

PEC 1

Empezando...

UOC

Información relevante:

- Fecha límite de entrega: 17 de octubre de 2022
- Peso en la nota de EC: 10%.

Contenido

Información docente	3
Prerrequisitos	3
Objetivos	3
Resultados de aprendizaje	3
Enunciado	4
Tarea 1 - Configuración del entorno básico	4
Tarea 2 - Configurando IntelliJ	6
Ejercicio 1 - Debugging (2 puntos)	13
Tarea 3 - Testing con JUnit	14
Ejercicio 2 (4 puntos)	19
Ejercicio 3 (4 puntos)	21
Formato y fecha de entrega	24

Información docente

Esta PEC está vinculada con el módulo teórico “Introducción al paradigma de la programación orientada a objetos” de los apuntes de la asignatura. Léelo antes de empezar la PEC.

Prerrequisitos

Para hacer esta PEC necesitas saber:

- Conceptos básicos de algorítmica (aprendido en la asignatura anterior).
 - Conocer el paradigma de la programación estructurada.
 - La sintaxis de un lenguaje imperativo de uso común, p.ej. C, Python, etc.
 - Declaración y uso de variables de tipo primitivo (`int`, `float`, etc.).
 - Bucles (`while`, `for` y `do-while`).
 - Condicionales (`if-else` y `switch`).
 - Uso de arrays.
 - Creación y uso de funciones (en nuestro contexto llamadas “métodos”).

Objetivos

Con esta PEC el equipo docente de la asignatura busca que:

- Uses Java como un lenguaje de programación imperativo y así conozcas la sintaxis básica de este lenguaje.
- Te familiarices con la dinámica de la asignatura: metodología, tipología de enunciados, nomenclaturas, etc.

Resultados de aprendizaje

Con esta PEC debes demostrar que eres capaz de:

- Configurar un entorno de trabajo básico para programar con Java: JDK y Notepad++.
- Implementar programas sencillos en Java basados en el paradigma de la programación estructurada.
- Entender qué hace un programa o código dado por un tercero.
- Configurar y usar con cierta soltura un entorno de desarrollo integrado (IDE) como IntelliJ.
- Ser capaz de depurar (*debug*) en un IDE un programa con errores para corregirlo.
- Utilizar test unitarios para determinar que un programa es correcto.

Enunciado

Esta PEC contiene 3 tareas (actividades no evaluables) y 3 ejercicios (sí evaluables). Debes entregar tu solución de los 3 ejercicios evaluables (ver el último apartado).



Debido a que las actividades están encadenadas (i.e. para hacer una se debe haber comprendido la anterior), **es altamente recomendable hacer las tareas y ejercicios en el orden en que aparecen en este enunciado.**

Tarea 1 - Configuración del entorno básico

Antes de empezar debes:

- Instalar el JDK en tu ordenador. Para ello lee el apartado “1.1. Instalando JDK” de la Guía de Java. **Recomendamos instalar JDK 17.** Además se recomienda leer el apartado 1.2 de la Guía de Java.
- Usar un editor de texto plano (p.ej. el que viene con tu sistema operativo): vim, notepad, TextEdit, etc. No obstante, para Windows recomendamos [Notepad++](#).

Una vez realizados los pasos anteriores, puedes hacer la siguiente tarea para comprobar que el entorno básico para desarrollar programas en Java lo tienes bien instalado. Para ello, sigue estos pasos:

1. Abre el directorio `PAC1Task1` que te proporcionamos con el enunciado.
2. Allí encontrarás un fichero llamado `PAC1Task1.java`.
3. Abre un terminal o procesador de comandos (cmd).
4. En el terminal ubícate en el directorio `PAC1Task1` y una vez en él ejecuta el comando:


```
javac PAC1Task1.java
```
5. Si todo va bien, en el terminal no sucederá absolutamente nada. Esto quiere decir que el fichero `PAC1Task1.java` no ha producido ningún error. Sin embargo, si miras dentro del directorio `PAC1Task1`, debería haberte aparecido un nuevo fichero llamado `PAC1Task1.class`. Este nuevo fichero es el *bytecode* de `PAC1Task1.java`.
6. A continuación vamos a ejecutar `PAC1Task1.class`. Para ello, ejecuta el siguiente comando en el terminal que tienes abierto (que está ubicado en el directorio `PAC1Task1`):

```
java PAC1Task1
```

7. Verás que el programa escribe el siguiente mensaje: Draw
8. Ahora ejecuta el programa `PAC1Task1` pasándole argumentos. Por ejemplo:

```
java PAC1Task1 1 2 3 4 5
```

9. La ejecución anterior debería producir la siguiente salida por pantalla: Odd
10. Haz más pruebas pasándole diferentes argumentos. Después abre el fichero `PAC1Task1.java` y observa el código que contiene. Asimismo prueba otras ejecuciones del programa. ¿Qué crees que hace este programa? Por ejemplo, calcular la media de los números pasados como argumentos

11. Respuesta:

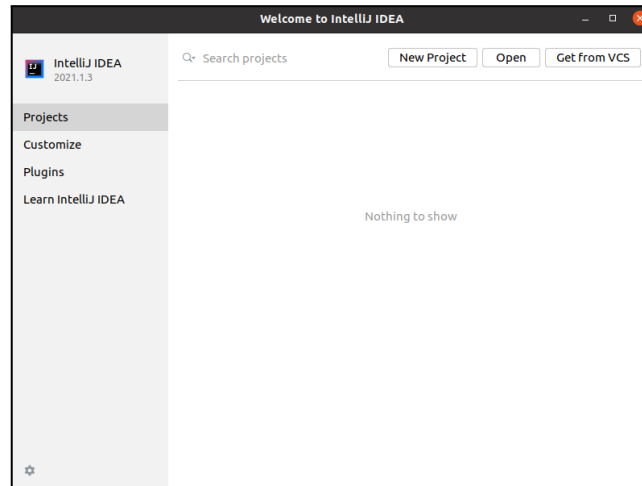
Este programa indica por pantalla si en el listado de número
sfacilitado hay más números pares que impares (salida even),
más números impares que pares (salida odd) o igual cantidad
(salida draw).

Tarea 2 - Configurando IntelliJ

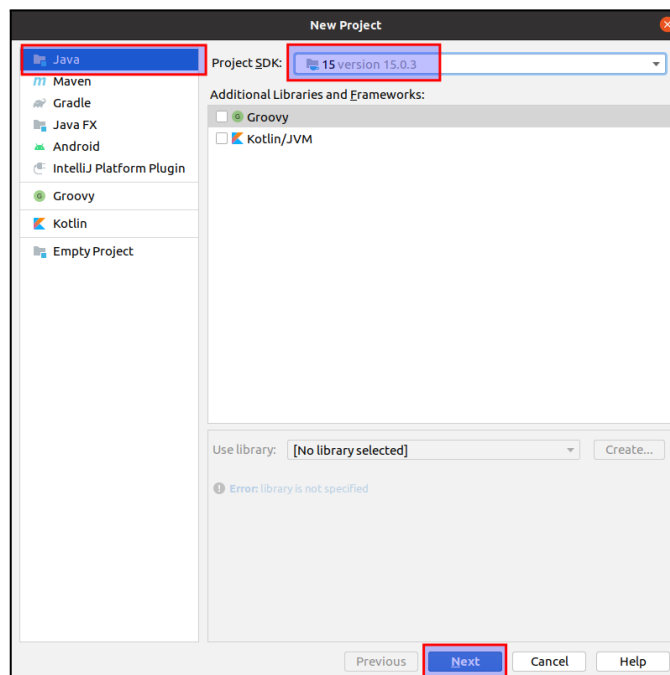
Antes de continuar debes:

Instalar y configurar el IDE IntelliJ en tu ordenador. Para ello lee el documento **Guía_IntelliJ_IB.pdf** que encontrarás en la actividad del aula.

