

# Tema 5. Colecciones de tamaño fijo: arrays. Librerías de clases: la clase String

---

## Principales conceptos que se abordan en este tema

- Colecciones de tamaño fijo: arrays.
- La librería Java (Java API),
  - Conocer las clases que proporciona, cómo localizarlas, cómo leer y entender la documentación
- Nos centraremos en la clase String.

## Número de sesiones

Se estiman un total de 44 horas

## 5.1 Arrays en Java

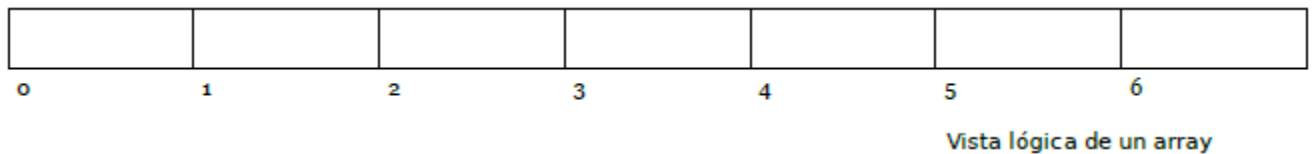
---

A menudo es necesario agrupar una serie de valores para ser tratado de una forma homogénea:

- la nota media de cada uno de los 20 alumnos de un grupo
- la frecuencia de aparición de cada una de las caras de un dado en una serie de lanzamientos
- la relación de canciones que hay en un CD
- el nº de días que tiene cada uno de los meses del año

Es posible agrupar todos estos valores creando una *estructura de datos de tamaño fijo*, un **array**.

Un array es una colección de valores, todos del mismo tipo, que se tratan de una manera específica. Físicamente los elementos de un array se sitúan en memoria uno detrás de otro. Los elementos del array se identifican por la posición que ocupan dentro de él.



Los valores que puede almacenar un array son:

- de un tipo primitivo – int, float, double, char, boolean, .....
- de un tipo referencia – array de objetos

## 5.1.1 Declaración de un array

---

Un array en Java se declara: *tipo[] nombre\_array;*

Ej.

```
int[] arrayEnteros;  
char[] arrayCaracteres;  
float [] notas;
```

La declaración de un array indica el nombre de la variable array y el tipo de valores que va a almacenar. En la declaración no se especifica nada acerca del tamaño del array (no se reserva memoria para él).

### Ejercicio 5.1

Define:

- un array precios que almacene los precios de los diferentes artículos de una tienda
- un array plazasParking que indica si las plazas de un parking están libres o no
- un array que almacene los nombres de una serie de alumnos
- un array que guarde la cantidad de lluvia caída en cada uno de los días del mes de noviembre en una determinada ciudad

**Solución**

- float [] precios;
- boolean [] plazasParking;
- String [] alumnos;
- float [] cantidadLluvia;

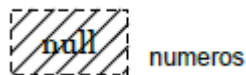
## 5.1.2 Creando objetos arrays

Un array es un objeto que se construye de una manera especial (no se llama al constructor con un nombre de clase como hemos visto hasta ahora con los objetos) a través del operador `new`.

**Ej.**

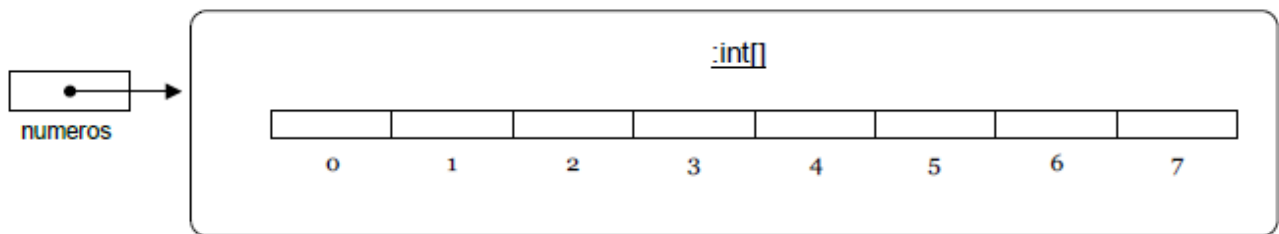
```
int[] numeros; // declaración del array  
numeros = new int[8]; //creación del array
```

En este ejemplo, a través del operador `new` se crea un objeto array con capacidad para almacenar 8 n<sup>os</sup> enteros y se asocia la variable referencia `numeros` (la variable array) con el objeto creado.



