Estructuras de Datos – Curso 2025/26
Grados en Ingeniería de Computadores y en TSI.

PRÁCTICA 1.

1. OBJETIVOS:

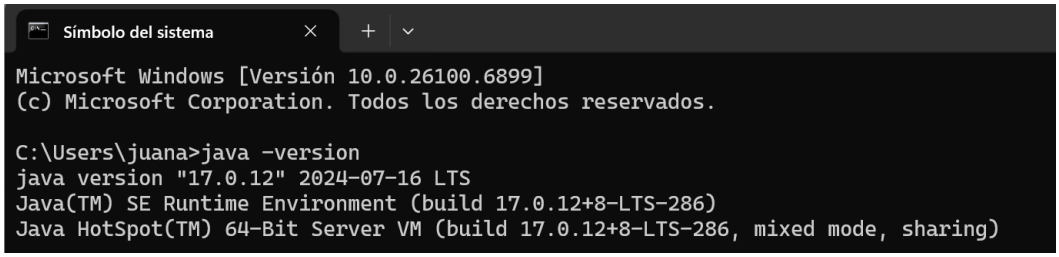
- Familiarizarse con el Entorno de Desarrollo Integrado *IntelliJ IDEA*.
- Primera toma de contacto con el lenguaje Java.
- Desarrollo de una aplicación básica para la gestión de notas de los alumnos de una asignatura ("Estructuras de datos")

2. El IDE IntelliJ IDEA

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

2.1. Instalación de IntelliJ IDEA.

En primer lugar, hay que tener instalado un Kit de desarrollo para Java (JDK). Para seguir los ejemplos y prácticas de la asignatura es válida cualquier versión desde JDK 8 a JDK 23. Para saber si tienes instalado Java, ejecuta el comando ***java -version*** en una consola (si tienes Windows, puedes abrir una consola con *cmd*).



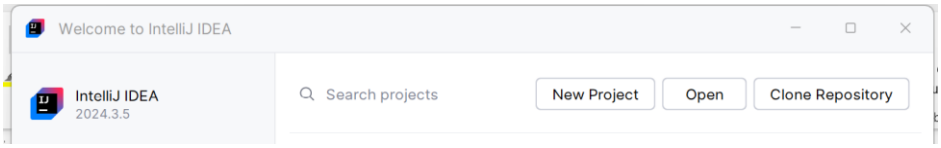
En caso de no tener ninguna versión de Java instada, puedes descargártela de la web de Oracle: https://www.oracle.com/java/technologies/downloads/.

A continuación, se instalará el IDE de IntelliJ, que se puede obtener en el siguiente enlace: https://www.jetbrains.com/idea/download.

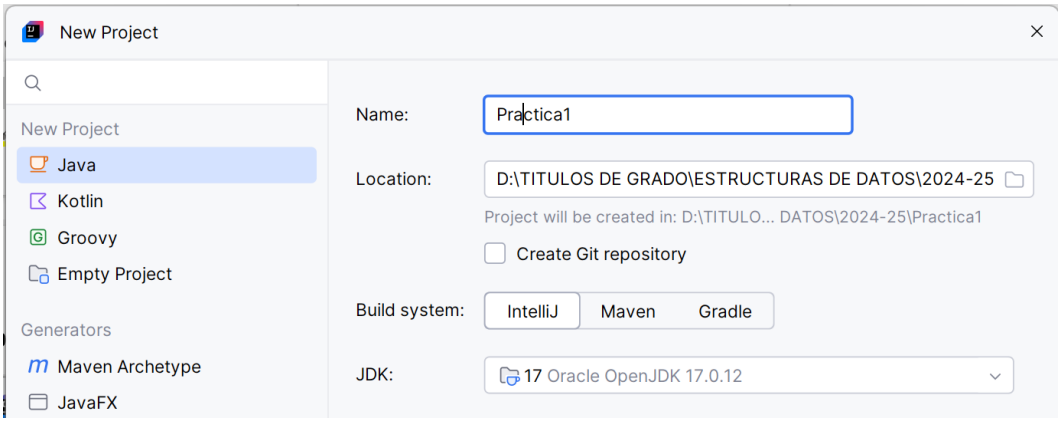
2.2. Crear un proyecto.

Para programar una aplicación Java con *IntelliJ*, siempre debe hacerse en el contexto de un **proyecto**. Un proyecto está compuesto por distintos tipos de ficheros: fuentes de java (.java), ficheros compilados java (.class), ficheros de configuración, etc. Todos estos ficheros se encuentran dentro de un mismo directorio o carpeta. Cuando se quiera abrir un proyecto en IntelliJ, se debe indicar cual es el directorio asociado.

