CURSO 2018-19.
Actividad práctica conceptos básicos de estructuras de datos.

1. OBJETIVOS:

- Familiarizarse con el IDE 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/jdk8-downloads-2133151.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*.

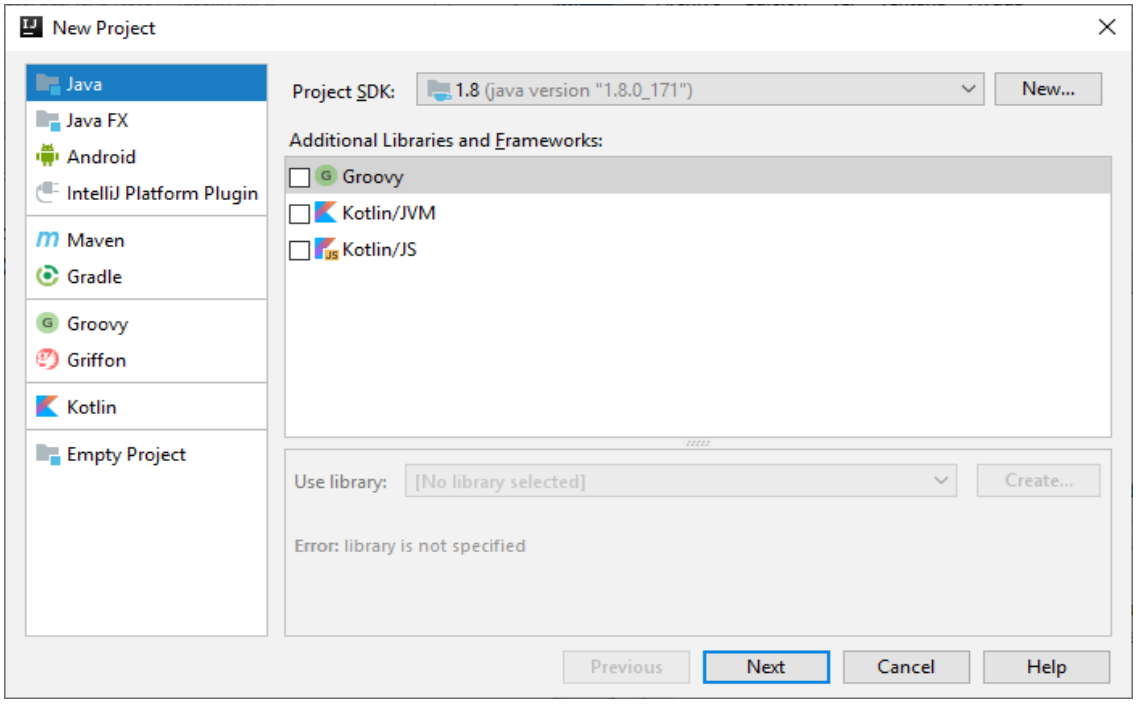
2.2. Crear un proyecto.

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.

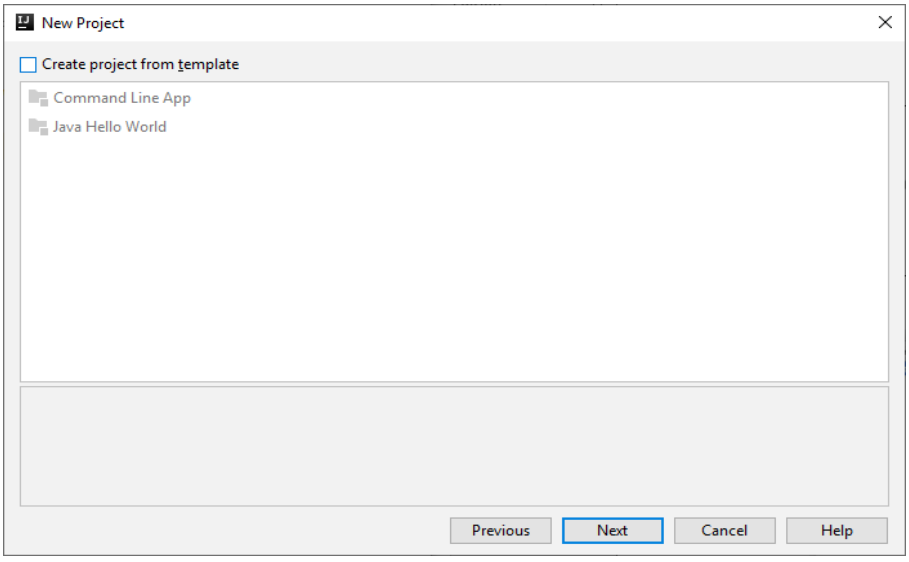
Al arrancar IntelliJ, nos solicita crear un nuevo proyecto o cargar uno ya creado:

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

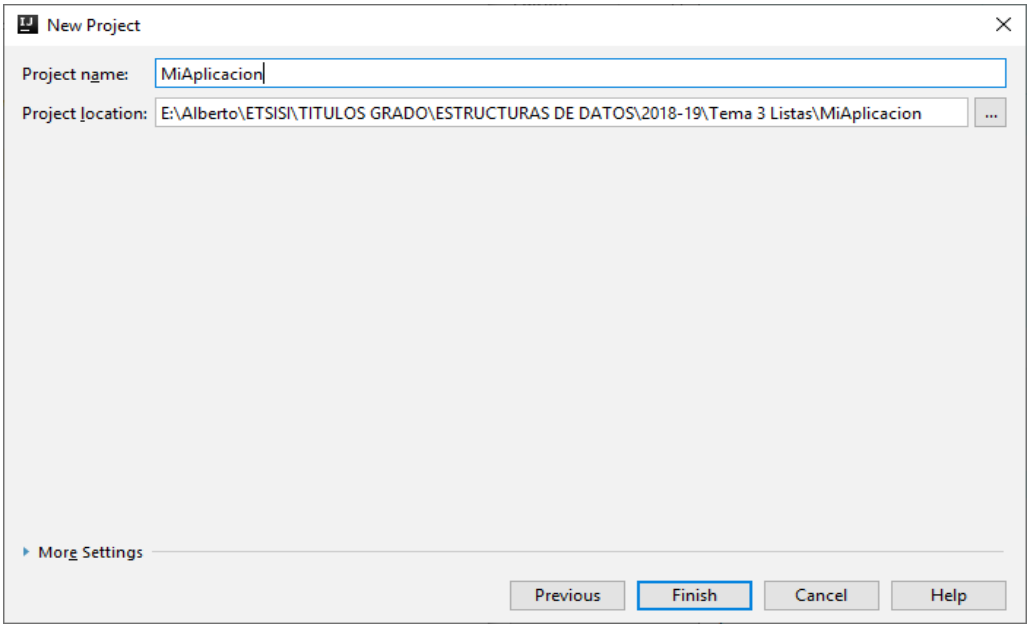
Si se pulsa “Create New Project” aparece la siguiente ventana:



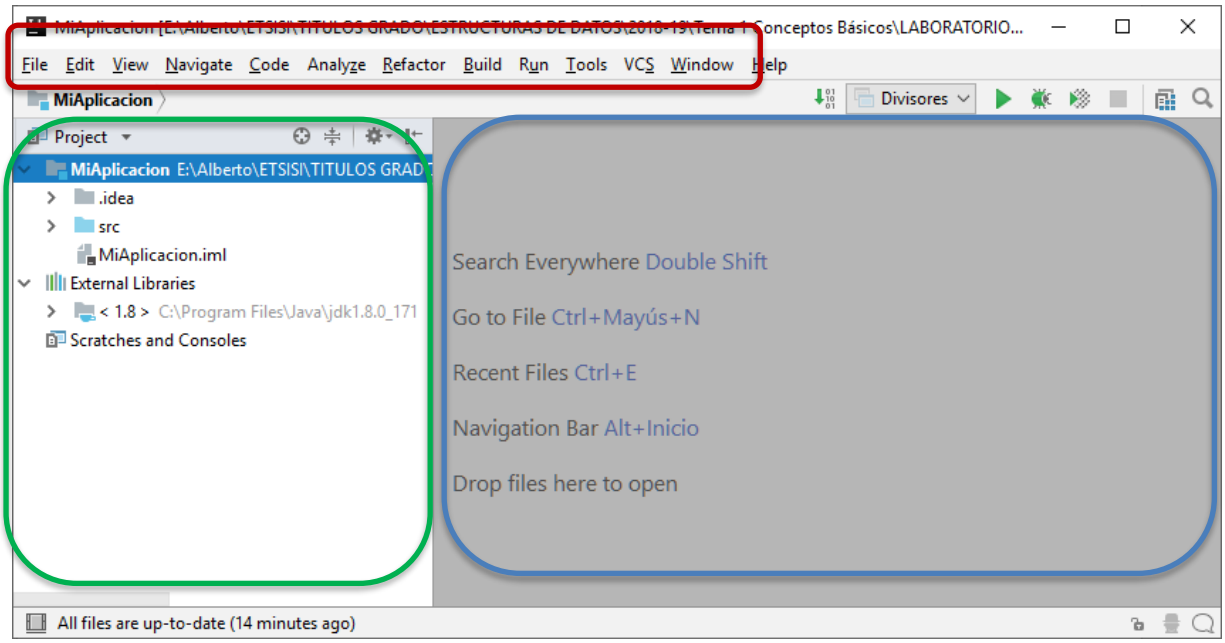
- Seleccionar en primer lugar que se va a crear un proyecto de tipo Java.
- Determinar el JDK de Java a utilizar. Si no aparece ninguno se le indicará la ubicación del JDK que se ha instalado. IntelliJ utilizará ese JDK para desarrollar en Java: uso de librerías estándar, compilar, ejecutar la máquina 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 1.8).

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*.

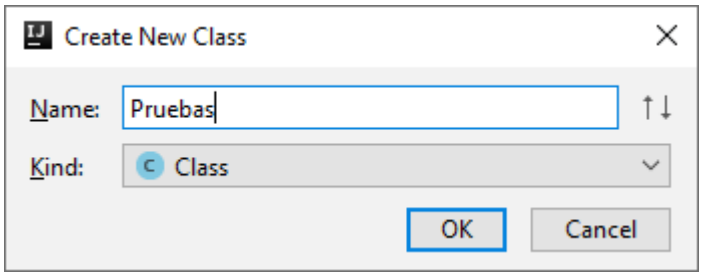
La ejecución de una aplicación Java debe comenzar por una clase que contenga un método *main*, con la siguiente cabecera:

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

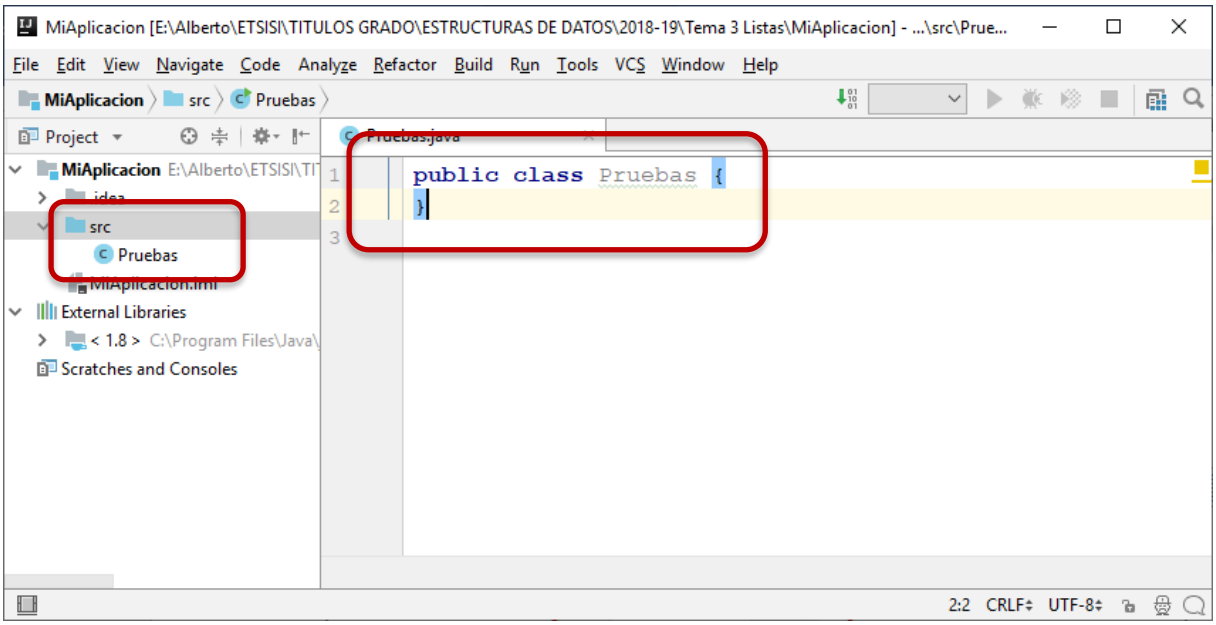
Aunque lo normal es tener un solo *main* en una aplicación Java, es posible tener varias clases con un método *main* cada una. De manera que se puede elegir qué *main* ejecutar.

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

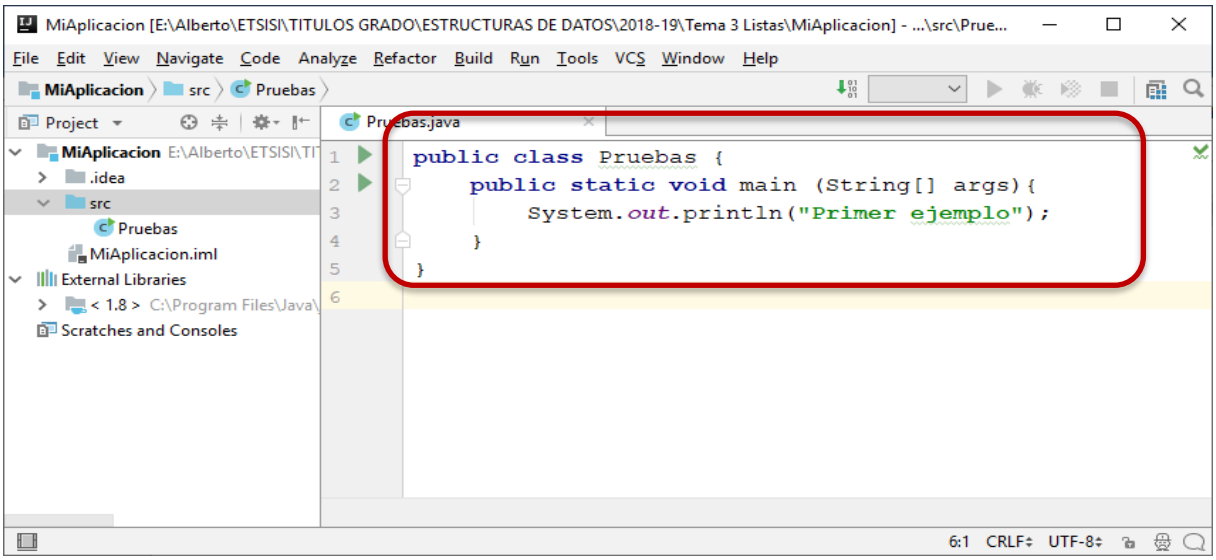
File --> New --> Java Class



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:



Puedes probar a cometer algún error al codificar *main* para ver cómo IntelliJ notifica los errores.

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.

Como comentario final, si ahora se desease crear un nuevo proyecto:

File --> New --> Project

Y si lo que se desea es abrir un proyecto ya creado anteriormente:

File → Open

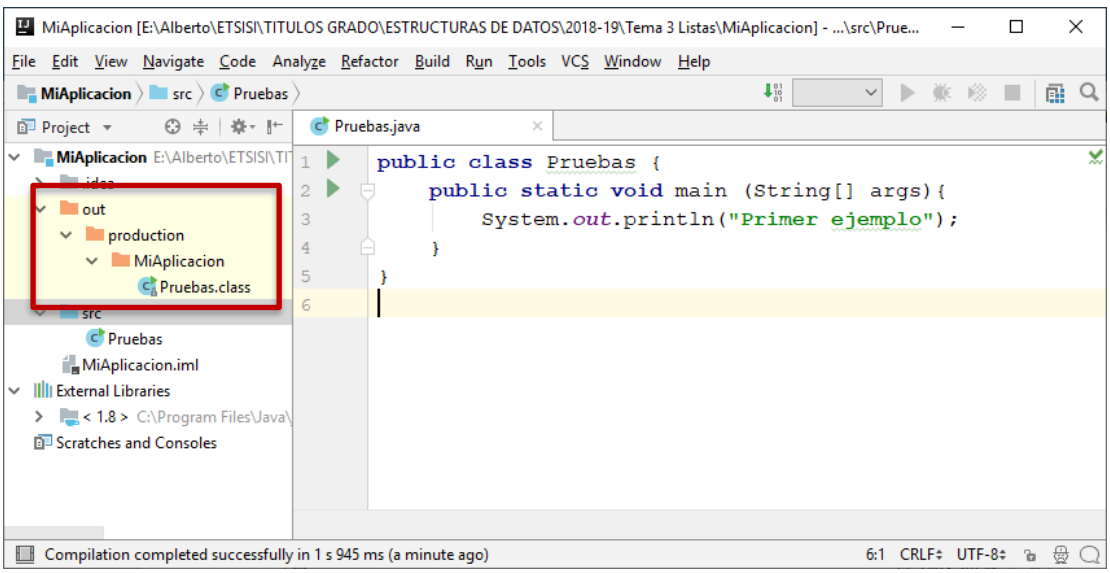
En ambos casos, IntelliJ te da la posibilidad de que el nuevo proyecto aparezca en una nueva ventana, o que sustituya al proyecto que está abierto en la ventana actual. Lo que no es posible en IntelliJ es tener dos proyectos abiertos en la misma ventana.