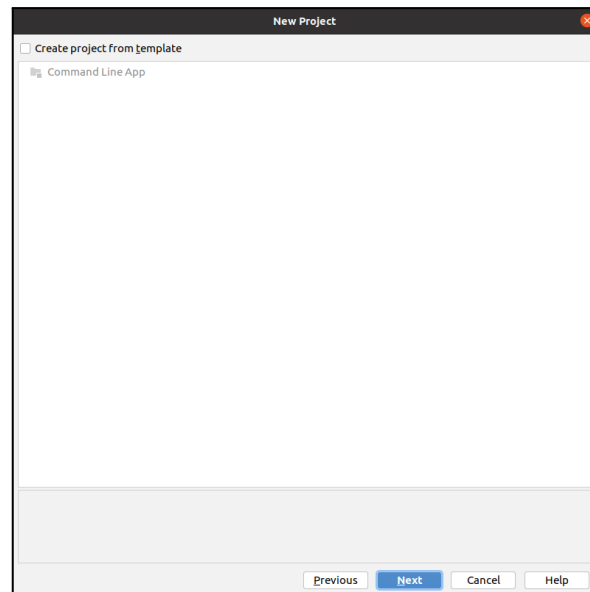
Una vez realizado el paso anterior, **ejecuta IntelliJ y crea un nuevo proyecto** con el botón **New Project**:



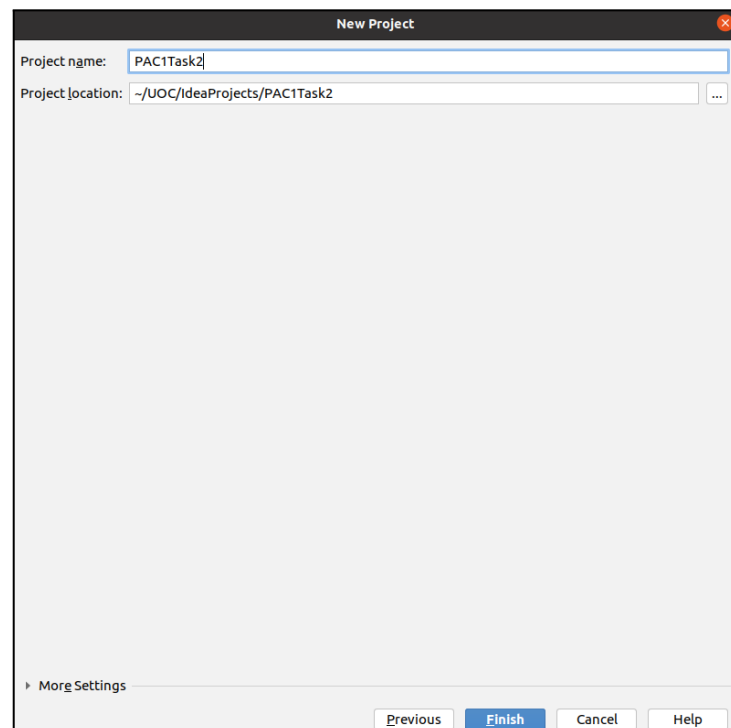
Una vez clicado el botón para crear un nuevo proyecto, IntelliJ te mostrará una ventana de configuración del proyecto. Por defecto IntelliJ selecciona como tipo de proyecto **Java** (lenguaje que usaremos durante toda la asignatura) y como SDK (*Software Development Kit*) el JDK que tienes instalado en tu máquina (JDK 17). Clicamos el botón **Next**.



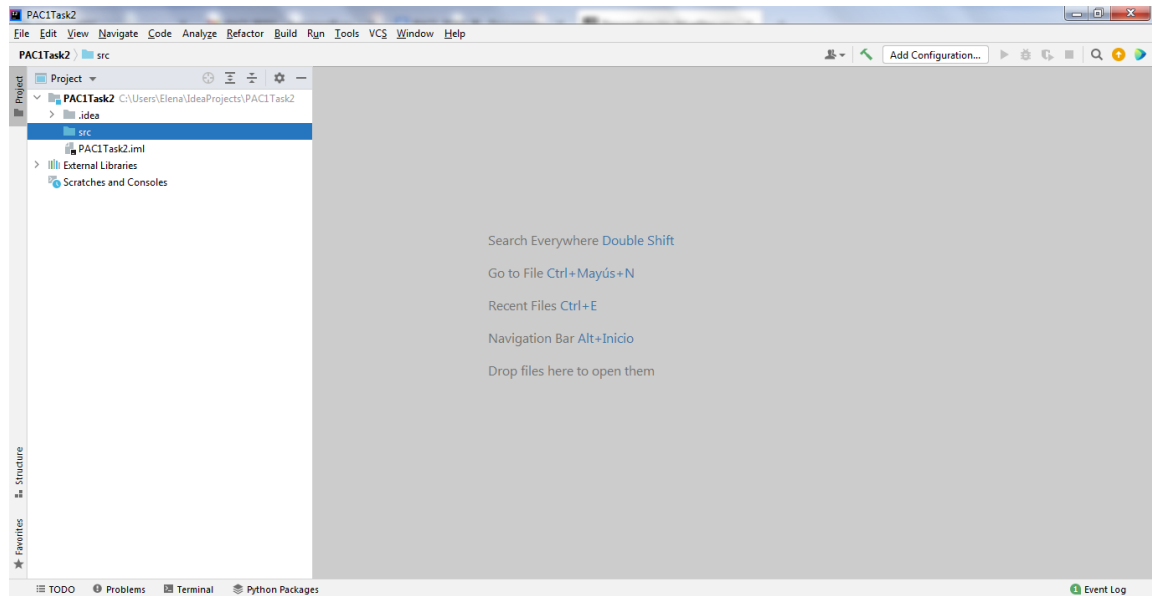
En la siguiente pantalla clics el botón `Next`, ya que no usaremos ninguna plantilla predefinida:



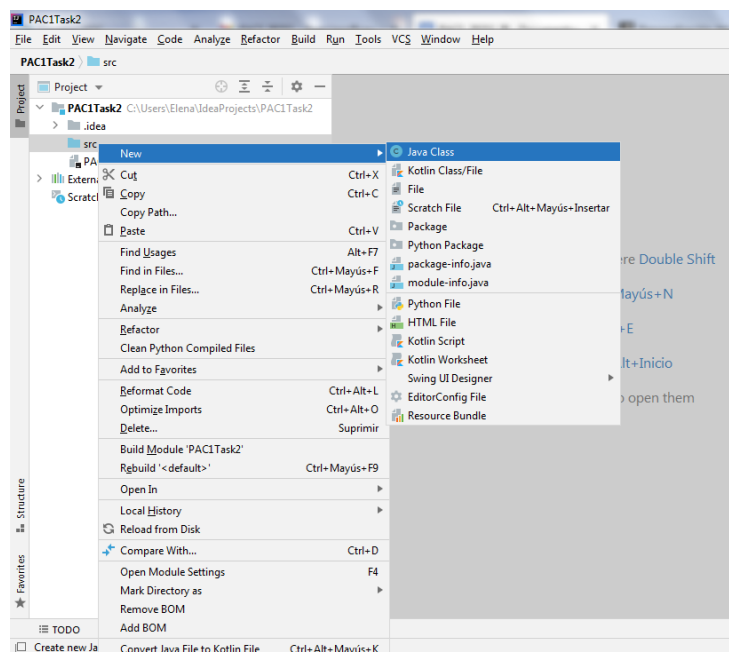
Por último, introduce el nombre del proyecto, en este caso, `PAC1Task2`, y selecciona la carpeta que hará de *workspace* (i.e. espacio de trabajo) de tus proyectos. IntelliJ ya te aconseja un directorio y un nombre para tu espacio de trabajo (i.e. `IdeaProjects`). Clics en el botón `Finish` para crear este proyecto Java.



