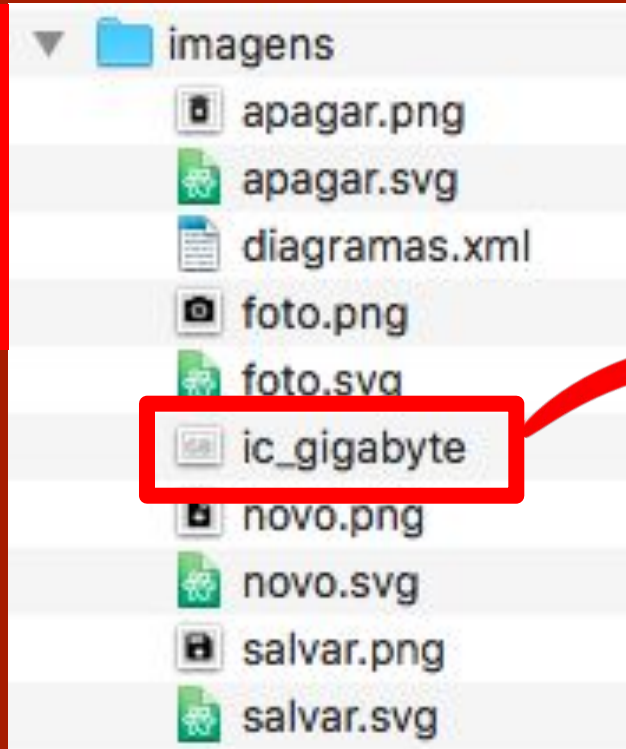


# Resolvendo o Exercício

```
strings.xml x
1 <resources>
2     <string name="app_name">DicaBoa</string>
3
4     <string name="form_label_marca">Marca</string>
5     <string name="form_label_sistema">Sistema</string>
6     <string name="form_label_processador">CPU</string>
7     <string name="form_label_usado">Usado?</string>
8     <string name="form_label_usado_on">Sim</string>
9     <string name="form_label_usado_off">Não</string>
10    <string name="form_radio_desktop">Desktop</string>
11    <string name="form_radio_notebook">Notebook</string>
12    <string name="form_seekbar_memoria">Memória RAM</string>
13    <string name="form_label_hd">HD</string>
14    <string name="form_label_ssd">SSD</string>
15 </resources>
```

# Resolvendo o Exercício

Copie o arquivo *ic\_gigabyte.png* disponível no EAD para a pasta *drawable* do projeto.



# Resolvendo o Exercício

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:theme="@style/form_text_size"
        android:text="HD" />
    <EditText
        android:id="@+id/form_edittext_hd"
        android:inputType="number"
        android:gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/form_text_size"
        android:drawableRight="@drawable/ic_gigabyte"
        android:maxLength="4" />
</TableRow>
```

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:theme="@style/form_text_size"
        android:text="@string/form_label_ssd" />
    <EditText
        android:id="@+id/form_edittext_ssd"
        android:inputType="number"
        android:gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/form_text_size"
        android:drawableRight="@drawable/ic_gigabyte"
        android:maxLength="4" />
</TableRow>
```

Acrescente os dois trechos do *TableRow* no *activity\_cadastro.xml*.

# Resolvendo o Exercício

Por fim, crie as propriedades no *CadastroActivity.java*

```
private EditText editTextHD;  
private EditText editTextSSD;
```

E depois obtenha os *EditText*'s através do ID.

```
editTextHD = (EditText) findViewById(R.id.form_edittext_hd);  
editTextSSD = (EditText) findViewById(R.id.form_edittext_ssd);
```

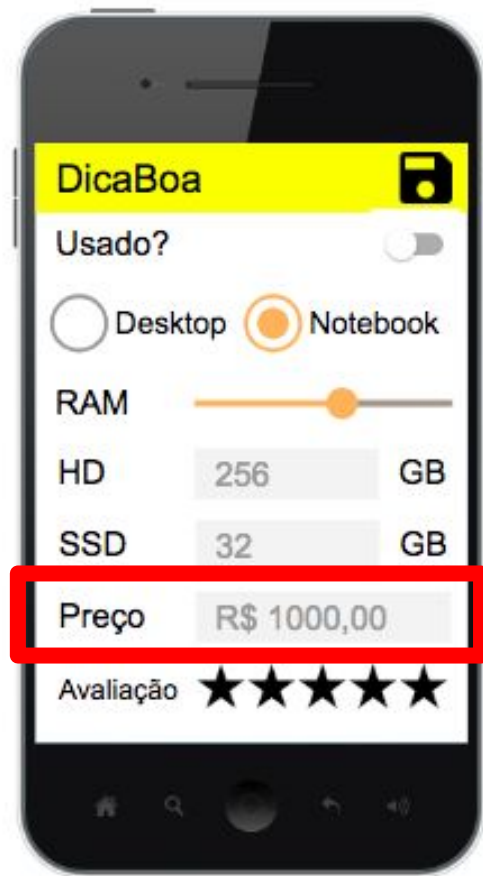


# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

A ideia por trás da formatação do texto de um *EditText* é relativamente simples. Veja os passos a seguir:

- Adicionar um *listener* para escutar todas as alterações do texto do *EditText*
- No *listener*, obtenha o valor digitado e aplique o formato desejado.
- Defina um novo valor para o *EditText* com base no formato aplicado.

**Atenção:** como vamos alterar o texto do *EditText* neste ponto, devemos evitar o *looping* infinito removendo temporariamente o *listener*.





## Utilizando *EditText* com formatação (*DicaBoa\_v6*)

Supondo que o *EditText* tenha o ID *form\_edittext\_preco*, podemos adicionar um *listener* do tipo *TextWatcher()* conforme o código a seguir.

```
editTextPreco = (EditText) findViewById(R.id.form_edittext_preco);
editTextPreco.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence charSequence, int start, int count, int after) {}

    @Override
    public void onTextChanged(CharSequence charSequence, int start, int before, int count) {}

    @Override
    public void afterTextChanged(Editable editable) {}
});
```

Os passos apresentados no slide anterior devem ser aplicados na implementação do método *onTextChanged()*. É nele que o listener captura o valor depois que o usuário terminar de digitar.



# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

```
@Override
public void onTextChanged(CharSequence valor, int start, int before, int count) {

    /* Evita invocação circular */
    editText.removeTextChangedListener( watcher: this);

    /* Remove caracteres não numéricos para tratamento */
    String cleanString = valor.toString().replaceAll( regex: "[R$,.] ", replacement: "").trim();
    double parsed = Double.parseDouble(cleanString);

    /* Converte para moeda real */
    Locale ptBr = new Locale( language: "pt", country: "BR");
    String formatted = NumberFormat.getCurrencyInstance(ptBr).format( number: parsed/100);

    /* Define o texto e posicionamento do cursor */
    editText.setText(formatted);
    editText.setSelection(formatted.length());

    /* Configura novamente este listener */
    editText.addTextChangedListener( watcher: this);

}
```

# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

```
@Override  
public void onTextChanged(CharSequence valor, int start, int before, int count) {
```

```
    /* Evita invocação circular */  
    editText.removeTextChangedListener( watcher: this);
```

Nesse trecho nós vamos remover o *listener* para evitar looping infinito...

```
    /* Converte para moeda real */  
    Locale ptBr = new Locale( language: "pt", country: "BR");  
    String formatted = NumberFormat.getCurrencyInstance(ptBr).format( number: parsed/100);  
  
    /* Define o texto e posicionamento do cursor */  
    editText.setText(formatted);  
    editText.setSelection(formatted.length());  
  
    /* Configura novamente este listener */  
    editText.addTextChangedListener( watcher: this);
```

```
}
```

# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

```
@Override
public void onTextChanged(CharSequence valor, int start, int before, int count) {

    /* Evita invocação circular */
    editText.removeTextChangedListener( watcher: this);

    /* Remove caracteres não numéricos para tratamento */
    String cleanString = valor.toString().replaceAll( regex: "[R$,.] ", replacement: "").trim();
    double parsed = Double.parseDouble(cleanString);
```

Aqui vamos remover todos os caracteres que não são numéricos (no caso, como o *EditText* terá um *inputType* igual a **number**, os únicos caracteres diferentes serão aqueles que nós vamos acrescentar neste formatador). Depois convertemos o valor para *double*.

```
    editText.setSelection(formatted.length());

    /* Configura novamente este listener */
    editText.addTextChangedListener( watcher: this);
```

```
}
```

# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

```
@Override
public void onTextChanged(CharSequence valor, int start, int before, int count) {

    /* Evita invocação circular */
    editText.removeTextChangedListener( watcher: this);
```

