# Tutorial – Programação em Android

**MIEEIC/MIEGSI/MSI/MIETI – Sistemas Ubíquos 2014/2015**

## Pré-requisitos

Antes de começar, vai precisar de instalar as ferramentas indicadas no seu computador:

1. *Eclipse IDE and SDK (http://www.eclipse.org/)*
2. Android Development Tools *(ADT) eclipse* plugin*: (http://developer.android.com/tools/sdk/eclipse-adt.html)*

## Introdução

Neste tutorial vamos desenvolver incrementalmente um aplicação que permite aos utilizadores enviarem mensagens descrevendo o seu estado (*Tweets*) para um serviço *Twitter* disponível na *Cloud* e também receber a sua *timeline* atualizada [1]. Neste desenvolvimento não vamos comunicar com o serviço Twitter.com real, mas sim com um outros serviço que implementa uma API compatível com o serviço Twitter.com e facilita os testes da aplicação. Este serviço, disponível em http://yamba.marakana.com, implementa a Twitter API 1.0.

Este desenvolvimento está dividido em várias partes. Nesta parte vamos criar uma atividade para submeter um *tweet* e enviar esse *tweet* para o serviço Web, desenhar a interface, aceder aos objetos da interface a partir do código Java, definir permissões de acesso à Internet e criar uma *thread* para executar as operações de forma concorrente à *thread* que recebe os eventos de UI do utilizador.

## Criar um projeto

**T1** Crie o projeto "Hello world" (http://developer.android.com/training/basics/firstapp/creating-project.html). Neste projeto é criada uma atividade principal. Esta atividade tem um ecrã principal com um *layout* que contêm um *widget* de texto (TextView).

O projeto é formado por um conjunto de diretorias e ficheiros. Consulte o site http://developer.android.com/training/basics/firstapp/running-app.html para obter informação sobre as diretorias e ficheiros mais importantes. Em particular consulte a diretoria `/src` onde se encontra o código fonte da atividade gerada (`MainActivity.java`). Esta classe é subclasse da classe `ActionBarActivity` e vai permitir ao programador reescrever os métodos responsáveis pela transição de estado da atividade.

O ficheiro `AndroidManifest.xml` descreve as características fundamentais da aplicação e define cada uma das suas componentes.

No ficheiro `activty_main.xml` na diretoria `layout` são descritos os elementos da interface da atividade. Neste caso, temos um `layout` com um elemento do tipo `TextView`. Este elemento refere um recurso de texto que está definido no ficheiro `values/strings.xml`. Os recursos *String* (assim como outros tipos de recursos) são mantidos num ficheiro de configuração à parte dos ficheiros fontes, pois assim torna-se mais fácil encontrar os textos definidos na aplicação para serem alterados, por exemplo.

**T2** Execute a aplicação. As instruções estão definidas em http://developer.android.com/training/basics/firstapp/running-app.html.

## Criar uma Interface

Vamos alterar a interface associada à atividade principal do projeto "Hello World". Esta nova interface vai permitir ao utilizador submeter a sua mensagem de estado (*tweet*) para o serviço central (equivalente ao serviço *twiiter*) e vai conter os seguintes elementos:

- Um botão para enviar uma mensagem de estado (*tweet*)
- Uma área de texto para submeter a mensagem
- E o *layout* que encapsula os dois elementos anteriores.

Uma interface é constituída por um *layout* (*ViewGroup*) principal. Este *layout* pode conter outros *layouts* ou um grupo de *Widgets* (*Views*) de texto, edição de texto ou botão (consultar http://developer.android.com/guide/topics/ui/overview.html) .

Os elementos de uma interface podem ser criados de uma forma declarativa num ficheiro xml ou criados programaticamente. Na forma declarativa, podemos editar o ficheiro xml (criado para uma atividade na diretoria `/res/layout/<nome-atividade>.xml`) ou editar o *layout* graficamente.

A atividade `MainActivity.java` já tem um *layout* associado. Para concluir a concepção deve selecionar o *layout*, selecione o ficheiro `/res/layout/activity_main.xml`. No eclipse pode aceder às vistas xml e gráfica dos *layouts* a partir dos *tabs* ao fundo da janela do eclipse.

