- Laboratorio Lab 1:
Clases y objetos
La clase SimpleNumber

1. Introducción

En esta sesión de prácticas vamos a empezar a trabajar con conceptos de POO, que se verán aplicados mediante el uso de Java. Para ello, vais a programar una clase que se denominará `NúmeroNatural` y que va a encapsular el valor de **un número entero positivo o cero**, y nos proporcionará un conjunto de métodos para operar con dicho valor.

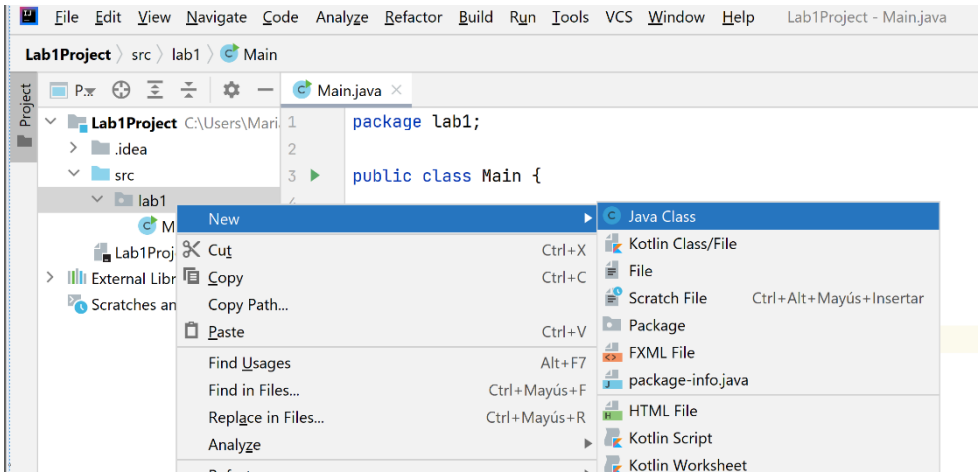
Es muy importante que os esforcéis en comprender el funcionamiento de los métodos que vais a implementar, y es deseable que vayáis incorporando comentarios al código que sin ser *farragosos* ayuden a que se entienda bien.

2. La clase `NúmeroNatural`

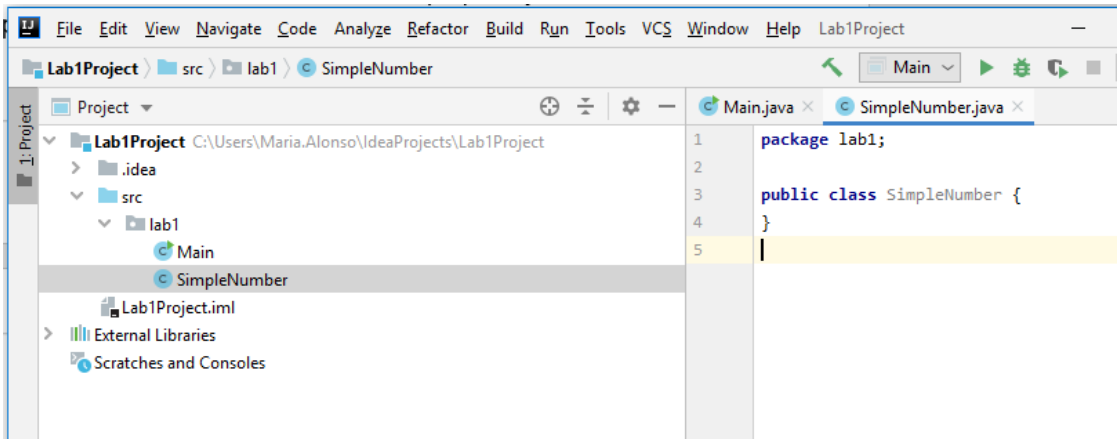
2.1 Creación de la clase y comprobación de su funcionamiento mediante un método main

Empezaremos creando un proyecto Java (File→New→Project) que se llamará `Lab1Project`. Además, vamos a usar un paquete que se llame `lab1`. Dentro de ese paquete, se creará una clase `Main`, un programa (*Main Java Class*). Recordad que, para poder ejecutar un proyecto, al menos una de las clases necesita tener un método *main*.

Ahora ya podemos crear la clase `NúmeroNatural`.
Desde la pestaña Project, con botón derecho en el paquete `lab1`, creamos una nueva *Java Class* con nombre `NúmeroNatural`.



