Background

https://github.com/androidannotations/androidannotations/wiki/WorkingWithThreads#background

Permite iniciar el método anotado en segundo plano sin la necesidad de realizar el desarrollo de AsyncTasks. Es muy importante saber que estos métodos (los métodos anotados `@Background`) no pueden interactuar con objetos de UI, si se hace acabará generando una excepción (como veremos próximamente). Un ejemplo de método anotado con `@ Background`:

```
@Background
void someBackgroundWork(String aParam, long anotherParam) {

}
```

Hemos comentado que la anotación `@Background` no puede interactuar con objetos de UI, vamos a ver que pasa si interactuamos con el debbuger y con un objeto de UI.

Test @Background: Logs

Crea un nuevo proyecto y añade las siguientes líneas de código:

```
@EActivity(R.layout.activity_main)
public class MainActivity extends AppCompatActivity {

    private final String INIT_COMMENT = "**************************

    @AfterViews
    void initAfterViews() {
        // First explanation
    final String OPERATION = "hello";
    System.out.println(INIT_COMMENT + "Launching " + OPERATION + " o
    addOperationBackground("hello", 42);
    System.out.println(INIT_COMMENT + "End!");
    }

    @Background
    void addOperationBackground(String aParam, long init) {
    long result = init;

    for (int c=0; c<15; c++) {
        try {
            sleep(500);
            result = result + 2;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
        }

        System.out.println("addOperationBackground: " +INIT_COMMENT + aP
        Log.v("addOperationBackground - DETAIL", INIT_COMMENT + aParam +
        Log.i("addOperationBackground - INFO", INIT_COMMENT + aParam + "
        Log.e("addOperationBackground - ERROR", INIT_COMMENT + aParam +
        }

        private void sleep(int sleepTime) throws InterruptedException {
            Thread.sleep(sleepTime);
        }
    }
```

Como verás después de 7500ms verás:

```
I/System.out: ************************************************> Lau
I/System.out: ************************************************> End

[Debugger info]

I/System.out: addOperationBackground: ********************************
V/addOperationBackground - DETAIL: ***********************************
I/addOperationBackground - INFO: *************************************
E/addOperationBackground - ERROR: ************************************
```

Por lo tanto podremos añadir `prints` y `loggers` en métodos anotados con `@Background`.

Test @Background: Actualizando TextView

Ves a `activity_main.xml` y añade un id al TextView, por ejemplo `tv_example`. Posteriormente añade crea la siguiente variable global a la clase:

```
@ViewById
TextView tv_example;
```

Y actualiza el método (por ejemplo `addOperationBackground`) que permite actualizar el texto del TextView que hemos creado anteriormente (`tv_example`):

```
        tv_example.setText(aParam + " is: " + result);
```

Esta línea lanzará el siguiente error:

```
E/AndroidRuntime: FATAL EXCEPTION: pool-1-thread-1
    Process: com.example.openwebinar.concurrence, PID: 30086
    android.view.ViewRootImpl$CalledFromWrongThreadException: Only t
    at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:7753)
    at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:122
```

```
at android.view.View.requestLayout(View.java:23093)
```

Opciónes del @Background

A continuación vamos a ver las distintas opciones que tiene la anotación `@Background`

Background - Id

https://github.com/androidannotations/androidannotations/wiki/WorkingWithThreads#id

Es una buena práctica identificar (poniendo un id) el método de background. Uno de los beneficios que otorga este id es poder matar las tareas en background. Vamos a escribir estos nuevos métodos:

```
@Background(id="cancellable_task")
void cancellableTaskBackground() {
Log.i("cancellableTaskBackground", INIT_COMMENT + "I go to die i

try {
    sleep(6000);
    Log.e("cancellableTaskBackground", INIT_COMMENT + "You won't
} catch (InterruptedException e) {
    e.printStackTrace();
    Log.i("cancellableTaskBackground", INIT_COMMENT + "'cancella
}
}

@Background
void iAmTheKillerBackground() {
try {
    sleep(3000);
} catch (InterruptedException e) {
    e.printStackTrace();
}

// http://javadox.com/org.androidannotations/androidannotations-
BackgroundExecutor.cancelAll("cancellable_task", true);
Log.i("iAmTheKillerBackground", INIT_COMMENT + "Sayonara baby 8D
}
```