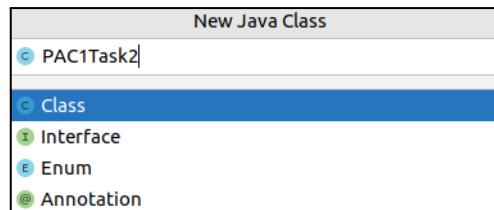
Una vez creado el proyecto **PAC1Task2**, se abrirá el proyecto mostrando el *project tool window* en el lateral izquierdo. El *project tool window* muestra un árbol de directorio del proyecto. En él verás la carpeta `src` vacía. Esta carpeta/directorio será donde pondrás el código fuente de tu programa.



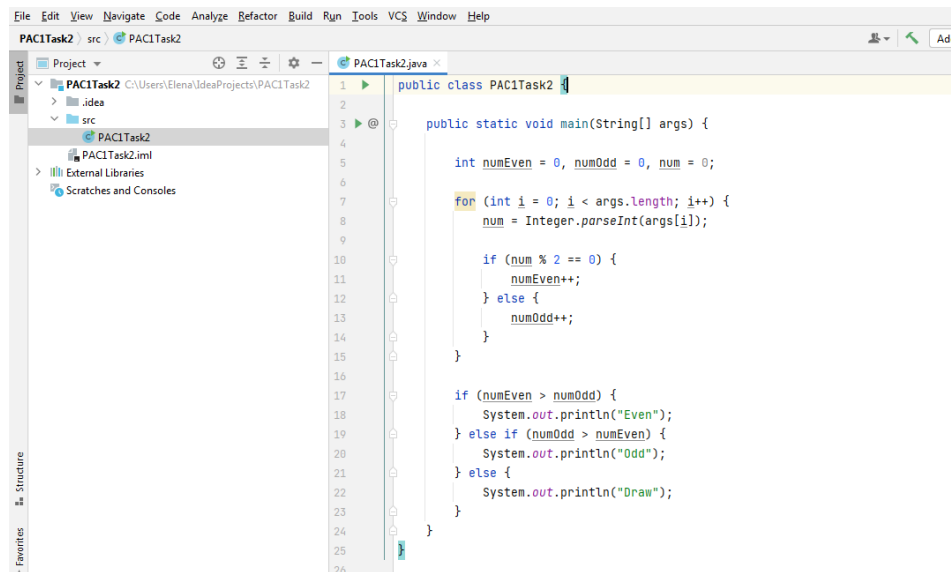
Haz click con el botón derecho encima de la carpeta `src` y escoge la opción: `src` → `New` → `Java Class`.



En la ventana que aparecerá, introduce el nombre de la clase, en este caso, **PAC1Task2** (no hace falta ponerle la extensión `.java`, IntelliJ la pondrá por ti).



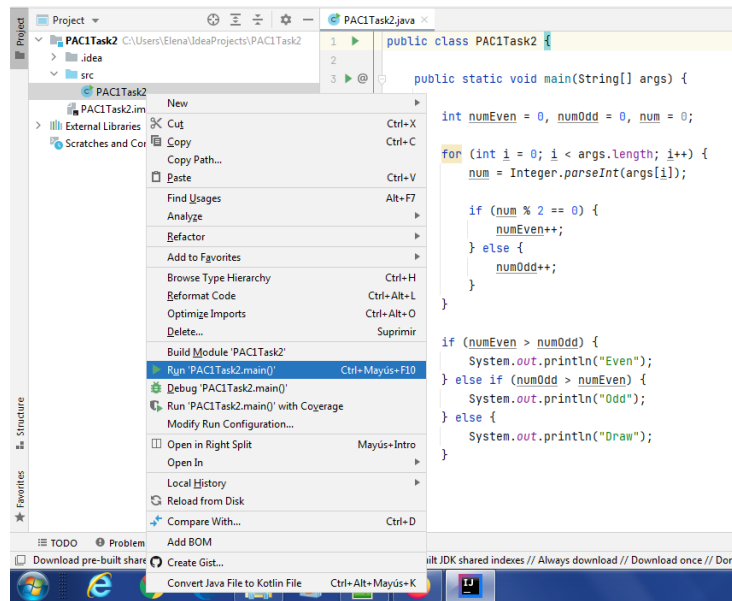
Una vez creada la clase, copia en el fichero `PAC1Task2.java` todo el código del método `main` (cabecera incluida) del fichero llamado `PAC1Task1.java` (puedes abrirlo con Notepad++) que encontrarás en la carpeta `PAC1Task1` que te hemos proporcionado con el enunciado y que has utilizado en la tarea anterior. Debería quedarte algo parecido a esto:



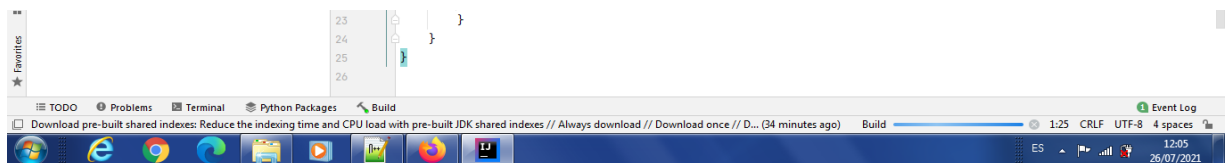
Truco: Si el código te aparece mal indentado (es decir, mal formateado), puedes hacer `Ctrl+Alt+L` (Windows) o `Cmd+Alt+L` (MacOS) para que IntelliJ indente el código correctamente.

Si quieres saber más atajos (*shortcuts*) de IntelliJ, mira: <https://www.jrebel.com/system/files/cheat-sheet-rebel-intellij-idea-web.pdf>.

