CURSO 2019-20.
Práctica 1: Introducción al lenguaje Java.

1. OBJETIVOS:

- Familiarizarse con el Entorno de Desarrollo Integrado *IntelliJ IDEA*.
- Primera toma de contacto con el lenguaje Java.
- Definición de clases en Java.
- Instanciación de objetos en Java y envío de mensajes a objetos.

2. El IDE IntelliJ IDEA

IntelliJ IDEA es un Java IDE (Integrated Development Environmet) desarrollado por JetBrains.

2.1. Instalación de IntelliJ IDEA.

Para realizar prácticas fuera del CIC es necesario:

- En primer lugar, hay que tener instalado un Kit de desarrollo para Java (JDK). En caso de no tenerlo, se puede obtener en la web de Oracle: https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html
- A continuación, se instalará el IDE de IntelliJ, que se puede obtener en el siguiente enlace: https://www.jetbrains.com/idea/download, edición *Community*.

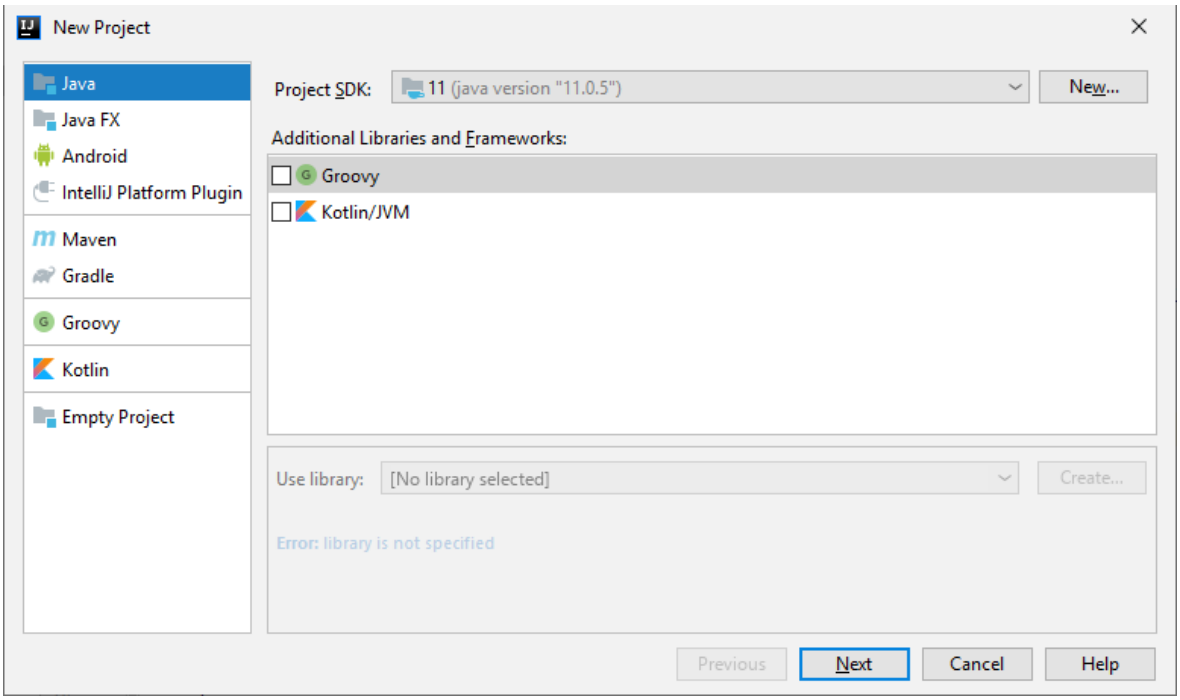
2.2. Crear un proyecto.

Para programar una aplicación Java con *IntelliJ*, siempre debe hacerse en el contexto de un **proyecto**. Dentro de un proyecto puede haber: ficheros fuente java (.java), ficheros compilados java (.class), librerías, ficheros de configuración, etc.

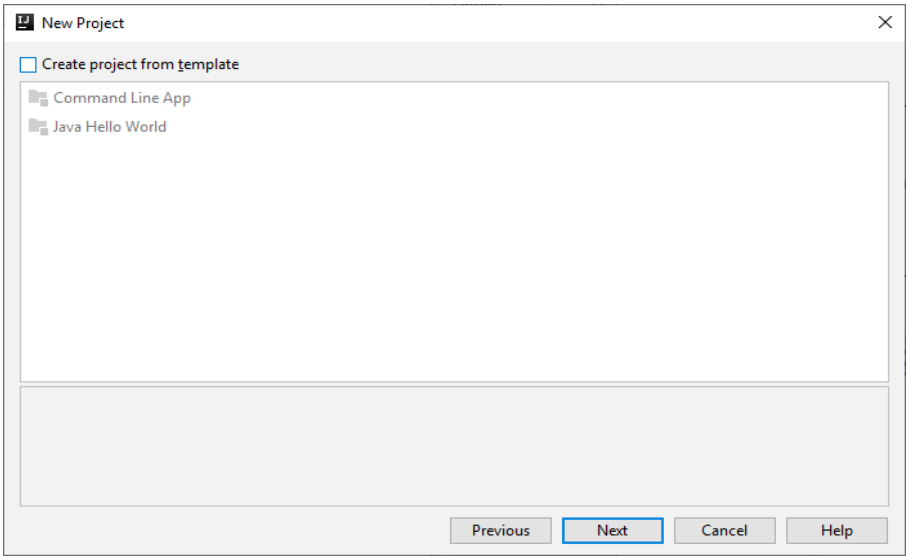
Al arrancar *IntelliJ*, nos solicita crear un nuevo proyecto o cargar uno ya creado:

Si se pulsa “Open” se carga un proyecto anterior realizado con *IntelliJ*.