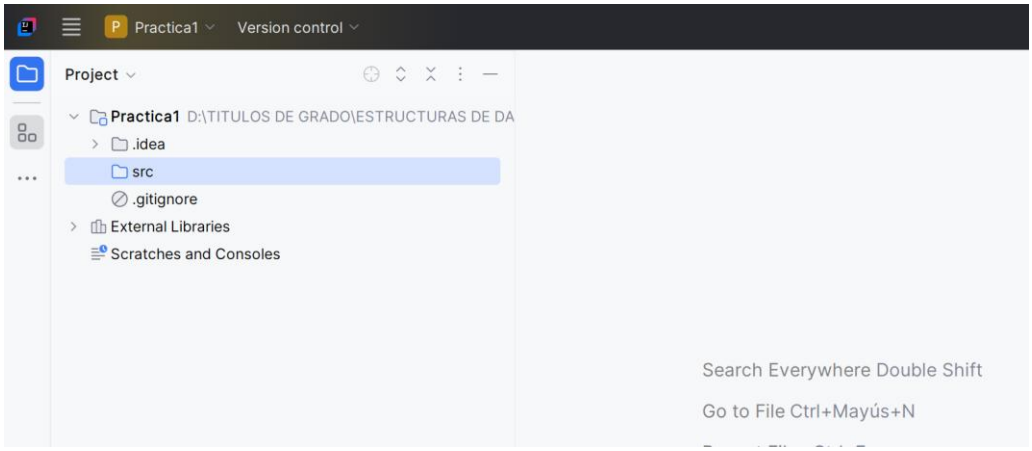
La primera vez que arrancamos IntelliJ, nos da la opción de crear un proyecto nuevo o cargar uno ya creado:



Pulsamos “New Proyect” para crear un nuevo proyecto:



Indicamos el nombre del proyecto y la localización donde se creará el directorio asociado al proyecto. Indicamos también que sea IntelliJ quien se encargue de las tareas de compilación, y seleccionamos el JDK que utilizaremos (de entre los que tenemos instalados).



- En la parte superior tenemos una barra de herramientas en la que podemos desplegar el menú con todas las opciones de IntelliJ.
- En la parte izquierda tenemos la estructura del proyecto. Cabe destacar la carpeta “src” (sources) en donde debemos meter todos los ficheros Java. Puedes abrir el mismo directorio en el explorador del sistema operativo y verás que tiene la misma estructura que te muestra IntelliJ.
- En la parte derecha tendremos la ventana de edición cuando estemos trabajando con un fichero Java.

2.3. Crear un archivo con una clase Java.

En cada archivo se puede almacenar una o más clases Java, aunque habitualmente meteremos sólo una. En ese caso, el nombre de la clase debe coincidir el nombre del archivo. Por ejemplo, la clase "Alumno" debe almacenarse en un archivo llamado "Alumno.java".

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

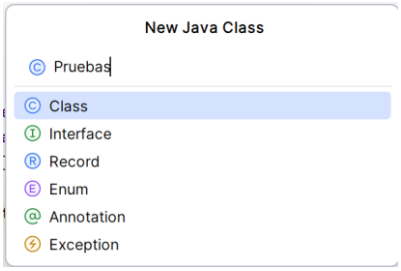
```
public static void main(String[] args)
```

No es necesario memorizar toda esta cabecera. Si a IntelliJ le ponemos psvm (siglas de public static void main), automáticamente saca esta cabecera.

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 *Practica1* un archivo con una clase Java que únicamente contenga un método *main*. Para ello, hay dos opciones.

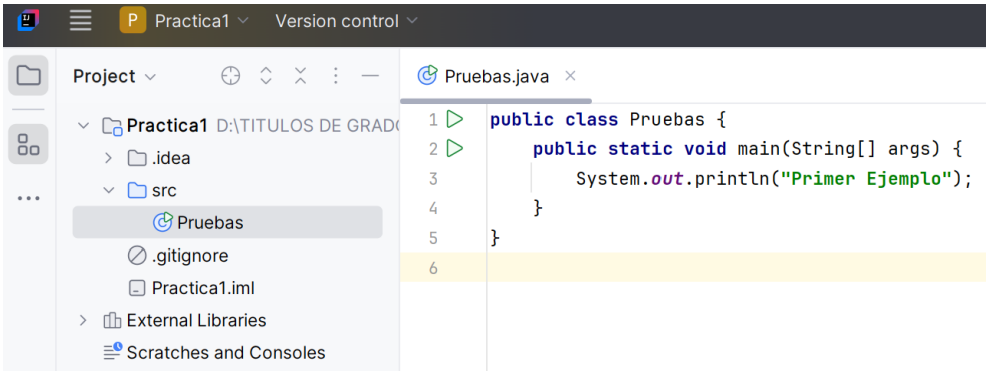
1. Menú: File --> New --> Java Class.
2. Pinchar con el botón derecho del ratón la carpeta "src" --> New --> Java Class



Le indicamos el nombre de la clase, este caso Pruebas. Recuerda que los nombres de las clases en Java deben empezar siempre por letra mayúscula.

