Práctica final

Adaptación

Avengers_info/build.gradle

```
allprojects {
    repositories {
    google()
    jcenter()
    maven {
        url 'https://repo.spring.io/libs-milestone'
    }
    }
}
```

Avengers_info/app/build.gradle

```
android {
...
    packagingOptions {
    exclude 'META-INF/notice.txt'
    exclude 'META-INF/license.txt'
    }
}

def AAVersion = '4.5.2'
def SAVersion = '2.0.0.M3'
def JVersion = '2.9.8'
dependencies {
    annotationProcessor "org.androidannotations:androidannotations:$
    implementation "org.androidannotations:androidannotations-api:$A

    annotationProcessor "org.androidannotations:rest-spring:$AAVersi
    implementation "org.androidannotations:rest-spring-api:$AAVersio

    implementation "org.springframework.android:spring-android-rest-

    implementation "com.fasterxml.jackson.core:jackson-databind:$JVe
...
}
```

Añade: Avengers_info/app/src/main/res/xml/
network_security_config.xml

AndroidManifest.xml

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

```
    <application
        android:networkSecurityConfig="@xml/network_security_config"
        ...
        <activity
            android:name=".MainActivity_"
            ...
```

`Avengers_info/app/src/main/java/com/example/openwebinar/avengersinfo/recoverinfo/dto/HeroInfo.java`

- `private int hero = 0;` ---> `private String hero = null;`
- Actualizar: contructor, setters y getters

Avengers_info/app/src/main/java/com/example/openwebinar/avengersinfo/recoverinfo/RequestInfoActivity.java

Cambiar todos los int de R.drawable.* por un String

Por ejemplo dejar como string el atributo (de R.drawable. *serán todos los*)

Avengers_info/app/src/main/java/com/example/openwebinar/avengersinfo/views/CustomView.java

En el método "setInfo", comentar la línea "imageView.setImageResource"

Compilar y lanzar la aplicación. Se debería cargar la app, pero sin imágenes.

Desarrollo

API

Del proyecto anterior, el cual fue empleado para explicar la teoría, podemos copiar la carpeta:

```
    rest
     |--- ApiRestClient.java
     |--- Avenger.java
```

En la interfaz `ApiRestClient` cambiaremos el prototipado de `getById` por:

```
    @Get("/getAll")
    List<Avenger> getAll();
```

Respecto al cliente del API no tocaremos más, tampoco del objeto "Avenger".

UI

Vamos a añdir un nuevo componente, un Floating Action Button (FAB) ya implementado para agilizar el desarrollo. Para ello añadiremos la librería "com.getbase:floatingactionbutton" en "Avengers_info/app/build.gradle" sección de "dependencies":

```
implementation 'com.getbase:floatingactionbutton:1.10.1'
```

En la interfaz gráfica, es decir en el "activity_main.xml" añadiremos:

```
<com.getbase.floatingactionbutton.FloatingActionsMenu
    android:id="@+id/fab_menu"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    fab:fab_addButtonColorNormal="?attr/colorPrimary"
    fab:fab_addButtonSize="normal"
    fab:fab_labelsPosition="left">

    <com.getbase.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab_menu_add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@android:drawable/ic_input_add"
        fab:fab_colorNormal="?attr/colorAccent"
        fab:fab_size="mini"
        fab:fab_title="Favorito" />

    <com.getbase.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab_menu_update"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@android:drawable/ic_menu_edit"
        fab:fab_colorNormal="?attr/colorAccent"
        fab:fab_size="mini"
        fab:fab_title="Buscar" />

    <com.getbase.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab_menu_delete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@android:drawable/ic_menu_delete"
        fab:fab_colorNormal="?attr/colorAccent"
        fab:fab_size="mini"
        fab:fab_title="Añadir a la cesta" />
</com.getbase.floatingactionbutton.FloatingActionsMenu>
```

Esto nos mostrará un FAB con sus tres opciones (añadir un elemento en la API, actualizar y eliminar).

Recursos

Nuestro objetivo será eliminar la lógica anterior, que atacaba contra los recursos, y hacer que ataque contra la API.

