

# Programación Frontend y Backend

*BLOQUE JAVA*

API Núcleo

01

## API Núcleo



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



## Collections

### PROBLEMÁTICA DE LOS ARRAYS

- Un Array es la manera más eficiente de almacenar y acceder a un conjunto de referencias a objetos.
- Permite almacenar tanto referencias a objetos como tipos de datos primitivos.
- Se puede definir un array de un tipo determinado (String []) lo cual ayuda a evitar “bugs” en el programa.
- Desventaja: Su tamaño tiene que ser fijado y no puede cambiarse en tiempo de ejecución.

## Collections

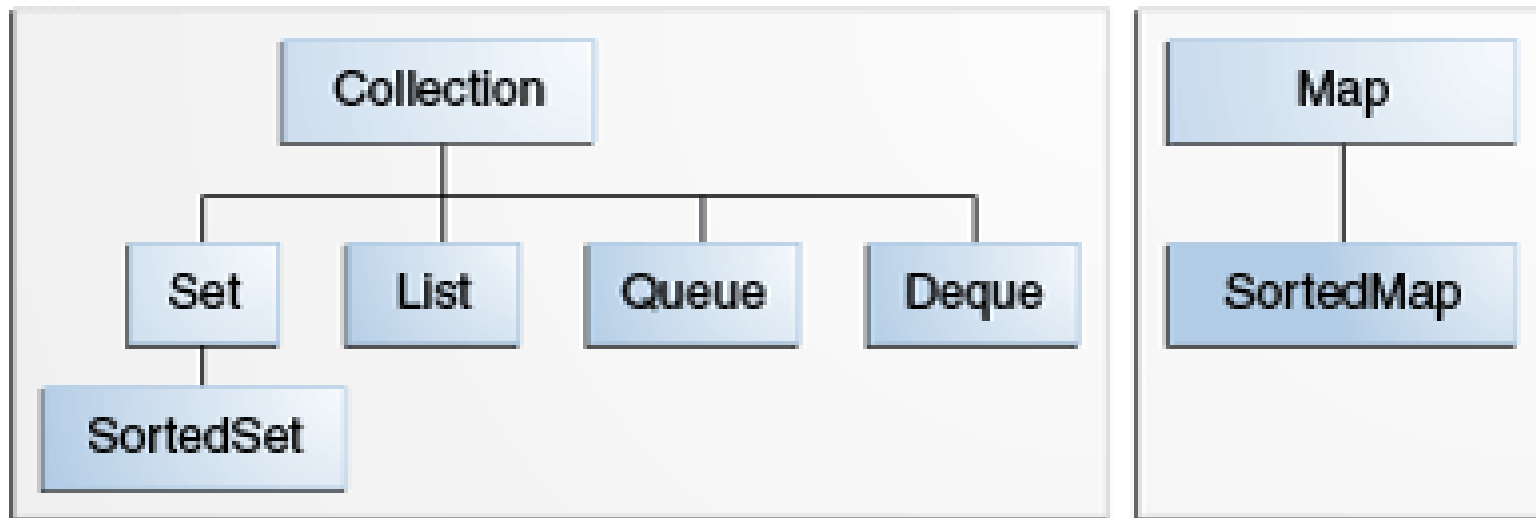
### SOLUCION

Utilizar Colecciones.

- Se expanden automáticamente si es necesario, esto es, el tamaño de la colección puede cambiar en tiempo de ejecución.
- Poseen un poderoso conjunto de métodos y una jerarquía especializada.
- Su interfaz es unificada lo que facilita la rápida adaptación a otras situaciones con mínimos cambios (polimorfismo).

