



```
@link https://dev
*
* @package _s
*/
if ( ! function_exists( 'incode_starter_setup' ) ) :
    /**
     * Sets up theme defaults and registers support for
     *
     * Note that this function is hooked into the after_
     * runs before the init hook. The init hook is too late
     * as indicating support for post thumbnails.
     */
    function incode_starter_setup() {
        /*
         * Make theme available for translation.
         * Translations can be filed in the /languages/ directory.
         * If you're building a theme based on _s, use a text domain
         * to change 'incode_starter' to the name of your theme.
         */
        load_theme_textdomain( 'incode_starter', get_template_directory() . '/languages' );

        // Add default posts and comments RSS feed links to head.
        add_theme_support( 'automatic-feed-links' );

        /*
         * Let WordPress manage the document title.
         * See https://codex.wordpress.org/Post_Title_Tag
         */
        add_theme_support( 'title-tag' );

        /*
         * Enable support for Post Thumbnails on your posts and pages.
         * See https://codex.wordpress.org/Feature-Flags#Post_Thumbnails
         */
        add_theme_support( 'post-thumbnails' );

        // Add support for the custom logo.
        add_theme_support( 'custom-logo' );

        // Add support for the custom header.
        add_theme_support( 'custom-header' );

        // Add support for the custom background.
        add_theme_support( 'custom-background' );

        // Add support for the custom color.
        add_theme_support( 'custom-color' );

        // Add support for the custom font.
        add_theme_support( 'custom-font' );

        // Add support for the custom menu.
        add_theme_support( 'custom-menu' );

        // Add support for the custom sidebar.
        add_theme_support( 'custom-sidebar' );

        // Add support for the custom widget.
        add_theme_support( 'custom-widget' );

        // Add support for the custom footer.
        add_theme_support( 'custom-footer' );

        // Add support for the custom header.
        add_theme_support( 'custom-header' );

        // Add support for the custom background.
        add_theme_support( 'custom-background' );

        // Add support for the custom color.
        add_theme_support( 'custom-color' );

        // Add support for the custom font.
        add_theme_support( 'custom-font' );

        // Add support for the custom menu.
        add_theme_support( 'custom-menu' );

        // Add support for the custom sidebar.
        add_theme_support( 'custom-sidebar' );

        // Add support for the custom widget.
        add_theme_support( 'custom-widget' );

        // Add support for the custom footer.
        add_theme_support( 'custom-footer' );
    }
endif;
```

6 NÍVEL

Manejo de estructuras de 2 dimensiones y persistencia
{Persistencia y manejo del estado inicial}

CASO — CAMPEONATO DE FUTBOL

Aplicación para manejar los resultados de los partidos en un campeonato de fútbol (cada par de equipos solo se enfrenta una vez).