Al pulsar Intro puede observarse cómo en el directorio "src" del proyecto aparece el archivo "Pruebas" con una clase vacía cuyo nombre es *public class Pruebas*. Es fundamental que el archivo creado quede centro de la carpeta "src". De no ser así, lo hemos creado mal; borrarlo y volver a crearlo correctamente, ya que no sería considerado como un archivo fuente en Java.

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



Prueba a cometer algún error al codificar el *main* para ver cómo *IntelliJ* te notifica los errores. Por ejemplo, borra un punto y coma o algún paréntesis.

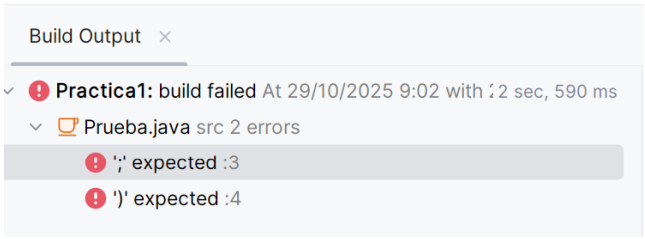
Fíjate en el menú **File** de IntelliJ. Ahí tenemos opciones para crear nuevos proyectos, clases y ficheros de otros tipos. También podemos abrir un proyecto ya creado. Recuerde que para abrir un proyecto hay que indicar el directorio. En este menú **File** podemos también cerrar un proyecto o ver los proyectos con los que hemos trabajado recientemente.

En *IntelliJ* se pueden tener abiertos varios proyectos a la vez (cada uno en una ventana)

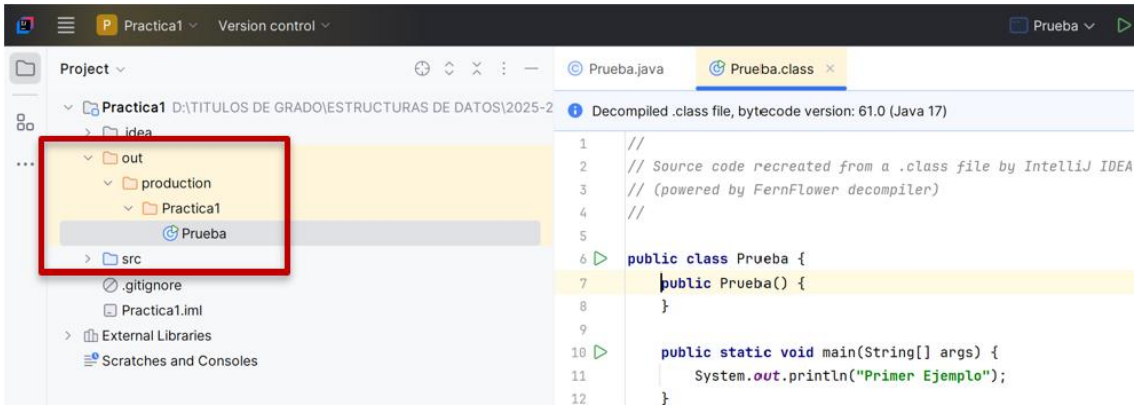
2.4. Ejecutar la aplicación.

Para ejecutar la aplicación podemos utilizar el icono con un triángulo verde, ya sea en la barra de herramientas o en el código de la clase que contiene el *main* (en la imagen anterior puedes ver que a la izquierda del método *main* aparece dicho triángulo verde). También puedes utilizar el menú de *IntelliJ*. Por ejemplo, ***Run --> Run 'Prueba.java'*** para ejecutar el *main* que contiene el archivo *Prueba.java*. En caso de que tuvieras varios *main* en el mismo proyecto o aplicación, tendrías que tener cuidado cual ejecutas.