Si se pulsa “Create New Project” aparece la siguiente ventana:

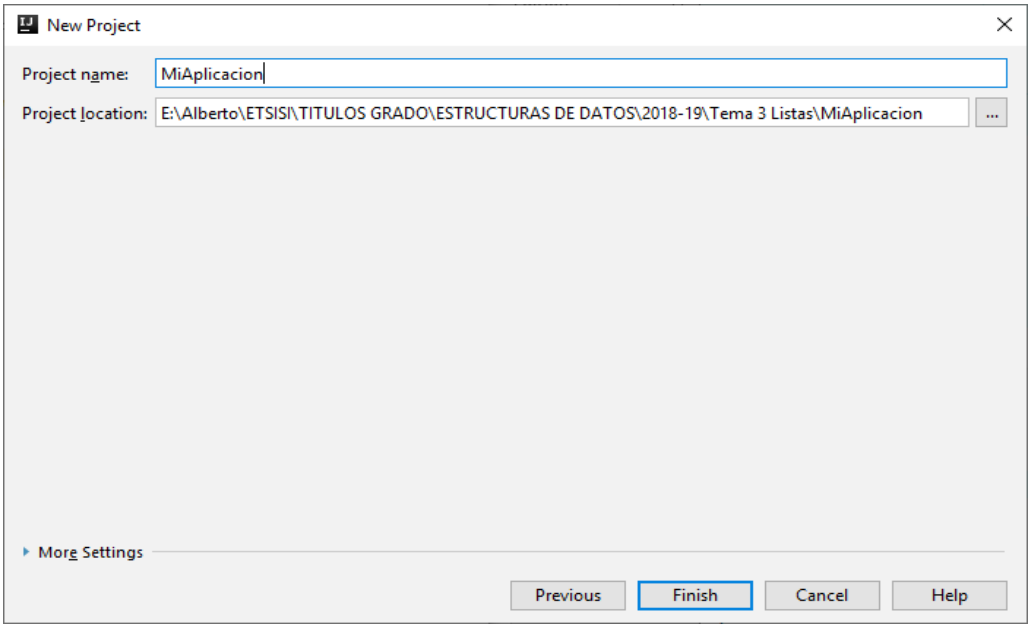


- En primer lugar, seleccionar que se va a crear un proyecto de tipo Java.
- Determinar el JDK de Java a utilizar. Si no aparece ninguno, se le indicará la ubicación del JDK instalado (*New*). *IntelliJ* utilizará ese JDK en sus tareas: uso de librerías estándar de Java, compilar, ejecutar la máquina virtual de Java, etc.
- No es necesaria ninguna librería más ni ningún *framework* de desarrollo.
- Pulsar Next.

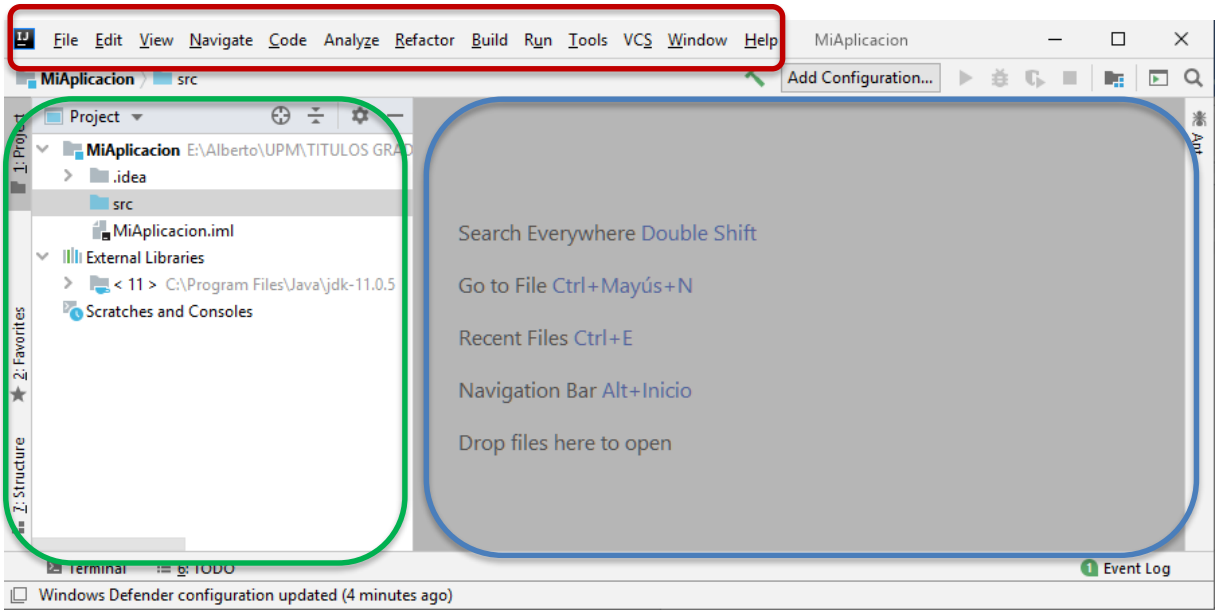


- Para crear un proyecto vacío, no se seleccionará ningún *Template*.
- Pulsar *Next*.

Darle un nombre al proyecto. Dicho nombre coincidirá con el nombre del directorio.



- Pulsar *Finish*.



Pueden observarse 3 zonas diferentes en la pantalla:

- La parte superior muestra el menú principal.
- La parte derecha es la ventana de edición, donde veremos el código Java cuando empecemos a crearlo.
- La parte izquierda es la estructura del proyecto “MiAplicacion”. Puede observarse un directorio “src”, que ahora está vacío, donde se almacenarán los archivos Java. El directorio “.idea” y el archivo “MiAplicacion.iml” son ficheros de configuración de *IntelliJ*. También puede observarse en “External Libraries” que el proyecto tiene disponible la biblioteca estándar de Java (JDK 11).

2.3. Crear un archivo en Java.

Una aplicación Java puede contener un número indefinido de clases. En cada archivo se puede almacenar una o más clases Java, aunque como mucho una con el modificador *public*. Además, el nombre de la clase con modificador *public* debe coincidir con el nombre del archivo. Por ejemplo, la clase publica "Alumno" debe almacenarse en un archivo llamado "Alumno.java". Además, en ese archivo se pueden almacenar más clases, pero sin modificador *public*.