## Collections

### Interfaces de Collections Framework



## Collections

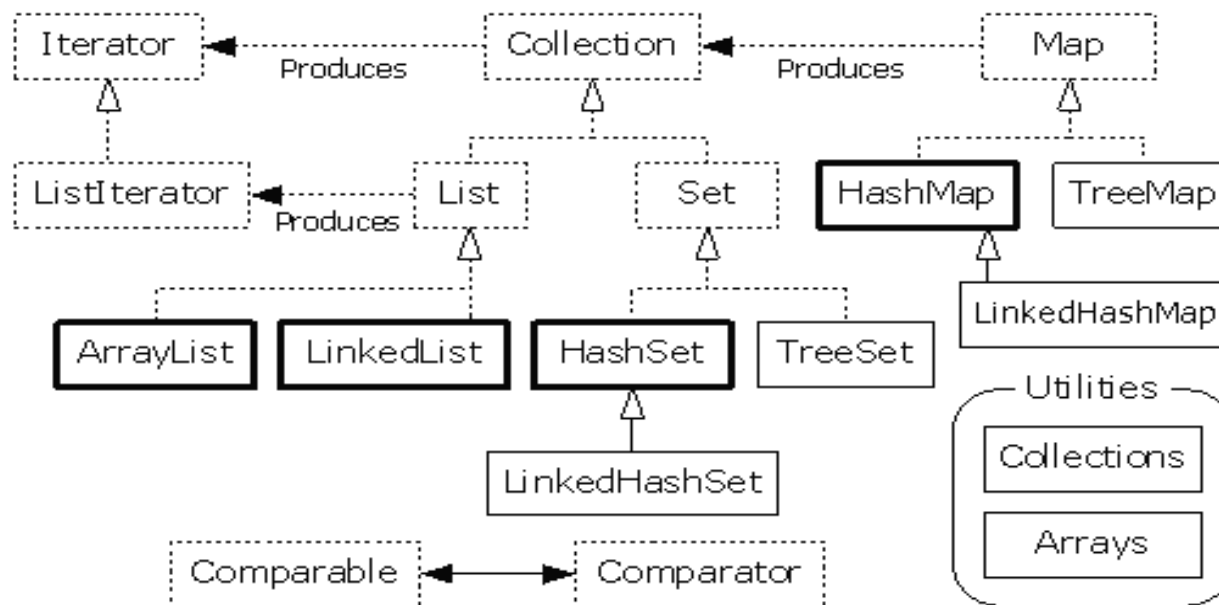
### INTERFACES

**Collection:** Grupo de elementos individuales, quizás con algunas reglas de restricción:

- **Set:** No puede tener elementos duplicados.
- **List:** Almacena los elementos en algún orden secuencial.
- **Queue:** Cola FIFO.
- **Map:** Grupo de pares (key, object).

## Collections

### INTERFACES



## Collections

### METODOS COLLECTION

<b>boolean add(Object)</b>	Añade un objeto a la colección.
<b>boolean addAll(Collection)</b>	Añade todos los objetos de una colección a la presente colección.
<b>void clear( )</b>	Borra todos los elementos de la colección.
<b>boolean contains(Object)</b>	<b>true</b> si el objeto pertenece a la colección.
<b>boolean containsAll(Collection)</b>	<b>true</b> si esta colección tiene todos los objetos de la colección pasada como parámetro.
<b>boolean isEmpty( )</b>	<b>true</b> si no tiene elementos.
<b>Iterator iterator( )</b>	Retorna un <i>iterator</i> que nos permite movernos por la colección.
<b>boolean remove(Object)</b>	Si el argumento está en la colección, una instancia del mismo es eliminada. Retorna <b>true</b> si se ha eliminado alguna instancia.





## Collections

### METODOS COLLECTION

<b>boolean retainAll(Collection)</b>	Retiene en la colección sólo aquellos elementos contenidos en el argumento (intersección de conjuntos). Retorna <b>true</b> si la colección ha cambiado como consecuencia de esta invocación.
<b>int size( )</b>	Retorna el número de elementos actuales de la colección.
<b>Object[] toArray( )</b>	Retorna un array de objetos con la colección.
<b>Object[] toArray(Object[] a)</b>	Retorna un array con los elementos de la colección, pero del tipo indicado en el argumento. Necesita cast: <i>Object[] a = c.toArray(new String[0]);</i> <i>(String)a.substring.....</i>



## Collections

### METODOS COLLECTION

Las interfaz **LIST** Añade los métodos siguientes a *Collection*:

- `add(int index)`
- `get(int index)`
- `indexOf (Object o)`
- `lastIndexOf (Object o)`
- `set (int index, Object o)`
- `sublist(int fromIndex, int toIndex)`

## Collections

### LIST

La lista (**List**) es un tipo de colección en el que se mantiene el orden de inserción.

- **ARRAYLIST**: Lista respaldada por arrays. Importante definir el tamaño inicial. Cuando el tamaño supera el máximo actual, aumenta su tamaño un 50%
- **LINKEDLIST**: Lista doblemente enlazada. Adecuada para operaciones de inserción y eliminación de elementos.
- **VECTOR**: Similar al ArrayList pero su acceso es “sincronizado”. Cuando el tamaño supera el máximo actual, este dobla su tamaño. Optimo acceso secuencial, con rápidos *insert* y *remove* del medio de la lista. Relativamente lento para acceso aleatorio. Añade métodos **addFirst( )**, **addLast( )**, **getFirst( )**, **getLast( )**, **removeFirst( )**, **removeLast( )**, tomados de la interface **Queue** a la cual también implementa.