Si hay algún error en el código de algún archivo Java del proyecto, la compilación (*build*) no tiene éxito. En este caso, *IntelliJ* ofrece un listado de dichos errores, incluyendo el número de línea:



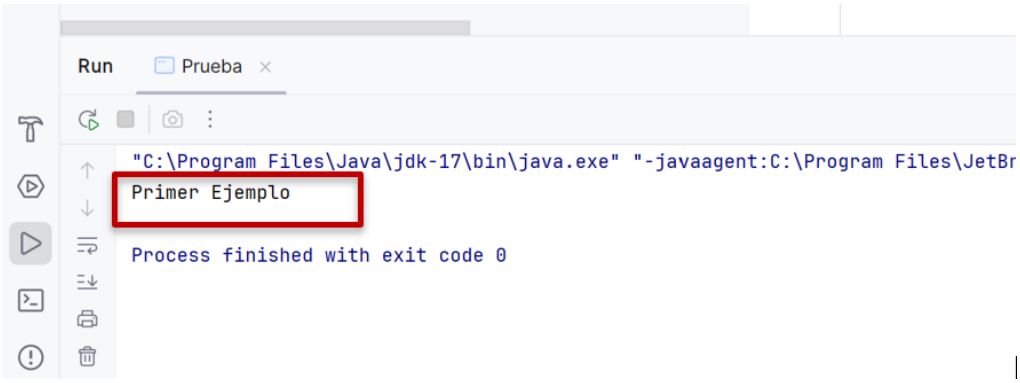
Pero si todo es correcto, podrás ver que el resultado de la compilación son los archivos *.class* con el código intermedio en *ByteCodes*. Concretamente, se encuentran en el directorio "out" del proyecto, en la carpeta "production":



Fíjate que el archivo *Prueba.class* se encuentra decompilado para que puedas ver su contenido, pero no intentes editarlo, ya que el contenido real es ininteligible (*ByteCodes*). Para comprobarlo, puedes intentar visualizarlo directamente desde el explorador del sistema operativo.

Este código en *ByteCodes* será el que se ejecuta en la máquina virtual de Java (JVM) de tu ordenador (a través del *IntelliJ*). También podrías obtener un archivo de tipo *.jar* con todos los archivos *.class* de un proyecto, de manera que ese archivo se podría ejecutar en cualquier máquina (con cualquier sistema operativo y procesador) que tuviera instalada la máquina virtual de Java. Si estás interesado, puedes consultar en internet cómo generar el archivo .jar de un proyecto usando *IntelliJ*.

Además de compilar todos los archivos Java, la operación *Run* nos ejecuta el proyecto y nos ofrece el resultado en la consola que nos abre en la parte inferior:



También podemos compilar los fuentes de nuestro proyecto sin ejecutarlos. Para ello, seleccionamos en el menú ***Build --> Build Proyect***. Obtendríamos los posibles mensajes de error, o los archivos *.class* si todo es correcto.

2.5. Ejercicio propuesto.

Crear una nueva clase ***Ejercicio*** en el proyecto "Práctica1", y añadirle el siguiente método *main*:

```
import java.util.Scanner;

public class Ejercicio {
    public static void main(String[] agrs) {
        int numero;
        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) {
                    visualizarDivisores(numero);
                }
            }
        } while (numero != 0);
    }
```

Este código lo que hace es pedir un número entero al usuario por el teclado y llamar a *visualizarDivisores* con dicho número como parámetro. Este proceso se repite hasta que el usuario introduce 0 como número. Además, si se introduce un número negativo se visualiza un mensaje para indicar que el número no vale.

El ejercicio consiste en definir la función *visualizarDivisores* a continuación del *main* que sea determine y visualice los números que son divisores del número recibido como parámetro (excluyendo el 1 y el propio número).

```
public static void visualizarDivisores(int valor) {
        // Completar la función

}
```

Se pretende conseguir el siguiente comportamiento:

```
Introduzca un valor entero positivo (0 para terminar): -5
Número no válido
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
```

2.6. Errores en ejecución.

Ejecuta el *main* de la clase *Ejercicio* e introduce por teclado algo que no sea un número. Verás que se produce un error durante la ejecución del programa y **que éste se para**. Concretamente, el resultado que se obtiene será similar al siguiente:

```
Introduzca un valor entero positivo (0 para terminar): hola
Exception in thread "main" java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at Ejercicio.main(Ejercicio.java:10)
```

Se ha producido un error de ejecución (en Java se dice que se ha producido una excepción) porque no sabe cómo tratar la entrada "hola". Lo interesante es que nos informa del punto de ejecución donde se ha producido el error. En concreto, en la línea 10 de la función main de Ejercicio.java. Si pinchamos nos lleva al método *nextInt*, que no ha sido capaz de obtener un entero.

El conocer el motivo de error durante la ejecución de nuestro programa puede ser muy importante para poder mejorar nuestro programa y que sepa responder ante situaciones de error. Así evitaremos que se pare de forma anticipada.