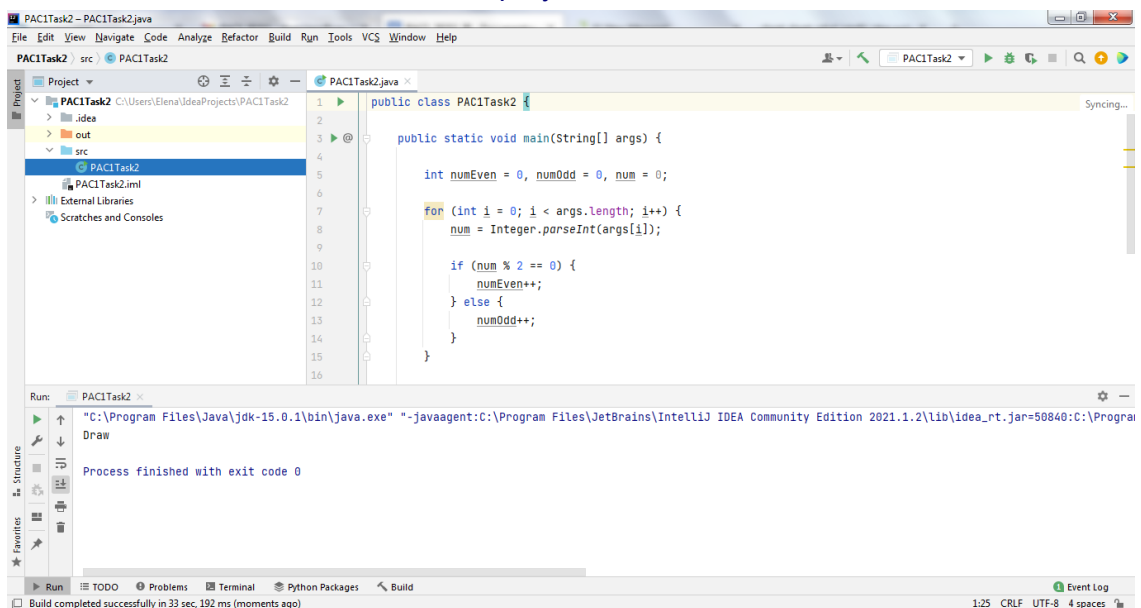
Ahora, en el *project tool window*, selecciona el fichero que acabas de crear `PAC1Task2.java` (la extensión `.java` no aparece visible en el *project tool window*), y haz click con el botón derecho del mouse. En el menú contextual que aparecerá escoge la opción: `Run 'PAC1Task2.main()'`.




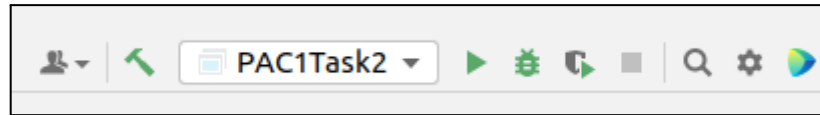
El programa será compilado (la primera vez puede tardar un poco) y, si no hay errores, se ejecutará. Fíjate en la parte inferior derecha de IntelliJ para ver qué está haciendo el IDE con tu proyecto. Indicará el estado en el que está, p.ej. *“Build”*, *“Parsing”*, *“Build completed”*, etc.



Una vez ejecutado debe aparecerte la consola con el resultado *Draw*. Ahora el código de la Tarea 1 lo hemos transformado en un proyecto IntelliJ.

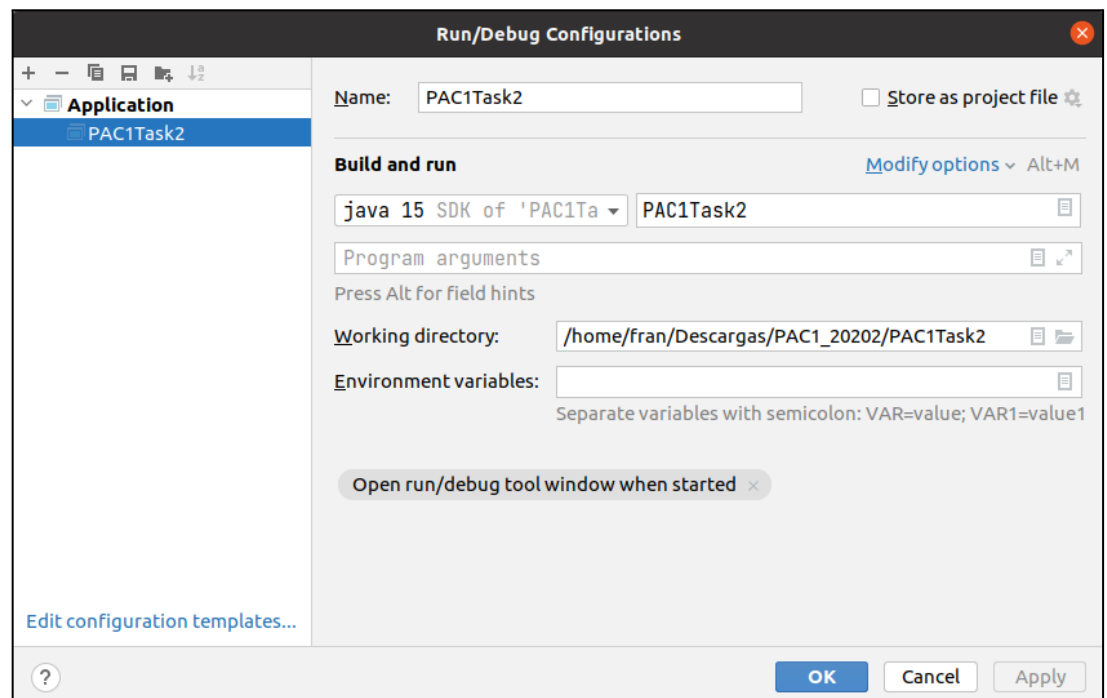


A continuación fíjate que en la parte superior derecha aparece la información de la última ejecución que hemos lanzado. Ahora puedes lanzar la ejecución directamente desde el botón Run .

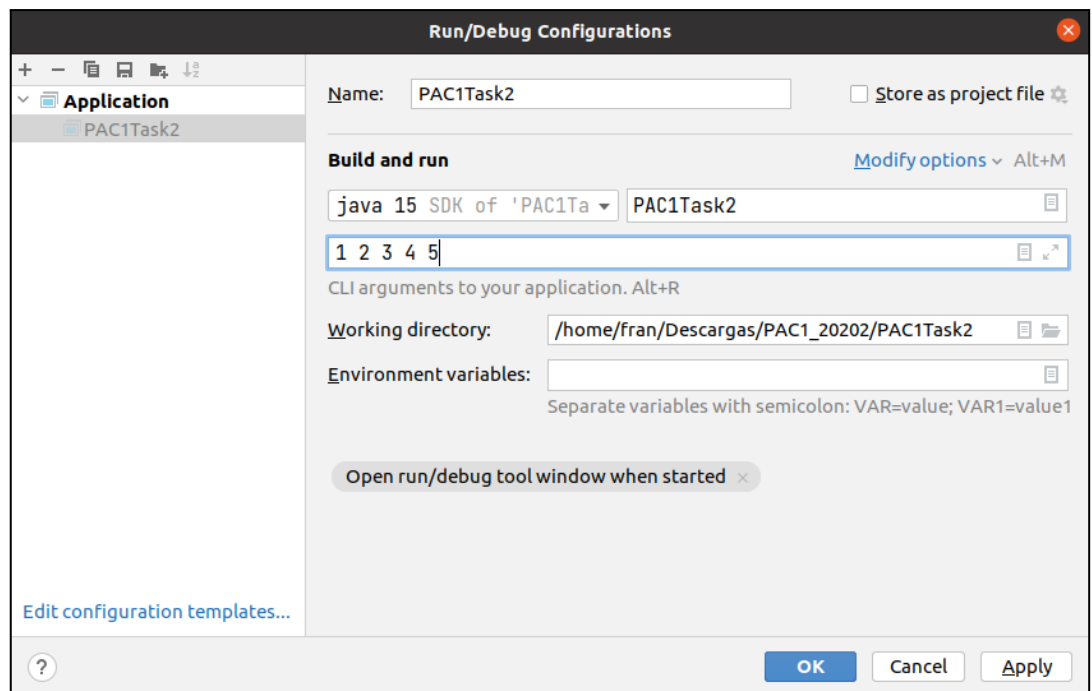



Si siempre le das a Run, siempre te aparecerá "Draw". Es normal ya que no le estás pasando argumentos. ¿Cómo pasar argumentos a la ejecución?