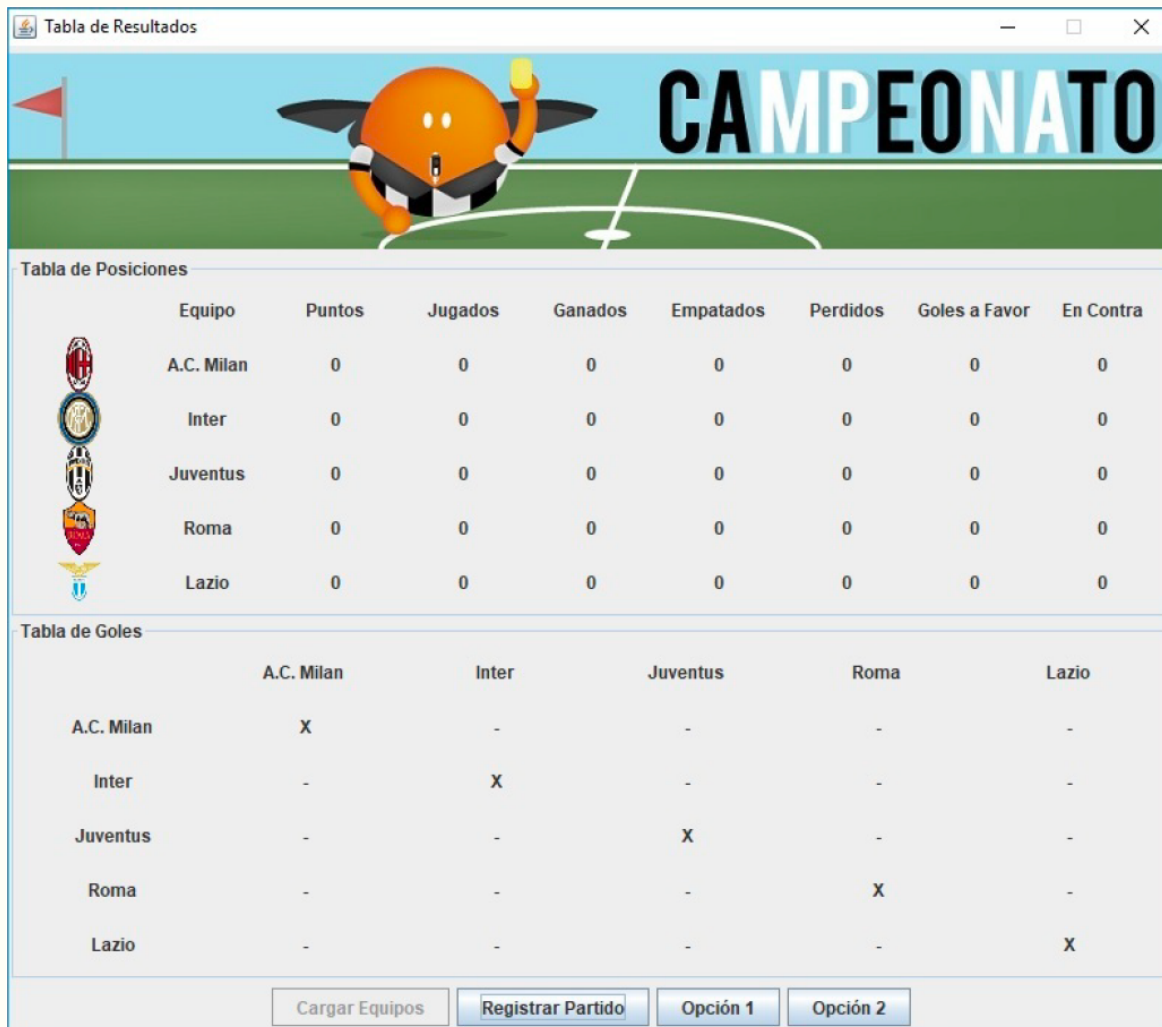


Tabla de Resultados

CAMPEONATO

Tabla de Posiciones

Equipo	Puntos	Jugados	Ganados	Empatados	Perdidos	Goles a Favor	En Contra
A.C. Milan	0	0	0	0	0	0	0
Inter	0	0	0	0	0	0	0
Juventus	0	0	0	0	0	0	0
Roma	0	0	0	0	0	0	0
Lazio	0	0	0	0	0	0	0

Tabla de Goles

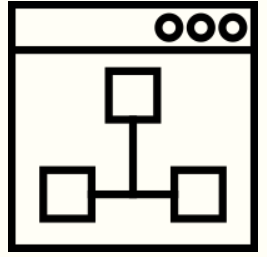
	A.C. Milan	Inter	Juventus	Roma	Lazio
A.C. Milan	X	-	-	-	-
Inter	-	X	-	-	-
Juventus	-	-	X	-	-
Roma	-	-	-	X	-
Lazio	-	-	-	-	X

Cargar Equipos Registrar Partido Opción 1 Opción 2

La información de los equipos que participan del campeonato está definida en un archivo que la aplicación debe leer para construir el estado inicial.