Obtendremos entonces una plantilla únicamente con el nombre del paquete y de la clase. Ahora la clase no tendrá método *main*, pues no es una clase principal, sino una *Java Class*:



La clase `NúmeroNatural` contendrá simplemente un único atributo de tipo entero (int) y que se va a llamar `número`.

Para terminar, crearemos una instancia de nuestra clase, dentro del método *main* de la clase `Main`, que nos servirá para comprobar que todo funciona correctamente. Sería algo como:

```
NúmeroNatural número1 = new NúmeroNatural ();
```

2.2. Creación de unos métodos sencillos

Ahora que nuestra clase ya está declarada, y 'funciona', vamos a definir un conjunto de métodos bastante facilitos:

- **`setNúmero`:** No devuelve nada (es un procedimiento), recibe un entero. Asignará el valor pasado como parámetro (si es >=0) al atributo `número`.
- **`getNúmero`:** Devuelve un entero (función), no recibe nada. Devolverá el valor que contenga el atributo `número`.
- **`incrementar`:** No devuelve nada, no recibe nada. Simplemente añadirá 1 al valor que ya tenga el atributo `número`, y lo almacenará ahí mismo. Es decir, su efecto es que `número` se incrementa en uno.
- **`decrementar`:** No devuelve nada, no recibe nada. Simplemente restará 1 al valor que ya tenga el atributo `número`, si se puede, y lo almacenará ahí mismo. Es decir, su efecto es que `número` se decrementa en uno.

Tened en cuenta que deberemos comprobar que el valor de `número r` nunca es inferior a 0, así que tendréis que ver dónde y cómo imponer esa condición. Debería ser sencillo.

Después, una vez que compile, añade algunas líneas de código a tu programa principal para comprobar que los métodos que has definido están funcionando como deberían. Esto es importante que lo realices minuciosamente, no solo para comprobar tu código, sino porque también practicarás la creación de objetos y cómo se debe de llamar a sus métodos.

2.3 Implementación de algunos métodos para hacer comprobaciones

En este paso añadiremos una serie de métodos sin parámetros, que hacen comprobaciones:

- **`esPar`:** Devuelve un valor booleano, true si el número es par, o false si es impar.
- **`esPrimo`:** Devuelve un valor booleano, true o false según el número sea primo o no lo sea.
- **`esPerfecto`:** Devuelve un valor booleano, true o false según el número sea *perfecto* o no. (Se considera que un número es perfecto cuando su valor coincide con la suma de los valores de todos sus divisores, por ejemplo, el número 6, porque 6 = 1 + 2 + 3).

Como antes, deberás añadir algunas líneas de código en la clase principal (programa) para poder demostrar que estos métodos que has programado están funcionando correctamente.

2.4 Algunas operaciones matemáticas sencillas

En este punto, vamos a añadir una serie de métodos que implementan operaciones matemáticas bastante sencillas, que son:

- **`pow`:** Devolverá un valor entero, recibirá un valor entero. Retorna el resultado de elevar `número` a la potencia especificada por el argumento.
- **`half`:** Devuelve un valor de tipo double, no recibe nada. Devolverá el valor de `número` dividido por dos.

Añade código a la clase principal para comprobar estos dos métodos. ¿Está funcionando correctamente el método `half` o te encuentras con el problema de que no devuelve el valor esperado? Si sucede esto, ¿cuál sería la explicación? Para solucionar el problema, intenta multiplicar el atributo `number` por un literal neutro (el resultado de la multiplicación es el mismo que el original), antes de realizar la división, es decir: `(number*1.0)/2`.
¿Qué sucede ahora? ¿Por qué crees que se da este comportamiento?

2.5 Otros métodos adicionales usando un objeto de la clase `NúmeroNatural` como parámetro

Aquí probaremos conceptos sencillos sobre instanciación de objetos. Para ello, tienes que definir estos métodos:

- **`distancia`:** Devuelve un `NúmeroNatural`, recibe un objeto de la clase `NúmeroNatural` como parámetro. Ha de devolver el valor absoluto de la diferencia entre dos `NúmeroNatural`.
- **`syncToMinor`:** No devuelve nada, recibe un objeto de la clase `NúmeroNatural` como parámetro. Este método compara los dos números encapsulados en los objetos (atributos `número`), y el que tiene un número mayor va a cambiar éste por el de un número menor, de forma que al llamar este método los dos objetos acabarán teniendo el mismo valor (el menor de los dos al inicio de la llamada).

Nos queda comprobar que estos dos métodos funcionan y hacen lo que se les pide. Como antes, añade código al main de la clase Main para ello.