1. Clica en el desplegable que hay al lado del botón Run en el que pone PAC1Task2 y verás que despliega un menú.
2. En él haz click en la opción Edit configurations...
3. Se abrirá una ventana como la que se te muestra a continuación.



4. En la caja de texto que pone Program arguments escribe unos números separados por espacios.



5. Si ahora haces click en el botón **OK** de la ventana y, posteriormente, en el botón **Run**  de la barra de ejecución, se ejecutará con los valores que has escrito como argumentos en `Program arguments`, siendo el resultado `Odd` (si has puesto los números del ejemplo). Si quieres modificarlos, debes ir a `Edit configurations...` y repetir el proceso.

Ejercicio 1 - Debugging (2 puntos)

Para hacer este ejercicio copia la carpeta PAC1Ex1 que te hemos proporcionado dentro de tu carpeta workspace (normalmente llamada IdeaProjects). Si abres la carpeta/proyecto PAC1Ex1 directamente con IntelliJ sin copiarla en el *workspace*, entonces los cambios se verán reflejados en el proyecto que te hemos dado (puedes hacerlo, pero está bien tener una copia de seguridad y, a la vez, saber que todas tus entregas están dentro del *workspace*).

Una vez abierto verás que aparecen errores. Los errores que IntelliJ muestra, o bien son sintácticos, o bien hacen referencia a elementos que el IDE no conoce (estos errores también entran dentro del apartado sintáctico porque para IntelliJ son como si estuvieran mal escritos). Un error sintáctico también puede ser la ausencia o repetición de una llave {}, o que el orden de las palabras clave e identificadores no está bien (p. ej. `myName int;`), etc.

Este programa lo que hace es calcular si el número que se introduce por teclado es primo o no. Utilizamos la clase `Scanner` de Java para ingresar el dato por teclado. El programa hace una validación del número introducido, el cual no puede ser 0 o negativo. Para saber si un número es primo utilizamos una función auxiliar llamada `isPrime`. Esta función comprueba si el número introducido es divisible por algún número dentro del intervalo $[2, \text{número}/2]$. Esto es así porque ningún número es divisible por un número mayor a su mitad. Si el número introducido es divisible por alguno de los números del intervalo $[2, \text{número}/2]$, entonces el número introducido no es primo.

- a) Corrige, con la ayuda de IntelliJ, los errores sintácticos del programa de manera que no contenga ninguno.

(1 punto)

Ahora si ejecutas el programa verás que el resultado no es el esperado. ¿Por qué? Pues porque contiene errores de lógica, es decir, la funcionalidad no está bien programada. Evidentemente IntelliJ no es mágico y no sabe qué pretendes programar. Así pues, debes comprobar mediante “trazas” (i.e. ejecutar el programa paso a paso) qué está haciendo el programa y detectar dónde están los problemas. Para ello puedes, o bien poner en ciertos puntos del programa instrucciones que impriman textos por pantalla (p.ej. `println`), o bien usar la funcionalidad de debugación (en español, depuración) que ofrece IntelliJ (ver el videotutorial que encontrarás en el *site* de la asigantura ([UOCoders](https://www.uoc.edu/programas/ingenieria-informatica/segunda-grado/segunda-grado-ingenieria-informatica)), o el vídeo <https://www.youtube.com/watch?v=bb7Ny3M6cpY> y el tutorial: <https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html>). Te recomendamos que le dediques un poco de tiempo a aprender a debugar mínimamente con IntelliJ, ya que te será muy útil en futuras actividades (y en el mundo profesional).

- b) Corrige los errores lógicos (también llamados errores semánticos) del programa de manera que se ejecute como se espera.

(1 punto)



Requisito mínimo para evaluar este ejercicio: el programa debe funcionar como es esperado, i.e. se deben realizar los apartados (a) y (b) correctamente.

Tarea 3 - Testing con JUnit

En el Ejercicio 1 la única manera que has tenido para saber si el programa se comportaba como era esperado ha sido ejecutándolo varias veces y viendo su salida por pantalla (o por el debugador). Hacer test y detectar errores utilizando el método del propio lenguaje que escribe por pantalla (p.ej. `printf`, `print`, `println` o el equivalente del lenguaje que utilices habitualmente) es una manera muy básica –muy de principiante–, aunque todo el mundo la suele utilizar en algún momento por su comodidad. Por su parte, la debugación se utiliza para detectar errores locales dentro del código.

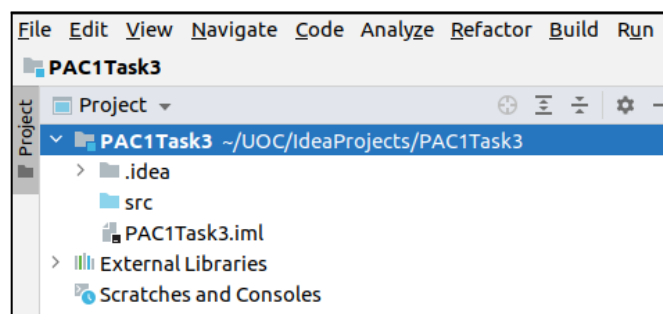
Dado que el *testing* es muy importante para garantizar que un programa es correcto y se comporta como se espera, con el tiempo ha surgido una manera de desarrollar software basada justamente en el testeo. Es lo que se conoce como TDD o *Test-Driven Development* (i.e. desarrollo conducido/dirigido por test). Esta forma de desarrollar tiene como lema: *“un programador debe escribir los casos de test antes que el código que dichos test van a comprobar”*.

A partir de ahora en esta asignatura escribirás o te proporcionaremos test para todos los ejercicios que requieren codificación. Para ello usaremos la librería JUnit. Es por este motivo que antes de continuar debes:

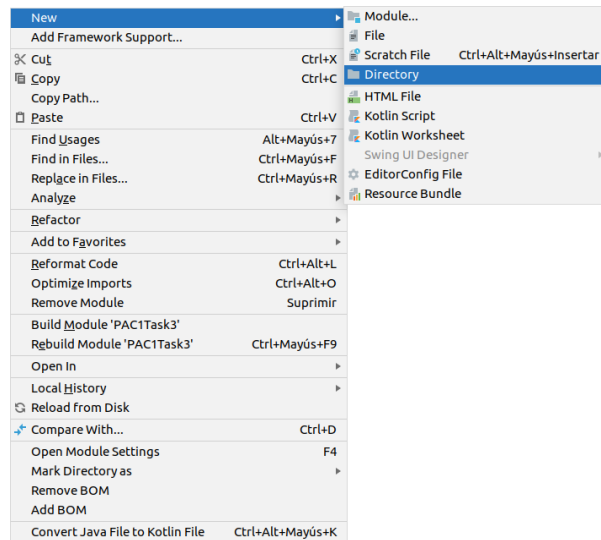
Leer el apartado **“5.3.1. JUnit”** de la Guía de Java, que da una pequeña introducción sobre qué son los test unitarios.

Y por otro lado, seguir los siguientes pasos para añadir, configurar y usar JUnit en un proyecto Java en el IDE IntelliJ (sin ningún soporte de un gestor de dependencias como Maven o Gradle... esto ya lo trabajaremos en la PEC2).