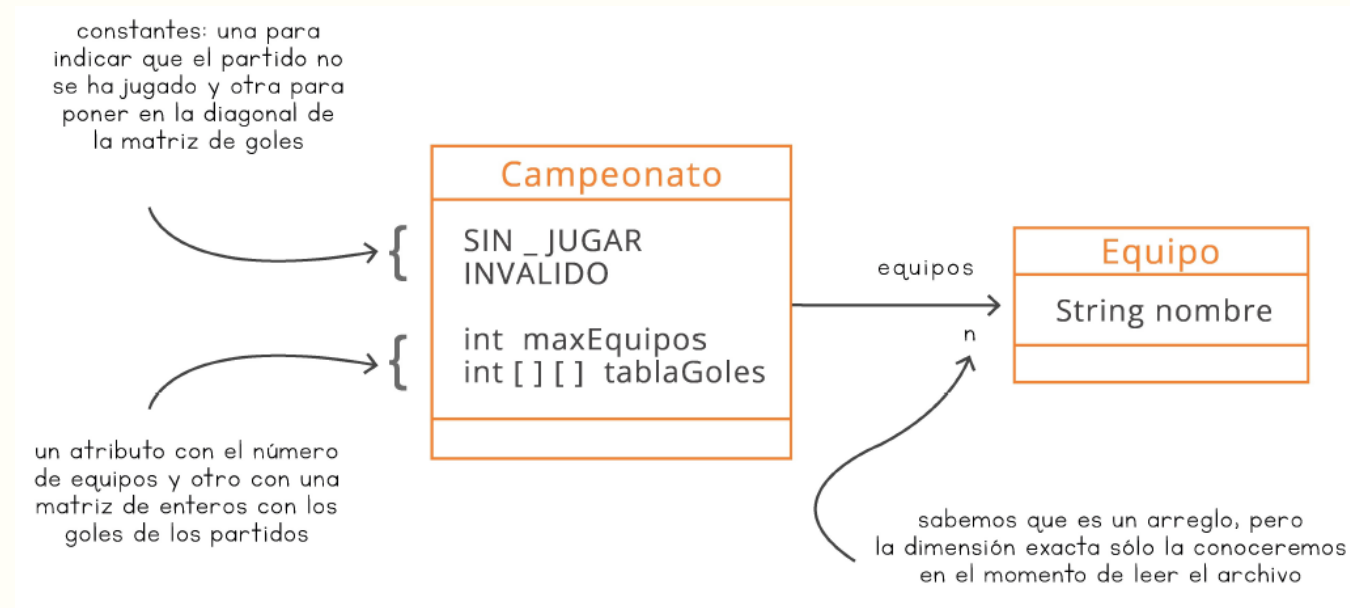
Debe mostrar la tabla de goles, también debe mostrar la tabla de posiciones, indicando para cada equipo el número de puntos, los partidos jugados, ganados, empatados y perdidos y los goles a favor y en contra. La aplicación permitirá registrar los resultados de cada uno de los partidos.

MODELO DEL MUNDO



Requerimientos Funcionales

- Cargar equipos
- Registrar un resultado
- Mostrar la tabla de goles
- Mostrar la tabla de posiciones



MANEJO DEL ESTADO INICIAL



En varios de los casos de estudio hemos utilizado archivos para configurar el estado inicial de la aplicación. Por ejemplo, teníamos la foto del empleado en un archivo y para la tienda teníamos en un archivo la imagen de cada producto. En este caso, el visor de imágenes utiliza un archivo para leer la imagen que será manipulada por la aplicación. Estos ejemplos tienen en común que la información del archivo se emplea para inicializar el estado de la aplicación

Un archivo es una entidad que contiene información que puede ser almacenada en la memoria secundaria del computador (Disco Duro, CD).

MANEJO DEL ESTADO INICIAL



Nombre Completo:	C:\dev\uniandes\cupi2\empleado\mundo\Empleado.java
Nombre Corto:	Empleado.java
Extensión o Apellido:	.java
Ruta o Camino:	C:\dev\uniandes\cupi2\empleado\mundo

- Para acceder a la información que se encuentra en los archivos debemos saber en dónde está y qué tipo de información contiene el archivo, para leerla.
- Los archivos que manejaremos tienen un formato especial que llamamos de “propiedades” (properties).
- Las clases Java que manejan archivos desde un programa están en el paquete `java.io`, y la clase que maneja las propiedades están en el paquete `java.util`.