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.

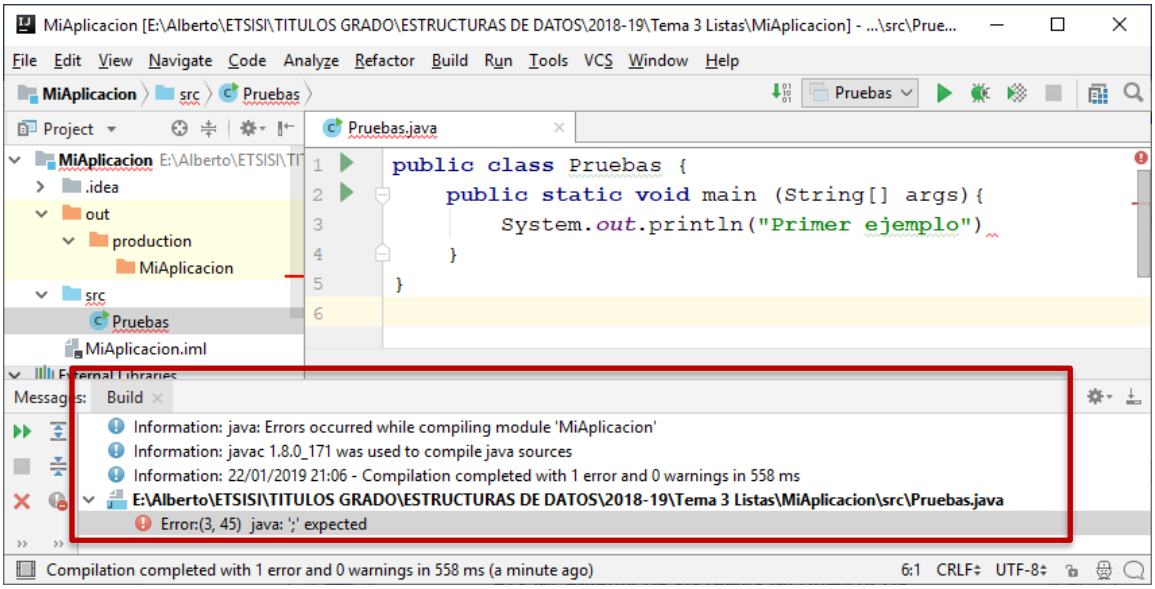
Para traducir los archivos Java a Bytecodes se debe realizar la operación **Build** del proyecto. Para ello:

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.

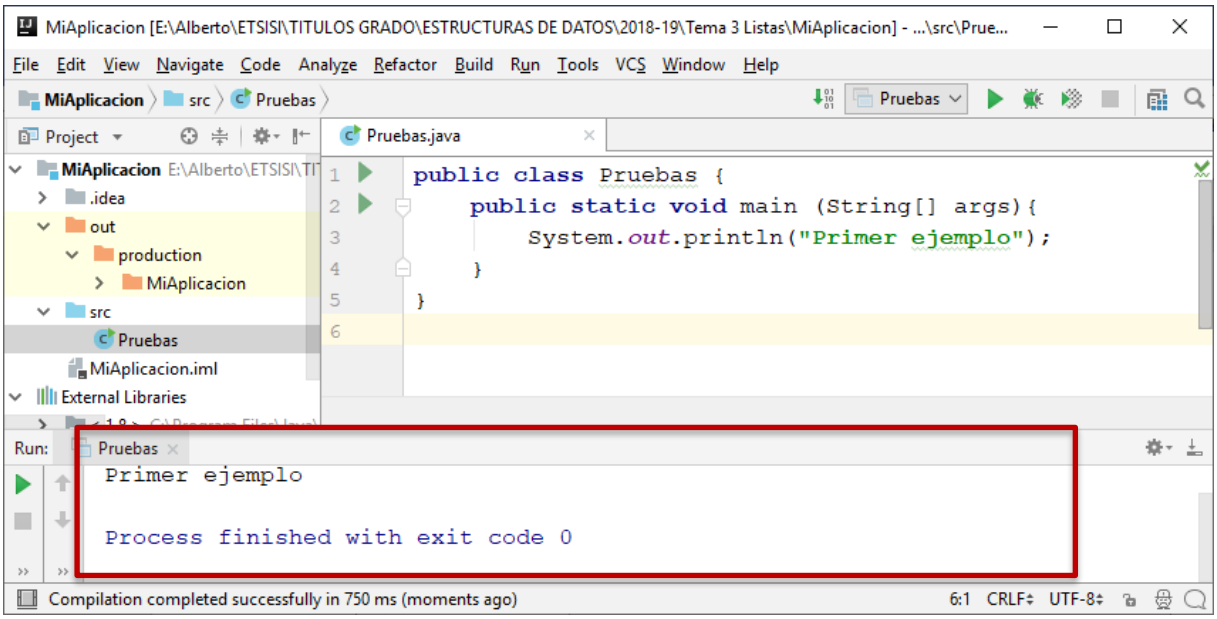
Si al hacer la operación **Build** se detecta algún error, aparece una nueva zona en la ventana notificando el error. Obviamente, en este caso no se habrán generado los archivos .class.