**T3** Insira os diferentes elementos da interface:

Adicione primeiro o botão para enviar a alteração de estado do utilizador. Procure um **Button Widget** na secção de **Widgets** da Palete e arraste-o para o canto superior direito do ecrã.

De seguida deve criar uma área de texto para submeter o *tweet*. Vai ser utilizado um **EditText Widget**. Localize um na secção de **Text Fields** da palete (Multiline Text) e arraste-o para abaixo do botão, encostando-o ao lado esquerdo.

Para aumentar a área de texto, selecione-a com o rato e clique com o botão direito do rato. Escolha **Layout Width→Match Parent**. Faça o mesmo para a altura (*Height*)

É agora necessário alterar o texto dos elementos. Relembramos que o sistema Android mantém os dados em ficheiros separados. Desta forma os dados de texto (como nomes de botões, títulos, ou qualquer texto em elementos da interface são definidos num ficheiro chamado `/res/values/`

`strings.xml`. Desta forma é possível disponibilizar várias versões dos recursos *string*, por exemplo para línguas diferentes.

Para este efeito, selecione o botão com o rato e clique com o botão direito do rato. Escolha **Edit Text**. Em vez de alterar a palavra "Button" para outra qualquer mais apropriada, escolha **New String**. No campo **New R.string**. escreva *button_tweet*. Este será o identificador do recurso que define o valor concreto deste texto em particular. No campo **String** escreva o nome que realmente quer que apareça por cima do botão na interface, por exemplo "Tweet". Clique, ok, e depois ok outra vez.

Um dos atributos mais importantes é o ID do elemento da interface, principalmente para os elementos que serão manipulados programaticamente pelas classes Java. Um ID tem o formato @+id/*someName* em que *someName* é qualquer nome que queremos usar para identificar o recurso. A convenção é usar o tipo do recurso, seguido do nome.

Neste sentido, é conveniente nomear os IDs dos elementos que serão acedidos programaticamente: neste caso, será necessário referir o botão e a área de texto a partir das classes Java, daí ser conveniente usar IDs mais significativos. Para alterar os IDs criados por defeito, clique com o botão direito do rato no elemento e selecione "Edit ID". Insira **buttonTweet** e **editStatus** para o botão e área de texto respetivamente.

Consultar o site http://developer.android.com/guide/topics/ui/declaring-layout.html para obter mais informação sobre todos os atributos dos elementos de uma interface.

**T4** Analise o ficheiro xml criado. Pode aceder a esse ficheiro a partir de um *tab* no fundo da janela eclipse. Pode também analisar o ficheiro `Strings.xml` na diretoria `/res/values`.

**T5** Execute novamente a aplicação no emulador. Poderá observar a nova interface.

### *Análise do código da classe MainActivity.java*

Como acontece com quase todos os componentes do modelo de aplicação Android (como atividades, serviços, *broadcast receivers* e *content providers*), começa-se sempre por criar uma subclasse de classes existentes no framework Android e reescreve-se os seus métodos. Neste caso, criamos uma subclasse da classe `ActionBarActivity` e rescrevemos o método `onCreate`(). Este método irá executar duas tarefas maiores que passam por inicializar o elemento botão de modo a responder a *clicks* e fazer a ligação com o serviço Web.

Tipicamente, a primeira ação a ser executada no método onCreate() é carregar a interface para memória e criar os objetos correspondentes. Para cada atributo de um elemento xml será definido um atributo no objeto Jvava. A linah de código responsável por esta tarefa é:

```
setContentView(R.layout.activity_status)
```

Para já vamos ignorar os outros métodos.

**T6** Analise o código do ficheiro `MainActivity.java`.

### *Carregar a interface e os seus elementos na classe MainActivity.java*

Depois de carregar os elementos da interface para memória, será necessário obter as referências para os objetos que vamos necessitar de aceder programaticamente e atribuir a varáveis Java. Para tal, as variáveis são normalmente declaradas como *private* e globais à classe. De seguida é utilizado o método estático `findViewById()` para procurar os objetos pelo seu identificador. Neste caso `R.id.editStatus` e `R.id.buttonTweet`.