LEYENDO DATOS - PROPIEDADES

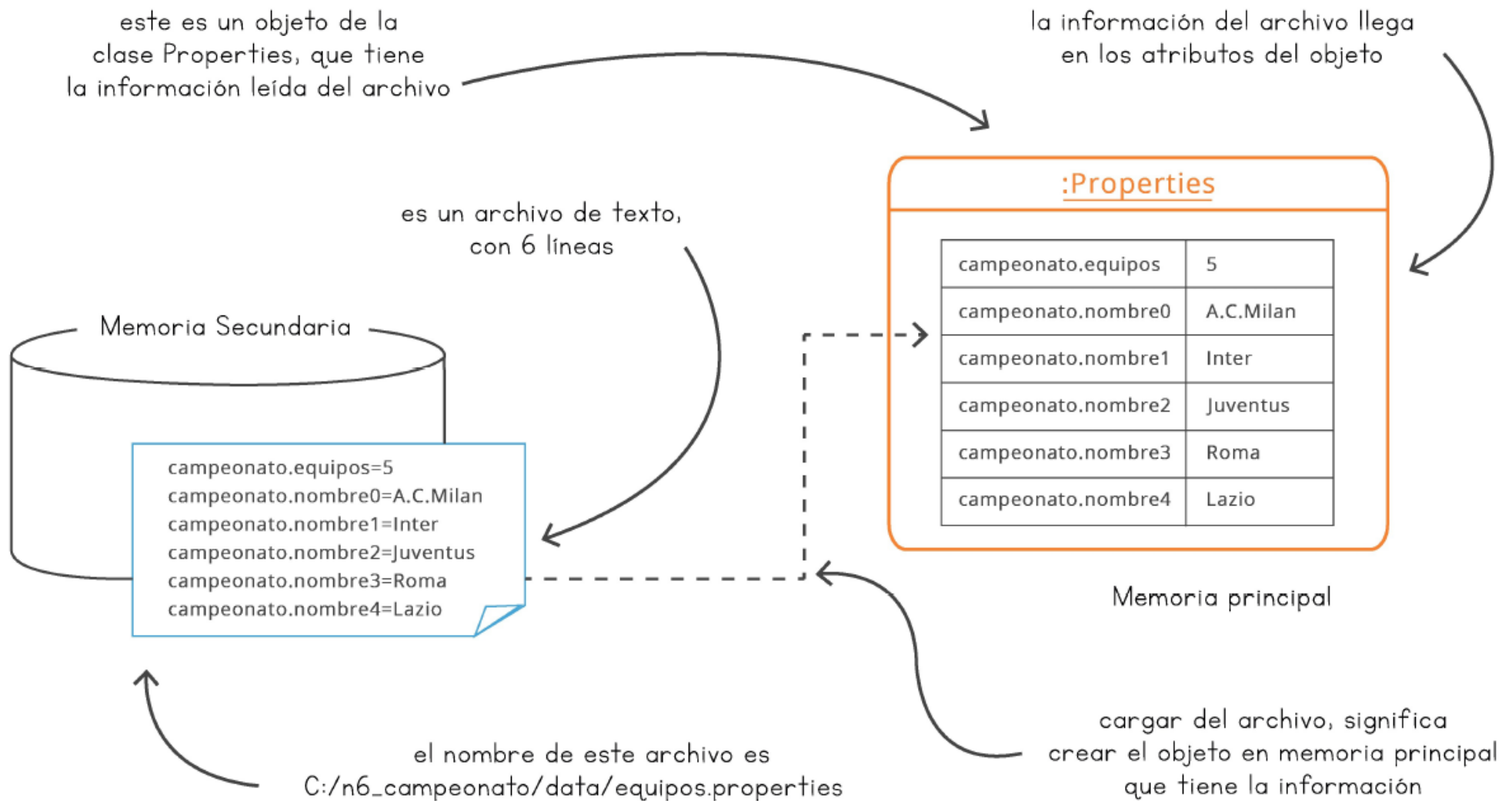
Una Propiedad se define como una pareja nombre = valor.
Por ejemplo, para expresar en un archivo que la propiedad llamada campeonato.equipos tiene por valor 5.



```
campeonato.equipos = 5
```

En Java existe una clase llamada Properties que representa un conjunto de propiedades persistentes. Por persistentes queremos decir que estas propiedades pueden ser almacenadas en un archivo en memoria secundaria y pueden ser leídas a la memoria del programa desde un archivo que ha sido escrito siguiendo las condiciones nombre = valor.

PROPIEDADES



PROPIEDADES

El archivo es un archivo de texto que tiene una lista de propiedades. Cada propiedad es una línea del archivo.



Si el objeto de la clase Properties está referenciado desde una variable llamada pDatos, una vez leído el archivo en memoria, se pueden utilizar los métodos de dicha clase para obtener el valor de los elementos.

Por ejemplo si queremos saber el valor de la propiedad campeonato.nombre0, podemos utilizar el siguiente método, cuya respuesta será: “A.C.Milan”.

```
String nombre = pDatos.getProperty ( "campeonato.nombre0" );
```



Por convención para los nombres de las propiedades utilizamos una secuencia de palabras en minúsculas, separadas por un punto.

