Desenvolvimento de aplicações para Android

Interface, layouts e views

Criação de projeto

- Criar um projeto *Android*
  - API: 22
  - Indicar como ecrã inicial uma "Empty Activity"

Activity

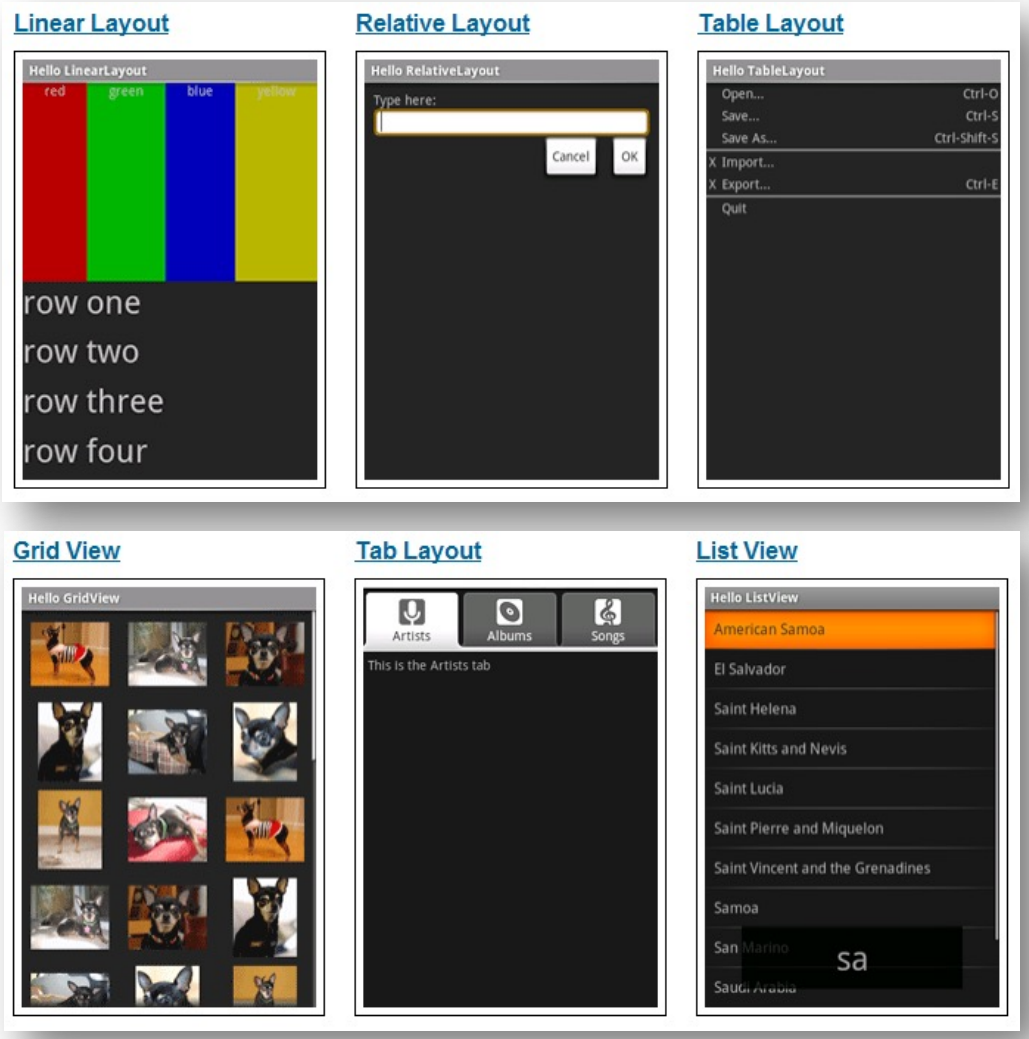
- Com a criação do projeto com as configurações referidas, é criada automaticamente uma atividade

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Interface

- A interface das aplicações é constituída por uma hierarquia de objetos `View`
  - Existe um tipo de objeto `ViewGroup`, que deriva de `View`, que permite conter outros objetos `View`
    - Estes objetos permitem organizar os componentes existentes nas interfaces e designam-se por objetos "*layout*", exemplos:
      - `FrameLayout`
      - `LinearLayout`
      - `RelativeLayout`
      - `TableLayout`
      - ~~`AbsoluteLayout`~~
      - `...`
      - `ConstraintLayout` (*Android Jetpack – androidx*)

Tipos de Layout



Interface

- Os objetos podem ser
  - criados programaticamente
    - Criando objetos `View` e derivados
    - Associando-os à atividade com:
      - `setContentView(view: View)`
  - construídos a partir de ficheiros *xml*
    - O layout da aplicação é definido através de *xml*
    - Associando à atividade com:
      - `setContentView(layoutResID: Int)`
    - Quando associado à atividade a informação é lida do ficheiro e os objetos são criados e configurados um-a-um de forma automática (operação '*inflate*')

