Laboratório #03

Multithreading e armazenamento no Android

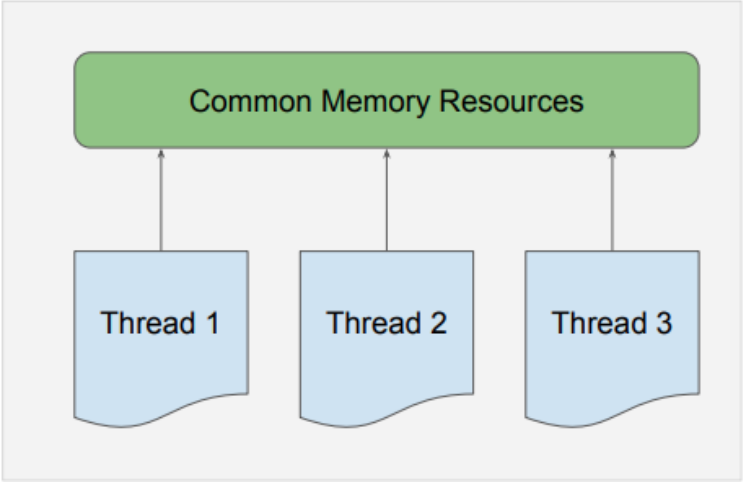
Objectivos

- Introdução ao *multithreading* no Android: *UI thread*, *worker threads*, e *AsyncTasks*.
- Visão geral dos principais mecanismos do Android para armazenamento local de dados: ficheiros e bases de dados.

Introdução a multithreading no Android

- **Processos, Threads e Componentes**
  - Quando uma aplicação é iniciada, o Android inicia um novo processo do Linux
    - O processo executa uma instância de máquina virtual Android Runtime (ART)
  - ART inicia uma única *thread* de execução chamada thread principal
  - A *thread* principal lida com todos os componentes da aplicação
  - Se um componente da aplicação for iniciado e existir um processo para esta aplicação, então:
    - O componente é iniciado dentro deste processo
    - Utiliza a mesma *thread* de execução
  - Os componentes podem ser organizados para serem executados em processos separados
  - É possível criar *threads* adicionais para qualquer processo
- **Multithreading**
  - Uma *thread* é uma unidade concorrente de execução
  - Cada *thread* tem sua própria pilha de chamadas
  - A pilha de chamada é utilizada na chamada de método, passagem de parâmetro e armazenamento para as variáveis locais do método chamado (revise **Arquitectura de Computadores**)

joao.costa@isptec.co.ao



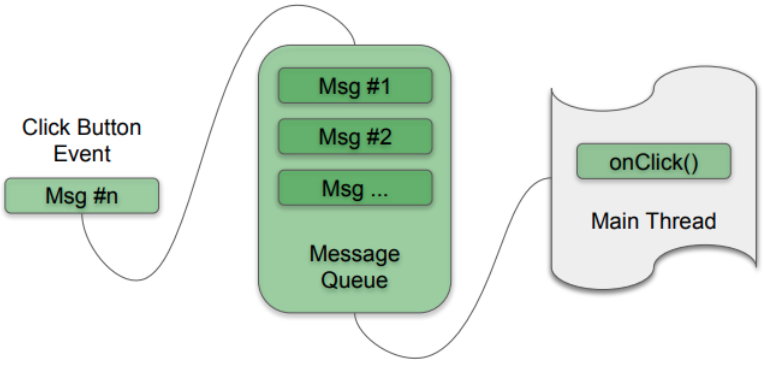
- **Criação de *thread***
  - Implementação do código da *thread*:

```
Runnable tLogic = new Runnable() {
     public void run() {
          // do some work
     }
}
```

  - Criação da *thread* e inicialização

```
Thread t = new Thread(tLogic, "My thread");
t.start();
```

- ***Thread* principal (também conhecido como *UI Thread*)**
  - Thread muito importante: lida com a interface do utilizador
    - É responsável por despachar eventos para os *widgets* de interface de utilizador apropriados, incluindo eventos de desenho
  - O sistema não cria uma *thread* separada para cada instância de um componente
    - Todos os componentes executados no mesmo processo são instanciados em *UI thread*
  - As chamadas do sistema para cada componente são despachadas nesta *thread*
    - Portanto, os métodos que respondem aos retornos de chamada do sistema (por exemplo, onKeyDown()) sempre são executados na UI do processo
- **Manipulação de eventos de IU**



