Práctica #1

Crea un nuevo proyecto con una empty activity.

En la vista principal añade un nuevo TextView (deberá haber dos). A poner a cada uno su Id, por ejemplo:

```
tv_prime
tv_fibonacci
```

Añade los siguientes resources dentro de la carpeta "values":

```
strings.xml
    <string name="on_progress">Performing calculation…</string>
    <string name="only_natural_numbers">Only natural numbers</string
    <string name="prime_number">: is prime number. Good election!</s
    <string name="no_prime_number">: isn\'t prime number. Try again<
    <string name="fibonacci_succession">Fibonacci succession: </stri
integers.xml
    <integer name="prime_input_value">123457</integer>
    <integer name="fibonacci_size_input_value">40</integer>
```

Para finalizar la UI, inicializa el texto de ambos TextView (`tv_prime` y `tv_fibonacci`) con el recurso `on_progress`.

Abre la clase `MainActivity` e instancia (empleando AndroidAnnotations) los siguientes elementos:

```
tv_prime, tv_fibonacci
prime_input_value, fibonacci_size_input_value
only_natural_numbers, prime_number, no_prime_number, fibonacci_succe
```

Para hacer la operación de cálculo de números primos hay que tener en cuenta lo siguiente:

- Sólo se va a hacer de números positivos
- Desde `0` hasta `3` serán números primos
- Un número primo es aquel número que sólo se puede dividir entre si mismo y entre `1`
- Es una operación costosa, por lo que se debe hacer en background

Con estas premisas, la operación de la validación del número primo sería:

```
@Background
void prime(){
    boolean isPrime=true;
    String result = "";

    if (prime_input_value < 0) {
```

```
            result = only_natural_numbers;
        } else {
            if (prime_input_value <= 3) {
                result = prime_input_value + prime_number;
            } else {
                for (int c = 4; c < prime_input_value; c++) {
                    if (isMultipleNumber(prime_input_value, c)) {
                        result = prime_input_value + no_prime_number;
                        isPrime = false;
                        break;
                    }
                }
                if (isPrime) {
                    result = prime_input_value + prime_number;
                }
            }
        }

        updateTvPrimeUiThread(result);
    }

    private boolean isMultipleNumber(int multiple, int divisor) {
        return (multiple%divisor) == 0;
    }
```

Finalmente se debe mostrar el resultado en la UI:

```
    @UiThread
    void updateTvPrimeUiThread(String message) {
        tv_prime.setText(message);
    }
```

Respecto al cálculo de Fibonacci, a tener en cuenta lo siguiente

```
    Fórmula: F(n) = F(n-1) + F(n-2)
        Es decir el elemento n-esimo es igual al elemento anterior más d
```

- El elemento primero: `0`
- El elemento segundo: `1`
- El elemento tercero: `0+1`

Ejemplo: `0, 1, 1, 2, 3, 5...`

- Es una operación costosa, por lo que se debe hacer en background

Con estas premisas, la generación hasta el elemento n-esimo de Fibonnaci sería:

```
    @Background
    public void fibonacci(){
```

```
        StringBuilder result = new StringBuilder(fibonacci_succession);

        if (fibonacci_size_input_value > 0) {
            for (double i = 0; i < fibonacci_size_input_value; i++) {
                double calculatedResult = calculateFibonacci(i);
                String singleElement;

                if (WRONG_ELEMENT == calculatedResult) {
                    singleElement = only_natural_numbers;
                } else {
                    singleElement = Double.toString(calculatedResult).re
                }
                result = result.append(" ").append(singleElement);
            }
        } else {
            result.append(" ").append(only_natural_numbers);
        }

        updateTvFibonacciUiThread( result.toString() );
    }

    private double calculateFibonacci(double fibonacciElement) {
        if (fibonacciElement > 1) {
            return calculateFibonacci(fibonacciElement-1) + calculateFib
        }
        else if (fibonacciElement == 1) {
            return 1;
        }
        else if (fibonacciElement == 0) {
            return 0;
        }
        else {
            return WRONG_ELEMENT;
        }
    }
```

Finalmente se debe mostrar el resultado en la UI:

```
    @UiThread
    void updateTvFibonacciUiThread(String message) {
        tv_fibonacci.setText(message);
    }
```

Tener en cuenta que `WRONG_ELEMENT` es:

```
    final double WRONG_ELEMENT = -1;
```

Los dos métodos de background (el de la verificación de número primo y la

sucesión de Fibonacci) serán invocados desde el AfterViews:

```
@AfterViews
void initAfterViews() {
    prime();
    fibonacci();
}
```