Cuando se declara: `int[] numeros;`

Después de hacer: **numeros = new int[8];**



En general, para construir un array: `new tipo[expresión_entera];`  
donde:

- tipo – especifica el tipo de los valores que almacenará el array
- expresión\_entera – especifica el tamaño del array, la cantidad máxima de valores que podrá almacenar

Es posible declarar y crear un array en la misma sentencia,

```
double[] precios = new double[10];
```

Una vez se ha creado un array su tamaño no puede ser modificado.

Como al resto de las variables, Java asigna a los elementos de un array un valor por defecto, cero si son enteros o reales, '\u0000' para caracteres, false para valores lógicos, *null* para los objetos.

### Ejercicio 5.2

Suponiendo las definiciones del ejercicio anterior, crea:

- un array para guardar los precios de 20 artículos
- un array para las n plazas de un parking (n es una variable entera)

- un array para almacenar los nombres de MAX alumnos (MAX es una constante de valor 25)
- un array que guarde la cantidad de lluvia caída en cada uno de los días del mes de noviembre

#### Solución

- `float [] precios=new float[20];`
- `boolean [] plazas=new boolean[n];`
- `String [] alumnos=new String[MAX];`
- `float [] cantidadLluvia=new int[30];`

## 5.1.3 Acceso a los elementos de un array

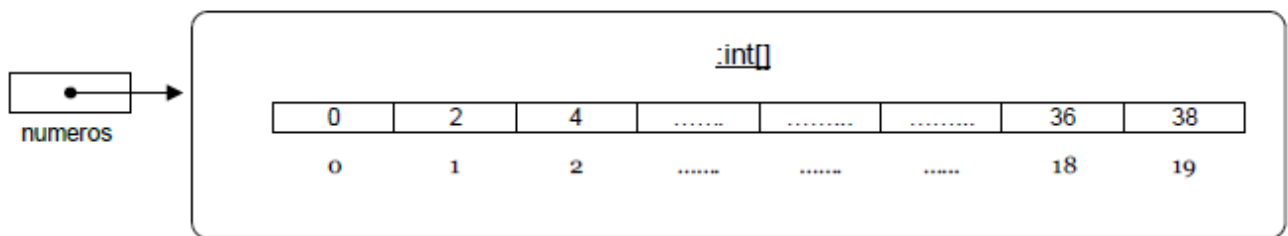
Los arrays tienen un atributo público (no un método), `length`, que indica la longitud del array (recordemos que los arrays son objetos y, por tanto, pueden tener atributos y métodos).

```
int[] numeros = new int[20];  
numeros.length devuelve el valor 20
```

Para acceder a los elementos de un array hay que indicar la posición o índice que ocupa el elemento dentro de él. El valor del índice varía entre **0 y `length - 1`**. Si se especifica un índice fuera del rango se genera un mensaje de error *`ArrayIndexOutOfBoundsException`*.

Ej.

```
int[] numeros = new int[20];  
.....  
for (int i = 0; i < numeros.length; i++)  
    numeros[i] = 2 * i;
```



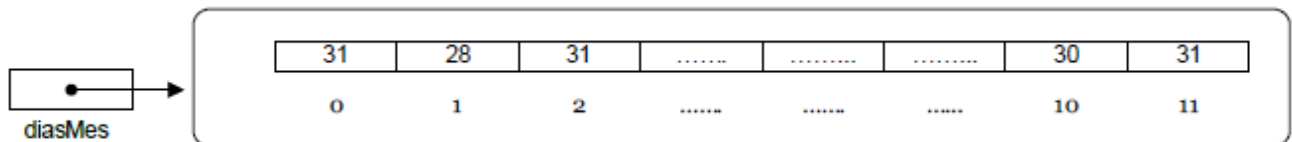
Cada uno de los elementos de un array puede ser tratado como cualquier otra variable. El elemento `numeros[2]` es una variable cuyo valor es 4 y podemos operar con ella de la misma manera que lo haríamos con cualquier otra variable entera.