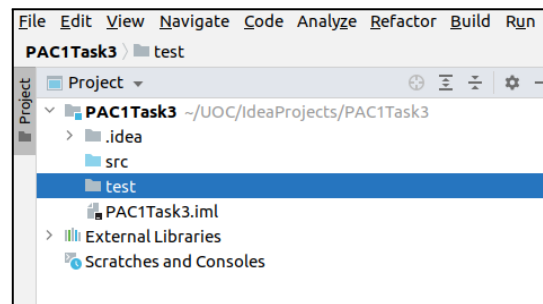
1. Crea un nuevo proyecto Java en IntelliJ desde `File → New → Project` llamado **PAC1Task3**.



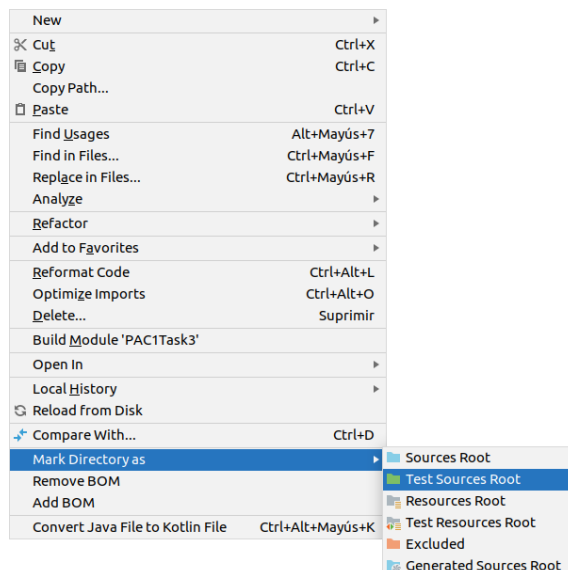
2. Desde la raíz del proyecto crea, con el menú contextual que aparece al hacer click en el botón derecho del mouse, un directorio (*Directory*) llamado `test` para almacenar los test unitarios JUnit.



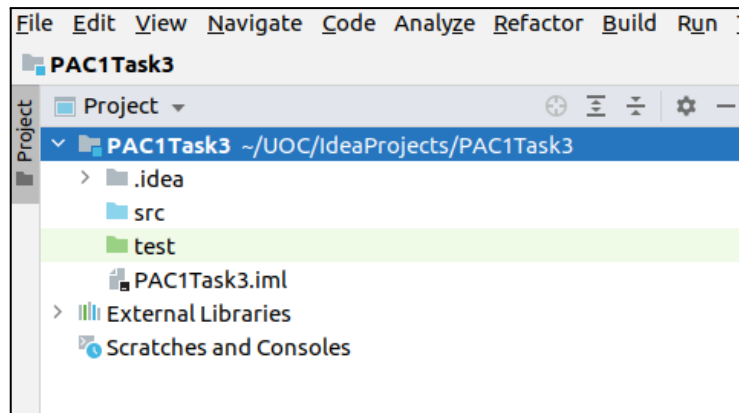
3. Una vez creado, verás en el *project tool window* una nueva carpeta *test*, al mismo nivel que la carpeta de código fuente *src*.



4. Selecciona la carpeta *test* y márcala como un directorio que será la raíz de las fuentes de los test unitarios (i.e. *Test Sources Root*). Para ello haz botón derecho sobre el directorio *test* y escoge la opción *Mark Directory as* → *Test Sources Root*.



Veràs que ha canviado el color de la carpeta `test` a verde.

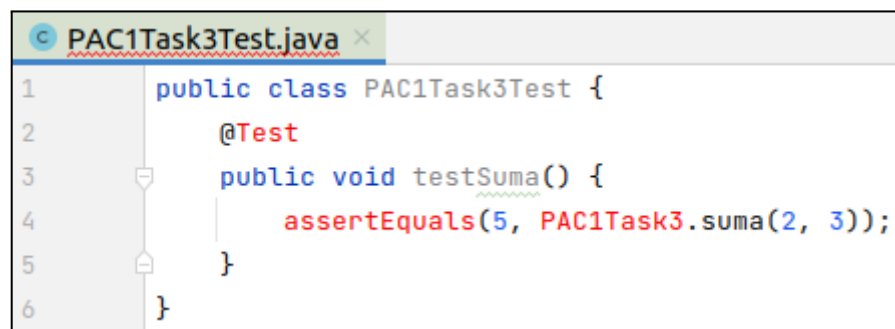



- Una vez aquí, en el directorio `test`, crea la clase `PAC1Task3Test` donde implementaràs los test unitarios. Ya has visto anteriormente cómo crear clases en los proyectos.
- Crea la siguiente función/método dentro de esta clase:

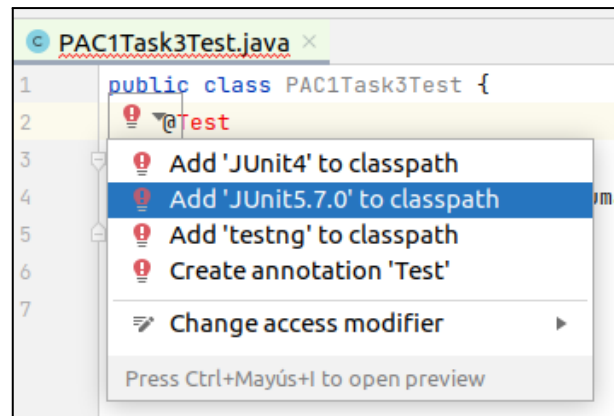
```
public void testSuma() {
    assertEquals(5, PAC1Task3.suma(2, 3));
}
```

- Añade la anotación `@Test` encima de la firma del método. Con esto estamos especificando que se trata de una método de test JUnit. Además, esto nos permite importar desde IntelliJ la librería JUnit.

Como veràs, si pones el cursor encima de la línea `@Test`, te marca un error de IntelliJ.

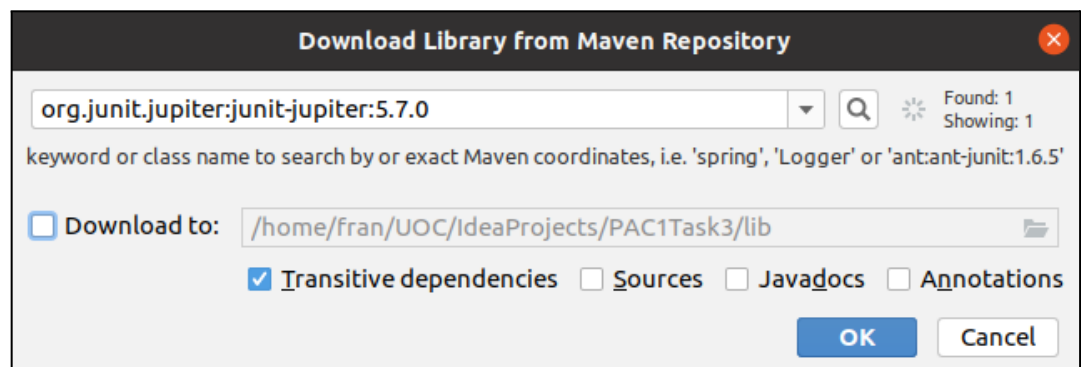


- Clica encima de este globo rojo  e IntelliJ te proporcionará una manera fácil de añadir la librería JUnit 5 a tu proyecto.



9. En la siguiente pantalla, IntelliJ te permite descargar desde el repositorio Maven la librería JUnit 5.x (esto no significa que utilices Maven como gestor de dependencias, únicamente estás diciendo que descargue esta librería desde el repositorio oficial de Maven; podría ser desde otro repositorio o link).

Puedes seleccionar el *check* Download to si quieres descargar los fichero de la librería en una carpeta (lib) de tu proyecto, o no marcarlo si quieres añadir la librería como External libraries. En este caso utilizaremos la segunda opción.