ESCOGER UN ARCHIVO - PROGRAMA

- En Java se puede hacer una abstracción del nombre específico del archivo a un nombre independiente utilizando la clase `File`.
- La clase `File` ofrece varios servicios útiles, como saber si el archivo existe, preguntar por las características del archivo, crear un archivo vacío, renombrar un archivo, y mas ... Para crear un objeto de la clase `File` que contenga la representación abstracta del archivo físico, debemos crear una instancia de dicha clase, usando la siguiente sintaxis:

```
File archivosDatos = new File( "C:\n6_campeonato\data\equipos.properties" );
```

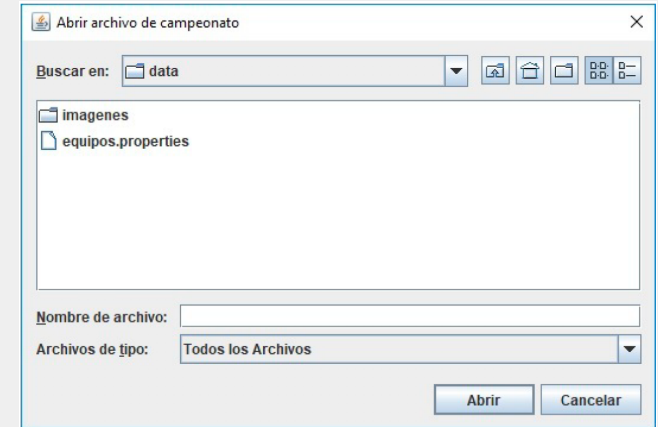


La variable `archivosDatos` está haciendo referencia a un objeto de la clase `File` que representa en abstracto el archivo que queremos leer. Esto se hace si conocemos el nombre del archivo.

Si invocamos el constructor de la Clase `File` con una cadena vacía (`null`), se disparará la excepción:
`java.lang.NullPointerException`

ESCOGER UN ARCHIVO

- Ventana a mostrar cuando el usuario seleccione el botón “Cargar Equipos”
- Además, queremos que aparezcan los archivos que están en el directorio “data” pues es allí donde almacenamos la información del campeonato.



```
public class InterfazCampeonato extends JFrame
{
    public void cargarEquipos ()
    {
        ...
        JFileChooser fc = new JFileChooser( "./data" );
        fc.setDialogTitle ("Abrir archivo de campeonato");
        ...
    }
}
```

Es responsable de preguntar al usuario el archivo del cual quiere cargar la información

Cambiamos el título de la ventana

Permite seleccionar un archivo. Se crea una instancia, pasándole en el constructor el directorio en el que queremos que comience.

ESCOGER UN ARCHIVO

```
public class InterfazCampeonato extends JFrame
{
    public void cargarEquipos ()
    {
        ...
        JFileChooser fc = new JFileChooser( "./data" );
        fc.setDialogTitle ("Abrir archivo de campeonato");
        File archivoCampeonato = null;
        int resultado = fc.showOpenDialog ( this );
        if ( resultado == JFileChooser.APPROVE_OPTION )
        {
            archivoCampeonato = fc.getSelectedFile ();
        }
        ...
        //Aquí debe ir la lectura del archivo
        ...
    }
}
```

Hace que la ventana de selección de archivos se abra. Y este método retorna un entero, que describe el resultado de la operación.

Obtenemos el objeto de la clase File que describe el archivo escogido por el usuario (Si este no canceló antes la operación)



ESTADO INICIAL

Para cargar el estado inicial del campeonato, debemos leer del archivo de propiedades la información sobre el número de equipos que van a participar (`campeonato.equipo`) y el nombre de los equipos (`campeonato.equipoX`), donde X es el índice que empieza en cero. Con esa información podemos iniciar el arreglo de equipos, y la matriz que representa la tabla de goles.

El constructor de la clase `Campeonato` será el encargado de inicializar y vamos a dividir en 3 sub-problemas:

- Cargar el problema del archivo en un objeto `Properties`
- Inicializar el arreglo de equipos con base en la información leída
- Inicializar la matriz que representa la tabla de goles

EL CONSTRUCTOR

```
public class Campeonato {
```

```
// Atributos
```

```
private int maxEquipos;  
private int [] [] tablaGoles;  
private Equipo [] equipos;
```

```
public Campeonato( File pArchivo ) throws Exception
```

```
{
```

```
    Properties datos = cargarInfoCampeonato( pArchivo );
```

```
    inicializarEquipos( datos );
```

```
    inicializarTablaGoles();
```

```
}
```

```
}
```

Recibe como parámetro el objeto de la clase *File* que describe el archivo con la información.

El objeto viene de la interfaz.

```
public class InterfazCampeonato extends JFrame ...
```

Si encuentra un problema leyendo el archivo o el formato es invalido.

Carga la información del archivo en un objeto llamado datos.