## Collections

### LIST

Las variables a utilizar deben ser siempre referencias a interfaces. Así, si en un momento dado queremos utilizar otra clase concreta, los cambios sólo se realizarían en el momento de creación de las variables:

Ejemplo:

**List x = new LinkedList();**

En un momento dado queremos utilizar ArrayList en vez de LinkedList. El cambio sería menos impactante, ya que cambiamos el objeto que se crea y todos los métodos que aceptan una lista aceptan también la otra (polimorfismo).

**List x = new ArrayList();**

## Collections

### SET

Un set es una coleccion de objetos. (INTERFAZ).

Cada elemento añadido debe ser único, es decir, no se añaden elementos duplicados.

No se garantiza ningún orden.

### IMPLEMENTACIONES:

- **HASHSET** : Implementación concreta donde el tiempo para búsquedas es importante. Es la clase que implementa **Set** más utilizada.
- **TREESET** : Es un **Set** ordenado con estructura de árbol.
- **LINKEDHASHSET** : Internamente es una lista enlazada. Por tanto garantiza un orden (está primero el que primero se ha insertado).

## Collections

### EJEMPLO SET

```
public class ListarConjunto {  
    public static void main(String[] args) {  
        Set<String> hs = new HashSet<String>();  
        hs.add("Victor");  
        hs.add("Amaya");  
        hs.add("Amaya"); // Los elementos solo pueden estar una vez  
        hs.add("Javier");  
        Iterator<String> iter = hs.iterator();  
        while (iter.hasNext())  
            System.out.println(iter.next());  
    }  
}
```

## Collections

### EJEMPLO LIST

```
public class ListarConjunto {  
    public static void main(String[] args) {  
        List<String> hs = new ArrayList<String>();  
        hs.add("Victor");  
        hs.add("Amaya");  
        hs.add("Amaya"); // Los elementos se pueden repetir  
        hs.add("Javier");  
        Iterator<String> iter = hs.iterator();  
        while (iter.hasNext())  
            System.out.println(iter.next());  
    }  
}
```

## COLLECTIONS

### ITERATOR

Es un objeto cuya función es moverse a través de una secuencia de objetos, y obtener cada objeto en la secuencia sin que el programador conozca la estructura de almacenamiento utilizada.

Operativa con *Iterators*:

Pedir al contenedor (*objeto collection*) un *iterator*: `Iterator it = coll.iterator();`

Obtener el primero objeto invocando **next()**: `it.next();`

Comprobar si existen más objetos en la secuencia: `if (it.hasNext())`

Opcionalmente eliminar el objeto a partir del *iterator*: `it.remove();`



## COLLECTIONS

### COMPARABLE

Una clase puede implementar esta interface para proporcionar un ordenamiento a sus instancias. Este ordenamiento se llama *ordenamiento natural*.

La clase que implemente esta interfaz debe implementar el método **compareTo(Object )** que devuelve un valor **negativo si la instancia actual es menor que el objeto del argumento, cero si es igual, mayor que cero si es mayor el actual objeto que el argumento.**

```
public class Cubo implements Comparable<Cubo> {  
  
    private int height;  
    private int width;  
    private int breadth;  
}
```

```
public int compareTo(Cubo o) {  
    if (this.getVolume() == o.getVolume()) {  
        return 0;  
    } else if (this.getVolume() > o.getVolume()) {  
        return 1;  
    } else {  
        return -1;  
    }  
}
```

## COLLECTIONS

## COMPARABLE

```
public static void main(String[] args) {  
  
    Cubo cubos[] = {  
        new Cubo(1, 1, 1),  
        new Cubo(2, 2, 2),  
        new Cubo(2, 1, 2),  
        new Cubo(1, 1, 2) };  
  
    List<Cubo> cubosList = Arrays.asList(cubos);  
  
    for(Cubo c : cubosList) {  
        System.out.println(c.getVolume());  
    }  
  
    Collections.sort(cubosList);  
  
    for(Cubo c : cubosList) {  
        System.out.println(c.getVolume());  
    }  
}
```