Em seguida, vamos utilizar a classe *NumberFormat* do Java para transformar o valor *double* em texto. Estamos dividindo o valor por 100 para incluir os decimais automaticamente na medida em que o usuário digita.

```
/* Converte para moeda real */
Locale ptBr = new Locale( language: "pt", country: "BR");
String formatted = NumberFormat.getCurrencyInstance(ptBr).format( number: parsed/100);

/* Define o texto e posicionamento do cursor */
editText.setText(formatted);
editText.setSelection(formatted.length());

/* Configura novamente este listener */
editText.addTextChangedListener( watcher: this);
}
```



# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

```
@Override
public void onTextChanged(CharSequence valor, int start, int before, int count) {

    /* Evita invocação circular */
    editText.removeTextChangedListener( watcher: this);

    /* Remove caracteres não numéricos para tratamento */
    String cleanString = valor.toString().replaceAll( regex: "[R$,.] ", replacement: "").trim();
```

Por fim, definimos o texto formatado e o posicionamento do cursor e configuramos novamente este listener no *EditText*.

```
R");
e(ptBr).format( number: parsed/100);
```

```
/* Define o texto e posicionamento do cursor */
editText.setText(formatted);
editText.setSelection(formatted.length());

/* Configura novamente este listener */
editText.addTextChangedListener( watcher: this);
```

```
}
```

# Utilizando *EditText* com formatação (*DicaBoa\_v6*)



Este código está disponível na classe *FormatadorMoeda.java* no link: <https://github.com/lgapontes/aulas/blob/master/desenvolvimento-mobile/apoio/FormatadorMoeda.java>

Branch: master ▾

[aulas](#) / [desenvolvimento-mobile](#) / [apoio](#) / **FormatadorMoeda.java**

Find file

Copy path

 **lgapontes** Formatador de moedal real para EditText

09e759e 27 minutes ago

1 contributor

47 lines (33 sloc) | 1.54 KB

Raw

Blame

History

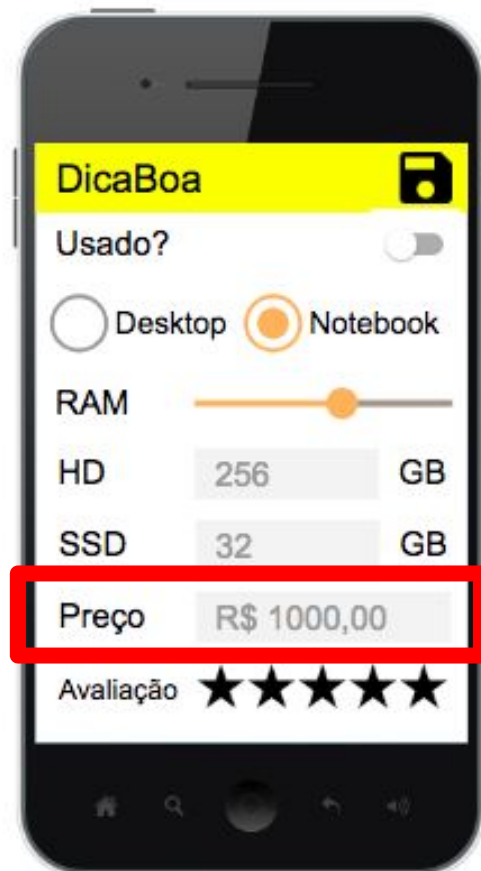
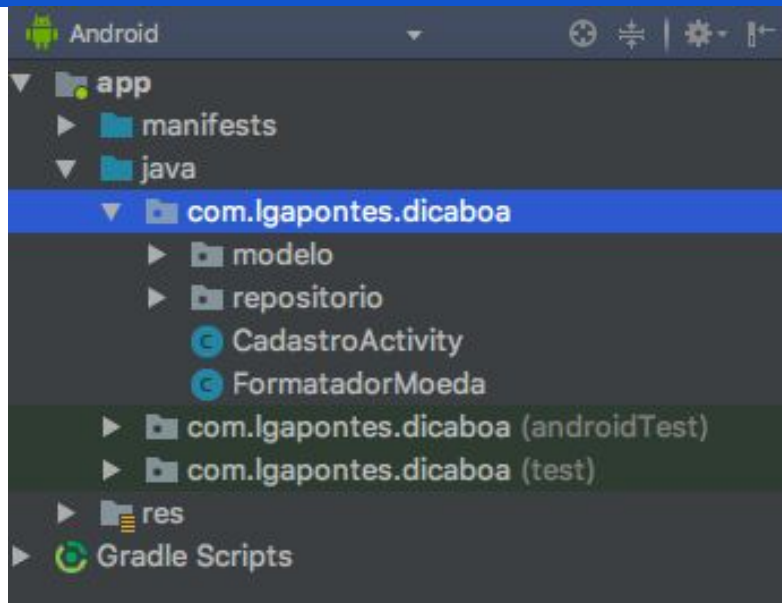


```
1  package com.lgapontes.dicaboa;
2
3  import android.text.Editable;
4  import android.text.TextWatcher;
5  import android.widget.EditText;
6
7  import java.text.NumberFormat;
```

# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

## PASSO 1

Baixe o arquivo do *FormatadorMoeda.java* e salve-o no pacote principal da aplicação: <https://bit.ly/2OOyK5p>



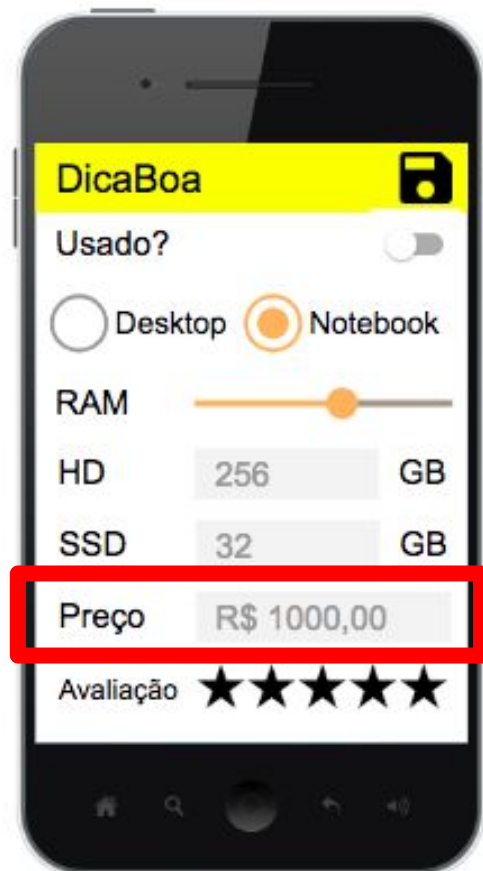


# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

## PASSO 2

Crie uma string com o valor "Preço"

```
strings.xml x
Edit translations for all locales in the translations editor.
1  <resources>
2    <string name="app_name">DicaBoa</string>
3
4    <string name="form_label_marca">Marca</string>
5    <string name="form_label_sistema">Sistema</string>
6    <string name="form_label_processador">CPU</string>
7    <string name="form_label_usado">Usado?</string>
8    <string name="form_label_usado_on">Sim</string>
9    <string name="form_label_usado_off">Não</string>
10   <string name="form_radio_desktop">Desktop</string>
11   <string name="form_radio_notebook">Notebook</string>
12   <string name="form_seekbar_memoria">Memória RAM</string>
13   <string name="form_label_hd">HD</string>
14   <string name="form_label_ssd">SSD</string>
15   <string name="form_label_preco">Preço</string>
16 </resources>
17
```



# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:theme="@style/form_text_size"
        android:text="Preço" />
    <EditText
        android:id="@+id/form_edittext_preco"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/form_text_size"
        android:inputType="number"
        android:maxLength="11"
        android:hint="R$ 0,00" />
</TableRow>
```

## PASSO 4

Adicione uma nova *TableRow* no Layout XML. Vejamos algumas particularidades do *EditText*:

**inputType:** vamos utilizar *number* para habilitar o teclado correto no celular.

**maxLength:** defina 11 para evitar valores muito grandes.

**hint:** melhora a usabilidade mostrando ao usuário como o valor será exibido.

# Utilizando *EditText* com formatação (*DicaBoa\_v6*)

## PASSO 5

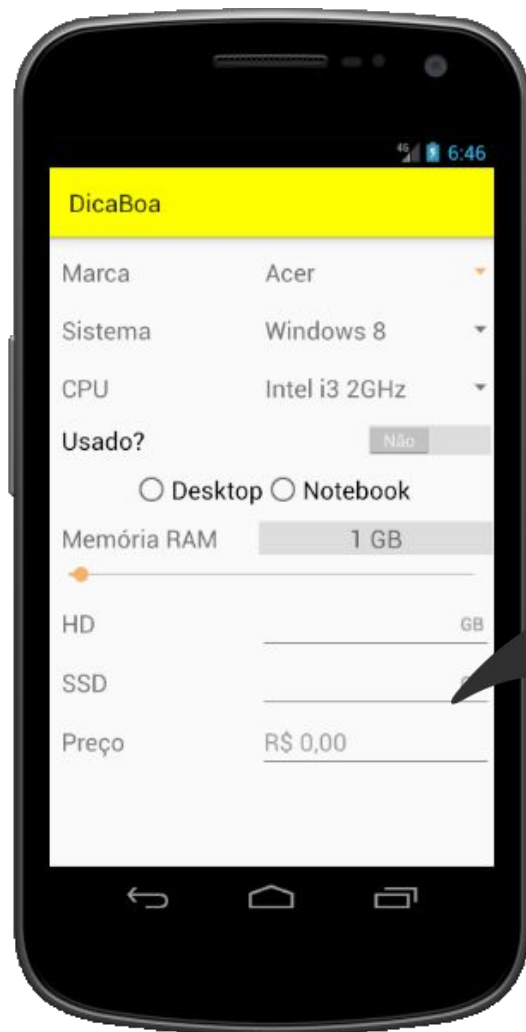
Crie a propriedade no *CadastroActivity.java*

```
private EditText editTextPreco;
```

## PASSO 6

Obtenha o *EditText* por ID e repasse-o para o método *configurar()* da classe *FormatadorMoeda.java*

```
editTextPreco = (EditText) findViewById(R.id.form_edittext_preco);  
FormatadorMoeda.configurar(editTextPreco);
```



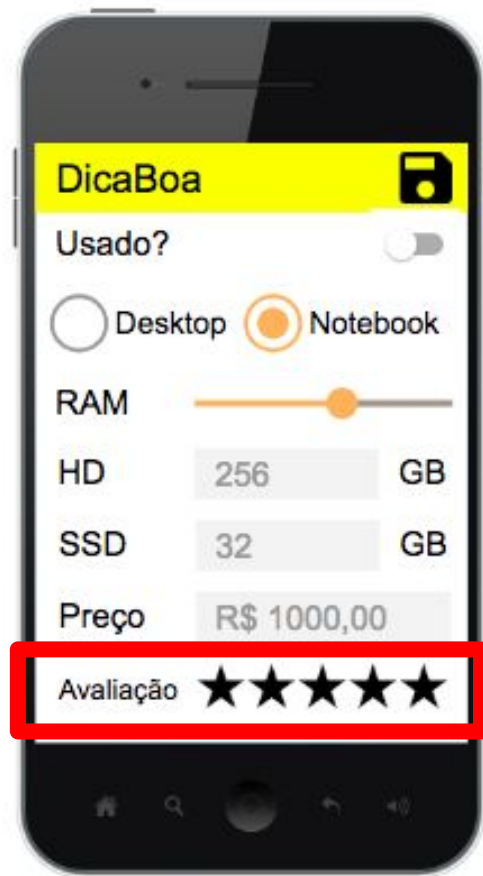
# Utilizando *RatingBar* (*DicaBoa\_v6*)

O view *RatingBar* é um componente que permite o usuário selecionar um valor clicando em 5 estrelas. Normalmente este elemento aceita valores com decimais, mas neste nosso exemplo vamos configurá-lo para obter apenas números inteiros.

## PASSO 1

Vamos começar definindo uma label no *strings.xml*

```
<string name="form_label_nota">Avaliação</string>
```





```
<TextView
```

```
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginLeft="10dp"  
    android:layout_marginTop="10dp"  
    android:theme="@style/form_text_size"  
    android:text="@string/form_label_nota" />
```

```
<LinearLayout
```

```
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal"  
    android:gravity="center">
```

```
    <RatingBar
```

```
        android:id="@+id/form_ratingbar_nota"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:numStars="5"  
        android:stepSize="1.0" />
```

```
</LinearLayout>
```

## PASSO 2

Vamos criar a *RatingBar* no LayoutXML. Às vezes a disposição do *RatingBar* em telas com baixa ou média densidade pode fazer com que ela saia da área útil horizontalmente. Para evitar isso, aos invés de colocar na mesma *TableRow*, vamos colocá-la abaixo da label.

**Atenção:** a *RatingBar* utilizada com *TableLayout* provocará problemas na exibição das estrelas. Para evitar isso, coloque-a dentro de um *LinearLayout*.

```
<TextView
```

```
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginLeft="10dp"  
    android:layout_marginTop="10dp"  
    android:theme="@style/form_text_size"  
    android:id="@+id/et_nota" />
```

O atributo *numStars* indica quantas estrelas vão aparecer.

O atributo *stepSize* indica o menor valor de variação entre as estrelas. Na prática isso fará com que o usuário selecione uma estrela por vez.

```
    android:numStars="5"  
    android:stepSize="1.0" />
```

```
</LinearLayout>
```

## PASSO 2

Vamos criar a *RatingBar* no LayoutXML. Às vezes a disposição do *RatingBar* em telas com baixa ou média densidade pode fazer com que ela saia da área útil horizontalmente. Para evitar isso, aos invés de colocar na mesma *TableRow*, vamos colocá-la abaixo da label.

**Atenção:** a *RatingBar* utilizada com *TableLayout* provocará problemas na exibição das estrelas. Para evitar isso, coloque-a dentro de um *LinearLayout*.



# Utilizando *RatingBar* (*DicaBoa\_v6*)

## PASSO 3

Crie a propriedade no *CadastroActivity.java*

```
private RatingBar ratingBarNota;
```

## PASSO 4

Obtenha o *RatingBar* pelo ID

```
ratingBarNota = (RatingBar) findViewById(R.id.form_ratingbar_nota);
```





*Ok, está funcionando,  
mas está cortado e não  
tem scroll...*

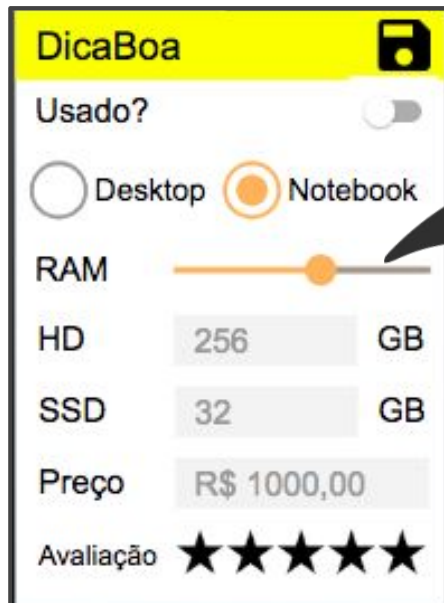




*Em aparelhos com  
baixa densidade cortou  
mais ainda...*



# Utilizando *ScrollView* (*DicaBoa\_v6*)



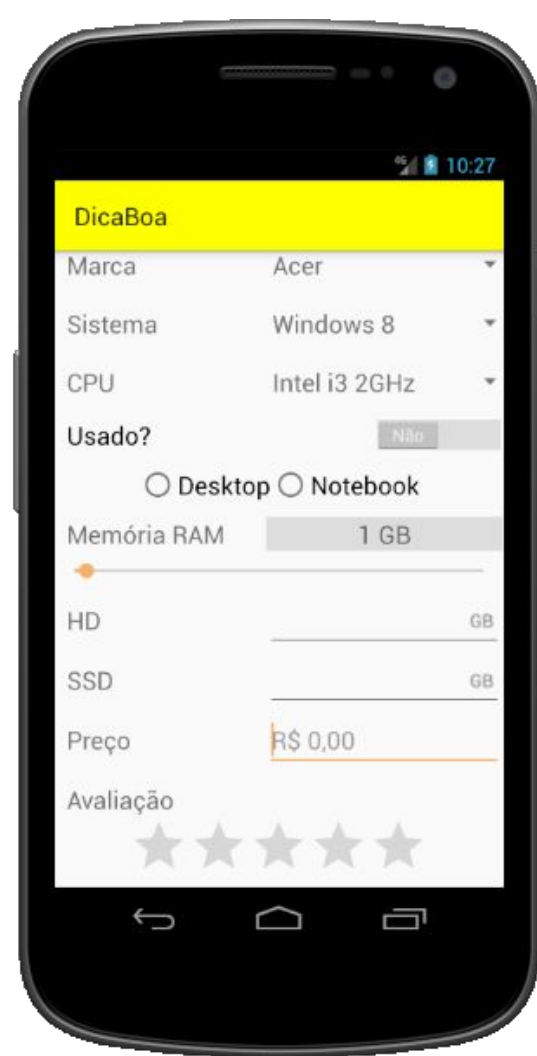
Este problema é fácil de resolver. Basta envolver todo o conteúdo da *TableView* em um *ScrollView*.

# Utilizando **ScrollView** (DicaBoa\_v6)