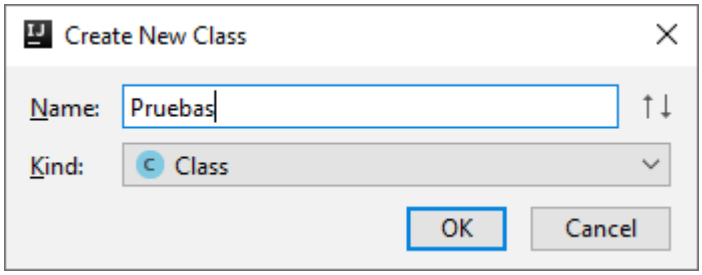
La ejecución de una aplicación Java debe comenzar por una clase que contenga un método *main*, con la siguiente cabecera:

```
public static void main (String[] args)
```

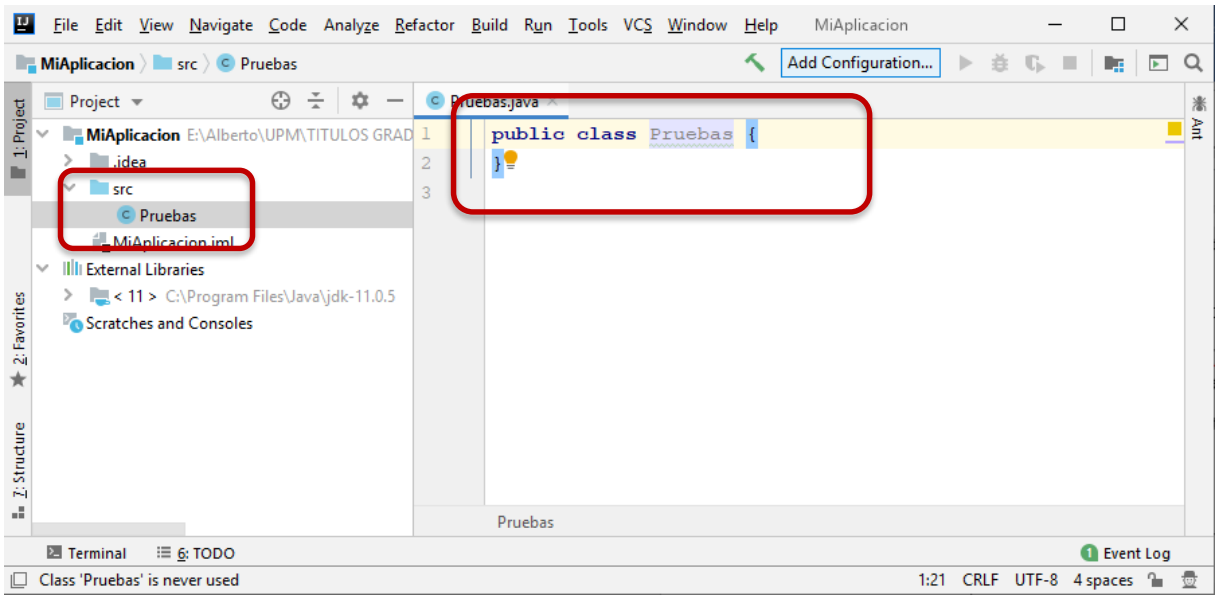
Aunque lo normal es tener un solo *main* en una aplicación Java, es posible tener varias clases con un método *main* cada una. De manera que se puede elegir qué *main* ejecutar.

A continuación, se creará en el proyecto *MiAplicacion* un archivo con una clase Java que únicamente contenga un método *main*:

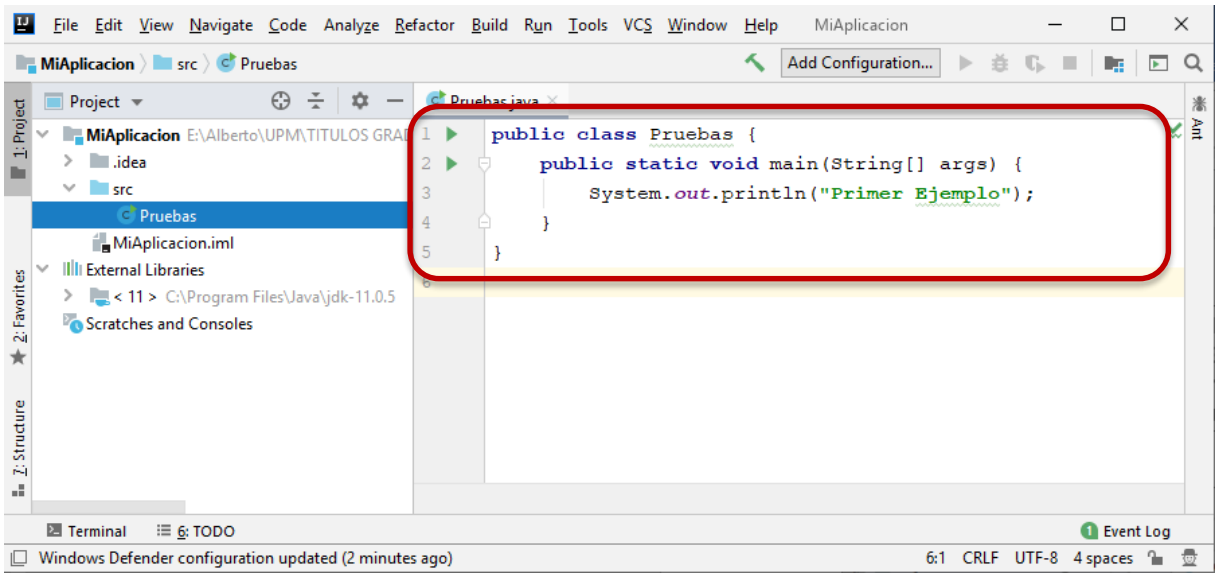
File --> New --> Java Class



Al pulsar OK, puede observarse cómo en el directorio ***src*** del proyecto aparece el archivo "Pruebas.java" con una clase vacía, cuyo nombre es precisamente "Pruebas".



Seguidamente codificamos un método *main* en la clase *Pruebas*:



Pruebe a cometer algún error al codificar *main* para ver cómo IntelliJ notifica los errores. Por ejemplo, borre algún punto y coma o algún paréntesis.

Para crear una nueva clase también se puede hacer pinchado el directorio "src" con el botón derecho del ratón. Aparece un menú contextual en el que figura la posibilidad de crear una clase nueva.