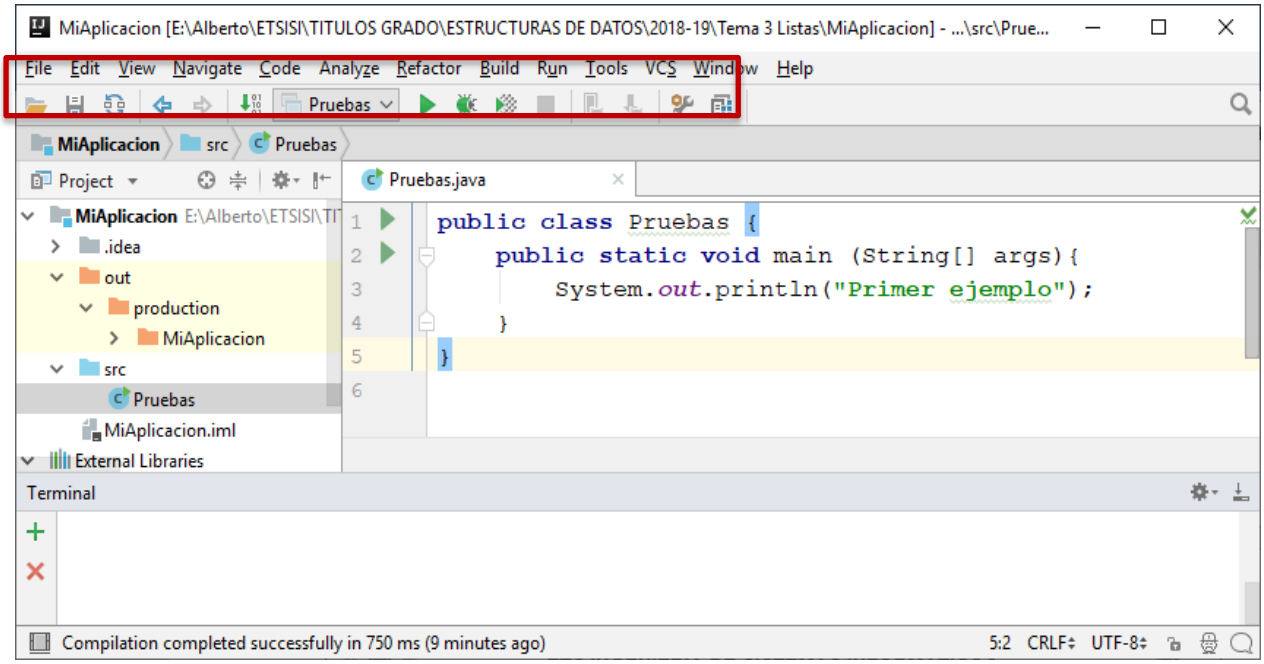
Una vez realizada la operación **Build** con éxito, la máquina virtual de java ya puede ejecutar este código. Para ello, IntelliJ te ofrece varias posibilidades:

- Sobre el código del fuente, botón derecho **--> Run 'Pruebas.main()'**
- Pinchar sobre el triángulo verde que aparece a la izquierda del *main*.
- Ir al proyecto, botón derecho sobre el archivo Pruebas **--> Run 'Pruebas.main()'**
- **Run --> Run 'Pruebas.main()'**

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



Consejo: habilitar la barra de herramientas mediante **View --> ToolBar**



En la barra de herramientas aparecen accesos directos a operaciones muy habituales: *build*, *run*, *debug*, salvar, abrir nuevo proyecto, etc.

Obviamente, el triángulo verde de la barra de herramientas sirve para la ejecución del proyecto. Nótese que a la izquierda del mismo viene marcada la clase principal (Pruebas), es decir, la que contiene el *main* que se quiere ejecutar. Si en la aplicación tuviera varios *main*, podría elegir aquí cuál de ellos ejecutar.

Realmente, cuando se realiza la operación ***Build*** sólo se compilan los archivos que hubieran sido modificados desde la última operación ***Build***. Además, 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. Sí que se debe ejecutar de forma explícita cuando lo que se quiere es únicamente comprobar si el código tiene errores.

Ejercicio propuesto:

Crear un nuevo proyecto con nombre *MiAplicacion2* en el que se incluya un archivo Java, llamado *Divisores*, para calcular los divisores de un número entero introducido por el usuario, excluyendo el 1 y el propio número. Introducir el código que aparece a continuación, y completar el método *visualizarDivisores*.

```
import java.util.Scanner;
public class Divisores {
    public static void main(String[] args) {
       int valor;
       Scanner lectura = new Scanner(System.in);
       System.out.print("Introduzca un valor entero positivo: ");
       valor = lectura.nextInt();
       if (valor < 1) {
           System.out.println("Número no válido");
       } else {
           visualizarDivisores(valor);
       }
    }
    public static void visualizarDivisores(int valor){
        // Completar
    }
}
```

Ejecución:

```
Introduzca un valor entero: 345
Divisores: 3 5 15 23 69 115
```

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 de contexto (botón derecho del ratón). Observe cómo cambia también automáticamente el nombre de la clase. Además, si esta clase se utilizara desde otro fichero, automáticamente cambiaría el nombre en todos los usos de la clase.