```
activity_cadastro.xml x
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:orientation="vertical">
7
8      <ScrollView
9          android:layout_width="match_parent"
10         android:layout_height="match_parent">
11
12         <TableLayout...>
13
14         </TableLayout>
15
16     </ScrollView>
17 </LinearLayout>
```

# Utilizando *ScrollView*

Problema resolvido!

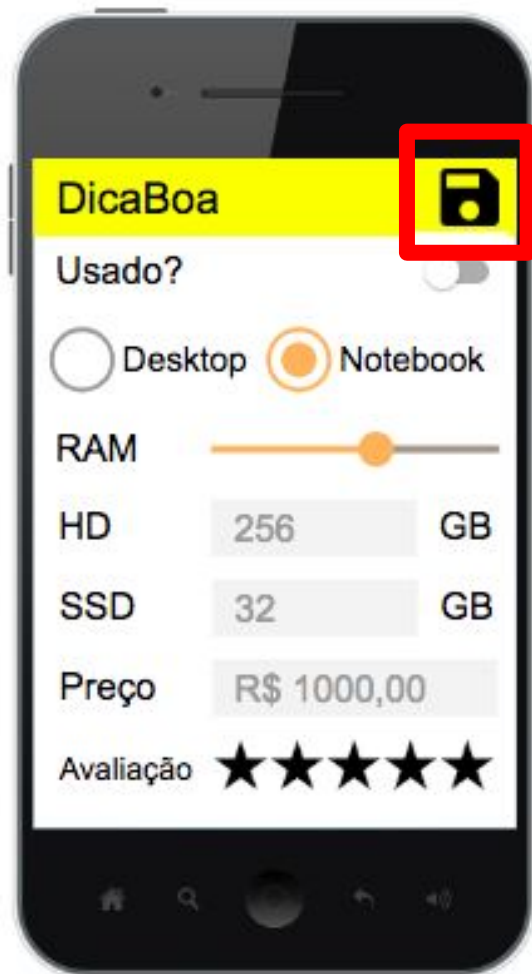




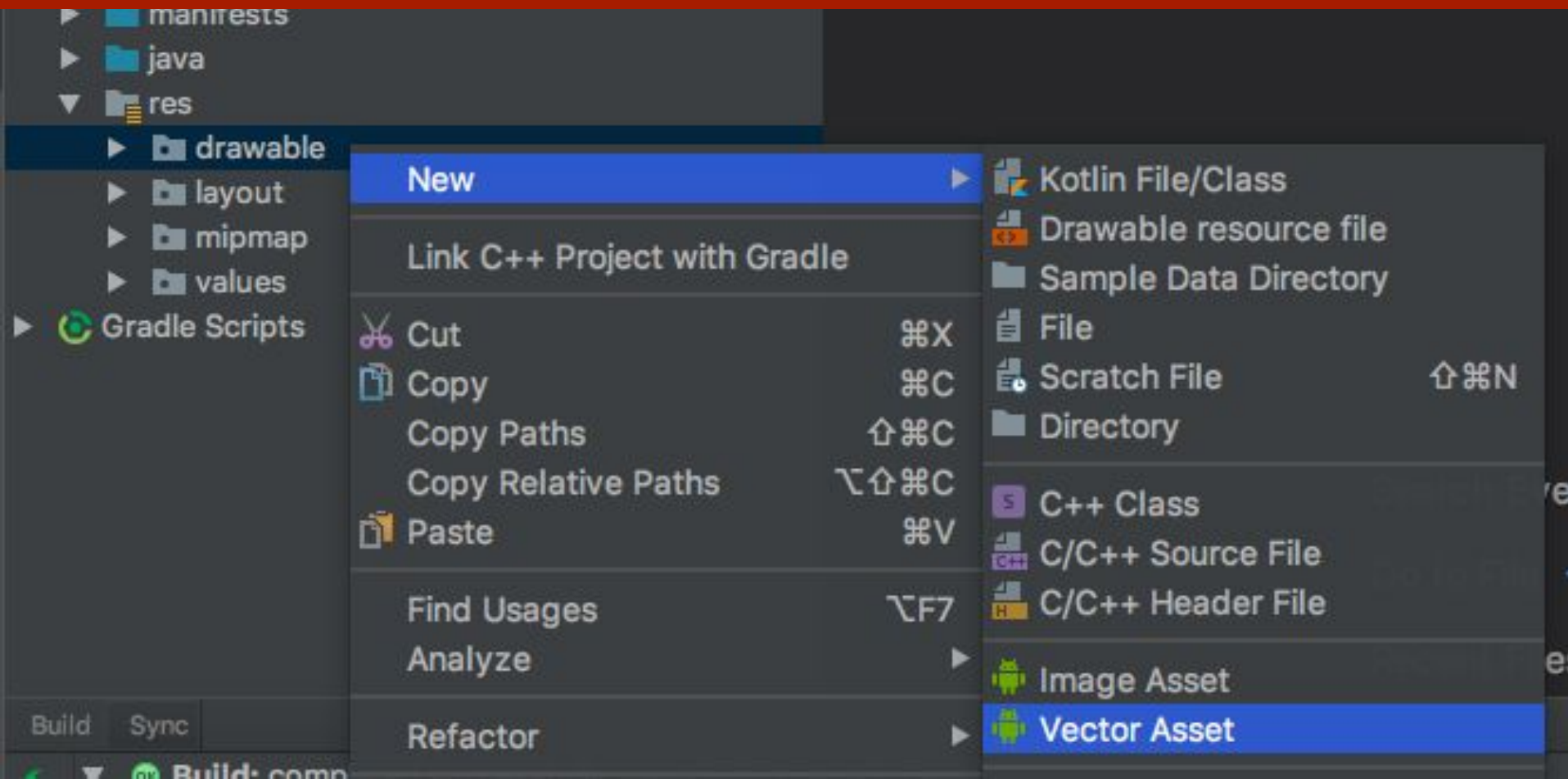
# Exercício em Sala

Continuando o desenvolvimento com base no nosso protótipo, adicione um item no menu do *ActionBar* conforme a imagem ao lado.

Carregue o arquivo *salvar.svg* (que encontra-se disponível na pasta imagens do EAD) como imagem vetorial no projeto e defina-o como ícone para o menu. Ao clicar neste item, o App deve exibir um Toast "Dados salvos com sucesso!".