**T7** Insira as linhas de código seguintes (a negrito) à classe MainActivity.java.

```
import android.widget.EditText;
import android.widget.Button;
...
private EditText editStatus;
private Button buttonTweet;

@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main););

      editStatus = (EditText) findViewById(R.id.editStatus);
      buttonTweet = (Button) findViewById(R.id.buttonTweet);
}
...
```

### *Atendimento de eventos gerados pelo utilizador na interface*

O atendimento de eventos do utilizador em Android é semelhante ao atendimento de eventos em linguagens que utilizam elementos gráficos (como o Java).

Para que seja possível receber um evento relacionado com o elemento botão (normalmente um *click*), a classe `MainActivity.java` terá de implementar a interface Java `OnClickListener`.
Os elementos da interface são objetos que disponibilizam métodos para definir o código a executar quando uma determinada ação é executado, como por exemplo o *click*. Para isso existe o método `onClick()` que deverá ser programado. Adicionalmente o objeto botão deverá criar um *listner* para os eventos gerados. Neste caso é a atividade `MainActivity.java`. Esta ação é feira através do método `setOnClickListener()` da classe botão.

**T8** Insira a seguinte linha de código no método `onCreate()`:
`buttonTweet.setOnClickListener(this) //` instrui o objeto botão para notificar a atividade `MainActivity` quando recebe um *click*

**T9** Defina o método `onClick()`. Este método é invocado quando é feito um *click* no botão.

```
@Override
      public void onClick(View view) {
      String status = editStatus.getText().toString();
      Log.d(TAG, "onClicked with status: " + status);

      }
```

A ação consiste na leitura do texto inserido no elemento `EditText`. A instrução:

```
Log.d(TAG, "onClicked with status: " + status);
```

Adiciona o texto passado com argumento a uma sistema de *logging* do Android. O argumento TAG é uma *string* que identifica a aplicação. Deve acrescentar o `import android.util.Log;` e a declaração da variável TAG como *private static final*:

```
private static final String TAG = "StatusActivity";
```

Deste modo pode verificar se esta etapa da aplicação aplicação funciona.

**T10** Execute novamente a aplicação (O código completo até ao final desta etapa, está disponível no ficheiro MyFirstApp-T10.java, disponível na página da UC no *moodle* ou *blackboard*)

## Acesso ao Serviço Web

Nesta fase vamos implementar na aplicação o acesso ao serviço *http://yamba.marakana.com*. Como já foi referido, este serviço implementa a Twitter API 1.0. Esta é uma interface de um serviço WebRESTful. Este Web serviço disponibiliza informação tanto em xml como em JSON.
Para aceder ao serviço vamos utilizar uma biblioteca (API) Java Android que contêm uma classe Java que interage com o serviço *http://yamba.marakana.com* que abstrai todos os pormenores da comunicação remota e serialização das invocações remotas[1].

**T11** Descarregue a biblioteca a partir da página da UC no *moodle* ou *blackboard*. Copie (arraste) o ficheiro `yambaclientlib.jar` para diretoria */lib* do projeto Eclipse.[2]

### *Atualização do ficheiro Manifest para autorizar ligação à Internet*

Existem certas operações, como o acesso à Internet, que requerem autorização explícita do utilizador. Essa autorização é expressa no ficheiro `/bin/res/AndroidManifest.xml`.

**T12** Aceda à vista de texto deste ficheiro e adicione o elemento:

```
<uses-permission android:name="android.permission.INTERNET" />
```

### *Processamento Multi-thread em Android*

Por defeito, uma aplicação Android executa numa única *thread.* Esta *thread* é também conhecida pela *thread UI*, pois é *a thread* que executa todos os comandos inseridos pelo utilizador pela UI. Esta *thread* é responsável por desenhar todos os elementos da interface no ecrã assim como processar todos os eventos do utilizador, como toques no ecrã, *clicks* de botões, etc. A execução de chamadas remotas pela Internet na mesma *thread* que executa as tarefas de UI, além de bloquear todas as tarefas de UI, não é permitida nas versões mais recentes do Android.

---

[1] Normalmente os serviços Web globais oferecem estas bibliotecas, sendo no entanto sempre possível usar as bibliotecas standard do Java para fazer o mesmo trabalho.