Esta funcionalidad de *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* por otro nombre. Para ello, recuerde que primero ha de seleccionar dicha variable en el código.

De forma similar, modifique también el nombre del proyecto.

Descoloque ahora las tabulaciones de algunas líneas del código (introduzca nuevas tabulaciones para desordenar el 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.

En el mismo proyecto anterior, defina un nuevo archivo *Factorial* con el siguiente contenido:

```
import java.util.Scanner;

public class Factorial {

    public static void main(String[] args) {
        int valor;
        Scanner lectura = new Scanner(System.in);
        System.out.print("Introduzca un valor entero: ");
        valor = lectura.nextInt();
        if (valor < 1) {
            System.out.println("El número ha de ser positivo");
        } else {
            System.out.println("El factorial de " + valor + " es " +
                                                    factorial(valor));
        }
    }

    public static long factorial(int numero) {
        long resultado;
        if (numero <= 1) {
            resultado = 1;
        } else {
            resultado = numero * factorial(numero - 1);
        }
        return resultado;
    }
}
```

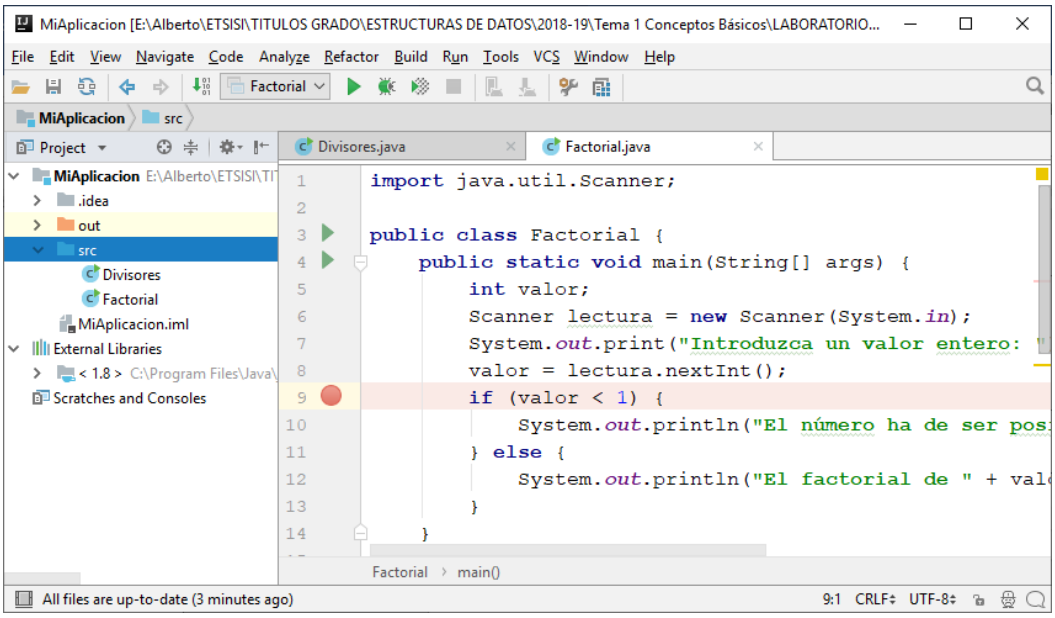
Ejecute el nuevo programa para comprobar que es correcto. Nótese que la función factorial de un número positivo es recursiva:

$$n! = \begin{cases} 1 & si\ n = 1 \\ n * (n-1)! & en\ otro\ caso \end{cases}$$

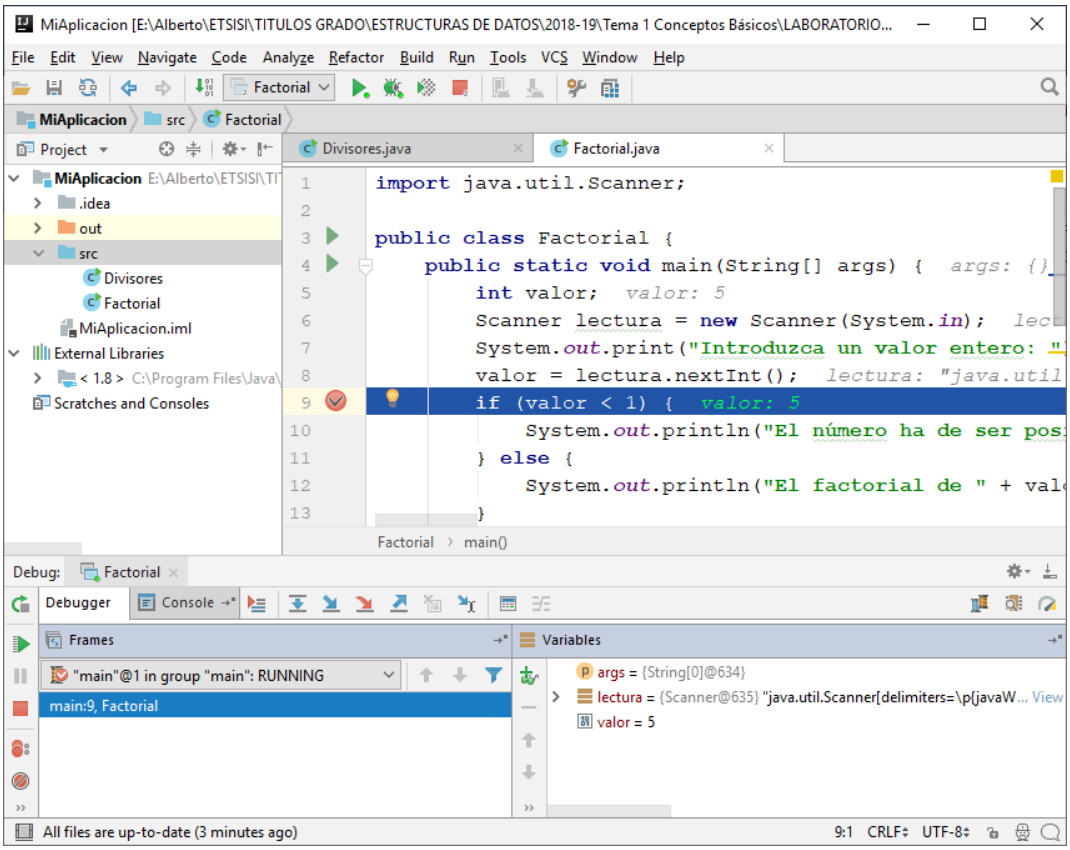
Por tanto, si se quiere calcular el factorial del número 5, se deben realizar 5 llamadas a la función factorial: *factorial(5)*, *factorial(4)*, *factorial(3)*, *factorial(2)* y *factorial(1)*. Estas 5 llamadas terminan en orden inverso a como se llamaron y cada una de ellas tiene sus propios datos (parámetros y variables). En este caso, cada una de ellas tiene su parámetro *numero* y su variable *resultado*.