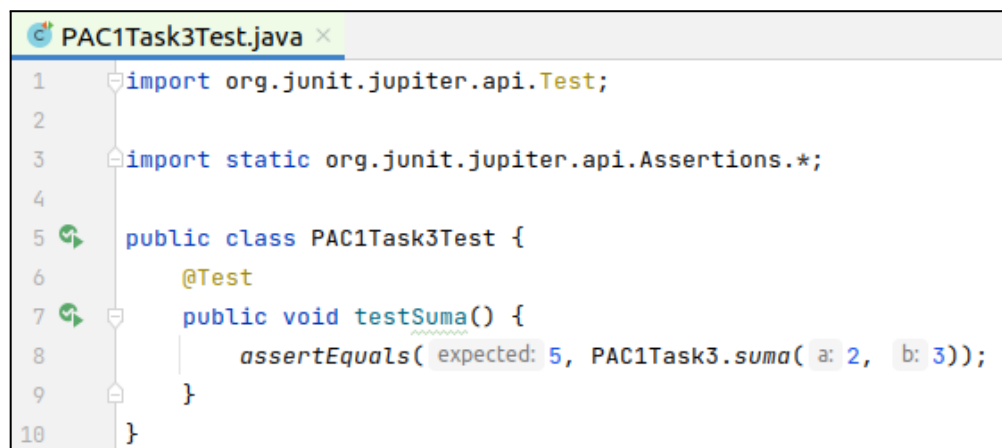
10. Una vez añadidas las librerías JUnit, verás que ya no tienes el error en la anotación @Test.
11. Ahora, en el directorio src, crea la clase PAC1Task3 donde tendrás la implementación del método testSuma que has codificado en la clase de test PAC1Task3Test.
12. Dentro del fichero PAC1Task3.java escribe el siguiente código:

```
public class PAC1Task3 {
    public static int suma(int a, int b){
        return a + b;
    }
}
```

13. Por último, ya sólo te queda resolver el error que hay en la clase `PAC1Task3Test`, puesto que falta un `import` para poder utilizar las `assertions` (i.e. métodos/funciones) de JUnit. Por lo tanto añade este `import` encima de la definición de la clase (si no lo ha hecho IntelliJ ya por ti):

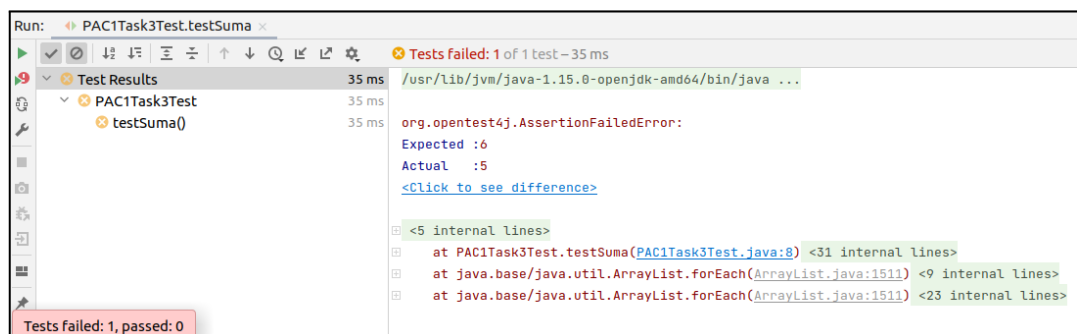
```
import static org.junit.jupiter.api.Assertions.*;
```

14. Ahora ya puedes ejecutar el test implementado en la clase `PAC1Task3Test`. Puedes ejecutarlo como hemos explicado anteriormente clicando en el botón derecho sobre la clase y accediendo a `Run`, o con los botones  que verás delante de la clase y los métodos que implementan los tests unitarios.



```
1 import org.junit.jupiter.api.Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 public class PAC1Task3Test {
6     @Test
7     public void testSuma() {
8         assertEquals( expected: 5, PAC1Task3.suma( a: 2, b: 3));
9     }
10 }
```

15. Modifica el valor de `expected` (5) para poner cualquier otro valor que sea incorrecto, vuelve a ejecutar y verás que el IDE te muestra los errores del JUnit de la forma: valor esperado y el actual, así como la traza de error.



Run: PAC1Task3Test.testSuma

Tests failed: 1 of 1 test - 35 ms

Test Results

- PAC1Task3Test (35 ms)
 - testSuma() (35 ms)

org.opentest4j.AssertionFailedError:
Expected :6
Actual :5
[Click to see difference](#)

<5 internal lines>

- at PAC1Task3Test.testSuma(PAC1Task3Test.java:8) <31 internal lines>
- at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
- at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>

Tests failed: 1, passed: 0

Ejercicio 2 (4 puntos)

Copia en tu *workspace* el proyecto `PAC1Ex2` que te facilitamos con el enunciado y ábrelo con IntelliJ. En este ejercicio vamos a ver cuándo dos monjes se encuentran en un camino circular. Hay que tener presente que la primera posición del camino es siempre 0 y que la máxima longitud indicada en cada caso para el camino es un índice no incluido. Como el camino es circular, cuando se llega al final, se vuelve a empezar por 0.

A continuación debes codificar los `TODO` (del inglés, “*to do*”, en español, “*por hacer*”, o usando una traducción más libre: “*pendiente*”). Este ejercicio consiste básicamente en codificar los siguientes 3 métodos/funciones:

- `stepForward`: esta función recibe la posición en la que está un monje, la velocidad a la que camina (i.e. número de posiciones que avanza cada día) y la máxima longitud del camino.

```
stepForward(1, 2, 7)
```

El resultado que debería devolver esta función tendría que ser: 3, ya que el monje está en la posición 1, se desplaza a velocidad 2, i.e. $1 + 2 = 3$. Sin embargo, la siguiente invocación es diferente:

```
stepForward(6, 2, 7)
```

El resultado debería ser 1. Al sumar 2 a la posición 6, como esta posición es la última, el monje vuelve al inicio y camina 2 posiciones, $6 \rightarrow 0$, $0 \rightarrow 1$.

Esta función devuelve -1 si la posición inicial es negativa o mayor o igual a la longitud del camino, o la velocidad es un valor negativo.

(1 punto: 0.5 puntos test proporcionados; 0.5 puntos calidad del código)

- `stepForward`: esta función recibe las posiciones en las que están los dos monjes, las velocidades a las que caminan y la máxima longitud del camino.

```
stepForward(1, 2, 6, 2, 7)
```

El primer monje parte de la posición 1 y camina a 2 casillas por día, mientras que el segundo monje parte de la posición 6 y camina a 2. La longitud del camino es 7. El resultado que debería devolver esta función tendría que ser: `[3, 1]`, donde el primer índice indica dónde está el primer monje después de moverse una vez/día, i.e. casilla 3, y el segundo índice dónde está el segundo monje, i.e. casilla 1.

Si para alguno de los monjes se da un caso anómalo como el explicado en la función anterior, entonces para ese monje el valor devuelto será -1. Por ejemplo:

```
stepForward(1, -2, 6, 2, 7)
```

El valor retornado por esta función será `[-1, 1]`, ya que la velocidad del primer monje es un valor negativo.

(1 punto: 0.5 puntos test proporcionados; 0.5 puntos calidad del código)

- `whenTheyMeet`: este método indica en qué día y en qué posición se encontrarán dos monjes.

```
whenTheyMeet(0, 2, 1, 1, 8)
```

Los dos primeros argumentos son la casilla inicial del primer monje y su velocidad; los siguientes dos argumentos son los mismos valores para el segundo monje; y el último valor es la longitud del camino.