Por ello quitaremos todos los recursos no necesarios:

colors.xml -> Eliminaremos los colores de Iron Man, Viuda Negra, Capitán América, Thor, Hulk y Spider Man (así como sus respectivos arrays). El archivo quedará con únicamente:

```
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

strings.xml -> Al igual que el XML anterior quitaremos todo lo relacionado con Iron Man, Viuda Negra, Capitán América, Thor, Hulk y Spider Man. Quedando únicamente en el fichero:

```
<resources>
    <string name="app_name">Avengers</string>
    <string name="app_name_sp">Vengadores</string>

    <string name="ib_next">next</string>
    <string name="ib_back">back</string>

    <string name="b_1">#1</string>
    <string name="b_2">#2</string>
    <string name="b_3">#3</string>
    <string name="b_4">#4</string>
    <string name="b_5">#5</string>

    <string name="b_select_en">EN</string>
    <string name="b_select_en_sp">IN</string>
    <string name="b_select_sp">SP</string>
    <string name="b_select_sp_sp">ES</string>
</resources>
```

Java

Respecto a los beans (`@EBean`), que eran los que tenían las traducciones, dejaremos únicamente lo relativo a la aplicación ya que las traducciones de los Vengadores vendrán directamente de la API.

- English -> Dejaremos las invocariones de recursos (con sus respectivos getters)

  @StringRes

  String b_select_en, b_select_sp, app_name;

- Spanish -> Dejaremos las invocariones de recursos (con sus respectivos getters)

  @StringRes

String b_select_en_sp, b_select_sp_sp, app_name_sp;

La clase de constantes (Constants) se quedará únicamente con:

- public static final String EN = "EN";
- public static final String ES = "ES";

La clase de "HeroInfo" únicamente cambiaremos:

private int image = 0; --> private String image = null;

Porque la imagen ya no será un recurso, sino una URL a un repositorio de imágenes. Como se está empleando un constructor para inicializar todos los atributos de la clase éste deberá ser actualizado, además del getter.

RequestInfoActivity

Esta será la clase que más cambios experimentará. La lógica que administra la API deberá estar aquí (con el objetivo de dejar la clase "MainActivity" únicamente con la lógica de UI.

Primeramente deberemos quitar las anotaciones `@Bean` ya que las traducciones no serán necesarias, así como `@IntArrayRes`. En su lugar añadiremos la invocación al cliente de API con la anotación `@RestService`:

```
@RestService
ApiRestClient apiRestClient;
```

Y adicionalmente crearemos una lista de vengadores global a la clase:

```
private List<Avenger> avengers = null;
```

Donde iremos guardando los vengadores con lo que hayamos interactuado.

El siguiente paso será recuperar la información de los Vengadores, anteriormente se hacía empleando el método `callHero` el cual recuperaba, según un id, el vengador de los recursos. Ahora deberemos emplear el `@RestService`. Vamos a renombrar el método `callHero` a `getHero` y llamaremos al cliente del API que permite recuperar elementos:

```
public HeroInfo getHero(int heroId) {
    HeroInfo heroInfo = null;

    initAvenger();
    if (avengers!= null && !avengers.isEmpty()) {
        Avenger anAvenger = avengers.get(heroId);
        heroInfo = avenger2HeroInfo(anAvenger);

    }

    return heroInfo;
```

```
    }
```

Este código lo que hace es recuperar todos los vengadores en la BBDD si es que no se han recuperado ya (`initAvenger();`) y con el ID de entrada al método se recupera el vengador. Como la vista no está funcionando directamente con los objetos que devuelve la API habrá que convertirlo (`avenger2HeroInfo(anAvenger);`) y retornarlo.