El objetivo ahora es ejecutarla, pero en modo *debug*, es decir, se va poder ver el contenido de variables y parámetros en cualquier momento de la ejecución, para comprobar que todo es correcto. Esto puede ser muy útil cuando un código no nos funciona correctamente, pero no somos capaces de encontrar el error.

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 la quinta línea del main (*if (valor < 1) {*). Para ello, se marcará en el margen izquierdo de dicha línea y aparecerá un punto de color rojo:

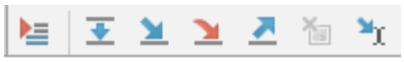


Ejecute ahora la aplicación en modo *debug* (**run --> debug 'Factorial'**, o directamente pulsar el escarabajo que aparece en la barra de herramientas). Después de pedir el número, la aplicación se para justo en el punto de ruptura:



Nótese que aparecen dos pestañas en la zona inferior de la ventana. Una para los resultados del *debugger* y otra para la consola de la aplicación. En el debugger se puede ver el valor de las variables

en el punto de ruptura. A partir de aquí se puede ir ejecutando paso a paso utilizando las opciones del *debugger*:



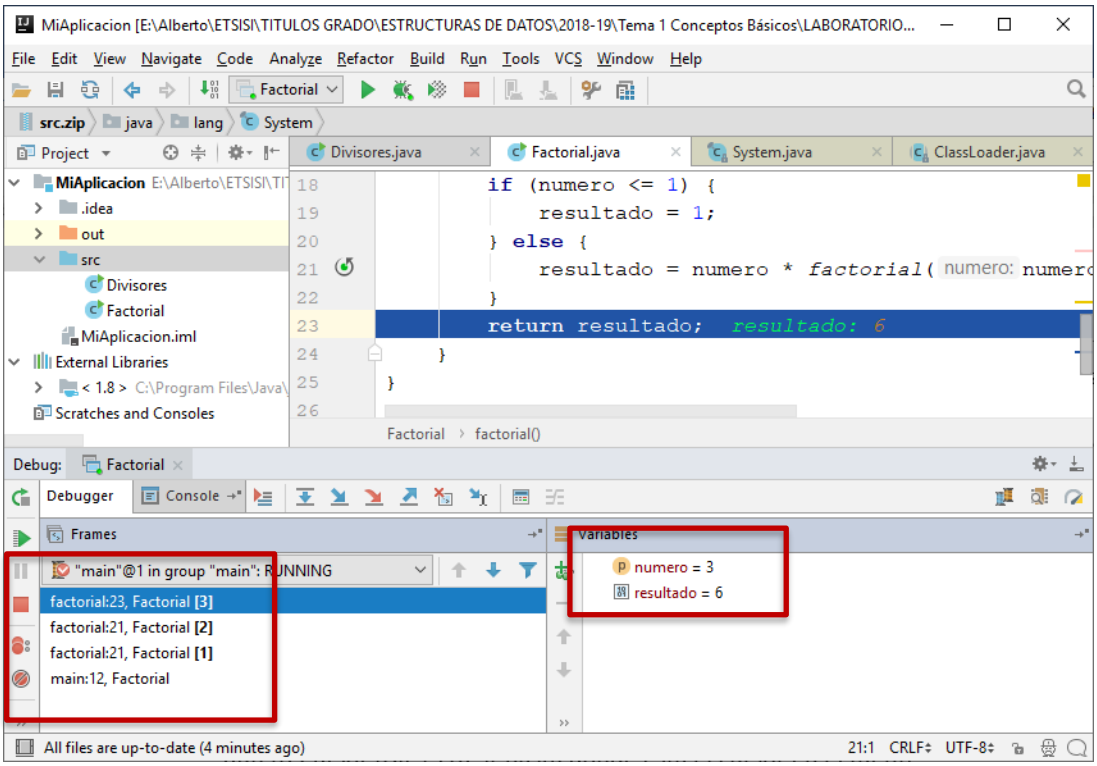
- Step Over (F8): ir a la sentencia siguiente.
- Step Into (F7): ir a la sentencia siguiente, pero si hay una función entrar dentro de ella.
- Force Step Into: Forzar entrar dentro. Llega hasta las funciones de librería.
- Step Out (Mayus + F8): salir de la función.
- Run to Cursor (Alt + F9): ir hasta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

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

Finalmente, se le puede indicar al *debugger* que termine la ejecución mediante: 

Ejercicio propuesto:

Realice el *debug* de la aplicación anterior introduciendo un 5 como dato para calcular el factorial. Realice ejecución paso a paso con Step Into (F7) para entrar en la función *factorial*. Compruebe que se realizan las 5 llamadas a dicha función y que van terminando en orden inverso. Analice también los datos que van teniendo cada una de las llamadas a la función en cada momento (*numero* y *resultado*).