&lt;terminated&gt; Mair

1  
8  
4  
2  
1  
2  
4  
8

GOBIERNO  
DE ESPAÑAMINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMOEscuela de  
organización  
industrialUnión Europea  
Fondo Social Europeo  
Iniciativa de Empleo Juvenil  
El FSE invierte en tu futuro

## Ejercicio 1

Vamos a modificar nuestra clase Persona para que podamos ordenar una lista de personas, a la hora de ordenar lo haremos por el nombre de la persona (alfabéticamente) en caso de tener el mismo nombre se ordenará por la edad de mayor a menor y en caso de coincidir también en edad ordenaremos por año de nacimiento de más antiguo a más reciente. La lista de personas es la siguiente:

ANA	26	2010	RESTO DE CAMPOS
ALBA	21	1988	RESTO DE CAMPOS
ANA	26	2009	RESTO DE CAMPOS
ALBA	26	2007	RESTO DE CAMPOS

## COLLECTIONS

### MAP

El mapa o **Map** es un tipo de dato que asocia una clave a un valor. No extiende de Collection.

#### IMPLEMENTACIONES:

- HashMap : Es la más utilizada.
- TreeMap : Ordenadas las *keys* de manera ascendente.
- LinkedHashMap : Doble lista enlazada. Ordenado por orden de llegada.

## COLLECTIONS

### METODOS INTERFAZ MAP

- *put(Object key, Object value)*: Añade un objeto asociado a una clave.
- *get(Object key)*: Recupera el objeto asociado con el argumento como clave.
- *boolean containsKey(Object key )*: Retorna **true** si contiene a Object como key.
- *boolean containsValue(Object value )*: Retorna **true** si contiene a Object como valor.
- *entrySet()*: Retorna un Set de pares (key, valor) (*Map.Entry*).
- *keySet()*: Retorna un Set con las keys.
- *values()*: Retorna una Collection con los values.

## Ejercicio 2

Vamos a modificar la clase Persona.java y vamos a añadirle el atributo DNI,  
Vamos a crear un HashMap de personas dónde el DNI será la clave y el objeto persona será el valor.  
Por último recorreremos el HashMap mostrando los datos de cada persona por consola.

## STRING UTILS

### Expresiones Regulares

- Una **expresión regular** define un patrón de búsqueda para cadenas de caracteres.
- La podemos utilizar para comprobar si una **cadena contiene o coincide con el patrón**. El contenido de la cadena de caracteres puede coincidir con el patrón 0, 1 o más veces.
- El patrón se busca en el String **de izquierda a derecha**. Cuando se determina que un carácter cumple con el patrón este carácter ya no vuelve a intervenir en la comprobación.

#### Ejemplo:

La expresión regular **"010"** la encontraremos dentro del String **"010101010"** solo dos veces: **"010101010"**.

## STRING UTILS

### Símbolos comunes en expresiones regulares

Expresión	Descripción
.	Un punto indica cualquier carácter
^expresión	El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
expresión\$	El símbolo \$ indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener las letras a ó b ó c.
[abc][12]	El String debe contener las letras a ó b ó c seguidas de 1 ó 2
[^abc]	El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a ó b ó c.
[a-z1-9]	Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos)
A B	El carácter   es un OR. A ó B
AB	Concatenación. A seguida de B



## STRING UTILS

### Meta caracteres

Expresión	Descripción
\d	Dígito. Equivale a [0-9]
\D	No dígito. Equivale a [^0-9]
\s	Espacio en blanco. Equivale a [ \t\n\x0b\r\f]
\S	No espacio en blanco. Equivale a [^\s]
\w	Una letra mayúscula o minúscula, un dígito o el carácter _ Equivale a [a-zA-Z0-9_]
\W	Equivale a [^\w]
\b	Límite de una palabra.

En Java debemos usar una doble barra invertida \\. **Ejemplo:**

- Para utilizar `\w` tendremos que escribir `\\w`.
- Si queremos indicar que la barra invertida es un carácter de la expresión regular tendremos que escribir `\\\\`.

## STRING UTILS

### Cuantificadores

Expresión	Descripción
{X}	Indica que lo que va justo antes de las llaves se repite X veces
{X,Y}	Indica que lo que va justo antes de las llaves se repite mínimo X veces y máximo Y veces. También podemos poner {X,} indicando que se repite un mínimo de X veces sin límite máximo.
*	Indica 0 ó más veces. Equivale a {0,}
+	Indica 1 ó más veces. Equivale a {1,}
?	Indica 0 ó 1 veces. Equivale a {0,1}

## STRING UTILS

### Expresiones Regulares

Para usar expresiones regulares en Java se usan las clases ***Pattern*** y ***Matcher*** del paquete **java.util.regex**.

