Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

Programação de Dispositivos Móveis
Guia para Aula Laboratorial 4
Licenciatura em Engenharia Informática

Programming of Mobile Devices
Guide for Laboratory Class 4
Degree in Computer Science and Engineering

Sumário
Análise de formas complementares de implementação de consumidores de eventos. Exercícios para análise do ciclo de vida de uma Atividade numa aplicação Android™, bem como para a prática do uso do registo de sistema disponibilizado pelo Android™ e conhecido por *logcat*.

Summary
Analysis of alternative ways of implementing event Handlers. Exercises to analyze the lifecycle of an Android™ application Activity, as well as for practicing the usage of log system provided by Android™ and known as logcat*.*

Pré-requisitos:
Algumas das tarefas enunciadas a seguir requerem o acesso a um sistema com o Android Studio e com o SDK Android™, bem como com a Gradle™ instalados ou, alternativamente, com permissões para instalação e configuração do IDE, *kit* e ferramenta. Serão suficientes permissões para criar diretorias e ficheiros num disco local e para configurar variáveis de sistema, nomeadamente a *path*. É necessário ter acesso a uma versão e imagem da plataforma Android™ ou a um dispositivo físico com o sistema operativo e com a opção de *debug* ativa. É igualmente necessário ter um compilador Java instalado.

1 Preliminares
Preliminaries

O guia laboratorial 2 elabora nos passos necessários a criação e compilação (*build*) de projetos de aplicações para a plataforma Android™ via linha de comandos. Esta abordagem, apesar de não comportar algumas das facilidades oferecidas por ambientes de desenvolvimento integrados, nomeadamente ambientes de edição da interface de utilizador *What You See Is What You Get* (WYSIWYG), permite conhecer em maior profundidade os detalhes de implementação de uma aplicação Android™, mas requer que, após instalação do Android Studio, se atualize e instalem as várias ferramentas do *Software Development Kit* (SDK) Android™[1]. Depois do sistema estar devidamente configurado, 4 passos são suficientes para criar um projeto Android™, gerar o ficheiro `.apk` e instalar a aplicação num dispositivo (virtual ou real):

1. Inicializar o dispositivo móvel virtual ou ligar um real ao computador[2];
2. Gerar o projeto através do Android Studio;
3. Compilar o projeto com a ferramenta Gradle™, emitindo o comando `$ gradlew assembleDebug` na raiz do projeto;
4. Instalar a aplicação com um comando semelhante a `$ adb install -r path\NomeApp-debug.apk`.

[1] Ver `https://developer.android.com/studio/intro/update`.
[2] Se o dispositivo for real, tem de ter a opção de depuração ativada.

Tarefa 1 *Task 1*

Como já vem sendo habitual, a primeira tarefa consiste em iniciar um *Android Virtual Device* (AVD). Para isso, pode emitir o comando `$ emulator`, incluído na pasta `emulator` do SDK, e lançar um AVD. Caso não exista nenhum AVD configurado, crie um[3]. O ideal será um emulador de uma versão superior à 6.0 do Sistema Operativo (SO).

Tarefa 2 *Task 2*

Verifique se as variáveis `ANDROID_HOME` e `PATH` estão devidamente definidas com comandos parecidos com:

```
$ echo %ANDROID_HOME%
$ echo %JAVA_HOME%
$ echo %PATH%
```

Caso as variáveis já estejam devidamente definidas, passe para a secção seguinte. Caso contrário, precisa de as definir antes de avançar, com comandos semelhantes a:

```
$ set JAVA_HOME=RAIZ do JAVA JDK
$ set ANDROID_HOME=C:\installation location\sdk
```

[3] O guia laboratorial 1 contém uma breve discussão acerca deste assunto.

```
$ set PATH=%PATH%; %JAVA_HOME%;
%ANDROID_HOME%\tools; %ANDROID_HOME%\emulator;
%ANDROID_HOME%\platform-tools
```

Nota: Para mais detalhes consultar o guia laboratorial 2.

2 Rotinas de Tratamento de Eventos

Event Handlers

O guia laboratorial anterior sugeria a criação de uma aplicação que imitasse uma simples calculadora. Também foi sugerido que associasse um consumidor (*listener*) para cada objeto interativo do tipo botão, definindo-lhe uma rotina de tratamento do evento *clique no rato*, para as várias operações aritméticas pedidas. A ideia desta parte do guia é a de mostrar outras formas de implementar as rotinas de tratamento de eventos ou de associar um evento do tipo *clique* a uma dessas rotinas.

Q1.: Só para relembrar: qual é o pacote que deve importar para usar objetos interativos como o botão?