## 5.1.4 Declarando e inicializando un array

Cuando se sabe a priori los valores que va a contener un array es posible declararlo e inicializarlo en un solo paso, omitiendo la llamada al operador new.

```
int[] diasMes = {31, 28, 31, 30, ....., 30, 31};
```

La sentencia anterior declara e inicializa una variable array que almacena una lista de 12 elementos cada uno de ellos con los valores especificados. Estos dos pasos no es posible separarlos.



Sí se admite:

```
int[] diasMes ;  
diasMes = new int[] {31, 28, 31, 30, ....., 30, 31};
```

### Ejercicio 5.3

- Asigna a la variable numeroElementos el tamaño del array diasMes
- Suponiendo que el año es bisiesto asigna a febrero 29 días

#### Solución

- numeroElementos=diasMes.length;
- diasMes[1]=29;

### Ejercicio 5.4

Escribe el siguiente método:

```
public void escribirArray(){  
    .....  
}
```

Declara dentro del método un array local de 10 elementos enteros e inicialízalo con los valores 1, 2, 3, ..., 10 (declara e inicializa a la vez). Escribe luego el contenido del array en pantalla.

#### Solución

```
int[] array={1,2,3,4,5,6,7,8,9,10};
for(int i=0; i<array.length; i++){
    System.out.println(array[i]);
}
```

## Ejercicio 5.5

Completa el siguiente método:

```
/**
 * @param diaSemana valor entre 1 7
 */
public String encontrarNombreDia (int diaSemana) {
    String[ ] nombres = {"Lunes", "Martes", "Miércoles",
        "Jueves", "Viernes", "Sábado", "Domingo"};
    .....
}
```

Codifica dentro del método una **sentencia if** que devuelva el nombre del día de la semana correspondiente al parámetro que se pasa al método. Si el valor del parámetro es incorrecto se devolverá la cadena **"Día incorrecto"**;

#### Solución

```
if(diaSemana<1 || diaSemana>7){
    return "Día incorrecto";
} else {
    return nombres [diaSemana];
}
```



## 5.1.5 Arrays como argumentos

---

Un array puede ser pasado como argumento a un método. En este caso se pasa la referencia al array, el nombre del array. Esto quiere decir que si el método modifica el parámetro los cambios se reflejarán en el array original.

```
public void invertir(int[] valores){
    int longitud = valores.length;
    int aux;
    int limiteDerecha = longitud - 1;
    for (int i = 0; i < (longitud / 2); i++){
        aux = valores[i];
        valores[i] = valores[limiteDerecha];
        valores[limiteDerecha] = aux;
        limiteDerecha--;
    }
}
```

```
public void escribir(int[] valores){
    for (int i = 0; i < valores.length; i++){
        System.out.print(valores[i] + ",");
    }
    System.out.println();
}
```

Si el array original es:

```
int[] valores = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

y hacemos,

```
invertir(valores);
escribir(valores);
```

el resultado es 9, 8, 7, 6, 5, 4, 3, 2, 1,

## 5.1.6 Array como valor de retorno en un método

---

Un método puede devolver a través de la sentencia `return` un array. El ejemplo anterior podría modificarse de la siguiente manera:

**Ej.**

```
public int[] invertir(int[] valores){
    int longitud = valores.length;
    int[] resultado = new int[longitud];

    for (int i = 0; i < longitud ; i++){
        resultado[i] = valores[longitud - i - 1];
    }

    return resultado;
}
```

En este ejemplo el array que se pasa como parámetro no cambia. Se crea uno local sobre el que se realiza la inversión del array original y es el array local el que se devuelve a través de la sentencia **`return`**.

### Ejercicio 5.6

Construye el siguiente método que calcula la media del array que recibe como parámetro:

```
public double calcularMedia (double [] notas) {

    .....