# Resolvendo o Exercício





# Configure Vector Asset

Android Studio

Asset Type: ☐ Clip Art ☒ Local file (SVG, PSD)

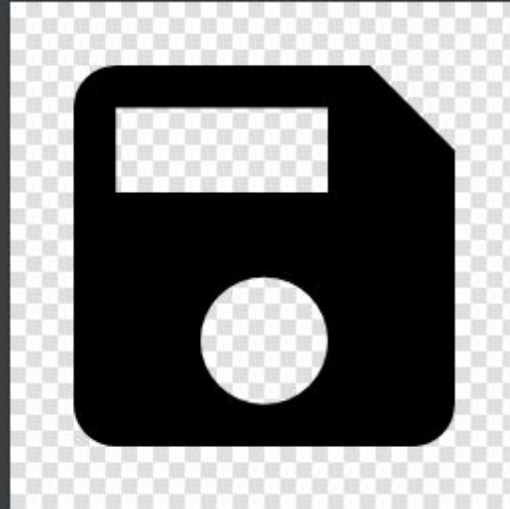
Name:

Path:  ...

Size:  dp X  dp ☐ Override

Opacity:  100 %

☐ Enable auto mirroring for RTL layout



Vector Drawable Preview



Cancel

Previous

Next

Finish



# Confirm Icon Path

Android Studio

Res Directory:

main

Output Directories:

▼ main  
▼ res  
▼ drawable  
ic\_salvar.xml



Cancel

Previous

Next

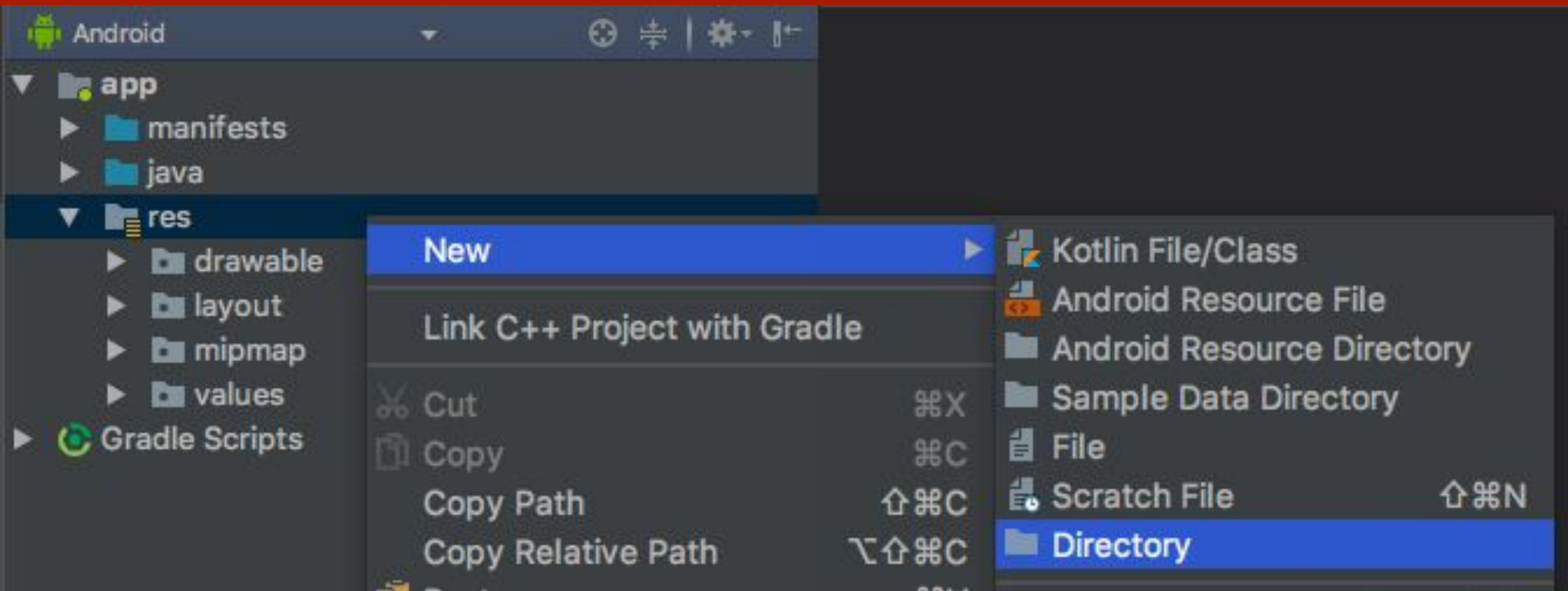
Finish

# Resolvendo o Exercício

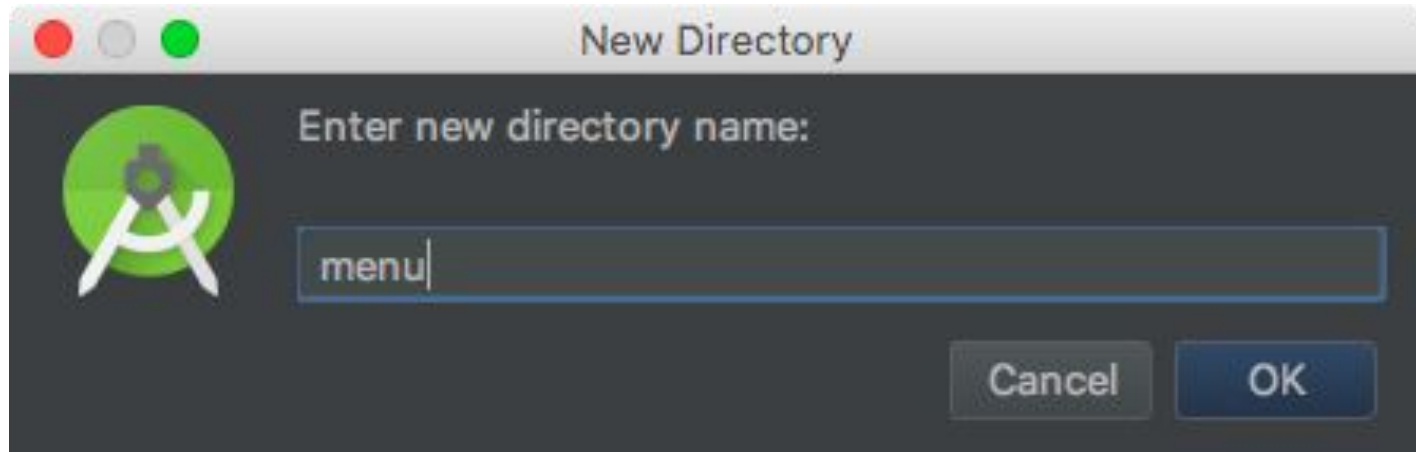
```
strings.xml x
Edit translations for all locales in the translations editor.