**Pattern**: Un objeto de esta clase representa la expresión regular. Contiene el método **compile**(String regex) que recibe como parámetro la expresión regular y devuelve una instancia de la clase **Pattern**.

**Matcher**: Esta clase compara el String y la expresión regular y contiene los siguientes métodos útiles para determinar coincidencias:

**boolean: matches**(CharSequence stringToValidate).

**boolean: find**() indica si el String contienen el patrón.

## STRING UTILS

Para Comprobar si un String está formado por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10:

```
Pattern pat = Pattern.compile("[a-zA-Z]{5,10}");  
Matcher mat = pat.matcher(cadena);  
if (mat.matches()) {  
    System.out.println("SI");  
} else {  
    System.out.println("NO");  
}
```

## STRING UTILS

```
m ↗ replaceFirst (String regex, String replacement)
m ↗ replace (char oldChar, char newChar)
m ↗ concat (String str)
m ↗ replaceAll (String regex, String replacement)
m ↗ substring (int beginIndex)
m ↗ substring (int beginIndex, int endIndex)
m ↗ toLowerCase ()
m ↗ toUpperCase ()
m ↗ trim ()
m ↗ charAt (int index)
m ↗ contains (CharSequence s)
m ↗ endsWith (String suffix)
m ↗ equals (Object anObject)
m ↗ equalsIgnoreCase (String anotherString)
m ↗ indexOf (String str)
m ↗ isEmpty ()
m ↗ lastIndexOf (String str)
m ↗ length ()
m ↗ matches (String regex)
m ↗ split (String regex)
m ↗ split (String regex, int limit)
m ↗ startsWith (String prefix)
m ↗ toCharArray ()
```



## EJERCICIOS STRING

- Convertir la siguiente cadena "Esto es una prueba" a mayúsculas e imprimir el resultado por pantalla.
- Separar la siguiente cadena "La clase String, tiene varios métodos, muy útiles, como saber su longitud, trocear la cadena, etc" en una lista que contiene todos los string que van separados por coma ',' e imprimir cada uno de ellos por pantalla.
- Dentro de la siguiente cadena "Esto es un ejemplo usado para determinar un indice", imprime el indice donde aparece la subcadena "ejemplo".
- Escribir un método que reciba un String con una fecha especificada con el formato dd/MM/yyyy ("04/11/2016" por ejemplo) que muestre por pantalla la fecha escrita de forma "legible". Ejemplos:
  - muestraFecha("01/02/2016") mostraría "Es el 1 de Febrero de 2016".
  - muestraFecha("22/04/2017") mostraría "Es el 22 de Abril de 2017".
  - muestraFecha("01-02-2016") mostraría "La fecha '01-02-2016' no tiene el formato correcto".
- Introduce una frase por consola e imprime a continuación el número de veces que aparece la a, la e, la i, la o y la u, se debe tener en cuenta que la frase puede estar en minúsculas o mayúsculas a la hora de contabilizar las vocales.

## CALENDAR

### CALENDAR

- Clase abstracta para representar y el tiempo y la fecha.
- La clase ***java.util.GregorianCalendar*** es una implementación de ***java.util.Calendar***.
- Para crear un calendario con el instante actual:  
*Calendar cal = Calendar.getInstance();*
- Ofrece un conjunto de métodos y campos que nos permiten obtener información sobre el calendario y realizar operaciones de manipulación.

## CALENDAR

t.get(Calendar.YEAR)	Año
t.get(Calendar.MONTH)	Mes (0-11)
t.get(Calendar.DAY_OF_MONTH)	Día del mes (1-31)
t.get(Calendar.DAY_OF_WEEK)	Día de la semana (1-7)
t.get(Calendar.HOUR)	Hora (0-11)
t.get(Calendar.HOUR_OF_DAY)	Hora (0-23)
t.get(Calendar.MINUTE)	Minuto (0-59)
t.get(Calendar.SECOND)	Segundo (0-59)
t.get(Calendar.MILLISECOND)	Milisegundo

<https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html>



## GregorianCalendar

Implementacion de Calendar

### Constructores

[GregorianCalendar\(\)](#)

[GregorianCalendar](#)(int year, int month, int dayOfMonth)

[GregorianCalendar](#)(int year, int month, int dayOfMonth, int hourOfDay, int minute)