Si ahora se desease crear un nuevo proyecto: **File --> New --> Project**

Y si lo que se desease es abrir un proyecto ya creado anteriormente: **File → Open**

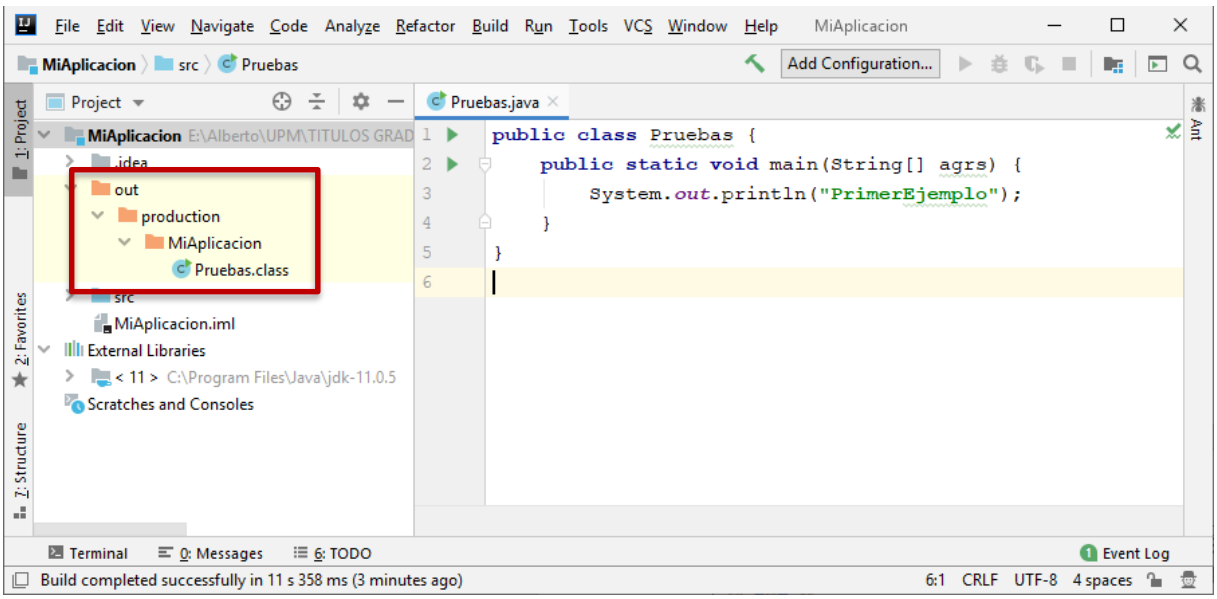
En ambos casos, *IntelliJ* da la posibilidad de que el nuevo proyecto aparezca en una ventana nueva o que sustituya al proyecto que está abierto en la ventana actual. Lo que no es posible en *IntelliJ* es tener dos proyectos abiertos en la misma ventana.

2.4. Ejecutar la aplicación del proyecto.

Para ejecutar la aplicación, en primer lugar, hay que traducir el código Java en código intermedio de *Bytecodes*. Este código intermedio es el que ejecuta la máquina virtual de Java (JVM).

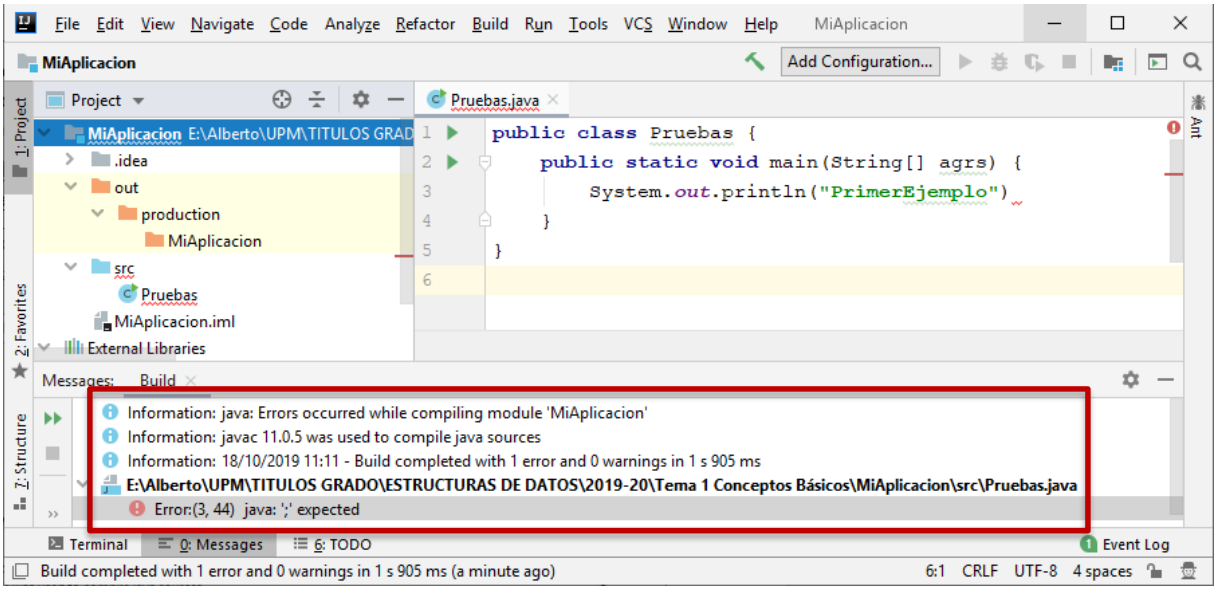
Para traducir los archivos Java a *Bytecodes* se debe realizar la operación **Build** del proyecto. Para ello:

Build --> Build Proyect



Se Puede apreciar que ha aparecido en el proyecto un nuevo directorio "out", en donde se ha creado el archivo "Pruebas.class" en formato de *Bytecodes*. Si ahora se quiere visualizar dicho archivo con IntelliJ, éste aplica automáticamente un decompilador de *ByteCodes* y se visualiza código Java.