```
    private void initAvenger() {
        if (avengers == null) {
            avengers = apiRestClient.getAll();
        }
    }

    private HeroInfo avenger2HeroInfo(Avenger anAvenger) {
        int[] colors = new int[]{anAvenger.getColor_1().intValue(), anAv
        return new HeroInfo(
                anAvenger.getId().intValue(),
                anAvenger.getLang(),
                anAvenger.getUrlimage(),
                anAvenger.getName(),
                anAvenger.getActor(),
                anAvenger.getDescription(),
                colors);
    }
```

El siguiente método que haremos es el de eliminar un vengador:

```
    public  void deleteAvenger(long id, String lang) {
        if (avengers != null) {
            int index = getIndexFromAvengers(id);
            apiRestClient.deleteAvenger(id);
            avengers.remove(index);
            if (ES.equals(lang)) {
                apiRestClient.deleteAvenger(id - 1);
                avengers.remove(index - 1);
            } else {
                apiRestClient.deleteAvenger(id + 1);
                avengers.remove(index);
            }
        }
    }
```

Para ello recuperaremos el Id que tiene el vengador en la BBDD (`getIndexFromAvengers(id);`) y posteriormente lo eliminaremos (`apiRestClient.deleteAvenger(id);`) adenas de eliminarlo en la lista de la aplicación. De forma adicional, no tenemos únicamente un vengador, sino que hay dos (por cada idioma), por ello habrá que eliminarlo.

```
private int getIndexFromAvengers(long id) {
    for (int c=0; c<avengers.size(); c++) {
        Avenger anAvenger = avengers.get(c);
        if ( anAvenger.getId() == id ) {
            return c;
        }
    }

    return -1;
}
```

Finalmente, el método de actualizar y añadir respectivamente serán:

```
public void replaceAvenger(long id, Avenger changeAvenger) {
    apiRestClient.replaceAvenger(id, changeAvenger);
    avengers = apiRestClient.getAll();
}

public void addAvenger(Avenger newAvenger) {
    apiRestClient.addAvenger(newAvenger);
    avengers = apiRestClient.getAll();
}
```

MainActivity

Como variable global a la clase únicamente añadiremos una que será la que nos permita ver el tamaño que tiene la lista de vengadores:

```
private int allAvengers = 0;
```

A contnuación lo que habrá que hacer es recuperar la lista de vengadores desde la API, este proceso se deberá hacer en background de lo contrario Android nos tirará la aplicación con una excepción como ya se ha visto anteriormente. Para evitar problemas de concurrencia todos los métodos de background se ejecutarán de forma secuencial.

Para recuperar la lista de vengadores tendremos que llamar al siguiente método desde el `@AfterViews`:

```
@Background(serial = "mainProcessAvengers")
void countAllAvengersBackground() {
    allAvengers = receiverAction.totalAvengers();
    Log.i("MainActivity", "Recovered number of avengers: " + allAven
}
```

Este método recuperará (en caso de que no se tenga ya en memoria) todos los vengadores y devolverá el número de items recuperados.

Posteriormente habrá que recuperar el primer vengador y mostrarlo en la UI:

```
@Background(serial = "mainProcessAvengers")
void recoverAvengerBackground(int id) {
    hero = receiverAction.getHero(id);
    updateAvenger(hero);
}
```

Quedando el método `@AfterViews`:

```
@AfterViews
void initAfterViews() {
    countAllAvengersBackground();
    recoverAvengerBackground(0);
}
```

El método `updateAvenger` se encarga de actualizar el custom view que definimos en prácticas anteriores, no hará ninguna operación más:

```
@UiThread
void updateAvenger(HeroInfo update) {
    customView.setInfo(update);
}
```

Respecto al botón que se encarga de cambiar la app al inglés, en este caso vamos a evitar que si está activado pueda volverse a activar (es un mero cambio estético) para ello sólo ejecutaremos su operatoria si el contador de los vengadores es impar. En tal caso iremos al elemento anterior y actualizaremos la UI:

```
@Click(R.id.b_english_change)
void changeToEnglish() {
    if ( (counterAvenger%2) == 1 ) {
        counterAvenger = counterAvenger - 1;
        changeTextFromToolbar(english.getAppName(), english.getBSele
        changeCustomView();
    }
}
```

En cuanto el botón de traducción al español, será la misma lógica que el cambio al inglés pero en este caso nos deberemos cerciorar de no ejecutar la operación si estemos en un elemento de la lista de vengadores impart:

```
@Click(R.id.b_spanish_change)
void changeToSpanish() {
    if ( (counterAvenger%2) == 0 ) {
        counterAvenger = counterAvenger + 1;
        changeTextFromToolbar(spanish.getAppName(), spanish.getBSele
```