2 Se pretender analisar o código fonte desta biblioteca, descarregue o ficheiro `YambaClientLib-master.zip` da página da UC no *moodle* ou *blackboard*.

A solução para este problema reside na execução em múltiplas *threads*. Assim as operações mais longas correm numa *thread* independente. Neste caso, a execução de chamadas remotas vai então executar numa *thread* independente.

O mecanismo mais vulgar de criação de *threads* em Java não é o mais adequado para as aplicações Android, pois há algumas restrições na interação com a UI de um aplicação Android: esta só pode ser alterada pela *thread* que a criou. Esta restrição dificulta a comunicação entre a *thread* que executa uma chamada remota e o utilizador.

O sistema operativo Android disponibiliza um mecanismo específico para lidar com tarefas de execução longa que necessitam de comunicar com a UI: a classe **AsyncTask**.

Para recorrermos a este mecanismo, será necessário criar na MainActivity classe uma subclasse da classe **AsyncTask** e implementar os métodos `doInBackground()`, `onProgressUpdate()`, e `onPostExecute()`. Basicamente, é necessário programar as tarefas a executar em *background* na nova *thread*, as tarefas a executar quando existe algum progresso e as tarefas a executar quando a *thread* termina.

*Além de implementar os métodos referidos, será necessário acrescentar no método `onClick()` a instrução:*

```
new PostTask().execute(status);
```

Esta instrução cria uma instância de uma *thread* do tipo **PostTask** e invoca o método `execute()` na nova instância, passando-lhe a mensagem a submeter ao serviço Web. Este parâmetro será passado ao método `doInBackground()` que define as tarefas a executar pela nova thread. Esta *string* é convertida posteriormente num *array* de *strings*.

O ficheiro `MainActivity-ParteI-final.java` contém a implementação final da classe `MainActivity.java`. Está disponível na página da UC no *moodle* e *blackboard*.

**T13** Analise este ficheiro.
A classe `PostTask` é uma *inner class* da classe `MainActivity`, É uma subclasse da classe `AsyncTask`. `usado o mecanismo de *Java generics* para descrever os parâmetros dos três métodos da classe.

O método `doInBackground()` especifica o código a ser executado quando a *thread* é criada:

- A instrução `new YambaClient("student", "password")` cria uma instância da classe que serve de interface ao service Web[3].
- A instrução `yambaCloud.postStatus(params[0])` invoca o método da biblioteca que serializa a invocação ao serviço Web. O parâmetro corresponde ao texto submetido pelo utilizador que foi passado via o método `execute()`.

[3] Pode consultar a documentação da biblioteca *yambaclientlib* disponível na diretoria das fontes desta biblioteca.

O método `onPostExecute()` é invocado quando a tarefa executada na *thread* termina. Neste caso corresponde a receber os dados do serviço Web. Este é uma espécie de *callback* à `MainActivity` que avisa que a *thread* terminou e permite avisar o utilizador a partir da UI:

- Neste caso particular recorremos à funcionalidade **Toast** das interfaces Android para fazer o display de uma mensagem rápida no ecrã:

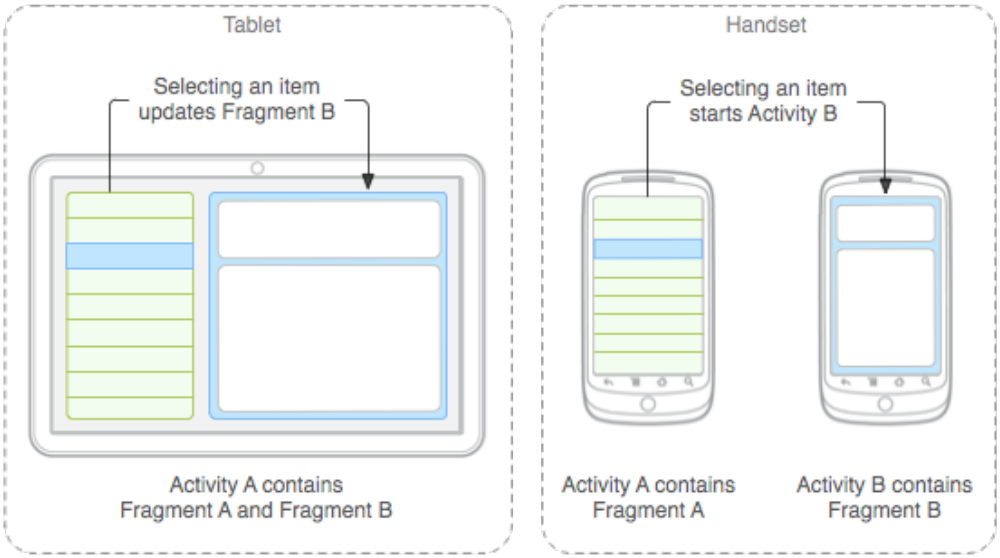
  ```
  Toast.makeText(MainActivity.this,result,Toast.LENGTH_LONG).show()
  ```