IMPORTANTE: Hay que tener presente que los dos monjes hacen un desplazamiento diario y al mismo tiempo. Como mínimo se necesitará un día para encontrarse, incluso cuando ambos monjes parten de la misma casilla inicial.

En el ejemplo anterior, el resultado debe ser el array `[1, 2]`, donde el primer índice del array significa el día de encuentro y el segundo índice la casilla en la que se encuentran. En este ejemplo, el primer día el primer monje va del `0→2` y el segundo va del `1→2`. Así pues, al final del día ambos monjes están en la casilla `2` y sólo se ha necesitado un día para encontrarse. Como ves los días empiezan en `1`.

Si en algún momento hay un error, por ejemplo porque la velocidad de uno de los monjes es negativo, entonces el valor devuelto por esta función debe ser `[-1, -1]`.

(2 puntos: 1.5 puntos test proporcionados; 0.5 puntos calidad del código)

Requisito mínimo para evaluar este ejercicio: el programa debe pasar todos los test proporcionados para los dos métodos `stepForward`.

Ejercicio 3 (4 puntos)

Copia en tu *workspace* el proyecto `PAC1Ex3` que te facilitamos con el enunciado y ábrelo con IntelliJ. En este ejercicio vamos a calcular diferentes aspectos relacionados con la pérdida de poder adquisitivo del profesorado de la FUOC en base a los datos extraídos del [INE](#) (Instituto Nacional de Estadística). El [convenio colectivo de la FUOC](#) se firmó en 2012 y éste no incluye la revisión de los sueldos en función del IPC. Así pues, desde entonces, los salarios de los profesores de la FUOC están congelados, siempre y cuando el profesor no haya logrado un cambio de categoría fruto de la obtención de méritos propios.

A continuación debes codificar los `TODO` (del inglés, “*to do*”, en español, “*por hacer*”, o usando una traducción más libre: “*pendiente*”). Este ejercicio consiste básicamente en codificar los siguientes 4 métodos/funciones:

- `calculateSalaryWithAbsoluteCPI`: este método calcula el salario que debería cobrar un profesor a día de hoy a partir de un salario base y un IPC dado. Así pues, si queremos calcular cuánto debería cobrar en mayo de 2022 un profesor de la categoría “Profesor Lector 2” sabiendo que el IPC acumulado entre mayo de 2012 y mayo de 2022 es del 16.9%, haríamos:

```
calculateSalaryWithAbsoluteCPI(32000, 16.9)
```

El resultado de este método tendría que ser: 37408. Sin embargo, la realidad es que un profesor que sigue siendo en 2022 profesor Lector 2, sigue cobrando 32000 €, lo mismo que en 2012.

Este método devuelve el salario introducido como parámetro, si el salario es cero o negativo, o bien si el IPC introducido como parámetro es negativo o cero.

(1 punto test proporcionados)

- `calculateAccumulateCPI`: esta función calcula el IPC acumulado en un periodo de tiempo. Este método recibe el número de año inicial (consideramos que 0 es el año 2012), el número de mes del año inicial (enero es el mes 0), el número de año final que hay que incluir y el número del mes del año final a incluir.

```
calculateAccumulateCPI(0, 0, 0, 11)
```

La invocación anterior debe devolver el valor del IPC en el año 2012. Fíjate, `añoInicial = 0` (2012), `mesInicial = 0` (enero), `añoFinal = 0` (2012) y `mesFinal = 11` (diciembre). El resultado es 3.



IMPORTANTE: Fíjate que te damos una variable llamada `evolutionCPI` que contiene el valor del IPC para cada mes desde enero de 2012 hasta mayo de 2022. Esta variable es un array de arrays, donde cada casilla/índice del primer array representa un año y cada año guarda un array con los valores de IPC de los meses de ese año.

Ten en cuenta que esta función debe devolver `-1`:

- si el valor del año inicial o del final es menor a cero o alguno de los dos es mayor al número de años que contiene `evolutionCPI`.
- si el valor del año inicial es mayor que el año final.
- en caso de que el año inicial y el final sean el mismo, devolverá `-1` si el mes inicial es mayor al mes final.
- si el valor del mes del año inicial o del año final no existe en `evolutionCPI`. Piensa que para el año 2022, por ejemplo, no tenemos el valor de agosto.

(2 puntos: 1.5 test proporcionados; 0.5 calidad del código)

- `calculateSalaryWithIntervalCPI`: esta función calcula el salario que debería tener un trabajador en un periodo dado. Este método recibe un salario base (se entiende que es el que tiene al inicio del periodo), un año inicial, un mes dentro del año inicial, un año final y un mes dentro del año final. Si existe algún problema durante el cálculo (p.ej. parámetros incorrectos), entonces devuelve `-1`. En caso contrario, devuelve el nuevo salario.



Consejo: Utiliza las dos funciones que has codificado anteriormente para codificar `calculateSalaryWithIntervalCPI`.

(0.5 puntos test proporcionados)

- `printSalaryInARange`: esta función imprime por pantalla el salario de un trabajador por año si se le aplicara el IPC correspondiente. Para ello esta función recibe un salario base, el año inicial y el año final.

```
printSalaryInARange(28000, 0, 0)
```

La invocación anterior debe imprimir por pantalla:

```
Year 2012, your salary should be 28840.00
```

Si nos fijamos estamos pidiendo el salario del año 0 (2012) al año 0 (2012). Es decir, devuelve el salario que tendría el trabajador al final del año 2012.

Otro ejemplo:

```
printSalaryInARange(32000, 0, 1)
```

La invocación anterior debe imprimir por pantalla:

```
Year 2012, your salary should be 32960.00
Year 2013, your salary should be 33058.88
```

Si nos fijamos estamos pidiendo el salario del año 0 (2012) al año 1 (2013).

Si se produce algún error, por ejemplo porque el año no tiene 12 meses, entonces se debe imprimir los años anteriores con normalidad e imprimir, en una nueva línea, el mensaje "[ERROR] There was an error with printSalaryInARange" y no mostrar los siguientes años.



Consejo: Utiliza la función `calculateSalaryWithIntervalCPI` dentro de esta función. Asimismo debes utilizar el `decimalFormat` que hemos definido en la implementación para que formatee el número de salida con dos decimales. Se debe usar de la siguiente manera:

```
decimalFormat.format(doubleValue)
```

(0.5 puntos test proporcionados)



Requisito mínimo para evaluar este ejercicio: el programa debe pasar todos los test proporcionados de los métodos `calculateSalaryWithAbsoluteCPI` y `calculateAccumulateCPI`.

Formato y fecha de entrega

Tienes que entregar un fichero *.zip, cuyo nombre tiene que seguir este patrón: loginUOC_PEC1.zip. Por ejemplo: dgarciaso_PEC1.zip. Este fichero comprimido tiene que incluir los siguientes elementos:

- El proyecto de IntelliJ PAC1Ex1 completado siguiendo las peticiones y especificaciones del Ejercicio 1.
- El proyecto de IntelliJ PAC1Ex2 completado siguiendo las peticiones y especificaciones del Ejercicio 2.
- El proyecto de IntelliJ PAC1Ex3 completado siguiendo las peticiones y especificaciones del Ejercicio 3.

El último día para entregar esta PEC es el **17 de octubre de 2022** antes de las 23:59. Cualquier PEC entregada más tarde será considerada como no presentada.