1  <resources>
2      <string name="app_name">DicaBoa</string>
3
4      <string name="form_label_marca">Marca</string>
5      <string name="form_label_sistema">Sistema</string>
6      <string name="form_label_processador">CPU</string>
7      <string name="form_label_usado">Usado?</string>
8      <string name="form_label_usado_on">Sim</string>
9      <string name="form_label_usado_off">Não</string>
10     <string name="form_radio_desktop">Desktop</string>
11     <string name="form_radio_notebook">Notebook</string>
12     <string name="form_seekbar_memoria">Memória RAM</string>
13     <string name="form_label_hd">HD</string>
14     <string name="form_label_ssd">SSD</string>
15     <string name="form_label_preco">Preço</string>
16     <string name="form_label_nota">Avaliação</string>
17     <string name="form_menu_salvar">Salvar</string>
18 </resources>
19
```

# Resolvendo o Exercício

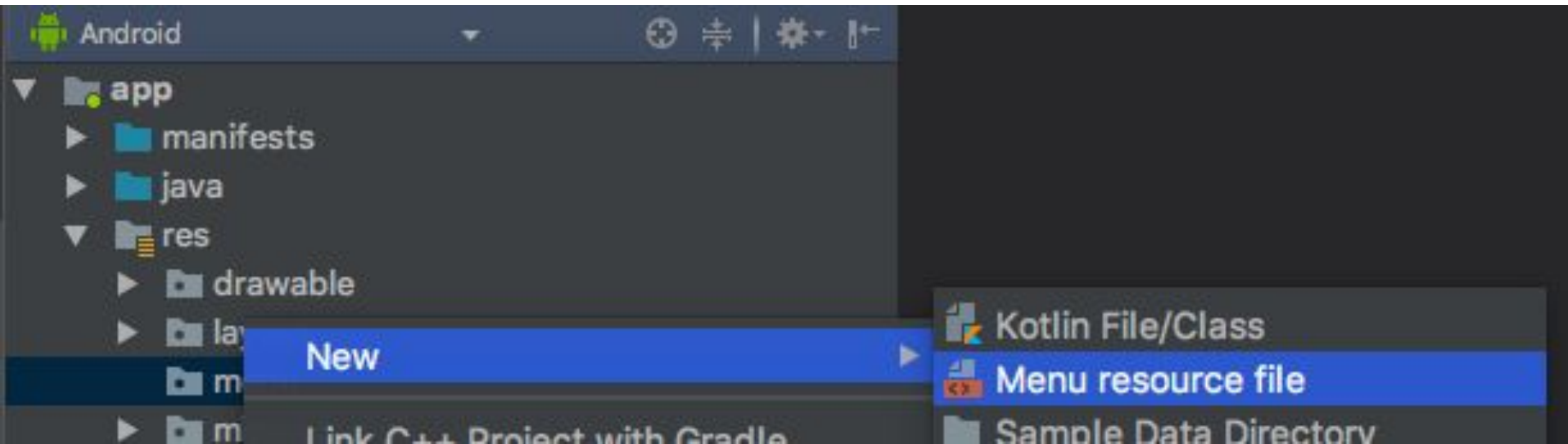


# Resolvendo o Exercício

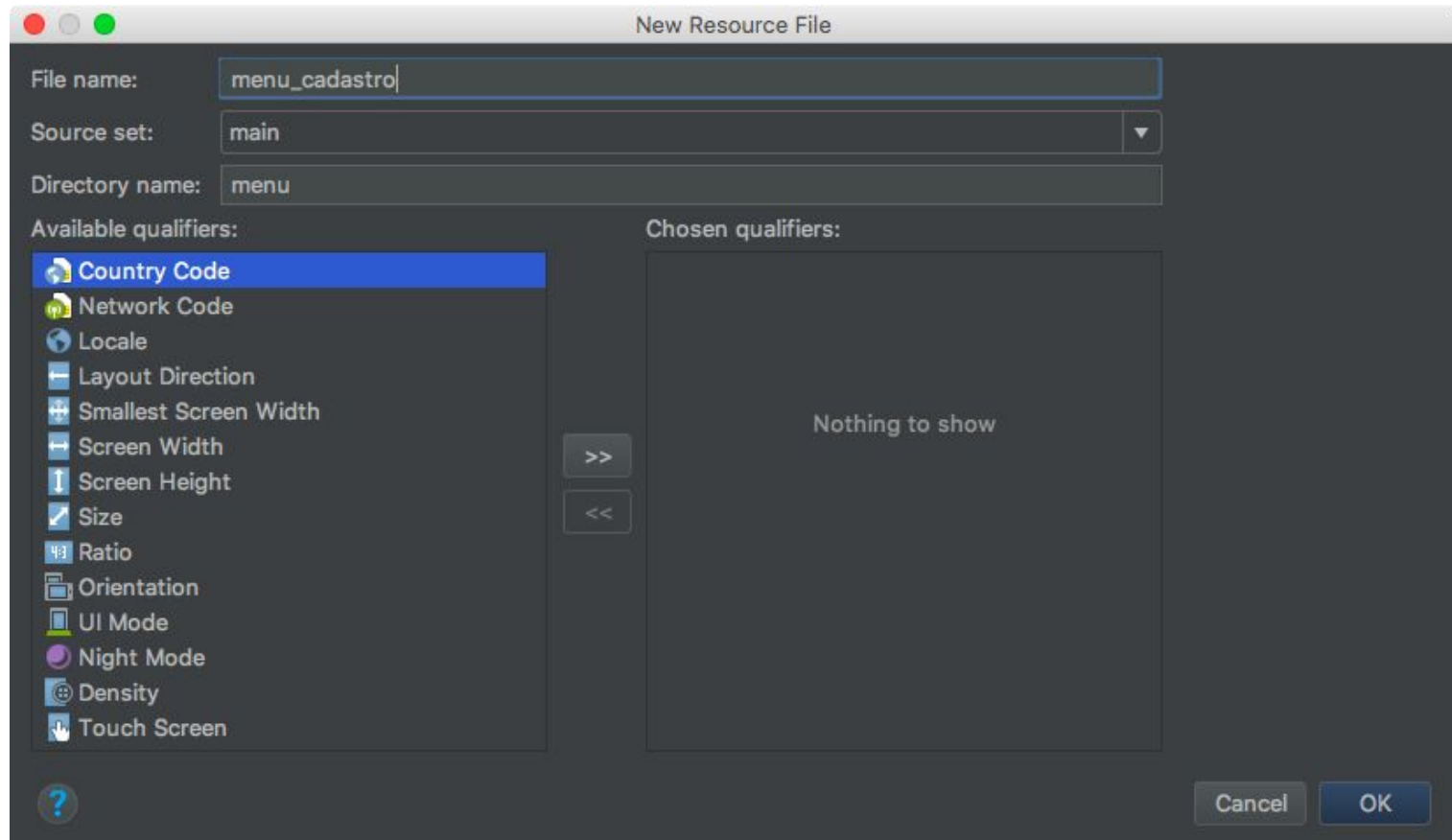




# Resolvendo o Exercício



# Resolvendo o Exercício



# Resolvendo o Exercício

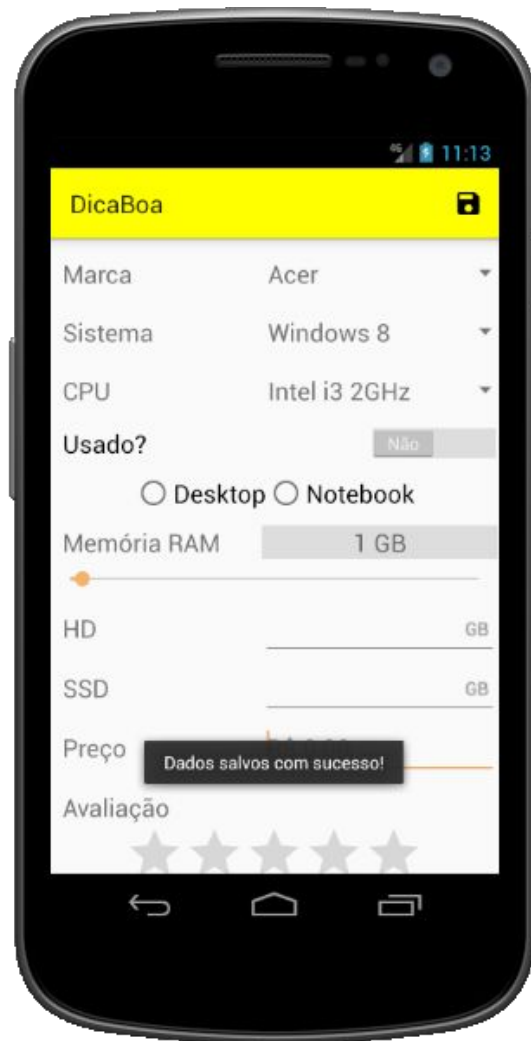
```
menu_cadastro.xml x
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto">
4      <item
5          android:id="@+id/form_cadastro_menu_salvar"
6          android:title="@string/form_menu_salvar"
7          android:icon="@drawable/ic_salvar"
8          app:showAsAction="always" />
9  </menu>
```

# Resolvendo o Exercício

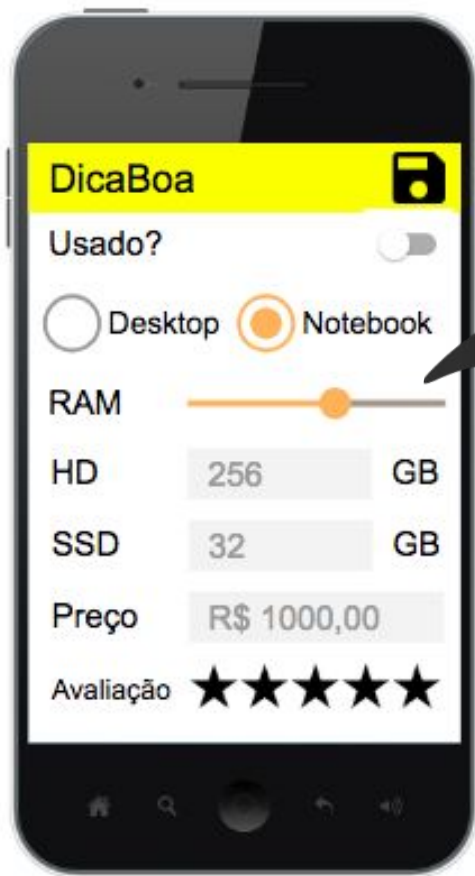
```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_cadastro, menu);
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.form_cadastro_menu_salvar:
            Toast.makeText(
                context, this, text: "Dados salvos com sucesso!",
                Toast.LENGTH_SHORT).show();
            break;
    }