2.7. Algunas funcionalidades interesantes de IntelliJ.

Podemos cambiar el nombre de una variable de forma automática, es decir, la cambio una vez y el IDE se encarga de cambiarla en todos los sitios en donde aparezca. Para ello, nos posicionamos en su nombre, pulsamos el botón derecho y seleccionamos ***Rename*** en el menú. Pruebe a cambiar el nombre de la variable *numero* del *main*.

De esta manera podemos cambiar también el nombre de una función o el nombre de una clase. Pruebe a cambiar el nombre de la función *visualizarDivisores*.

Otra funcionalidad interesante es la que me permite organizar automáticamente las tabulaciones del código de para poder verlo de forma clara. Esto se consigue en el menú principal ***Code -> Reformat Code.***

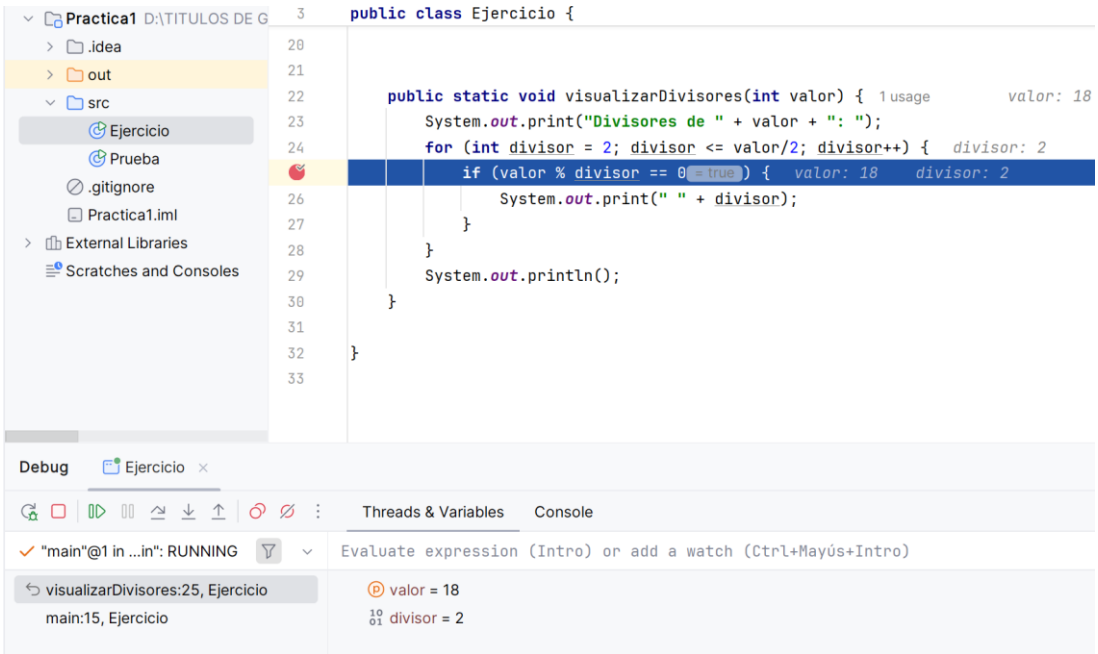
2.8. Ejecutar la aplicación en modo debug.

El objetivo ahora es ejecutar la aplicación de los divisores, pero en modo **debug**, es decir, se ejecutará paso a paso e iremos viendo el contenido de variables y parámetros. 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.

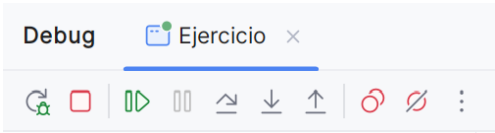
En primer lugar, hay que marcar al menos un punto de ruptura en el fuente (punto en donde va a detener la ejecución). En este caso, como queremos probar visualizarDivisores, pondremos el punto de ruptura en la primera sentencia del for. La idea es que me vaya diciendo el valor de las variables cada vez que pase por este punto. Marcamos en el margen izquierdo de dicha línea y aparece un punto de color rojo:

```
22      public static void visualizarDivisores(int valor) {  1 usage
23          System.out.print("Divisores de " + valor + ": ");
24          for (int divisor = 2; divisor <= valor/2; divisor++) {
25              if (valor % divisor == 0) {
26                  System.out.print(" " + divisor);
27              }
28          }
29          System.out.println();
30      }
```