- **Cuidado com operações longas na *UI Thread*!**

```
public void onClick(View v) {
     Bitmap b = loadImageFromNetwork(...); // long op
     mImageView.setImageBitmap(b);         // update UI
}
```

    - Operações de longa duração bloquearão toda a interface do utilizador
        - Nenhum evento pode ser despachado: a aplicação parece estar travada
        - Se bloqueado por muito tempo, a caixa de diálogo "aplicação não está a responder" é apresentada
    - Observação: o Android 3.0 (API de nível 11) e superior exige que execute operações de rede em uma *thread* diferente do UI thread principal; caso contrário, um `NetworkOnMainThreadException` será lançada.

- **1ª tentativa: Long Ops no Worker Thread**

```
public void onClick(View v) {
     new Thread(new Runnable() {
          public void run() {
               Bitmap b = loadImageFromNetwork(...); // do long op
               mImageView.setImageBitmap(b);         // update UI
          }
     }).start();                                     // execute thread
}
```

    - Bom: não bloqueia a *UI thread*
    - Problema: *Android UI toolkit* não é *thread-safe* e deve ser sempre manipulado na *UI thread*
        - Neste código, o `ImageView` é manipulado em uma *worker thread*
        - Gera `CalledFromWrongThreadException`

- **Acessar *UI Thread* a partir da *Worker Threads***
    - Várias maneiras (mas o código começa a ficar complicado)
        - `Activity.runOnUiThread(Runnable)`
        - `View.post(Runnable)`
        - `View.postDelayed(Runnable, long)`
        - `Handler`

```
public void onClick(View v) {
     new Thread(new Runnable() {
          public void run() {
               final Bitmap b = loadImageFromNetwork(...);
               mImageView.post(new Runnable() {      // update UI
                    public void run() {
                         mImageView.setImageBitmap(b);
                    }
               });
          }
     }).start();
}
```

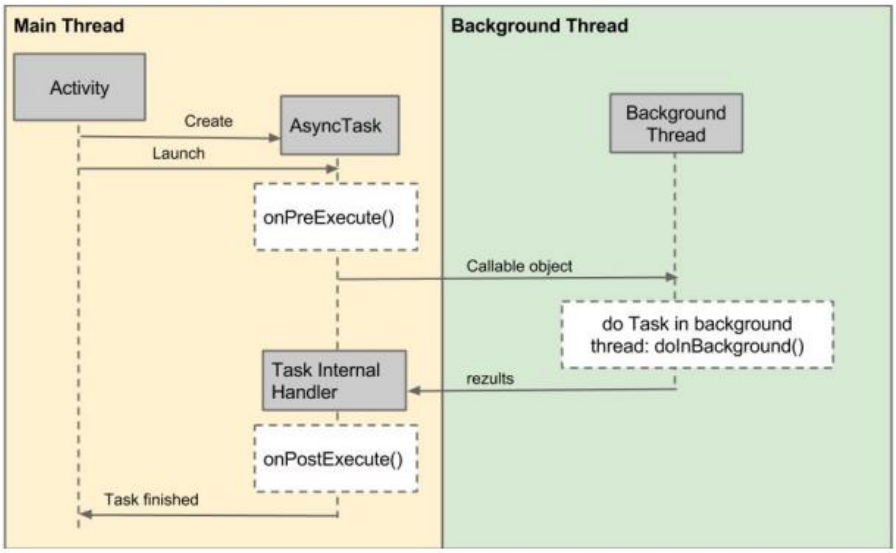
- **Tarefas de execução longa com `AsyncTask`**
    - Simplifique a criação de tarefas de execução longa que precisam se comunicar com a UI

```
public void onClick(View v) {
     new DownloadImageTask().execute("http://...");
}

private class DownloadImageTask extends AsyncTask {
     protected Bitmap doInBackground(String ...urls) {
          return loadImageFromNetwork(urls[0]);
     }

     protected void onPostExecute(Bitmap result) {
          mImageView.setImageBitmap(result);
     }
}
```