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.

Ejercicio propuesto:

1. Crea un nuevo proyecto Java en IntelliJ de nombre Practica1.
2. Define la **clase *Alumno*,** con las siguientes características:

   - **Atributos**:
     - *Nombre*, *apellidos* y *matrícula*: 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.
   - **Métodos**:
     - Constructor sin parámetros, inicializando los atributos con los siguientes valores:
       - Nombre, apellidos y matricula con un *String* vacio
       - Calificación y numAsigs con valor 0
       - Además, se creará el *array* de asignaturas, dejándolo vacio.
     - Constructor con 4 parámetros para inicializar los atributos: nombre, apellidos, matrícula y calificación. El resto de atributos se inicializarán como en el constructor sin parámetros.
     - Métodos *getters* y *setter* para *nombre apellidos*, *matrícula* y *calificación*.
     - Método *getter* para *numAsignaturas*.
     - Método void *anadirAsignatura(String asig)*, que añade a la lista de asignaturas del alumno el *String* que recibe como argumento, y actualiza el valor de numAsigs. Si se intenta añadir más asignaturas del máximo permitido se indicará que no es posible mediante un mensaje.
     - Método *void mostrarAsignaturas()*, que muestra en pantalla las asignaturas en las que está matriculado el alumno

       Ejemplo:
       - Estructuras de Datos.
       - Matemática Discreta.

       Si el alumno no está matriculado en ninguna asignatura, simplemente aparecerá el mensaje: “El alumno no está matriculado en ninguna asignatura”
     - void mostrarAlumno(): muestra por pantalla los datos del alumno:

       Ejemplo:

       García García, María. Matr: zz1234 (9.0)
       - Estructuras de Datos.
       - Matemática Discreta.

3. Crea clase pública *Principal* con un método *main* en el que se realice lo siguiente:
   - Instanciar 4 objetos de la clase Alumno con los siguientes datos:

	apellidos	nombre	matricula	calificaci
0	Arias Gonzalez	Felipe	aa1253	3.50
1	Garcia Sacedón	Manuel	ax0074	8.35
2	Lopez Medina	Margarita	mj7726	7.70
3	Sanchez Arellano	Estela	bc2658	6.75

   - Añadir dos asignaturas al último alumno de la lista anterior. Utilizando el método correspondiente, mostrar las asignaturas del último alumno en pantalla
   - Utilizando el método correspondiente, mostrar los datos del primer y del último alumno de la lista anterior.

4. Añadir al proyecto una clase *GrupoAlumnos* con las siguientes características
   - **Atributos**:
     - *ListaAlumnos*, *array* de objetos *Alumno*, que contendrá los alumnos que componen un grupo.
     - *NumAlumnos*, entero que indica cuantos alumnos hay en el grupo en cada momento.
     - *Máximo*, entro que indica el máximo de alumnos que puede haber en el grupo. Este valor servirá para inicializar el array de alumnos con este máximo.
     - *Nombre*, *String* que indicará el nombre del grupo.
   - **Métodos:**
     - Constructor sin parámetros que inicializará un objeto *GrupoAlumnos* de nombre "GrupoDesconocido" y valor máximo de 10 alumnos (El array listaAlumnos será de 10 posiciones)
     - Constructor *GrupoAlumnos(int max, String nom)*, constructor que inicializará un objeto *GrupoAlumnos* con nombre *nom* y con un número máximo de *max* alumnos.
     - Los métodos *getNumAlumnos*, *getMaximo*, *getNombre* y *setNombre*.
     - El método *Alumno alumnoPos(int i)*, que devuelve el alumno que se encuentra en la posición *i*. Si no existe alumno en dicha posición, devolverá *null*.
     - El método *boolean insertarAlumno(Alumno a)*, que añade un nuevo alumno a la lista si no está llena. Devuelve un valor booleano indicando si ha sido posible o no la inserción.
     - El método *void mostrarGrupo()*, que muestra en pantalla el nombre del grupo y los datos de todos los alumnos que lo forman.

       Ejemplo:

       Arias Gonzalez, Felipe. Matr: aa1253 (3.5)
       El alumno no está matriculado en ninguna asignatura
       Garcia Salcedon, Manuel. Matr: ax0074 (8.35)
       El alumno no está matriculado en ninguna asignatura
       García García, María. Matr: zz1234 (9.0)
       - Estructuras de Datos.
       - Matemática Discreta.

5. Añadir lo siguiente en el método *main* de la clase *Principal*:
   - Instanciar un objeto *grupoAlumno* de nombre *g1* y con un máximo de 20 alumnos.
   - Insertar en este grupo todos los alumnos instanciados en el punto 3.
   - Insertar un alumno nuevo con vuestro nombre, apellidos y número de matrícula, y la calificación que queráis. (Cada alumno debe meter sus datos).
   - Añadir a este alumno al menos dos asignaturas.
   - Mostrar en pantalla los datos del grupo.
6. Añadir a la clase *grupoAlumno* los siguientes métodos:
   - *double mediaCalif()*, que devuelve la media de las calificaciones de los alumnos del grupo.
   - *void maxCalif()*, que escribe en pantalla los datos del alumno con la máxima calificación. Si hay más de uno escribirá los datos del primero de la lista con esa calificación máxima.
7. Escribir mensajes en la clase principal que permitan probar los métodos anteriores.