Ejecuta ahora la aplicación en modo *debug* (**run --> debug 'Principal'**), o directamente pulsa sobre el escarabajo que aparece en la barra de herramientas. 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* (*Threads & Variables*) y otra para la consola de la aplicación (*Console*). En la primera se puede ver los datos *valor y divisor*. A partir de aquí se puede ir ejecutando paso a paso utilizando las opciones del *debuger*:



- Resume program (F9) para avanzar al siguiente punto de ruptura. Como tenemos sólo un punto de ruptura y está al comienzo del bucle, lo que va a pasar es que avanza a la siguiente iteración del bucle. Podemos ir viendo cómo evolucionan *valor* y *divisor*.
- Step Over (F8): dar un paso en la ejecución (una sentencia). Si hay una llamada a una función, la considera como un solo paso.
- Step Into (F7): dar un paso en la ejecución, pero si hay una llamada a una función va dando los pasos (sentencias) que tenga esta llamada.
- …

Prueba a hacer *debug* con el número 18. Imagínate que has hecho mal el bucle de la función *visualizarDivisores* y que has puesto < *valor/2*, en lugar de <= *valor/2*. Aplica los comandos del *debug* hasta que te des cuenta del error (nunca intenta ver si el 9 es divisor).

2.9. Trasladar un proyecto de una plataforma a otra.

Cuando se quiere seguir trabajando con un proyecto de *IntelliJ* en otra máquina, basta con llevarnos el directorio que contiene dicho proyecto. Sin embargo, es habitual que aparezca un error relativo a la versión de JDK. Esto es debido a que el proyecto ha sido configurado para una versión de JDK, y la nueva máquina no conoce dicha versión (“**Project JDK is not defined”**). Este problema es muy fácil de solucionar. A continuación, se proponen dos métodos:

1. Junto al mensaje de error “Project JDK is not defined” aparece un enlace “**Setup SDK**”. Al pulsarlo, aparecen los JDK de los que tiene constancia *IntelliJ*. 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 ***SDK***. En ella aparecen los JDK de los que tiene constancia *IntelliJ*, se seleccionará uno cualquiera. Cuidado en ***Language Level*** no vayas a seleccionar una versión superior (ante la duda elije ***SDK default***)

   En caso de que *IntelliJ* no tenga constancia de ningún JDK, en **Platform Settings, SDKs** se podrá añadir (+) un JDK ya instalado, o un instalar un nuevo JDK.

3. Aplicación para la gestión de notas en ED.

En este ejercicio se trata de realizar una pequeña aplicación para la gestión de las calificaciones en la asignatura de "Estructuras de Datos". De esta manera, además de realizar un primer ejercicio sobre programación orientada a objetos, nos ayudará a entender mejor la forma de evaluación de esta asignatura.

3.1. Definir un TAD Alumno.

Empezaremos abriendo el proyecto "Gestion Notas ED" que tenemos en Moodle y crearemos una clase ***Alumno*** para gestionar las calificaciones de un alumno. El objetivo que nos proponemos es crear un TAD Alumno que nos proporcione el siguiente interfaz:

- *public Alumno(String nombre, String matricula)*
- *public String getNombre()*
- *public String getMatricula()*
- *public void calificarParcial1(double nota)*
- *public void calificarPractica2(double examen, double entrega)*
- *public void calificarParcial2(double nota)*
- *public void calificarPractica3(double examen, double entrega)*
- *public double notaTeoría()*
- *public double notaPráctica2()*
- *public double notaPráctica3()*
- *public double notaPrácticas()*
- *public double notaFinal()*
- *public void mostrarDetalleNotas()*

Con este objetivo, empezamos definiendo los atributos en la clase *Alumno*:

- *nombre* de tipo *String*
- *matricula* de tipo *String*
- *parcial1* y *parcial2* de tipo *double* para las notas de ambos parciales.
- *examenPr2* y *entregaPr2* de tipo *double* para el examen y la entrega de la práctica 2.
- *examenPr3* y *entregaPr3* de tipo *double* para el examen y la entrega de la práctica 3.

Implementamos ahora el constructor, en el que recibimos el nombre y la matrícula del alumno. Inicializamos los atributos para un alumno que todavía no se ha evaluado de nada.

Los 6 primeros métodos de la interfaz (*getNombre*, *getMatricula*, *calificarParcial1*, *calificarPractica2*, *calificarParcial2* y *calificarPractica3*) son inmediatos. Sólo hay que asignar la/s nota/s a el/los atributo/s correspondiente/s.

Para el método *notaTeoría()* obtenemos la calificación media de los dos parciales. Sólo hay que tener en cuenta que ambos tienen el mismo peso.

