Anotacion Bean

La documentación de esta anotación la veremos en el siguiente enlace:

https://github.com/androidannotations/androidannotations/wiki/Enhance-custom-classes

Para esta sección puedes copiar el proyecto que hemos visto en el tema anterior, es decir el de la sección "*2.1_2.2_2.3_2.4_Enhanced_components*", pero deberás eliminar las lineas que se muestran a continuación dentro de la clase "com.example.openwebinar.enhancedcomponents.newappication.SubApplication":

```
@Bean //We will see in class "Anotación Bean" from "Inyección" secti
RandomNumbers randomNumbers;
```

Creando el EBeans

Crearesmos 3 nuevas clases y las anotaremos con "@EBean" (https://github.com/androidannotations/androidannotations/wiki/Enhance-custom-classes#enhancing-custom-classes) dentro del paquete "com.example.openwebinar.enhancedcomponents.newappication.beans". Éstas clases serán:

RandomOctal

RandomBinary

RandomChar

Cada clase generará un número aleatorio como se muestra a continuación:

RandomOctal --> Generará números aleatorios desde 0 a 8

RandomBinary --> Generará números aleatorios desde 0 a 1

RandomChar --> Generará chars aleatorios

Adicionalmente crearemos un nuevo paquete llamado "randomhex" bajo el paquete "com.example.openwebinar.enhancedcomponents.newappication.beans". En este nuevo path crearemos los siguiente objetos:

RandomHex --> Será la interface con

- int generateRandomHexNumber --> generará números aleatorios desde 0 a 16

- String stringValue(int hexValue) --> Mostrará el valor con su correcto formato (desde 0 a F)

RandomHexImpl --> Implementación de la Interface

Llamando al Beans

La anotación "@Bean" es usado para llamar a las clases anotadas con "@EBean". Existen tres métodos diferentes para usarlo.

Global class variable

https://github.com/androidannotations/androidannotations/wiki/Enhance-custom-classes#injecting-enhanced-classes

Puedes usar una clase mejorada con la anotación "@EBean" dentro de, por ejemplo, "@EActivity" u otro bean:

```
@EBean
public class MyOtherClass {

  @Bean
  MyClass myClass;

}
```

U otro ejemplo:

```
@EActivity
public class MyActivity extends Activity {

  @Bean
  MyOtherClass myOtherClass;

}
```

Interface

https://github.com/androidannotations/androidannotations/wiki/Enhance-custom-classes#injecting-implementations-in-assignable-fields

Si tienes que utilizar una dependencia en el código a través de una clase padre o una interfaz), puede especificar la clase de implementación como el parámetro de valor de la anotación @Bean. En este último caso, debe anotar la clase de implementación y no la interfaz.

```
@EActivity
public class MyActivity extends Activity {

    /* A MyImplementation instance will be injected.
     * MyImplementation must be annotated with @EBean and implement
     */
    @Bean(MyImplementation.class)
    MyInterface myInterface;

}
```

¡! -*IMPORTANTE*- ¡! ¿Genera realmente ventajas de desacoplamiento**?** La clase aún sabe acerca de la implementación de sus dependencias, pero al menos el código que usa esas dependencias tiene que confiar en la API, que es un objetivo interesante**.**

Method

https://github.com/androidannotations/androidannotations/wiki/Enhance-custom-classes#method-based-injection

El método anotado con "@Bean" se ejecutará en el momento de inicialización**.**

Hablaremos más del momento de inicialización en la clase de "Ciclo de vida de una app"**.**

Podrás usar la anotación "@Bean" en la definición del método o en los parámetros de entrada al método:

```
@Bean
void setOneBean(MyClass myClass){
    // do something with myClass
}
```

O en los parámetros de entrada:

```
void setMultipleBeans(@Bean MyClass myClass, @Bean MyOtherClass myOt
    // do something with myClass and myOtherClass
}
```

Development

En la clase "SubApplication" crearemos los siguientes métodos:

```
publicString octalSum() {
    return null;
}

public String hexSum() {
    return null;
}

public void binarySum() {

}

public void sum() {

}
```

Y en la clase "MainActivity" añadiremos al final del método "onCreate()":

```
            System.out.println("OCT ======> " + subApplication.octalSum(
            System.out.println("HEX ======> " + subApplication.hexSum())
```

A continuación crearemos una variable global a la clase del bean:

```
    @Bean
    RandomOctal randomOctal;
```

Como podrás ver sólo se debe añadir la anotación.

Y ahora procederemos a actualizar el método "octalSum()":

```
        int randomOctalX = randomOctal.generateRandomNumber();
        int randomOctalY = randomOctal.generateRandomNumber();

        return "Octal sum: " + randomOctalX + " + " + randomOctalY + " =
                Integer.toOctalString( (randomOctalX + randomOctalY) );
```

Respecto al caso de la interface usaremos el bean "RandomHex" y el método "hexSum()". Vamos a crear la variable global:

```
    @Bean(RandomHexImpl.class)
    RandomHex randomHex;
```

Hemos añdido la anotación y la implementación. Ahora ya podrás usar el método "hexSum()":

```
        int randomHexX = randomHex.generateRandomNumber();
        int randomHexY = randomHex.generateRandomNumber();

        return "Hexadecimal sum: " + randomHex.stringValue(randomHexX) +
                randomHex.stringValue( (randomHexX + randomHexY) );
```

Respecto al modo método trabajaremos con dos diferentes casos. El primer caso únicamente ntedrá un parámetro de entrada y la definición del método tandrá la anotación "@Bean":

```
    @Bean
    public String binarySum(RandomBinary randomBinary) {
        String generatedInfo;

        if (randomBinary != null) {
            int randomBinX = randomBinary.generateRandomNumber();
            int randomBinY = randomBinary.generateRandomNumber();

            generatedInfo = "Binary sum: " + randomBinX + " + " + random
                    Integer.toBinaryString( (randomBinX + randomBinY) );
        } else {
```

```
            generatedInfo = "[not init]";
        }

        System.out.println("BIN ======> " + generatedInfo);
        return generatedInfo;
    }
```

Este código genera un error de compilación:

```
    error: org.androidannotations.annotations.Bean can only be used on a
```

Por lo que deberemos eliminar el retourn al final del método:

```
    @Bean
    public void binarySum(RandomBinary randomBinary) {
        String generatedInfo;

        if (randomBinary != null) {
            int randomBinX = randomBinary.generateRandomNumber();
            int randomBinY = randomBinary.generateRandomNumber();

            generatedInfo = "Binary sum: " + randomBinX + " + " + random
                    Integer.toBinaryString( (randomBinX + randomBinY) );
        } else {
            generatedInfo = "[not init]";
        }

        System.out.println("BIN ======> " + generatedInfo);
    }
```

El último caso será tener dos parámetros de entrada y cada uno anotado con "@Bean":

```
    public void sum(@Bean RandomNumbers randomNumbers, @Bean RandomChar
        if (randomChar != null) {
            System.out.println("CHAR ======> '" + randomChar.generateRan
        }

        if (randomNumbers != null) {
            int randomX = randomNumbers.generateRandomNumber();
            int randomY = randomNumbers.generateRandomNumber();

            System.out.println("SUM: " + randomX + " + " + randomY + " =
        }
    }
```

Ambos casos serán ejecutados en el momento de la creación.