- Fluxo do `AsyncTask`



- `AsyncTask` deve ser utilizado subclassificando-o
- Deve ser criado na *UI thread* e pode ser executado apenas uma vez (precisa ser instanciado toda vez que quiser usá-lo, por exemplo, dentro de um manipulador de cliques)
- Visão geral:
  - Podemos especificar o tipo, usando genéricos, dos parâmetros, os valores de progresso e o valor final da tarefa
  - `doInBackground()` é executado automaticamente em uma *worker thread*.
  - `onPreExecute()`, `onPostExecute()` e `onProgressUpdate()` são invocados na *UI thread*
  - O valor retornado por `doInBackground()` é enviado para `onPostExecute()`
  - Podemos chamar `publishProgress()` a qualquer momento em `doInBackground()` para executar `onProgressUpdate()` na *UI thread*
  - Podemos cancelar a tarefa a qualquer momento, de qualquer *thread*

Exercício I – Threads e Concorrência

O objectivo deste exercício é apresentar o comportamento da *UI thread*, *worker threads*, e *AsyncTasks*.

1. ***Threading* básico:**

    a. Crie uma aplicação Android com uma única actividade e uma *thread* de registo de tempo em execução em segundo plano. A *worker thread* mantém um contador interno que é incrementado a cada segundo. Para cada iteração, o valor do contador deve ser impresso no console de depuração do `LogCat`. Adicione dois botões à actividade principal: "Start" e "Stop". O botão "Start" deve zerar o contador e iniciar a contagem. O botão "Stop" deve parar a contagem. Pontas:

        i. Utilize `Log.d()` para imprimir as mensagens de tique-taque. Para permitir que o `LogCat` filtre mensagens com base no nome do pacote da aplicação, certifique-se de definir a opção `Tools > Android > Enable ADB Integration`.
        ii. Utilize `Thread.sleep()` para bloquear a *work thread* por um tempo programável.
        iii. Para interromper a *worker thread* a partir da *thread* principal, invoque o método *interrupt()* do *worker thread* e retorne do ciclo do *worker thread* se ocorrer uma interrupção.

    b. Inspecione as *threads* de sua aplicação usando o depurador. Execute a aplicação no emulador, pressione o botão "Start" e abra o *Android Device Monitor (ADM)*. No painel "Devices" do ADM (no lado esquerdo), selecione o ID da aplicação e clique no ícone "Update threads" na parte superior do painel. Identifique a *thread* principal e a *worker thread* em execução. Em seguida, pressione o botão "Stop". O que aconteceu com as *threads*?

2. **Passagem de mensagem entre threads.**

    a. Importe o projecto `SimpleImageDownload.zip` para o *Android Studio*. Compile-o, teste-o no emulador e estude seu código. O que esta aplicação faz?
    b. Modifique esta aplicação para que as mensagens de status actualmente impressas no console `LogCat` sejam escritas na UI (abaixo do botão "Download File"). Use um manipulador para permitir que a *worker thread* se comunique com a *UI thread*. Siga as etapas descritas a seguir e explique o que este código faz:
        i. Crie um objecto `Handler` e associe-o a *UI thread*. Para isso, na actividade principal, adicione a seguinte linha:

```
private Handler handler = new Handler(this);
```

        ii. Implemente a interface `Handler.Callback` na `Activity`. Primeiro, modifique a assinatura da actividade com o qualificador "`implements Handler.Callback`" e adicione a seguinte implementação de callback à actividade:

```
public boolean handleMessage(Message msg) {
    String text = msg.getData().getString("status");
    TextView statusText = (TextView) findViewById(R.id.status);
    statusText.setText(text);
    return true;
}
```

iii. Da *thread* de *download*, utilize o objeto `handler` para enviar uma mensagem contendo o novo texto de *status* para a *UI thread*. Mais concretamente, substitua as chamadas para `Log.i` por chamadas para `sendMessage` e implemente o método `sendMessage`:

```
private void sendMessage(String what) {
    Bundle bundle = new Bundle();
    bundle.putString("status", what);
    Message message = new Message();
    message.setData(bundle);
    handler.sendMessage(message);
}
```