}
```

**Solución**

```
double suma;
for (int i = 0; i < notas.length; i++){
    suma=suma+notas[i];
}
return suma/notas.length;
```

## Ejercicio 5.7

Define un método que reciba un array de enteros y devuelva el valor máximo.

### Solución

```
public int valorMax ( int [] arrayEnteros ) {  
    int max=0;  
    for (int i=0; i<arrayEnteros.length; i++){  
        if ( arrayEnteros[i] > max )    max=arrayEnteros[i];  
    }  
    return max;  
}
```

## Ejercicio 5.8

Define un método que reciba como parámetro un array de enteros y devuelva otro array con los valores pares encontrados en el original.

### Solución

```
public int[] arrayDePares ( int [] arrayEnteros ) {  
    int [] arrayPares = new int[arrayEnteros.length];  
    int j=0; //Índice para recorrer el array de Pares que voy a crear con  
    los elementos pares encontrados  
    for (int i=0; i<arrayEnteros.length; i++){  
        if(arrayEnteros[i] % 2 ==0) {  
            arrayPares [j]=arrayEnteros[i];  
            j++;  
        }  
    }  
    return arrayPares;  
}
```

## Ejercicio 5.9

Construye el siguiente método:

```
public void rotarDerecha(int[] numeros){  
    .....  
}
```

que rota una posición a la derecha el array numeros.

### Solución

```
int [] aux = new int [numeros.lenght];  
for (int i=0; i<numeros.lenght-1; i++) {  
    aux[i+1]=numeros[i];  
}  
aux[0] = numeros[numeros.lenght];  
numeros=aux;
```

## Ejercicio 5.10

Diseña el método:

```
public int[] generarArray(){  
    .....  
}
```

de la forma siguiente:

- define un array local numerosPares
- genera números aleatorios entre 1 y 50 y guarda en el array los que sean pares
- devuelve el array cuando esté completo (la constante MAX indica el tamaño máximo del array)

### Solución

```
int [] numerosPares = new int[MAX];  
int i=0;  
while (i != numerosPares.lenght){  
    int num=(int)(Math.random()*50)+1;  
    if (num % 2 == 0) {
```

```

        numerosPares[i]=num;
        i++;
    }
}
return numerosPares;

```

## Ejercicio 5.11

Completa el método:

```

public int[] calcularFrecuencias(){
    .....
}

```

que genera 100 nos aleatorios entre 1 y 9 y devuelve la frecuencia de aparición del valor 1, del valor 2, del valor 3, .....

### Solución

```

int [] aleatorios = new int [100];
//Creamos un array auxiliar con los números aleatorios
for (int i=0; i<aleatorios.length; i++){
    int numero=(int)(Math.random() * 9) +1;
    aleatorios[i]=numero;
}
//Creamos un array auxiliar dónde tendremos las frecuencias de cada
valor
int [] frecuencias = new int[9];
//Recorremos el array y vemos cuantos números de cada tipo tenemos
for (int i=0; i<aleatorios.length; i++){
    switch (aleatorios[i]){
        case 1: frecuencias[0] = frecuencias[0]+1;
            break;
        case 2: frecuencias[1] = frecuencias[1]+1;
            break;
        case 3: frecuencias[2] = frecuencias[2]+1;
            break;
        case 4: frecuencias[3] = frecuencias[3]+1;
            break;
        case 5: frecuencias[4] = frecuencias[4]+1;

```

```
        break;
    case 6: frecuencias[5] = frecuencias[5]+1;
        break;
    case 7: frecuencias[6] = frecuencias[6]+1;
        break;
    case 8: frecuencias[7] = frecuencias[7]+1;
        break;
    case 9: frecuencias[8] = frecuencias[8]+1;
        break;
    }
}
return frecuencias;
```

## 5.1.7 Arrays como atributos

Una clase puede tener atributos que sean arrays. En este caso, el array puede declararse e inicializarse tal como hemos visto en los ejemplos anteriores o, lo que es más habitual, se crea el array dinámicamente durante la ejecución de un método, por ejemplo, el constructor de la clase.

Ej.

```
public class EjemploArray{
    private int[] números = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    .....
}
int[9];

] numeros){
    numeros.length;i++)
        1;
```

```
public class EjemploArray{
    private int[] números;
    public EjemploArray(){
        numeros = new

        inicializar(numeros);
    }
    private void inicializar(int[

        for (int i = 0; i <

            numeros[i] = i +

        }
    }
```

### Ejercicio 5.12

- Define una clase **ContadorDias** cuyos objetos cuentan, dada una fecha, los días transcurridos desde principio de año.
- Define en la clase un atributo **díasMes** e inicialízalo (en la propia declaración) con los días que tienen cada uno de los meses del año.
- El constructor de la clase no hace nada.
- La clase tiene un método **contarDias()** que, a partir de tres parámetros, el día, el mes y el año, devuelve un entero que representa la cantidad de días transcurridos.
- Hay que tener en cuenta la posibilidad de que el año sea bisiesto. Esto se comprobará con ayuda del método **private boolean esBisiesto(int año)**. Asumimos que la fecha recibida por el método es correcta.

#### Solución