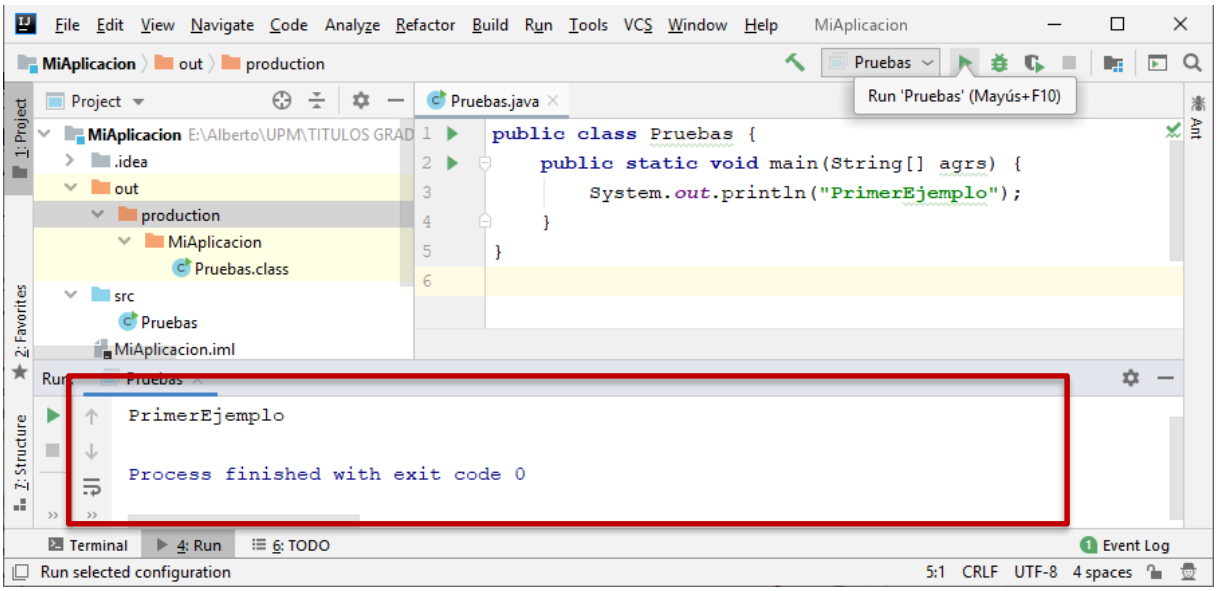
Si al hacer la operación **Build** se detecta algún error, aparece una nueva zona en la ventana notificando el error. Obviamente, en este caso no se habrán generado los archivos ".class".



Una vez realizada la operación **Build** con éxito, la máquina virtual de java ya puede ejecutar este código. Para ello, *IntelliJ* te ofrece varias posibilidades:

- Sobre la ventana con el código de "Pruebas.java", botón derecho **--> Run 'Pruebas.main()'**
- Ir a la ventaja del proyecto, botón derecho sobre el archivo Pruebas **--> Run 'Pruebas.main()'**
- Menú principal, Run --> Run 'Pruebas.main()'

En cualquier caso, el resultado de la ejecución se obtiene en la zona inferior de la pantalla:



Obviamente, el triángulo verde de la barra de herramientas en la parte superior derecha de la pantalla, también sirve para ejecutar del proyecto. Nótese que a la izquierda de dicho triángulo viene marcada la clase principal (Pruebas), es decir, la que contiene el *main* que se quiere ejecutar. Si en la aplicación hubiera varios *main*, podría elegirse aquí cuál de ellos ejecutar.

Realmente, para ejecutar el proyecto no es necesario realizar previamente de forma explícita la operación **Build**, ya que *IntelliJ* se encarga de hacerlo de forma automática cuando se solicita la ejecución del proyecto. Sí que se debe ejecutar de forma explícita **Build** cuando lo que se quiere es únicamente comprobar si el código contiene errores.

Ejercicio propuesto:

Crear un nuevo proyecto con nombre *MiAplicacion2* en el que incluyan dos archivos Java. El primero, llamado *Principal.java*, contendrá una clase *Principal* con un método *main*, encargado de leer un número de la entrada estándar, crear un objeto de la clase *Divisores* y enviarle el mensaje *visualizarDivisores* con el número leído. Este proceso se repite hasta que el usuario introduce el número 0. El contenido de *Principal* será el siguiente:

```
import java.util.Scanner;

public class Principal {
    public static void main(String[] agrs) {
        int numero;
        Divisores divisores = new Divisores();
        Scanner lectura = new Scanner(System.in);
        do {
            System.out.print(
                "Introduzca un valor entero positivo (0 para terminar): ");
            numero = lectura.nextInt();
            if (numero < 0) {
                System.out.println("Número no válido");
            } else {
                if (numero > 0) {
                    divisores.visualizarDivisores(numero);
                }
            }
        } while (numero != 0);
    }
}
```

En *Divisores.java* se definirá la clase *Divisores* con la siguiente estructura:

```
public class Divisores {
    public void visualizarDivisores(int valor) {
        // Completar método
    }
}
```

Se trata de completar el método *visualizarDivisores* para que la ejecución sea la siguiente:

```
Introduzca un valor entero positivo (0 para terminar): 18
Divisores de 18: 2 3 6 9

Introduzca un valor entero positivo (0 para terminar): 564
Divisores de 564: 2 3 4 6 12 47 94 141 188 282

Introduzca un valor entero positivo (0 para terminar): 0
```

Modifique ahora el nombre del archivo *Divisores* a *DivisoresEntero*. Para ello, seleccione el nombre del archivo en el proyecto y realice **Refactor --> Rename**, ya sea en el menú principal o en el menú contextual (botón derecho del ratón). Observe cómo cambia el nombre de la clase en todo el proyecto. Concretamente, en el código *Principal.java* se ha cambiado *Divisores* por *DivisoresEntero*.