- O argumento deste método é o valor retornado pelo método `doInBackground()`, neste caso uma *String*. A referência a `MainActivity.this` representa o Contexto em que a aplicação está.

**T14** Complete o código da **MainActivity** no seu projeto e execute a aplicação.

## Criar um Fragmento

Os Fragmentos forma introduzidos na API Android a partir da versão 11. Um Fragmento representa uma porção da UI numa atividade. Assim é possível combinar múltiplos fragmentos numa única atividade de modo a construir atividades com múltiplos painéis (equivalente às *frames* numa página Web) e reutilizar fragmentos em múltiplas atividades. Um fragmento é uma secção modular de uma atividade, com o seu próprio ciclo de vida, os seus próprios eventos de entrada[4]. Esta funcionalidade veio ao encontro das novas potencialidades oferecidas por dispositivos tipo *tablet* que oferecem um ecrã maior. Na figura (obtida de http://developer.android.com/guide/components/fragments.html) pode verificar alternativas de concepção de UIs para dois tipos de dispositivos e a sua relação com os fragmentos e atividades.




[4] Consultar o site http://developer.android.com/guide/components/fragments.html para obter mais informação sobre Fragmentos e o seu ciclo de vida.

Para converter uma atividade num fragmento (uma vez que já partimos de uma atividade), é necessário mover a lógica da atividade *MainActivity* para uma nova classe, subclasse da classe *Fragment*.

**T15** Crie uma classe Java em *File->New->Class*. Inserir *pt.uminho.su.myfirstapp* no campo **Package**, *MainFragment* no campo **Name**. Esta classe deverá ser subclasse da classe **Fragment** e implementar a interface `OnClickListener`. (Deve importar as classes `android.app.Fragment`, `android.view.LayoutInflater`, e `android.view.ViewGroup`).

A classe **MainFragment** deverá reescrever o método `onCreateView()` para criar uma *view* que executa todas as tarefas executadas pelo método `onCreate()` na atividade original **MainActivity**. Deste modo, deve implementar também o método `OnClickListener()` e definir a classe privada `PostTask` também definida previamente na atividade original **MainActivity**.

**T16** Copie o código do ficheiro `MainFragment.java` disponibilizado na página da UC (*moodle* ou *blackboard*) e analise o código.

A seguinte linha de código no método `onCreateView()`:

```
View view = inflater.inflate(R.layout.main_activity, container, false);
```

usa o método `inflate()` da classe `LayoutInflater()` para carregar o *layout* definido em `main_activity.xml` para o fragmento. O *layout* é o mesmo definido anteriormente no ficheiro `main_activity.xml`[5].

Repare que na linha de código, o argumento do método `Toast.makeText()` tem de ser uma referência a uma atividade. Deve ser referida a atividade que contém o fragmento, obtida através da invocação `MainFragment.this.getActivity()`.

```
Toast.makeText(MainFragment.this.getActivity(),result,
Toast.LENGTH_LONG).show();
```

**T17** Agora que o fragmento está definido, é necessário alterar a atividade **MainActivity** de modo a integrar o novo fragmento. A única tarefa a executar nesta atividade é carregar um novo *layout*. Este novo *layout* (a definir já de seguida) faz a ligação ao fragmento.

A nova versão do código da **MainActivity** está definido no ficheiro `MainActivity.java` disponibilizado na página da UC no *moodle* ou *blackboard*.