Para el método *notaPractica2()* consideraremos las calificaciones de *examenPr2* y *entregaPr2*. La nota se obtiene con un 60% de peso para el examen y un 40% para la entrega. Además, hay que tener en cuenta que si la calificación del examen es inferior a 5, entonces no se contabilizará la nota de la entrega.

Para el método *notaPractica3()* las mismas consideraciones.

En el método *notaPracticas()* sólo hay que tener en cuenta que la práctica 2 y la práctica 3 tienen el mismo peso.

En el método *notaFinal()* obtendremos la calificación final del alumno en la asignatura. Esta se calcula como el 70% de la nota de teoría y el 30% de la nota de prácticas. Hay que tener en cuenta que en esta asignatura es necesario al menos un 3.5 en la nota de teoría y un 3.5 en la nota de prácticas. Si no se cumple, su nota final no podrá ser superior a 4.0.

Finalmente, el método *mostrarDetalleNotas()* visualiza los datos y las notas del alumno con el siguiente formato:

```
Felipe Arias González (aa1253)  NOTA FINAL: 7.481
    Teoría: 7.55
        Parcial 1: 9.0
        Parcial 2: 6.1
    Prácticas: 7.32
        Práctica 2: 5.84
            Examen: 5.0
            Entrega : 7.1
        Práctica 3: 8.8
            Examen: 10.0
            Entrega : 7.0
```

Prueba el correcto funcionamiento del TAD *Alumno* construido. Para ello, utiliza el método *main* de la clase *Pruebas* en donde se encuentra creado el objeto Alumno correspondiente al ejemplo anterior.

Cambia las notas de *Felipe* en el *main* para comprobar que tu TAD funciona correctamente.

3.2. Definir un TAD Asignatura.

En el mismo proyecto crea ahora una clase *Asignatura* donde gestionaremos las calificaciones de todos los alumnos de una determinada asignatura. Más concretamente, la interfaz que nos proporcionará este TAD será la siguiente:

- *public Asignatura(String nombre, int maxAlumnos)*
  Se crea una asignatura con ese nombre y con capacidad para tener un máximo de alumnos (*maxAlumnos*)
- *public Asignatura(String nombre)*
  La asignatura puede llegar a tener un máximo de 50 alumnos
- *public boolean insertar(Alumno alumno)*
  Inserta un alumno en la asignatura. Si no cabe, devuelve false.
- *public int getNumAlumnos()*
- *public void verCalificaciones()*
  Muestra la calificación final de todos los alumnos de la asignatura.
- *public int aprobados()*
  Devuelve el número de aprobados en la asignatura
- *public double porcentajeAprobados()*
- *public double mediaCalificaciones()*
- *public Alumno alumnoMayorNota()*
- *public Alumno[] alumnosAprobados()*
- *public Alumno getAlumno(String matricula)*

Con este objetivo, comenzamos definiendo los atributos en la clase *Asignatura*:

- *nombre* de tipo *String*, para almacenar el nombre de la asignatura.

- *alumnos* que es un array de objetos *Alumno*. En él meteremos a todos los alumnos de la asignatura.
- *numAlumnos* de tipo int, para guardar el número de alumnos que tiene la asignatura.

Para implementar el primer constructor, inicializaremos los atributos de forma que la asignatura creada no tendrá ningún alumno, pero tendrá capacidad para almacenar un máximo de Alumnos *maxAlumnos*.

El segundo constructor es igual que el primero, pero *maxAlumnos* es 50.

Para el método *insertar* miraremos primero que haya sitio en el array *alumnos* para meter el nuevo alumno. De ser así, lo introduciremos en la primera posición libre del array.

El método *getNumAlumnos* sólo tiene que devolver el correspondiente atributo.

En el método *verCalificaciones* visualizaremos el nombre de la asignatura y el número de alumnos. Además, recorreremos el array *alumnos* (hasta donde indique el atributo *numAlumnos*) y, para cada alumno, sacaremos su nombre, su matrícula y su nota final. Más concretamente, el resultado de este método tendrá el siguiente aspecto:

```
Estructuras de Datos: 11 alumnos
    Pedro García Herrera (ab1234) NOTA: 7.307
    Ana Díaz López (ac0987) NOTA: 0.825
    Eusebio Alcazar de los Santos (ba0333) NOTA: 8.083
    Tatiana López Gómez (bb0955) NOTA: 4.0
    Sara Gómez Baños (bc0341) NOTA: 6.465
    Jorge Luis Becerra Rico (ah6351) NOTA: 5.635
    Adolfo Velasco Sánchez (bk0383) NOTA: 6.505
    Ana Belén Aguilar Alvaro (bl0611) NOTA: 4.0
    Carlos de la Torre Rodríguez (at1111) NOTA: 4.3
    Eva García de la Hoz (aa8888) NOTA: 0.0
    Sonia Olmos de Frutos (ay0088) NOTA: 4.0
```