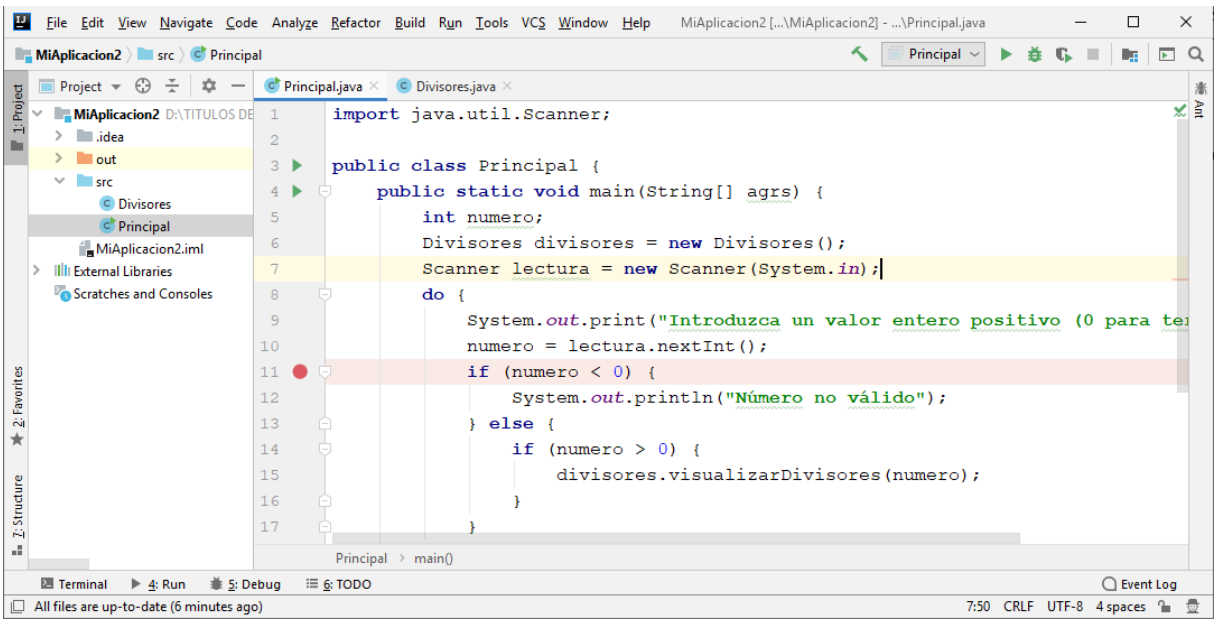
La funcionalidad *Refactor* sirve también para cambiar de forma automática el nombre de una variable o método todas las veces que aparezca. Pruebe a cambiar la variable *valor* del método *visualizarDivisores* por otro nombre. Para ello, recuerde que primero ha de seleccionar dicha variable en el código.

Descoloque ahora las tabulaciones de algunas líneas del código (introduzca y elimine tabulaciones para desordenar el código fuente). Para volver a ordenar el código de forma automática, ejecute **Code --> Reformat Code**. Como habrá observado, es una funcionalidad muy útil para clarificar el código.

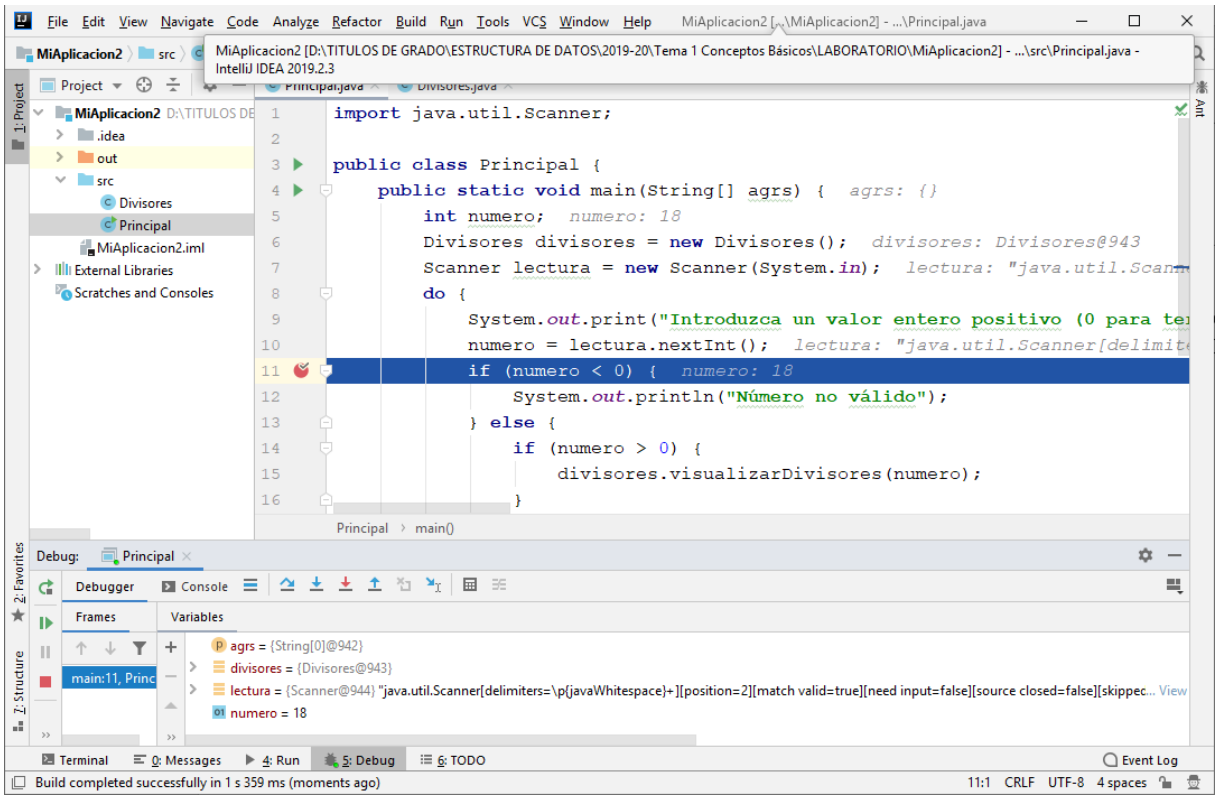
2.5. Ejecutar la aplicación en modo debug.

El objetivo ahora es ejecutar la aplicación de los divisores, pero en modo **debug**, es decir, se va a poder ejecutar paso a paso para poder ver el contenido de variables y parámetros en cualquier momento de la ejecución. Se trata de comprobar si la ejecución se realiza de la forma esperada. Lo cual puede ser muy útil cuando un código no nos funciona correctamente, pero no somos capaces de encontrar el error. Por ejemplo, imaginemos que al introducir en nuestro proyecto el número 18, nos sale como divisores el 2, el 3 y el 6; pero no nos sale el 9. Al ejecutarlo paso a paso podríamos ver cómo está funcionando el código cuando analiza el número 9.