    return super.onOptionsItemSelected(item);
}
```



# Obtendo os dados do formulário (*DicaBoa\_v6*)



The image shows a smartphone screen with a form titled "DicaBoa". The form has a yellow header bar with the title and a save icon. Below the header, there are several input fields: a toggle switch for "Usado?", radio buttons for "Desktop" and "Notebook" (with "Notebook" selected), a slider for "RAM", numeric input fields for "HD" (256 GB) and "SSD" (32 GB), a text field for "Preço" (R\$ 1000,00), and a star rating for "Avaliação" (5 stars).



`Computador.java`

O último passo desta aula será pegar todos os dados do formulário e criar um novo objeto *Computador*.

Isso será feito através do clique no botão salvar. Neste momento vamos aproveitar para validar o preenchimento dos dados. Caso algum dado esteja incorreto, exibiremos um *Toast* indicando o campo.



## Obtendo os dados do formulário (*DicaBoa\_v6*)

**PASSO 1:** crie um método *criarComputador()*. Por enquanto ele deve instanciar um novo *Computador()*, recuperar os objetos *marca*, *sistema* e *processador* e defini-los respectivamente pelos métodos *setMarca()*, *setSistema()* e *setCpu()*.

```
private void criarComputador() {  
  
    Computador computador = new Computador();  
  
    Marca marca = (Marca) spinnerMarcas.getSelectedItem();  
    computador.setMarca(marca);  
  
    Sistema sistema = (Sistema) spinnerSistemas.getSelectedItem();  
    computador.setSistema(sistema);  
  
    Processador processador = (Processador) spinnerProcessadores.getSelectedItem();  
    computador.setCpu(processador);  
}
```

## Obtendo os dados do formulário (*DicaBoa\_v6*)

**PASSO 2:** ainda no *criarComputador()*, pegue o valor do *switchUsado* com *isChecked()* e salve-o na instância do computador através do método *setUsado()*.

```
computador.setUsado(switchUsado.isChecked());

int idSelecionado = radioButtonTipo.getCheckedRadioButtonId();
if (idSelecionado == -1) {
    Toast.makeText(context, this, text: "Selecione o tipo!", Toast.LENGTH_SHORT).show();
    return;
}

RadioButton radioButtonSelecionado = (RadioButton) findViewById(idSelecionado);
computador.setTipo(radioButtonSelecionado.getText().toString());
```

**PASSO 3:** vamos obter o ID do *RadioButton* selecionado através do método *getCheckedRadioButtonId()*. O valor -1 indica que não houve seleção. Neste caso vamos exibir um Toast informando e terminar a execução de *criarComputador()*. Caso contrário, obtenha o *RadioButton* pelo ID e salve seu valor em *setTipo()* do computador.

## Obtendo os dados do formulário (*DicaBoa\_v6*)

**PASSO 4:** ainda no *criarComputador()*, vamos obter o valor da memória RAM direto do *SeekBar* através do método *getProgress()*.

Para obter o valor dos *EditText*'s de HD e SSD, devemos primeiro verificar se o usuário entrou com algum número. Como ambos são do tipo *number*, podemos garantir que havendo algum valor, a conversão para `int` é segura. Neste caso, convertemos os valores e guardamos nos métodos *setHd()* e *setSsd()* da instância *computador*.

```
computador.setMemoriaRam(seekBarMemoria.getProgress());
```

```
String hd = editTextHD.getText().toString();
```

```
if (!hd.equals("")) { computador.setHd(Integer.parseInt(hd)); }
```

```
String ssd = editTextSSD.getText().toString();
```

```
if (!ssd.equals("")) { computador.setSsd(Integer.parseInt(ssd)); }
```

## Obtendo os dados do formulário (*DicaBoa\_v6*)

**PASSO 5:** podemos obter o texto do *EditText* formatado normalmente. Vamos apenas aplicar uma regra para o caso do usuário não ter preenchido nenhum valor o sistema utilizar a String R\$0,00

```
String preco = editTextPreco.getText().toString();  
if (preco.equals("")) { preco = "R$0,00"; }  
computador.setPreco(preco);  
  
computador.setNota(ratingBarNota.getRating());
```

**PASSO 6:** Obtenha a nota do *RatingBar* pelo método *getRating()*. Note que em nosso modelo o atributo *nota* é do tipo *double*, enquanto que o método *getRating()* retorna um tipo *float*. O Java permite fazer essa conversão sem nenhuma perda de precisão porque o *double* armazena valores de ponto flutuante com 64 bits, contra os 32 bits do *float*. Mais detalhes: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.frm_cadastro_menu_salvar:
            Toast.makeText(context, this, text: "Dados salvos com sucesso!",
                           Toast.LENGTH_SHORT).show();
            break;
    }

    return super.onOptionsItemSelected(item);
}
```

Toast.makeText(context, this, text: "Dados salvos com sucesso!",  
Toast.LENGTH\_SHORT).show();



Copie o Toast de sucesso do clique no menu salvar para o final do método *criarComputador()*. Acrescente também um LOG exibindo o *toString()* do computador.

```
private void criarComputador() {
    Computador computador = new Computador();
    // Restante do código ...

    Toast.makeText(context, this, text: "Dados salvos com sucesso!",
                   Toast.LENGTH_SHORT).show();
    Log.i(tag: "COMPUTADOR", computador.toString());
}
```



# Obtendo os dados do formulário (*DicaBoa\_v6*)

## PASSO 8

No lugar do Toast (do clique de salvar), invoque o método *criarComputador()*

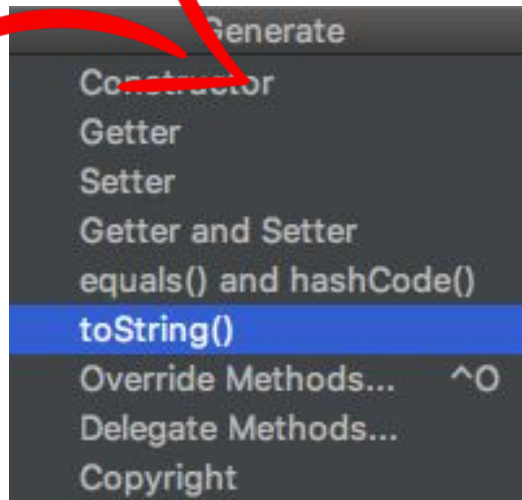
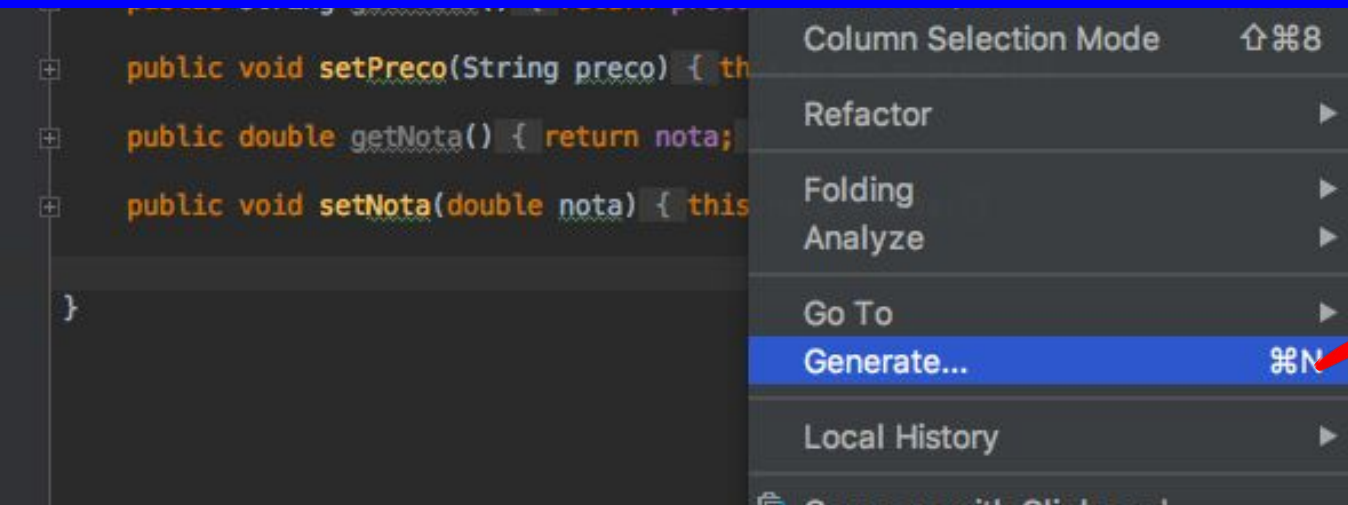
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.form_cadastro_menu_salvar:
            criarComputador();
            break;
    }