```
public class ContadorDias {
    private int [ ] diasMes = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};
    public ContadorDias( ) {
    }
```

```

public int contarDias (int dia, int mes, int año) {
    int dias;
    for (i=0; i< mes-1; i++){
        dias = dias + diasMes [i];
    }
    dias += dia;
    if (mes>2 && esBisiesto(año)) {
        dias ++;
    }
    return dias;
}

private boolean esBisiesto(int año) {
    if ((año % 4 == 0) && ((año % 100 != 0) || (año % 400 == 0)))
        return true;
    else
        return false;
}
}

```

## Ejercicio 5.13

Define una clase *EstadisticaLuzSolar* que incluye:

- un atributo *nombresMeses* con los nombres de los meses del año- un atributo *horas* con las horas de sol habidas en cada uno de los meses.

Estos valores son: {100, 90, 120, 150, 210, 250, 300, 310, 280, 230, 160, 120}

- un método *public double getMediaSol()* que devuelve la media de las horas de sol en el año
- un método *public String mesMasSoleado()* que devuelve el nombre del mes con más horas de sol
- un método *public String mesMenosSoleado()* que devuelve el nombre del mes con menos horas de sol

**Solución**

```

public class EstadisticaLuzSolar {

```



```

        private String nombresMeses = { "Enero", "Febrero", "Marzo", "Abril",
        "Mayo", "Junio", "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre",
        "Diciembre" };

        private int horasSol = { 100, 90, 120, 150, 210, 250, 300, 310, 280,
        230, 160, 120 };

        public EstadisticaLuzSolar ( ) {
        }

        public double getMediaSol ( ) {
            int horas = 0;
            for (int i=0; i< horasSol.length; i++){
                horas += horasSol[i];
            }
            return horas/horasSol.length;
        }

        public String mesMasSoleado ( ) {
            int index_maxHora, max=0;
            for (int i=0; i< horasSol.length; i++){
                if ( horasSol [i] > max ){
                    max = horasSol [i];
                    index_maxHora = i;
                }
            }
            return horasSol [index_maxHora];
        }

        public String mesMenosSoleado ( ) {
            int index_minHora, min=0;
            for (int i=0; i< horasSol.length; i++){
                if ( horasSol [i] < min ){
                    min = horasSol [i];
                    index_minHora = i;
                }
            }
            return horasSol [index_minHora];
        }
    }

```

## 5.2 Arrays de objetos (de tipos referencia)

El tipo base de los elementos de un array puede ser, además de un tipo primitivo, un tipo referencia. Esto permite construir arrays de objetos.

Los arrays cuyos elementos son de un tipo referencia se declaran de la misma manera solo que ahora el tipo de los elementos será una clase.

**Ej.**

```
String[] listaPalabras = new String[10];
```

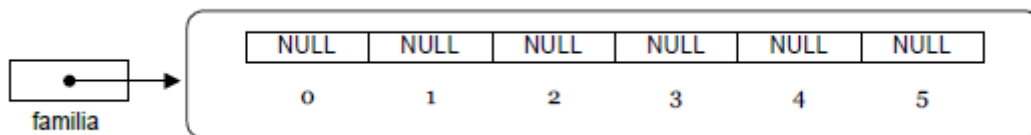
```
Persona[] familia = new Persona[6];
```