En primer lugar, hay que marcar al menos un punto de ruptura en el fuente (punto en donde va a detener la ejecución). Concretamente, se pondrá en línea 11 del *main* (*if (numero < 0) {*). Para ello, se marcará en el margen izquierdo de dicha línea y aparecerá un punto de color rojo:



Ejecute ahora la aplicación en modo *debug* (**run --> debug 'Factorial'**, o directamente pulsar sobre el escarabajo que aparece en la barra de herramientas de la parte superior derecha). Después de pedir el número, la aplicación se para justo en el punto de ruptura:



Nótese que aparecen dos pestañas en la zona inferior de la ventana. Una para los resultados del *debugger* y otra para la consola de la aplicación. En el *debugger* se puede ver el valor de los atributos, variables y parámetros en el punto de ruptura: *args*, *divisores*, *lectura* y *número*. Los tres primeros son referencias a objetos, y el último es la variable *número* con el valor que hayamos introducido. A partir de aquí se puede ir ejecutando paso a paso utilizando las opciones del *debugger*:

- Step Over (F8): ir a la sentencia siguiente.
- Step Into (F7): ir a la sentencia siguiente. Si contiene una llamada a una función, entrar dentro de ella.
- Force Step Into: forzar entrar dentro. Llega hasta las funciones de librería.
- Step Out (Mayus + F8): salir de la función.
- Run to Cursor (Alt + F9): ir hasta donde está el cursor en el código.

También se pueden marcar más puntos de ruptura y decirle al *debugger* que vaya al siguiente mediante  .

Se le puede indicar en cualquier momento al *debugger* que termine la ejecución mediante 

Ejercicio propuesto:

Realice el *debug* de la aplicación anterior introduciendo un 18 como dato para visualizar los divisores. Realice primero la ejecución paso a paso con **Step Over (F8)**, con lo que no entrará en el método *visualizarDivisores*.

Realice nuevamente el *debug* de la aplicación con el valor 18, pero ahora utilizando la ejecución paso a paso con **Step Into (F7)**. Al entrar dentro del método *visualizarDivisores* tendrá acceso a las variables y parámetros de dicho método en todo momento. De esta manera podrá comprobar si funciona correctamente o no. Puede también pulsar sobre la pestaña *Consola* para ver los resultados que va produciendo el método.

2.6. Otras funcionalidades de IntelliJ.

Ejercicio propuesto:

Dedique unos minutos a recorrer el menú principal para ver otras posibilidades de *IntelliJ* que pueden ser interesantes. Por ejemplo, **View --> Tool Windows** permite que aparezcan o desaparezcan las diferentes zonas de la pantalla.

3. Introducción a clases y objetos en Java.

1. Crear un nuevo proyecto Java en *IntelliJ* de nombre "ED 1 Practica".
2. Definir la clase ***Alumno***.

   Cada objeto de la clase *Alumno* contendrá encapsulados los datos o información de un alumno: el nombre completo, el número de matrícula, calificación final y asignaturas en las que está matriculado. Para almacenar estos datos, se decide utilizar los siguientes **atributos** en la clase *Alumno*:

   - *nombre* de tipo *String*.
   - *matricula* de tipo *String*.
   - *calificación*: de tipo *double.*
   - *asignaturas*: *array* de String, que contendrá la lista de asignaturas en las que el alumno está matriculado. El máximo número de asignaturas será de cinco.
   - *numAsigs*: de tipo entero, indica el número de asignaturas en las que el alumno está matriculado (máximo cinco).

   La interfaz de esta clase, es decir, las operaciones que ofrece la clase *Alumno*, vienen definidas por los siguientes **métodos**:

   - Constructor sin parámetros, inicializando los atributos con los siguientes valores:
     - *nombre* y *matricula* con un *String* vacio
     - *calificación* con valor 0
     - Además, el alumno no tendrá asignaturas en las que esté matriculado, por lo que numAsigs es 0. También se debe crear el *array* de *asignaturas*, aunque no se le inserte ninguna asignatura.
   - Constructor con tres parámetros para inicializar los atributos: *nombre*, *matrícula* y *calificación*. El resto de atributos se inicializarán como en el constructor sin parámetros.
   - Método que devuelve el nombre del alumno: *String getNombre()*.
   - Método que modifica el nombre de un alumno: *void setNombre(String nom)*.
   - Método que devuelve la calificación del alumno: *String getCalificacion()*.
   - Método que modifica la calificación de un alumno: *void setCalificacion(double cal)*.
   - Método que permite añadir una asignatura al alumno, para indicar así que se ha matriculado en ella*: void matricularAsignatura(String asignatura).* Si se intenta añadir más asignaturas del máximo permitido se indicará que no es posible mediante un mensaje.
   - Método que devuelve el número de asignaturas en las que está matriculado el alumno: *int getNumAsigs()*.
   - Método *void mostrarAsignaturas()*, que visualiza en pantalla las asignaturas en las que está matriculado el alumno con el formato que indica el siguiente ejemplo:

```
2 asignaturas:
    - Estructuras de Datos.
    - Matemática Discreta.
```

Si el alumno no está matriculado en ninguna asignatura, este método visualizará el mensaje: "No está matriculado en ninguna asignatura"

- Método *void mostrarAlumno()*, que muestra por pantalla todos los datos del alumno con el formato que indica el siguiente ejemplo:

```
María García García. Matr:zz1234 (9.0)
2 asignaturas:
    - Estructuras de Datos.
    - Matemática Discreta.
```

3. Crear la clase ***Principal*** con un método ***main*** en el que se realicen las siguientes tareas:
   - Instanciar 4 objetos de la clase *Alumno* con los siguientes datos:

	nombre	matrícula	calificación
0	Felipe Arias González	aa1253	3.50
1	Manuel García Sacedón	ax0074	8.35
2	Margarita López Medina	mj7726	7.70
3	Estela Sánchez Arellano	bc2658	6.75

   - Matricular a los cuatro alumnos en "Estructuras de Datos". Además, a "Estela Sánchez Arellano" en "Algebra" y "Estructuras de Computadores".
   - Mostrar las asignaturas de "Estela Sánchez Arellano".
   - Mostrar los datos de "Felipe Arias González" y de "Estela Sánchez Arellano".
4. Definir la clase ***GrupoAlumnos***.