3.3. Aplicación principal (main).

El proyecto “Gestion Notas ED” tiene un fichero de texto “ED.txt”, donde están almacenados todos los alumnos de la asignatura con el siguiente formato:

```
Pedro García Herrera, ab1234
6.5, 6, 8.2, 7.4, 9, 10
Ana Díaz López, ac0987
2.1, 1, 6, 0, 0, 0
Eusebio Alcazar de los Santos, ba0333
9, 10, 10, 6.5, 8, 7.3
```

En una línea están los datos de un alumno (nombre y matrícula) y en la línea siguiente se encuentran las notas que ha obtenido ese alumno en el curso:

- La primera es la nota del primer parcial.
- La segunda y la tercera son las notas de la práctica 2 (examen y entrega)
- La cuarta es la nota del segundo parcial
- La quinta y la sexta son las notas de la práctica 3 (examen y entrega)

Disponemos ya construida de una clase *Fichero*, cuya interfaz es la siguiente:

- *public Fichero(String ruta)*
  Inicializa el fichero indicado en *ruta* para lectura.

- *public String[] leerDatos()*
  Lee la siguiente línea de datos (nombre y matrícula) del fichero
  Devuelve un array con dos *String*, conteniendo el nombre y la matrícula.
  Si el fichero se ha acabado o hay un error de lectura, devuelve *null*.
- *public double[] leerNotas()*
  Lee la siguiente línea con las notas de un alumno del fichero
  Devuelve un array con seis *double*, conteniendo dichas notas.
  Si el fichero se ha acabado o hay un error de lectura, devuelve *null*.

Crear una nueva clase *Aplicacion* e incluir en ella un método *main* que realice lo siguiente:

- Abrir el fichero ED.txt utilizando un objeto *Fichero*.
- Crear un objeto *Asignatura* para "Estructuras de Datos" con un máximo de 100 alumnos.
- Leer los datos y las notas de todos los alumnos que contenga el ED.txt, e insertar un objeto *Alumno* en la *Asignatura* por cada uno de dichos alumnos. El proceso termina cuando una operación de lectura devuelva *null*.
- Visualizar los datos de la asignatura con *verCalificaciones*. Pero sólo en caso de que en la asignatura haya al menos un alumno.

3.4. Terminar la clase Asignatura y hacer pruebas en el main

Para el método *aprobados* recorreremos todo el array *alumnos* (hasta *numAlumnos*) y contabilizamos los que tegan una nota final >= 5.0.

El método *porcentajeAprobados* es inmediato utilizando el método *aprobados* y el atributo *numAlumnos*.

Probar ambos métodos en el *main* con la asignatura creada.

Para hacer el método *mediaCalificaciones*, recorremos todo el array *alumnos* y vamos sumando las notas de todos. Finalmente, devolvemos la media. Suponer que la asignatura tiene al menos un alumno (no comprobar).

Probar el método mediaCalificaciones en el *main*.

En el método *alumnoMayorNota* recorremos también el array *alumnos* para devolver el alumno que tenga la nota final más alta. Suponer también que cuando se llama a este método la asignatura tiene al menos un alumno (no comprobar)

Probar el método *alumnoMayorNota* en el *main*. Visualizar el nombre del alumno que tenga la mayor nota y su nota final.

Para hacer el método *alumnosAprobados*, creamos en primer lugar un array de objetos *Alumno* con un tamaño igual al número de aprobados (recuerda que tenemos ya un método que nos dice cuántos son). A continuación, recorremos el *array alumnos* para ir metiendo en nuestro array aquellos que tengan una nota final >= 5.

Prueba el método *alumnosAprobados* en el *main*. Para cada uno de los alumnos obtenidos con este método, visualiza el detalle de todas sus notas mediante el método *mostrarDetalleNotas*.

Finalmente, el método *getAlumno* nos tiene que devolver el objeto *Alumno* que tiene una determinada matrícula. Si en la asignatura no existe ningún alumno con esa matrícula, el método devolverá *null*. Nuevamente, para hacer este método recorremos el array *alumnos* buscando la matrícula. La búsqueda finaliza cuando encontramos al alumno o cuando se terminan los alumnos en el array.

Probar el método *getAlumno* en el *main*. Pide el alumno asociado a una matrícula y muestra los detalles de sus calificaciones (*mostrarDetalleNotas*). Si no existe el alumno en la asignatura, muestra un mensaje indicativo.