**T18** Crie um novo ficheiro `new_main_activity.xml` na diretoria `res/layout`. *Neste novo layout, a ser carregado pela atividade* **MainActivity**, deverá definir um elemento fragmento que deverá apontar para a classe `pt.uminho.su.myfirstapp.MainFragment`. O código para o novo *layout* da atividade

[5] Poderá renomear o ficheiro `activity_main.xml` para `fragment_main.xml` para maior clareza.

**MainActivity** está definido no ficheiro `new_main_activity.xml` disponibilizado na página da UC no *moodle* ou *blackboard*.

**T19** Complete o código da atividade **MainActivity,** do fragmento **MainFragment** e *layout* **new_main_activity** no seu projeto e execute a aplicação.

## Adicionar a *Action Bar*

A *Action Bar* em Android é um componente que disponibiliza menus standards às aplicações. Para criar uma *action bar* para uma aplicação é necessário (1) Adicionar o método `onCreateOptionsMenu()` à atividade que vai ter o menu; (2) criar os recursos em `/res/menu` associados à atividade que vai ter o menu; e (3) disponibilizar o atendimento de eventos no método `onOptionsItemSelected()` da atividade.

### *Criar a atividade para a Action Bar*

Já temos uma atividade (para fazer o *tweet*), mas vamos criar uma atividade que será o ponto de entrada na aplicação. Esta atividade vai ter um *action bar* e vai permitir a seleção das opções do menu.

**T20** Crie uma atividade em *File->New-> Other->Android*. Selecione *Blank Activity*. Insira o nome da atividade, por exemplo **EntryActivity**, Aceite as outras configurações por defeito. O eclipse regista esta atividade no ficheiro *AndroidManifest.xml,* mas será necessário declarar esta atividade como a o ponto de entrada da aplicação. Para isso, será necessário mover o código xml seguinte da atividade **MainActivity** para a atividade **EntryActivity** no ficheiro *AndroidManifest.xml*.

```
<intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Quando é criada a nova atividade, é criado o ficheiro `activity_entry.xml` que define o *layout* por defeito para esta atividade. Este *layout* apenas inclui um elemento `TextView` com a *string* "Hello world". Altere o texto para `@string/instructions` e define `instructions` no ficheiro `strings.xml` como "Choose action bar" por exemplo.

A versão final do ficheiro `activity_entry.xml` e `strings.xml` estão disponíveis na página da UC, no *moodle* ou *blackboard*.

### *Criar o Recurso Menu*

Assim que cria a atividade, é criado também em `res/menu` o ficheiro `entry.xml`. Neste ficheiro são definidos os *items* que constituem a *action bar*. Pode definir estes *items* em modo gráfico ou editando manualmente o ficheiro `entry.xml`. Cada item é caracterizado por um conjunto vasto de atributos, mas os mais importantes são o *Id* (identificador para o recurso item), *title* (título que vai ser visualizado para o item) e, *icon* (*icon*, pista visual) para ser apresentada com o texto.

**T21** Defina o *item* **action_tweet**. O ficheiro `entry.xml` está disponível na página da UC no *moodle* ou *blackboard*. Note que a *string tweet* terão de ser definidas no ficheiro `strings.xml`.

### *Carregar o menu para o programa Java*

A *action bar* é carregada pela atividade quando o utilizador pressiona o botão Menu do seu dispositivo Android. A primeira vez que o botão é pressionado, o sistema operativo invoca o método `onCreateOptionsMenu()` da atividade principal para carregar o menu para o programa Java a partir do recurso `menu.xml`: o código xml é lido e é criado um objeto Java para cada elemento.

É necessário alterar a atividade **EntryActivity** para carregar a *action bar*: essa alteração é feita no método `onCreateOptionsMenu()`:

```
@Override
	public boolean onCreateOptionsMenu(Menu menu) {
	// Inflate the menu; this adds items to the action bar if it is present.
		getMenuInflater().inflate(R.menu.entry, menu);
		return true;
	}
```

Este método é invocado na primeira vez em que o menu necessita de ser apresentado, como por exemplo quando a atividade é apresentada no ecrã. Esta ação é feita através do método `inflate` do objeto `MenuInflater`, obtido através da invocação do método `getMenuInflater()`.

### *Atendimento de eventos da action bar.*

Sempre que o utilizador seleciona um item da *action bar,* o método `onOptionsItemSelected()` é invocado para processar esse clique.