3. Tarefas assíncronas.

Conclua o projecto `SimpleImageDownload` para baixar a imagem utilizando um `AsyncTask` e mostre a imagem baixada em um *widget* `ImageView`. Para mostrar a imagem baixada no `ImageView`, utilize o método `setImageBitmap()`. Crie um `AsyncTask` conforme mostrado no manual de referência (https://developer.android.com/reference/android/os/AsyncTask) e substitua o método `onPostExecute()` para actualizar a visualização da imagem. Dica: Use o seguinte esqueleto para sua `AsyncTask`:

```
public class DownloadTask extends AsyncTask {

    public DownloadTask(ImageView imageView, TextView statusText) {
      ...
    }

    @Override
    protected void onPreExecute() {
      ...
    }

    @Override
    protected Bitmap doInBackground(String... inputUrls) {
      ...
    }

    @Override
    protected void onPostExecute(Bitmap result) {
      ...
    }
}
```

4. Ciclos de mensagem personalizados

Saiba mais sobre filas de mensagens e comunicação entre threads estudando uma implementação simples do problema clássico de produtores-consumidores. Importe o projeto `ProducerConsumerWithLooper.zip` para o Android Studio. Compile-o, execute-o e entenda seu código. Interprete a saída do programa no console `LogCat`. No código-fonte, qual é o papel da variável do `handler`? O que é um `Looper`?

Exercício II – Armazenamento de dados locais

1. Leitura e escrita de ficheiros

Faça *download* e importe o projecto `FileExplorer.zip` para o *Android Studio*. Compile o projecto e execute-o no emulador. Esta aplicação lhe ajudará a entender como ler e escrever ficheiros no armazenamento local interno e externo.

a. Inicie a aplicação e pressione o botão "Use Internal Storage". Escreva algum texto e clique em "Write". Em seguida, toque no botão "Read". Para garantir que o texto seja armazenado persistentemente, encerre e reinicie a aplicação. Em seguida, na actividade principal, selecione a mesma opção anterior e toque no botão "Read". Deve ver o seu texto. Explique como a aplicação está a executar as operações de leitura e escrita de ficheiros, estudando o código-fonte da aplicação. Qual é o nome do ficheiro de dados acessado pela aplicação?

b. Localize o ficheiro de dados no sistema de ficheiros do emulador utilizando a ferramenta `adb`. Proceda da seguinte forma:

   i. Abra um console e adicione o caminho da ferramenta `adb` à variável de ambiente `PATH`. O caminho para o `adb` é `<sdk>/platform-tools`. Para descobrir a localização do directório `<sdk>`, selecione "`File > Project Structure`" no Android Studio.
   ii. Obtenha o nome do seu emulador executando o comando: `adb devices`.
   iii. Abra um shell para o emulador. Supondo que o nome do emulador seja "`emulator-5554`", execute o comando: `adb -s emulator-5554 shell`.
   iv. Agora está logado no emulador. Pode executar os comandos de um console típico do Linux. Localize o ficheiro de dados da aplicação no caminho: `data/data/<package>/ficheiros`. Use o nome do pacote da aplicação. Verifique se o conteúdo do ficheiro de dados corresponde ao seu texto de entrada.

c. Repita essas duas perguntas para armazenamento externo. Na actividade principal, selecione a opção "Use External Storage". Observe que, se estiver a usar um emulador para este exemplo, precisará certificar-se de que criou um cartão SD para a instância

com a qual está a trabalhar. Use o adb para localizar o ficheiro no cartão SD. O caminho é `/sdcard/Android/data/<package>/ficheiros`.

d. Em nossas actividades de exemplo de manipulação de ficheiros, pode notar um **problema sutil em potencial**. Estamos a realizar operações de E/S a partir do thread principal (UI thread). Isso quase nunca é uma boa ideia. Ler e escrever dados de e para o sistema de ficheiros, interno ou externo, pode bloquear a UI thread. Corrija esse problema executando E/S a partir de um Thread ou AsyncTask.

2. **Ficheiros de preferências partilhados e bases de dados SQLite**

   Investigue dois mecanismos adicionais de armazenamento do Android – ficheiros de preferências partilhados e bases de dados SQLite – lendo " https://developer.android.com/training/data-storage#pref " e " https://developer.android.com/training/data-storage " da documentação oficial do Android.

Links úteis adicionais

- https://developer.android.com/guide/components/processes-and-threads.html
- http://mcatr.blogspot.pt/2013/06/android-thread-model.html
- https://developer.android.com/guide/index.html#threading