```
            changeCustomView();
        }
    }
```

El método `changeTextFromToolbar` será exactamente el mismo que teníamos anteriormente, mientras que el método "changeCustomView" únicamente cambiaremos las dos últimas líneas por la llamada al método:

```
recoverAvengerBackground(counterAvenger);
```

El cual ya hemos visto su funcionalidad. Para avanzar al siguiente vengador, evitaremos que el objeto `hero` no se haya quedado sin elementos o que no haya sido inicializado. Si está correcto, pasaremos al inmediatamente siguiente en su lenguaje:

```
@Click(R.id.ib_next)
void nextAvenger() {
    if (hero != null) {
        counterAvenger = (counterAvenger + 2) % allAvengers;
        recoverAvengerBackground(counterAvenger);
    }
}
```

Respecto al método de volver atrás será la lógica casi igual, salvo que tendrémos que tener especial cuidado de los contadores negativos. Para ello si `counterAvenger` al cual vamos a ir es negativo (el mudulo de dos posiciones más atrás) habrá que sumar para obtener el vengador actual el `counterAvenger` que será un número negativo, es decir: Estoy en la posición `1` y y quiero ir para atrás, me quedaré en el elemento `1-2=-1` (posición inexistente) por lo que habrá que generar la nueva posición con `allAvengers` (por ejemplo `12`), quedando `12+(-1)=11` (habrá que ir a la posición `11`).

```
@Click(R.id.ib_back)
void backAvenger() {
    if (hero != null) {
        counterAvenger = (counterAvenger - 2) % allAvengers;
        if (counterAvenger < 0) {
            counterAvenger = allAvengers + counterAvenger; // ¡¡Nega
        }
        recoverAvengerBackground(counterAvenger);
    }
}
```

En ambos casos, tras generar la posición habrá que recuperar el elemento y mostrarlo en la UI con `recoverAvengerBackground`.

En cuanto a las operaciones existentes en el FAB, serán:

- Eliminar:

```
    @Click(R.id.fab_menu_delete)
    void deleteAvenger() {
    deleteAvengerBackground();
    }
- Actualizar
    @Click(R.id.fab_menu_add)
    void fabMenuAdd() {
    addBackground();
    }
- Añadir
    @Click(R.id.fab_menu_update)
    void fabMenuUpdate() {
    updateBackground();
    }
```

La operación de eliminar, primero evalua si hay elementos y en caso positivo `counterAvenger = 0;` reinicia el listado a la posición `0`. Posteriormente eliminará el respectivo vengador haciendo una llamda al cliente de la API. Volverá a contar la lista de vengadores existentes y mostrará el primer vengador en la UI (mostrará el primero porque se ha reiniciado con `counterAvenger = 0`):

```
@Background(serial = "mainProcessAvengers")
void deleteAvengerBackground() {
    if (hero != null) {
        counterAvenger = 0;
        receiverAction.deleteAvenger(hero.getId(), hero.getLanguage(

        countAllAvengersBackground();
        recoverAvengerBackground(counterAvenger);
    }
}
```

El método de actualizar el vengador que se tiene actualmente en la UI actualizará la información del nombre del mismo y reiniciará la vista:

```
@Background(serial = "mainProcessAvengers")
void updateBackground() {
    counterAvenger = 0;

    Avenger newAvenger = updateFakeInfo();

    receiverAction.replaceAvenger(newAvenger.getId(), newAvenger);

    initAfterViews();
}
```

El método de añadir vengador creará un vengador con datos aleatorios y siempre la misma foto tanto pra inglés como para español y reiniciará la vista.

```
@Background(serial = "mainProcessAvengers")
void addBackground() {
    counterAvenger = 0;

    long newId = receiverAction.getLastId() + 1;
    Avenger newAvengerEN = createFakeAvenger(newId, "Encoded name is
    Avenger newAvengerES = createFakeAvenger(newId+1, "Nombre codifi

    receiverAction.addAvenger(newAvengerEN);
    receiverAction.addAvenger(newAvengerES);

    initAfterViews();
}
```

Como se puede ver el método anotado con `@AfterViews` se puede reusar, es importante tenerlo en cuenta tanto para esta anotación como para las demás.