[GregorianCalendar](#)(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second)

## CALENDAR

### Ejemplos de uso

```
/Inicializa el calendario a 01/01/2007 y le resta 1 año, 1 día, 4 horas y
//5minutos:
// inicializa calendario a 1 Jan 2007
Calendar calendar = new GregorianCalendar(2007,Calendar.JANUARY,1);

// sustrae 1año 1d 4h 5min
calendar.add(Calendar.YEAR,-1);
calendar.add(Calendar.DAY_OF_MONTH,-1);
calendar.add(Calendar.HOUR,-4);
calendar.add(Calendar.MINUTE,-5);
```

## DATE

- El método *Calendar.getTime()* devuelve un objeto *Date*, con la cantidad de milisegundos transcurridos desde 01/01/1970.
- Colabora con la clase *DateFormat* para convertir una fecha en *String* con un formato determinado.

## SIMPLE DATE FORMAT

- Subclase de la clase abstracta *DateFormat*.
- Parsea fechas, y posibilita conversiones entre *Date* y *String*.

## SIMPLE DATE FORMAT

### Ejemplo

```
private void printCalendar(Calendar calendar)
{
    // define el formato de la salida e imprime
    SimpleDateFormat sdf = new SimpleDateFormat("d MMM yyyy hh:mm aaa");
    String date_imprimir = sdf.format(calendar.getTime());
    System.out.println(date_imprimir);
}
```

## SIMPLE DATE FORMAT

### Parámetros

Letter	Date or Time Component	Examples
G	Era designator	AD
y	Year	1996; 96
M	Month in year	July; Jul; 07
w	Week in year	27
W	Week in month	2
D	Day in year	189
d	Day in month	10
F	Day of week in month	2
E	Day in week	Tuesday; Tue
a	Am/pm marker	PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in am/pm (0-11)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	55
S	Millisecond	978



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



Unión Europea  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



## SIMPLE DATE FORMAT

### EJEMPLOS

Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, 'yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o'clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2001-07-04T12:08:56.235-0700

## ENUM

### ENUMERADOS

Tipo de dato que representa un conjunto cerrado de elementos.

<https://www.baeldung.com/a-guide-to-java-enums>

```
public enum MiEnum {  
    VALOR1("texto1"), VALOR2("texto2"), VALOR3("texto3");  
  
    private String texto;  
  
    private MiEnum(String texto) {  
        this.texto = texto;  
    }  
  
    public String getTexto() {  
        return texto;  
    }  
}
```

## ENUM

### ENUMERADOS

```
1 public enum PizzaStatus {  
2     ORDERED,  
3     READY,  
4     DELIVERED;  
5 }
```

```
1 public class Pizza {  
2     private PizzaStatus status;  
3     public enum PizzaStatus {  
4         ORDERED,  
5         READY,  
6         DELIVERED;  
7     }  
8  
9     public boolean isDeliverable() {  
10         if (getStatus() == PizzaStatus.READY) {  
11             return true;  
12         }  
13         return false;  
14     }  
-- }
```



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



Unión Europea  
Fondo Social Europeo  
Iniciativa de Empleo Juvenil  
El FSE invierte en tu futuro





## Ejercicio 3

Mediante el uso de Enums o Enumerados, vamos a crear un programa que muestre el siguiente menú:

- 1- Nueva Calificación
- 2- Listar Calificaciones

En caso de introducir una opción distinta a 1 o 2 el programa finalizará su ejecución, cuando pulsemos la primera opción, el programa nos solicitará el nombre de un alumno y a continuación nos solicitará que introduzcamos una nota del 0 al 10, el programa validará que hayamos introducido una calificación válida en caso de no ser válida se solicitará de nuevo hasta introducirla correctamente.

Cuando pulsemos la segunda opción se mostrará por consola un listado con las calificaciones que habremos almacenado en la primera opción:

NOTABLE (7) – Alumno 1

INSUFICIENTE (3) – Alumno 2

## Ejercicio 3

Al menú anterior le añadiremos las siguientes opciones:

- 3- Listar Calificaciones (Ordenado por calificación ascendente)
- 4- Listar Calificaciones (Ordenado por calificación descendente)
- 5- Listar Calificaciones SOBRESALIENTES
- 6- Listar Calificaciones NOTABLES
- 7- Listar Calificaciones BIEN
- 8- Listar Calificaciones SUFICIENTE
- 9- Listar Calificaciones INSUFICIENTE