```
@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		switch (item.getItemId()) {
		case R.id.action_tweet:
			startActivity(new Intent(this, MainActivity.class));
			return true;
		default:
			return false;
		}
	}
```

Para já definimos apenas uma ação para atender o evento de input no item **action_tweet** (definido no ficheiro `entry.xml`). Neste caso é passada uma mensagem entre esta atividade e a **MainActivity** para que esta seja criada e inicie.

**T22** O ficheiro `EntryActiivty.java` está disponível na página da UC. Execute a aplicação.

## Adicionar um serviço

Vamos estender a aplicação com um serviço que corre em background (sem UI) responsável por obter a *timeline* do utilizador (*student*, *password*) no serviço Yamba.

Ao contrário de um atividade, um serviço não tem associada uma UI. Serviços são processos que executam independentemente das atividades ou outros componentes Os serviços podem sobreviver ao ciclo de vida de uma atividade.

O estado de um serviço é controlado normalmente por *Intents*. Sempre que uma atividade precisa de um serviço, esta vai invocar o serviço através de um *Intent*. Esta situação é conhecida como "iniciar um serviço". Um serviço em execução poderá receber uma mensagem "iniciar" repetidamente e em alturas imprevistas. É também possível parar um serviço, ou seja destruí-lo.

Um serviço pode ser do tipo *Bound* ou *Unbound*. Um serviço do tipo *Bound* é o servidor numa relação cliente/servidor. Um serviço deste tipo permite que outros componentes (por exemplo, atividades) se possam associar ao serviço, enviar pedidos e receber respostas. O tempo de vida de um serviço do tipo *Bound* está tipicamente associado ao tempo de vida de um outro componente e não executa indefinidamente em *background*. Neste tutorial, iremos focar os serviços tipo *unbound*, em que o ciclo de vida é independente do ciclo de vida de outros componentes. O ciclo de vida deste tipo de serviços tem apenas dois estados: *started* e *stopped*.[6]

### *Criar o serviço RefreshService*

Os passos necessários para criar um serviço são:

1. Criar a classe Java que implementa o serviço.
2. Registar o serviço no ficheiro AndroidManifest.xml.
3. Iniciar o serviço.

**T23** Crie uma classe Java em *File->New->Class*. Inserir *pt.uminho.su.myfirstapp* no campo **Package**, *RefreshService* no campo **Name**. Esta classe deverá ser subclasse da classe `android.app.IntentService` porque queremos que o atendimento a um ***Intent*** seja executado numa *thread* independente e não na *thread* associada à UI. Um serviço tipo ***IntentService*** é semelhante a um serviço regular à parte de duas questões: este serviço deverá reescrever o método `onHandleIntent()` que será invocado numa *thread* independente; e quando termina, o serviço será parado.

Será necessário reescrever alguns métodos do ciclo de vida de um serviço. O método `onBind()` não é usado nos serviços do tipo *unbound*. Deverá assim retornar `null`. Os métodos que vamos reescrever são, além do método `onHandleIntent()` já referido, o método `onCreate()` e o método `onDestroy()`:

[6] Consultar http://developer.android.com/guide/components/services.html para obter mais informação sobre Serviços Android.

- `onCreate()` é invocado quando o serviço é iniciado pela primeira vez. Não é invocado quando o serviço é iniciado das vezes seguintes. Serve para executar operações que apenas necessitam de ser executadas uma vez durante o ciclo de vida do serviço.
- `onDestroy()` é invocado imediatamente antes do serviço ser destruído quando é gerado um pedido para parar o serviço.

**T24** Para reescrever os métodos, pode utilizar a ferramenta do Eclipse *Source -> Override/Implement Methods* e selecione os métodos a reescrever.

A classe **IntentService** requer um construtor por defeito. Deve definir o construtor e invocar `super()` com um argumento tipo *String*, por exemplo o nome do serviço, ou uma avariável *static* tipo *String* como uma *tag* para escrever no *LogCat*, como exemplificado em baixo:

```
......
public class RefreshService extends IntentService {
	static final String TAG = "RefreshService";

	public RefreshService() {
		super(TAG);
	}
.....
}
```

### *Definir a lógica do serviço*

A lógica deste serviço vai consistir em fazer uma ligação ao serviço *http://yamba.marakana.com*, obter a *timeline* do utilizador (*student*, *password*) e disponibilizar esses dados no *LogCat* da aplicação. Vamos utilizar novamente a biblioteca **yambaclientlib.jar** que expõe os interfaces do serviço Yamba via a classe **YambaClien**t. Vamos utilizar o método desta classe `getTimeline()` que retorna os 20 *posts* mais recentes na *timeline* do utilizador. Esta lógica é implementada no método `onHandleIntent()`. A implementação é a seguinte:

```
@Override
protected void onHandleIntent(Intent intent) {

	YambaClient cloud = new YambaClient("student", "password");
	try {
		List<Status> timeline = cloud.getTimeline(20);
		for (Status status : timeline) {
			Log.d(TAG, String.format("%s: %s", status.getUser(),
		status.getMessage()));
		}
	} catch (YambaClientException e) {
		Log.e(TAG, "Failed to fetch the timeline", e);
		e.printStackTrace();
	}
return;
}
```

Neste momento, o serviço *Yamba* é contactado e os 20 *posts* mais recentes são escritos no *LogCat*. Deve também rescrever os métodos `onCreate()` e `onDestroy()`. Neste caso devem simplesmente invocar o construtor da *super* classe (a versão completa do ficheiro `RefreshService.java` está disponível na página da UC).

Para podermos testar o novo serviço é necessário atualizar o ficheiro **Manifest** de modo a comtemplar este novo componente na aplicação e também é necessário definir um meio de o iniciar e terminar.

### *Atualizar o ficheiro Manifest*

**T25** Edite o ficheiro *AndroidManifest.xml* da aplicação e adicione a seguinte *tag* `<service>` no elemento `<application>`.

```
...
        <application android:icon="@drawable/icon" android:label="@string/app_name">
        ...
                <service android:name=".RefreshService" />
        ...
        </application>
...
```

A versão completa do ficheiro *AndroidManifest.xml* está disponível na página da UC.

### *Iniciar o serviço através da Action Bar*

Uma forma imediata de iniciar o serviço será através do menu da *Action Bar*. Vamos adicionar um botão ao menu que criámos anteriormente.

**T26** Adicione o seguinte botão no ficheiro *entry.xml* (associado à atividade de entrada **EntryActivity**). Para isso, edite o ficheiro *entry.xml* e adicione as seguintes linhas:

```
<item android:title="@string/titleRefresh"
      android:id="@+id/itemServiceStart"
      android:icon="@android:drawable/ic_menu_rotate"></item>
```

Deve definir também a *string* `titleRefresh` no ficheiro *strings.xml*.

A versão completa dos ficheiros *entry.xml* e *strings.xml* estão disponíveis na página da UC.

**T27** Agora será necessário definir o atendimento a este *item* do menu. Para atender uma nova opção do menu, é necessário atualizar o método `onOptionsItemSelected()` da atividade **EntryActivity**. Edite o ficheiro `EntryActivity.java` e adicione o atendimento ao item `itemServiceStart`:

```
...
      case R.id.itemServiceStart:
            startService(new Intent(this, RefreshService.class));
            break;
...
```

É criado um *Intent* para iniciar o serviço **RefreshService**. Se o serviço ainda não existe, o *runtime* invoca o método `onCreate()`. De seguida, o método `onHandleIntent()` é invocado independente do serviço estar a ser criado pela primeira vez ou já estar a executar.

**T28** As versões completas dos ficheiros `AndroidManifest.xml`, `entry.xml`, `strings.xml`, `RefreshService.java` e `Entryactivity.java` estão disponíveis na página da UC. Execute a aplicação.

A opção "Refresh" só está disponível a partir da atividade principal, a **EntryActivity**. Para testar a sua aplicação deverá fazer "Refresh" antes de escolher qualquer outra opção.

Para analisar o resultado, consulte o LogCat, com o filtro definido pela Tag "RefreshService".

## Referências

*[1] Gargenta, Marko and Nakamura, Masumi, Learning Android. " O'Reilly Media, Inc.",* second edition*, 2014*