Ficheiros xml de layout

- Os ficheiros *xml* incluem a descrição de todos os objetos que constituem o *layout*
  - Gravados na secção *layout* dos *resources*
  - Editados em *xml* ou usando o editor disponibilizado
  - As *tags xml* permitem definir os objetos a criar, bem como os atributos referentes à sua configuração inicial
- A cada objecto é atribuído um ID que vai permitir a sua identificação
  - Para cada um destes ID's será gerado uma constante na classe 'R' criada e gerida pelo *Android Studio*
  - Estas constantes permitem o acesso aos elementos que representam

Ficheiros xml de layout

Exemplo de um ficheiro

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/tvMsg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

</LinearLayout>
```

Ficheiros xml de layout

- Para se obter a referência para os objetos criados pela operação *inflate*, de modo a usar os objetos no contexto das classes *Kotlin*, pode-se usar a função `findViewById`, passando como parâmetro o *id* atribuído (`R.id.<ID>`), por exemplo:

```
val tvMsg : TextView = findViewById(R.id.tvMsg)

tvMsg.text = "DEIS-AMOV"
```

View Binding

- Com *Kotlin* + *Android Jetpack* a tarefa de obter referências para as *views* está facilitada através da funcionalidade *View Binding*
  - incluir no ficheiro `build.gradle (module)`:

```
android {
    ...
    buildFeatures {
        viewBinding = true
    }
}
```

View Binding

- na classe *Kotlin* onde se pretende usar

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
        // o nome ActivityMainBinding é baseado no nome do XML

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.tvMsg.text = "DEIS-AMOV"
    }
}
```

Atributos de alinhamento

- Os elementos incluídos na interface podem ser configurados através de diferentes atributos que vão ter reflexo ao nível do seu alinhamento
  - No contexto do *layout*
    - Obrigatórios
      - `android:layout_width`
      - `android:layout_height`
        - Estes atributos podem assumir os valores `wrap_content`, `match_parent` (`fill_parent`) ou um valor específico para a altura e/ou largura no formato <valor><unidade>
          - As unidades aceites são as seguintes: *px* (*pixels*), *dp* (*density-independent pixels*), *sp* (*scaled pixels based on preferred font size*), *in* (*inches*), *mm* (*millimeters*)
    - Outros: `layout_weight`, `layout_margin`, …
      - Quando se definem pesos para os elementos gráficos a dimensão a ser considerada na divisão dos pesos (largura ou altura) deve ser colocada com `0dp`
  - Internos ao elemento
    - Ex: `gravity`, `padding`, …

Desenvolvimento de aplicações para Android

Processamento de eventos

Processamento de eventos

- Assumindo a definição de um botão através do seguinte ficheiro de layout

```
<?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" >
    <Button
        android:id="@+id/btnOk" android:text="OK"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"  />
</LinearLayout>
```

Processamento de eventos

- Os eventos gerados sobre os diversos elementos podem ser processados recorrendo a *listeners*
- Exemplo para um botão…
  - Obter referência para o botão

    ```
    val btn : Button = findViewById(R.id.btnOK)
    ```

  - Usar o método

    ```
    btn.setOnClickListener(ObjectoOnClickListener)
    ```

  - O *`ObjetoOnClickListener`* pode ter como origem:
    - Implementação da interface na própria classe (*this*)
    - Implementação da interface no âmbito de uma classe criada para o efeito (normal, *nested* ou *inner*)
    - Instanciação de um objeto através de um classe anónima
    - Criação e implementação *inline* de um objeto de uma classe anónima – *Lambda function*

Processamento do botões

- Os botões permitem ainda o processamento de cliques usando o atributo `onClick` na descrição dos componentes em XML
  - Propriedade não existente nas primeiras versões do *Android*
  - Definir na propriedade o nome do método a executar
  - O método tem que seguir o seguinte *template*

```
fun nomeMétodo(v : View)
{
    …
}
```

Estilo dos botões

- Pode-se alterar o estilo dos botões
  - Definir imagens sugestivas para cada um dos botões adaptadas a cada situação
    - Quando pressionado, quando tem o *focus*, quando está *disabled*
- Podem-se incluir imagens nos próprios botões alterando os atributos *drawable* do botão
  - `android:drawable[left|top|bottom|right]`
  - Existe também uma especialização do botão, designada por `ImageButton`, mais adaptada à configuração de botões apenas com imagens
- Ver…
  - http://developer.android.com/guide/topics/ui/controls/button.html

Exercício 1a

- Implementar uma calculadora básica que permita as operações +, -, *, /, ...