   Cada objeto de la clase *GrupoAlumos* contendrá encapsulados los datos de un grupo de clase de alumnos: el nombre del grupo, la lista de alumnos que actualmente pertenecen al grupo (cada alumno con los datos descritos en la clase *Alumno*), el número de alumnos que actualmente componen el grupo y el máximo número de alumnos que caben en ese grupo. Para almacenar estos datos, se decide utilizar los siguientes **atributos** en la clase *GrupoAlumnos*:

   - *nombre* de tipo *String*, que indicará el nombre del grupo.
   - *listaAlumnos* de tipo *array* de objetos *Alumno*, que contendrá los alumnos que componen el grupo.
   - *numAlumnos*, entero que indica cuantos alumnos hay en el grupo actualmente.
   - *máximo*, entero que indica el máximo de alumnos que puede haber en el grupo.

   La interfaz de esta clase, es decir, las operaciones que ofrece la clase *GrupoAlumnos*, vienen definidas por los siguientes **métodos**:

   - Constructor sin parámetros que inicializará un objeto *GrupoAlumnos* de *nombre* "GrupoDesconocido" y valor *máximo* de 10 alumnos. Además, el grupo no tendrá ningún alumno. También se debe crear el *array* de *listaAlumnos*, aunque no se le inserte ningún *Alumno*.
   - Constructor con dos parámetros para inicializar el tamaño máximo del grupo y el nombre del grupo. El resto de atributos se inicializarán como en el constructor sin parámetros.
   - Método que devuelve el nombre del grupo: *String getNombre()*.

- Método que modifica el nombre del grupo: *void setNombre(String nom)*.
- Método que devuelve el máximo de alumnos que puede haber en el grupo: *int getMaximo()*.
- Método que permite insertar a un alumno en el grupo *boolean insertarAlumno(Alumno alumno)*. Si el grupo está lleno, este método no inserta al alumno y devuelve *false*. En cambio, si el alumno se inserta correctamente, este método devuelve *true*.
- Método que devuelve el número de alumnos que pertenecen al grupo actualmente: *int getNumAlumnos()*.
- El método *Alumno getAlumno(int i)*, que devuelve el alumno que se encuentra en la posición *i*. Si no existe alumno en dicha posición, devolverá *null*.
- El método *void mostrarGrupo()*, que muestra en pantalla el nombre del grupo y los datos de todos los alumnos que lo componen, según el formato que se indica en el siguiente ejemplo:

```
GRUPO GX102: 2 alumnos
Estela Sánchez Arellán. Matr: bc2658 (6.75)
     No está matriculado en ninguna asignatura
María García García. Matr:zz1234 (9.0)
2 asignaturas:
     - Estructuras de Datos.
     - Matemática Discreta.
```

5. Realizar las siguientes tareas en el método ***main*** de la clase *Principal*:
   - Instanciar un objeto *grupoAlumno* con un máximo de 20 alumnos y con nombre *"GX11"*.
   - Insertar en este grupo todos los alumnos instanciados en el punto 3.
   - Mostrar en pantalla los datos del grupo.
   - Mostrar en pantalla los datos del alumno que ocupe la posición 1 en el grupo.
6. Añadir a la clase ***GrupoAlumnos*** los siguientes métodos:
   - *double mediaCalif()*, que devuelve la media de las calificaciones de los alumnos del grupo. En caso de que el grupo esté vacío, este método devuelve el valor -1.

     Desde el *main* de la clase *Principal*, visualizar la calificación media del grupo. Si el grupo no tiene alumnos visualizará un mensaje indicativo.
   - *Alumno mejorAlumno()*, que devuelve una referencia al alumno del grupo que tiene la máxima calificación. Si hay más de uno, devuelve el primero de la lista con esa calificación máxima. En caso de que el grupo esté vacío, este método devuelve null.

     Desde el *main* de la clase *Principal*, mostrar los datos del alumno que tiene la máxima calificación del grupo. Si el grupo no tiene alumnos visualizará un mensaje indicativo.
   - *boolean eliminarAlumno(String nombreAlumno)*, que borra del grupo al alumno cuyo nombre se corresponde con el parámetro. En caso de no existir el alumno, este método deja al grupo como estaba y devuelve *false*. Si el alumno se encuentra en el grupo, el método actualiza las referencias de *listaAlumnos* para no dejar huecos, y devuelve *true*.

     Desde el *main* de la clase *Principal*, eliminar del grupo a "Manuel García Sacedón" y a "Felipe Segovia López". En ambos casos, si la eliminación es correcta, mostrar el contenido el grupo; y en caso contrario, mostrar un mensaje indicativo.

7. Revise las clases *Alumno* y *GrupoAlumnos* para practicar con la referencia ***this***. Por ejemplo:
    1. En los constructores y/o métodos que crea oportuno, llamar igual a parámetros que a atributos. Resuelva la ambigüedad con *this*.
    2. Cuando en el código se envía un mensaje al propio objeto, utilizar de forma explícita *this*.
    3. En cualquiera de las dos clases, definir el constructor sin parámetros utilizando el constructor con parámetros.

Trasladar un proyecto de una plataforma a otra.

Cuando se quiere seguir trabajando con un proyecto de *IntelliJ* en otro ordenador, es habitual que aparezca un error debido a la versión de JDK. Esto es debido a que el proyecto ha sido configurado para una versión de JDK, mientras que la nueva máquina tiene instalada una versión distinta (“**Project SDK is not defined”**). Este problema es muy fácil de solucionar. A continuación, se proponen dos métodos diferentes:

1. Junto al mensaje de error “Project SDK is not defined” aparece un enlace “**Setup SDK**”. Al pulsarlo, aparecen los JDK de los que tiene constancia *IntelliJ*. Se elige uno cualquiera y el problema queda resuelto. En caso de que no aparezca ninguno, se definirá según el segundo método.
2. En la barra de menú de *IntelliJ*, pulsar ***File --> Project Structure***. En ***Project Settings***, ***Project***, existe una lista desplegable para ***Project SDK***. En ella aparecen los JDK de los que tiene constancia *IntelliJ*, se seleccionará uno cualquiera y el problema quedará resuelto.

   En caso de que *IntelliJ* no tenga constancia de ningún JDK, se pulsará el botón ***New*** para definirle uno. Se selecciona la opción JDK, se le indica la ubicación del JDK en el sistema de archivos, por ejemplo, C:\Program Files\Java\jdk-11.0.5 y se pulsa OK.