☐ `android.swing.*` ☐ `android.util.*`
☑ `android.widget.*` ☐ `android.view.*`

Q2.: Por curiosidade, qual é a classe pai (ou super classe) da classe `Button`?

☐ `java.lang.Object` ☐ `android.widget.Button`
☑ `android.view.View` ☐ `android.widget.TextView`

Q3.: Já agora, como se chama o cunhado do `Button`?

☐ ?
☐ ???
☑ ??????

Tarefa 3 *Task 3*

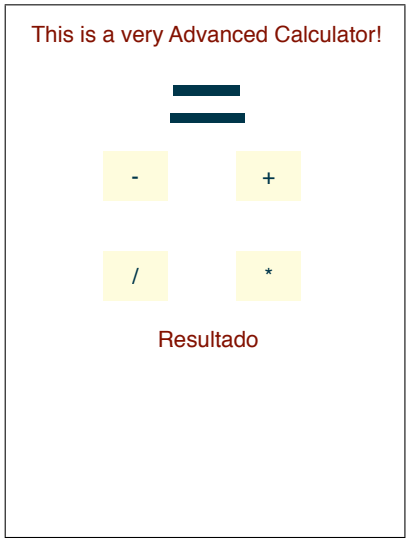
Juntamente com este guia laboratorial são disponibilizados dois arquivos com extensão `.zip`. Comece por descarregar e descompactar o arquivo `AdvCalculator2.zip`. Depois disso, considere analisar o ficheiro XML aí contido e que define a interface de utilizador de uma aplicação chamada `AdvCalculator2`, transcrito também para aqui por comodidade:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/
    android"
  xmlns:app="http://schemas.android.com/apk/res-auto
      "
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".ACalculator">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textview1"
    android:text="This is a very Advanced Calculator
        !"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=" "
    android:id="@+id/number1"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/
        textview1" />
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=" "
    android:id="@+id/number2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/number1
        " />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="+"
    android:id="@+id/SUM"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/
        number2"
    app:layout_constraintLeft_toRightOf="@+id/SUB" /
        >
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="-"
    android:id="@id/SUB"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@id/SUM"
    app:layout_constraintTop_toBottomOf="@id/number2
        " />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="x"
    android:id="@+id/MUL"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/SUM"
    app:layout_constraintLeft_toRightOf="@+id/DIV" /
        >
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="/"
    android:id="@id/DIV"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@id/MUL"
    app:layout_constraintTop_toBottomOf="@id/SUM" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Resultado"
    android:id="@+id/result"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/DIV" />
</android.support.constraint.ConstraintLayout>
```

Use o espaço em baixo para desenhar a interface de utilizador que vai ser gerada no dispositivo sem executar a aplicação, i.e., deduza-a da análise do XML incluído antes:



Tarefa 4 *Task 4*

Usando o Android Studio, crie um novo projeto com as seguintes características:

- Nome da aplicação — `AdvCalculator2`;
- Domínio — `pt.ubi.di.pmd`;
- Sem suporte para C++ ou Kotlin;
- Com uma `Empty Activity` chamada `ACalculator`;
- Peça para gerar o ficheiro de *layout* (o nome do ficheiro de *layout* deve ser `activity_acalculator.xml`);
- Retire qualquer suporte de retrocompatibilidade.

Tarefa 5 *Task 5*

Procure os ficheiros `activity_acalculator.xml` e `ACalculator.java` dentro da pasta `src/main/...` e substitua-os pelos dois ficheiros que vinham no ficheiro `.zip` que descarregou. **Não se esqueça de** compilar e testar o projeto no dispositivo virtual Android™ que deve ter inicializado antes.

Q4.: A interface de utilizador que desenhou em cima é semelhante à que realmente apareceu?
☐ É igualzinha!
☐ Ups, os botões não ficaram bem como eu os tinha colocado.
☐ Ups, o resultado não ficou bem como eu tinha pensado que ia ficar. Vou já ver o que é que compreendi mal.

Q5.: Experimentou os botões?
☐ Não sabia que era para experimentar.
☐ Não, não experimentei, mas vou experimentar.
☐ Experimentei, e não faziam nada, mas deviam fazer.
☑ Experimentei, e não faziam nada (como não podia deixar de ser).

Tarefa 6 *Task 6*

Note que parte do código Java que implementa a aplicação está comentado (por estar incompleto). Estes trechos de código permitem que seja definido **apenas um** (em vez de um para cada *widget*) objeto consumidor `onClickListener` e implementada uma só vez a rotina de tratamento `onClick()` para quando os botões são clicados. Contudo, **é preciso tratar cada botão individualmente dentro da rotina**, conforme já é sugerido. Note que falta código (muito pouco) para que o trecho de código comentado esteja completo.

Complete o código corretamente de modo a que os cliques nos botões produzam o resultado esperado. No final compile e certifique-se de que a aplicação funciona corretamente. **Sugestão:** use os IDs dos vários recursos nos vários casos do `switch`.

Se fez tudo bem, deve ter usado uma referência à classe Java chamada `R`. **Q6.: Onde está implementada esta classe? Por outras palavras, onde está o ficheiro `R.java`?**
☑ Algures em `build/generated/source/r/...`
☐ Em `build/generated/source/buildConfig`
☐ Este ficheiro é aquele que permite obter `Riot Points` no League of Legends.
☐ Não faço a mínima ideia, não encontro nenhum ficheiro `R` ...

Abra o ficheiro `R.java`. **Q7.: A que conceito se refere a instrução `R.id`?**
☑ A uma classe estática. ☐ A uma atividade.
☐ A um atributo estático. ☐ A um método.

Q8.: Em que situação é conveniente editar este ficheiro?
☐ Sempre que se quer adicionar um identificador novo a um *widget*.
☑ Só no dia de São Nunca à tarde.
☐ Sempre que se adicionam atividades.
☐ Antes de compilar.
☐ Depois de comer.
☐ Durante o sono.

Q9.: Afinal, o que são os IDs que utiliza no código para referenciar alguns recursos de uma aplicação Android™?
☐ São *strings*. ☑ São números inteiros.
☐ São números reais. ☐ São *cheesy* bytes.

Tarefa 7 *Task 7*

Na plataforma Android™, existem outras formas de associar um método consumidor a um botão, pelo menos para o evento `onClick`. Uma dessas formas consiste em indicar o nome do método que vai tratar o evento no ficheiro XML de *layout*. Pode encontrar mais informação acerca deste assunto na Web, nomeadamente em `http://developer.android.com/reference/android/widget/Button.html` mas, basicamente, esta associação faz-se definindo mais um atributo no elemento botão com a seguinte sintaxe:

```
android:onClick="CALLBACK_METHOD_NAME"
```

Esta tarefa consiste, portanto, na alteração do ficheiro de *layout* do projeto *ACalculator* de modo a que o botão de soma despolete o método `somar(View v)` que também já foi implementado no ficheiro `ACalculator.java`, **para sua conveniência**.

Q10.: De uma maneira geral, esta abordagem parece-lhe mais simples que as que já foram estudadas até aqui?
☑ De facto, parece-me mais simples (legível).
☐ É-me indiferente.

O método que acabou de definir como *Callback* tem um parâmetro de entrada do tipo `View`. **Q11.: Este parâmetro é mesmo necessário?**
☑ É sim. Sem ele, o *callback* não funciona.
☐ Não, neste caso não é preciso.
☐ É sim, e pode levar mais parâmetros adicionais.

Tarefa 8 *Task 8*

Depois de fazer a modificação necessária, compile e teste a aplicação, verificando que funciona.

Q12.: Por curiosidade, em que versão da *Application Programming Interface* (API) Android™ é que os botões foram disponibilizados?
☑ Na API 1. ☐ Na API 5. ☐ Na API 10.
☐ Na API 19. ☐ Na API 21. ☐ Na API 22.
☐ E como é que hei-de saber isso?

.

3 Registo do Sistema e Ciclo de Vida de uma Atividade

Logging and Activity Lifecycle

A segunda parte deste guia laboratorial tem como objetivo estudar o ciclo de vida das atividades.

Tarefa 9 *Task 9*

Usando o Android Studio, crie um novo projeto com as seguintes características:

- Nome da aplicação — `ActivityLifeCycle`;
- Domínio — `pt.ubi.di.pmd`;
- Sem suporte para C++ ou Kotlin;
- Com uma `Empty Activity` chamada `ActivityLifeCycle`;
- Peça para gerar o ficheiro de *layout*;
- Retire qualquer suporte de retrocompatibilidade.

Tarefa 10 *Task 10*

Descarregue para o seu sistema o arquivo `ActivityLC.zip`, também disponibilizado com este guia laboratorial. Procure o ficheiro `ActivityLifeCycle.java` dentro da pasta `src/main/...` e substitua-o pelo ficheiro que vem no arquivo `.zip` que descarregou. **Não se esqueça de** compilar e testar o projeto no dispositivo virtual Android™ que deve ter inicializado antes.

Tarefa 11 *Task 11*

Irá notar que há código comentado e em falta no ficheiro `ActivityLifeCycle.java`. Os locais onde falta código estão marcados com:

```
// FALTA CODIGO
```

Retire os escapes para comentário e complete o código que falta de modo a que a aplicação escreva entradas no registo do sistema (`logcat`). As mensagens a aparecer devem ser semelhantes a:

```
onCreate() method was called,
onStart() method was called,
...
onDestroy() method was called.
```

A *tag* a utilizar deve ser `ALC`. Já agora, estime se as questões seguintes ajudam na execução desta tarefa.

Note que vai precisar de importar a classe que permite escrever entradas no registo do sistema. **Q13.: Como se chama a classe que lhe permite fazer isso?**
☐ `LogCat` ☐ `Log` ☐ `Logarithm` ☐ `Register`

Q14.: Qual o pacote que terá de importar para poder usar a classe mencionada antes?
☐ `Java.util.*`
☐ `android.app.Activity`
☐ `android.util.*`
☐ `Java.View.*`

Tarefa 12 *Task 12*

Depois de alterar o código, compile, instale e teste a aplicação. Se estiver a funcionar, volte a fechá-la (garanta mesmo que fica fechada). Caso contrário, procure resolver o problema.

Tarefa 13 *Task 13*

Coloque o *logcat* a correr usando o comando `$ adb logcat`. Caso esteja a observar demasiados registos no *output* do programa, considere aplicar um filtro para a *tag* `ALC`. **Q15.: Como é que se aplicam filtros no logcat via linha de comandos?**

☐ Combinando a opção `-f` com `ALC`.
☐ Combinando a opção `-e` com `ALC`.
☐ Combinando a opção `-s` com `ALC`.
☐ Combinando a opção `-i` com `ALC`.

Certifique-se de que está a conseguir ver as entradas que aplicação faz no registo, executando e terminando a aplicação, e verificando se aparecem no logcat.

Tarefa 14 *Task 14*

Observe os fluxos possíveis de uma aplicação Android™ esquematizados na figura central do URL `http://developer.android.com/reference/android/app/Activity.html` (também incluída na aula teórica 4). **Q16.: Consegue simular um fluxo normal (completo) de execução da aplicação no seu *logcat*?** I.e., o fluxo `onCreate()` ⟶ `onStart()` ⟶ `onResume()` ⟶ `onPause()` ⟶ `onStop()` ⟶ `onDestroy()`

☑ Claro que consigo.
☐ Não consigo...

Use o espaço seguinte para descrever claramente quais os passos que tomou para conseguir simular este fluxo:

1. _____
2. _____
3. _____
4. _____
5. _____

Tarefa 15 *Task 15*

Simule de seguida o fluxo definido por `onCreate()` ⟶ `onStart()` ⟶ `onResume()` ⟶ `onPause()` ⟶ `onStop()` ⟶ `onRestart()` ⟶ `onStart()`

Use o espaço seguinte para descrever claramente quais os passos que tomou para conseguir simular este fluxo:

1. _____
2. _____
3. _____
4. _____
5. _____

Tarefa 16 *Task 16*

A documentação da plataforma Android™ diz que é possível que uma aplicação seja terminada sem passar pelos métodos `onStop()` e `onDestroy()`. Simule no *logcat* o fluxo definido por `onCreate()` ⟶ `onStart()` ⟶ `onResume()` ⟶ `onPause()` ⟶ `onStop()` ⟶ `onCreate()` ⟶ `onStart()` ⟶ `onResume()` ⟶ ...

Use o espaço seguinte para descrever claramente quais os passos que tomou para conseguir simular este fluxo:

1. _____
2. _____
3. _____
4. _____
5. _____

Tarefa 17 *Task 17*

A documentação da plataforma, nomeadamente o diagrama referido em cima também sugere que é possível passar de `onPause()` para `onResume()`. **Q17.: Consegue simular o fluxo definido a seguir no seu *logcat*?** `onCreate()` ⟶ `onStart()` ⟶ `onResume()` ⟶ `onPause()` ⟶ `onResume()` ⟶ ...

☐ Canja de galinha.
☐ Esta é impossível com a aplicação fornecida.
☐ Este está a oferecer resistência, mas penso que vou conseguir.

Use o espaço seguinte para descrever claramente quais os passos que tomou para conseguir simular este fluxo:

1. _____

_____

2. _____

_____

3. _____

_____

4. _____

_____

5. _____

_____

Q18.: Por curiosidade, o que é que acontece quando muda a orientação (e.g., de vertical (*portrait*) para horizontal (*landscape*)) do dispositivo, em termos do ciclo de vida de uma atividade?

- ☐ O facto é que, em termos de ciclo de vida, a atividade parece não ter sofrido qualquer alteração no processo.
- ☑ A atividade parece ter sido terminada e depois reiniciada!
- ☐ A atividade parece ter sido colocada em pausa e depois resumida.
- ☐ A atividade foi simplesmente terminada.
- ☐ A atividade passou a ser radioatividade.