Recibe el objeto e inicializa el arreglo de equipos

Aprovecha la información dejada en los atributos, para crear la matriz con la tabla de goles

CARGANDO INFORMACIÓN

Recibe un objeto de la clase File

```
private Properties cargarInfoCampeonato(File arch) throws Exception  
{
```

```
    Properties datos = new Properties ();
```

Se crea un objeto de la clase Properties, donde va a quedar el resultado del método.

```
    FileInputStream in = new FileInputStream( arch );
```

```
    try  
    {
```

```
        datos.load ( in );  
        in.close ();
```

```
    }
```

```
    catch ( Exception e )  
    {
```

```
        throw new Exception ( "Formato Invalido" );
```

```
    }
```

```
    return datos;
```

```
}
```

Se crea un objeto de la clase FileInputStream que nos ayuda a hacer la conexión entre la memoria secundaria y el programa. Como una especie de canal por donde se pasan los datos.

Se pasa como parámetro el "canal de lectura". Lanza una excepción si el formato del archivo no es correcto.

Se cierra el canal de lectura.

INICIALIZANDO LA MATRIZ

```
private void inicializarTablaGoles ( )  
{  
    tablaGoles = new int [ maxEquipos ] [ maxEquipos ];  
  
    for ( int i= 0; i < maxEquipos ; i++ )  
    {  
        for ( int j = 0; j < maxEquipos ; j++ )  
        {  
            if ( i != j )  
                tablaGoles [ i ] [ j ] = SIN_JUGAR;  
  
            else  
                tablaGoles [ i ] [ j ] = INVALIDO;  
        }  
    }  
}
```

Crea una matriz con una fila y columna por cada equipo.

Inicializa cada una de las casillas de la matriz.

En la diagonal deja el valor de inválido.

OBJETOS CLASE PROPERTIES

```
private void inicializarEquipos ( Properties pDatos )  
{  
    String strNumeroEquipos = pDatos.getProperty ( "Campeonato.equipos" );  
  
    maxEquipos = Integer.parseInt ( strNumeroEquipos );  
  
    equipos = new Equipo [ maxEquipos ];  
  
    for ( int i = 0; i < maxEquipos ; i++)  
    {  
        String nombreEquipo = ( "Campeonato.nombre" + i );  
  
        equipos[ i ] = new Equipo ( nombreEquipo );  
    }  
}
```

Se obtiene el número de equipos, el valor de una propiedad siempre es una cadena de caracteres.

Convertimos la respuesta en un entero.

Se crea el arreglo de equipos.

Se recuperan los nombres de los equipos, y se van creando los objetos de la clase y se guardan en las casillas del arreglo.

Los nombres de los equipos viene en las propiedades "campeonato.nombre0, campeonato.nombre1, etc " razón por la cual calculamos el nombre dentro del ciclo, agregando al final el índice en el que va la iteración.

REGISTRAR RESULTADO

```
public void registrarResults( int pEquipo1, int pEquipo2, int pGol1, int pGol2 ) throws Exception
{
    if( pEquipo1 < 0 || pEquipo1 >= maxEquipos || pEquipo2 < 0 || pEquipo2 >= maxEquipos ){
        throw new Exception( "Equipos incorrectos" );
    }
    if( pEquipo1 == pEquipo2 ){
        throw new Exception( "Son el mismo equipo" );
    }
    if( pGol1 < 0 || pGol2 < 0 ){
        throw new Exception( "Número de goles inválido" );
    }
    if( tablaGoles[ pEquipo1 ][ pEquipo2 ] != SIN_JUGAR || tablaGoles[ pEquipo2 ][ pEquipo1 ] !=
        SIN_JUGAR ){
        throw new Exception( "Partido ya jugado" );
    }
    tablaGoles[ pEquipo1 ][ pEquipo2 ] = pGol1;
    tablaGoles[ pEquipo2 ][ pEquipo1 ] = pGol2;
}
```

EJERCICIO — CLASE CAMPEONATO

Calcular el número total de partidos ganados por el equipo que se recibe como parámetro. `tablaGoles[2][0] > tablaGoles[0][2]`, indica que el equipo con índice 2 le gana al equipo con índice 0

```
public int partidosGanados( int pEquipo)
{

}
}
```

EJERCICIO — CLASE CAMPEONATO

Calcular el mayor número de goles marcados en un partido del campeonato (sumando los goles de los dos equipos).

```
public int calcularTotalGoles()  
{
```

```
}
```

PROCESO