    return super.onOptionsItemSelected(item);
}
```



# Obtendo os dados do formulário (*DicaBoa\_v6*)

## PASSO 9

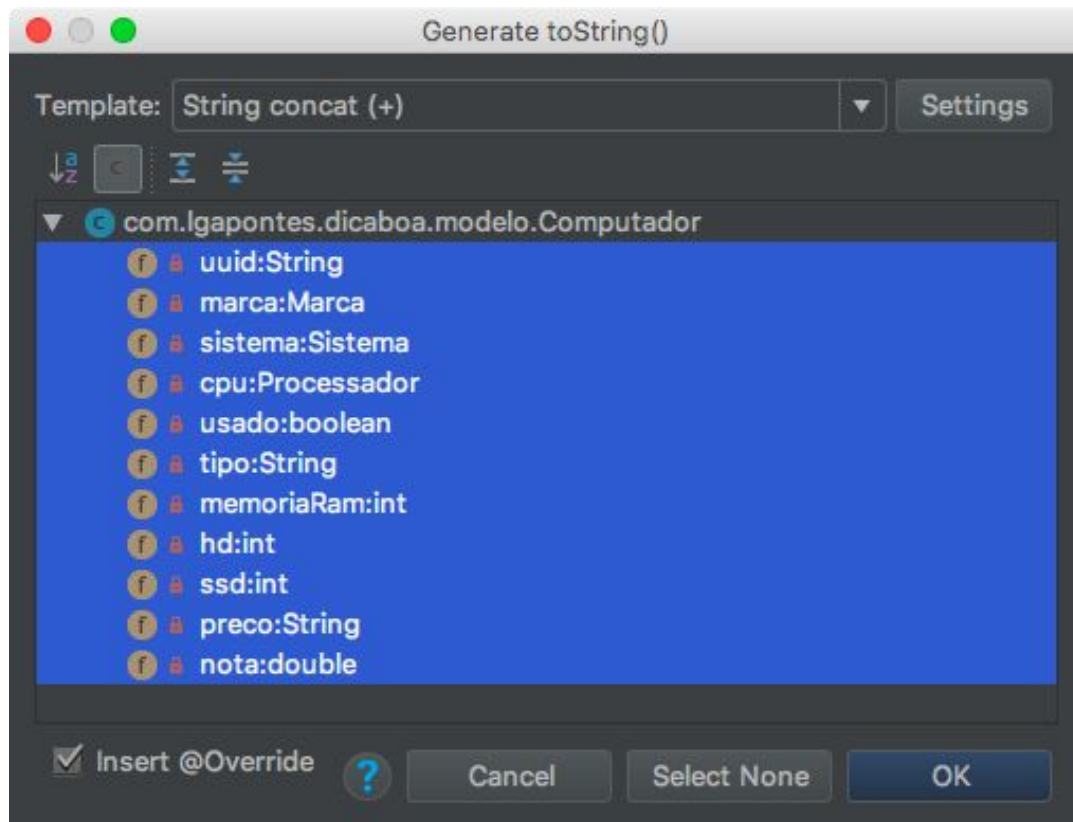


Por fim, na classe de modelo *Computador.java*, vamos sobrescrever o *toString()*. Clique com o botão direito no trecho onde o método deve ser inserido, opção *Generate...* Em seguida, selecione a opção *toString()*

# Obtendo os dados do formulário (*DicaBoa\_v6*)

## PASSO 10

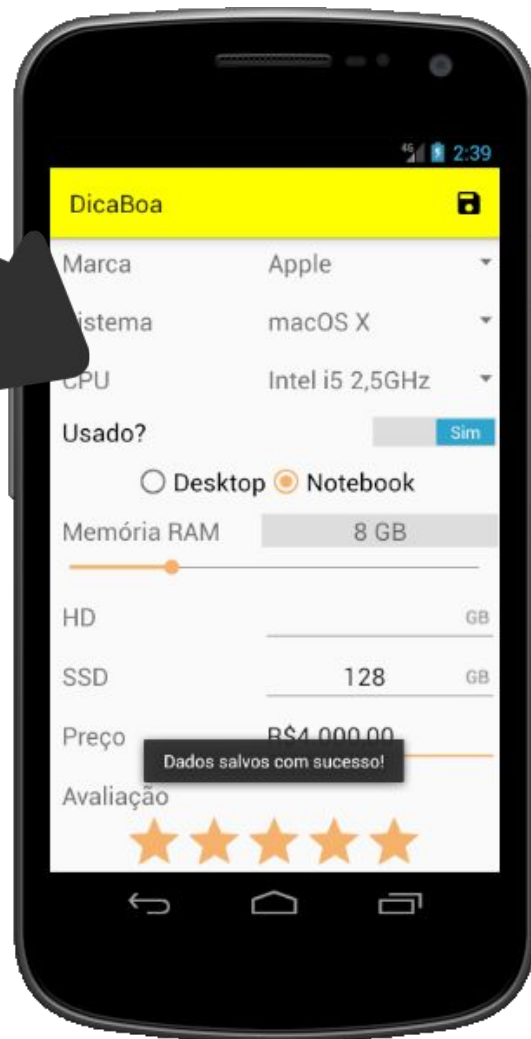
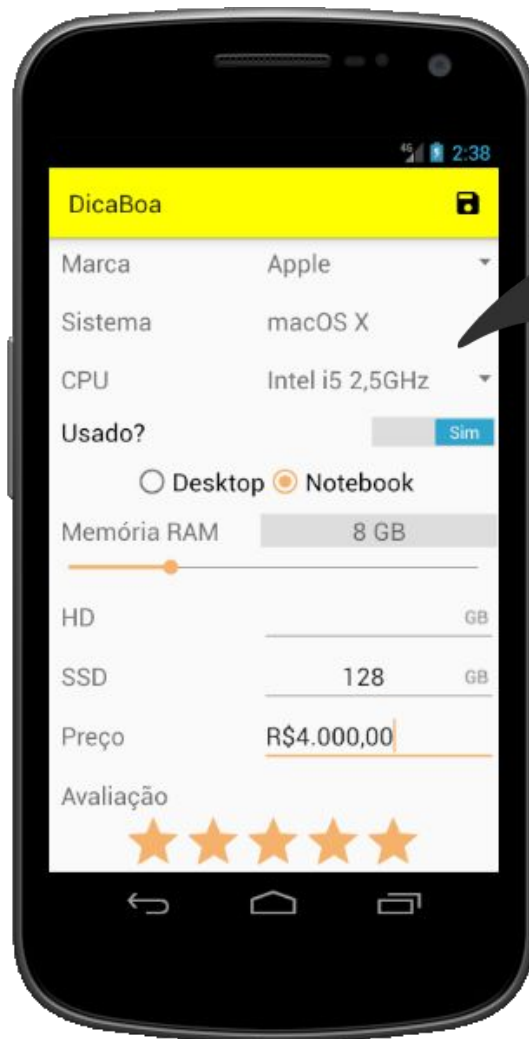
Selecione todos os atributos e clique OK

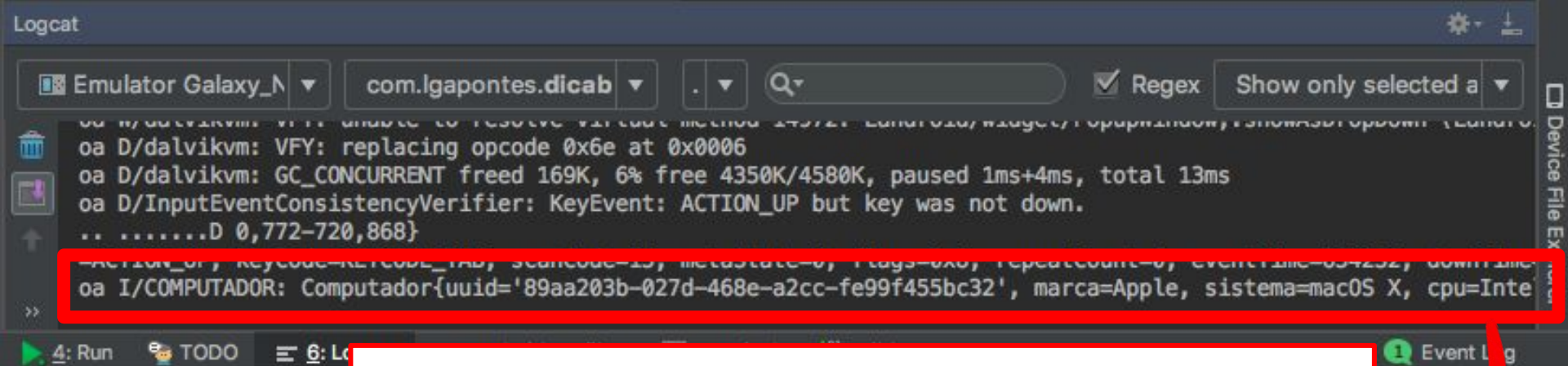


# Obtendo os dados do formulário (*DicaBoa\_v6*)

Mantenha o código gerado pelo Android Studio.

```
@Override
public String toString() {
    return "Computador{" +
        "uuid='" + uuid + '\'' +
        ", marca=" + marca +
        ", sistema=" + sistema +
        ", cpu=" + cpu +
        ", usado=" + usado +
        ", tipo='" + tipo + '\'' +
        ", memoriaRam=" + memoriaRam +
        ", hd=" + hd +
        ", ssd=" + ssd +
        ", preco='" + preco + '\'' +
        ", nota=" + nota +
        '}';
}
```





```
Computador{
    uuid='89aa203b-027d-468e-a2cc-fe99f455bc32',
    marca=Apple,
    sistema=macOS X,
    cpu=Intel i5 2,5GHz,
    usado=true,
    tipo='Notebook',
    memoriaRam=8,
    hd=0,
    ssd=128,
    preco='R$4.000,00',
    nota=5.0
}
```

**Obrigado!**

**Próxima aula: salvar dados no SQLite...**