El método `iAmTheKillerBackground` matará al método `cancellableTaskBackground` después de `3` segundos, por lo tanto nunca se llegará a ver el mensaje `You won't see me anymore. I'll die before :'(`. Este paso es especialmente útil para parar todas las tareas que se están ejecutando en background desde los eventos de "onDestroy" o "onStop".

Background - Serial

https://github.com/androidannotations/androidannotations/wiki/WorkingWithThreads#serial

La ejecución de los métodos en segundo plano es "aleatoria" o, más correctamente, **no** se puede **conocer el orden** de ejecución. En caso de necesitar una orden o sucesión de ejecución en segundo plano, deberemos agregar el parámetro "serial" a la anotación `@Background`.

Mira este ejemplo. Al final del método `addOperationBackground` agregue estas líneas:

```
serial1Background();
serial2Background();
serial3Background();
serial4Background();
serial5Background();
serial6Background();
```

Y ahora escribiremos estos métodos:

```
@Background
void serial1Background() {
Log.i("serial1Background","I should be the first");
}

@Background
void serial2Background() {
Log.i("serial2Background","I am the second");
}

@Background
void serial3Background() {
Log.i("serial3Background","Third one");
}

@Background
void serial4Background() {
Log.i("serial4Background","I will be the fourth");
}

@Background
void serial5Background() {
Log.i("serial5Background","I'm the fith");
}

@Background
void serial6Background() {
Log.i("serial6Background","Sixth, dude!");
```

```
        }
```

A continuación ejecutaremos el código. Como vemos no es secuencial:

```
    I/serial1Background: I should be the first
    I/serial2Background: I am the second
-------->I/serial4Background: I will be the fourth
-------->I/serial3Background: Third one
    I/serial5Background: I'm the fith
    I/serial6Background: Sixth, dude!
```

Y ejecutamos una vez más:

```
    I/serial1Background: I should be the first
    I/serial2Background: I am the second
    I/serial3Background: Third one
    I/serial4Background: I will be the fourth
-------->I/serial6Background: Sixth, dude!
-------->I/serial5Background: I'm the fith
```

Finalmente vamos a añadir la opción de "serial":

```
        @Background(serial = "serial_example")
        void serial1Background()...

        @Background(serial = "serial_example")
        void serial2Background()...

        @Background(serial = "serial_example")
        void serial3Background()...

        @Background(serial = "serial_example")
        void serial4Background()...

        @Background(serial = "serial_example")
        void serial5Background()...

        @Background(serial = "serial_example")
        void serial6Background()...
```

Si ejecutamos varias veces el código, siempre obtendremos el mismo resultado:

```
    I/serial1Background: I should be the first
    I/serial2Background: I am the second
    I/serial3Background: Third one
    I/serial4Background: I will be the fourth
    I/serial5Background: I'm the fith
    I/serial6Background: Sixth, dude!
```

Background - Delay

https://github.com/androidannotations/androidannotations/wiki/WorkingWithThreads#delay

Es posible retardar la ejecución de una tarea en background. Al comienzo del método anotado con `@AfterViews` (método `initAfterViews`) añadiremos:

```
// Delayed task
delayerTaskBackground();
```

Y escribiremos este método:

```
@Background(delay=10000)
void delayerTaskBackground() {
Log.i("delayerTaskBackground","I am the last one because I am a
}
```

Este log que hemos escrito será el último en mostrarse:

```
    I/serial1Background: I should be the first
    I/serial2Background: I am the second
    I/serial3Background: Third one
    I/serial4Background: I will be the fourth
    I/serial5Background: I'm the fith
    I/serial6Background: Sixth, dude!
------->I/delayerTaskBackground: I am the last one because I am a turtle
```