```
Alumno[] curso = new Alumno[MAX];
```

Tomemos la declaración,

```
Persona[] familia = new Persona[6];
```

Se está declarando un array de nombre familia dónde cada elemento es del tipo referencia Persona (Persona es una clase). Además se está creando un array con capacidad para 6 personas.



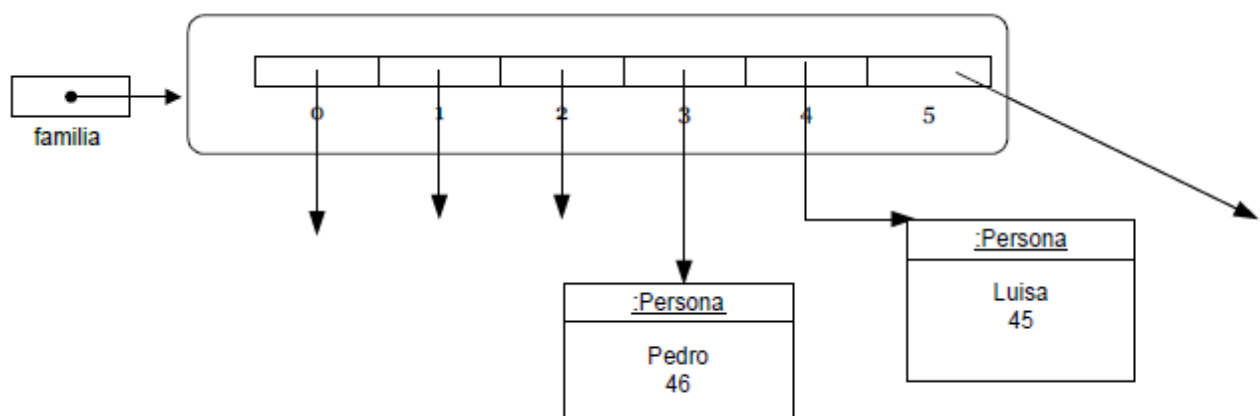
Cuando se crea el array familia cada una de las referencias vale NULL. Todavía no se han creado las personas de la familia. Hay que crear cada uno de los objetos Persona y asignarlo a cada elemento del array. (Esto puede hacerse de muchas formas dependiendo del problema).

**Ej.**

```
familia[3] = new Persona("Pedro",46);
```

```
familia[4] = new Persona("Luisa",45);
```

Después de crear los objetos Persona el array queda:



### Ejercicio 5.14

- Define una clase **Palabra** que modela una palabra de un diccionario inglés-castellano.
  - De cada palabra se guarda su expresión en castellano y en inglés.
  - Incluye el constructor con parámetros y métodos accesorios, `getCastellano()` y `getIngles()`, además del método `toString()` que devuelve un String de la forma “palabra castellano – palabra inglés”
- Define una clase **Diccionario** con un atributo `listaPalabras` que es un array del tipo `Palabra`.
  - La clase incluye también una constante de clase, `MAX_PALABRAS` que indica el nº máximo de palabras que almacenará el diccionario. Añade un atributo adicional `pos` que indicará la siguiente posición en la que se añadirá una nueva palabra al diccionario.
  - Añade a la clase el constructor con un parámetro `numPalabras` que indica el nº de palabras que tendrá el array (si `numPalabras > MAX_PALABRAS` se creará el array de tamaño máximo)
  - Añade un método `insertarPalabra()` que toma una palabra como argumento y la añade al diccionario (siempre se añade detrás de la última palabra introducida). Solo se puede insertar si hay sitio en la lista (esto nos lo puede indicar `pos`). Si no es posible insertar muestra un mensaje de error con `showMessageDialog()` de la clase `JOptionPane` (consulta la API).

*`JOptionPane.showMessageDialog(null, “Error”); --> Deberás importar la clase javax.swing.JOptionPane;`*

- Incluye un nuevo método, `obtenerPalabra()`, que devuelve la palabra de una determinada posición que se pasa como parámetro al método (si la posición no es correcta se devuelve `null`).
  - Define el método `traducirPalabra()` que dada una posición de la palabra en castellano devuelve un String, su traducción al inglés.
  - Incluye un método `toString()` en la clase `Diccionario` que devuelva la representación textual del diccionario
  - Añade el método `escribirDiccionario()` que muestra en pantalla el diccionario
- Crea una clase **InterfazDiccionario** que permita a un usuario interactuar con un diccionario mostrándole las diferentes opciones de que dispone:
    - añadir una nueva palabra,
    - mostrar el diccionario,
    - obtener la traducción de una palabra de una determinada posición,
    - ....
    - (construye adecuadamente esta clase tal como lo hemos hecho en otros ejercicios, mostrando un menú al usuario, incluyendo los métodos privados necesarios para que esta clase lleve a cabo sus responsabilidades)
  - Dibuja el diagrama de clases UML de este ejercicio

#### Solución

En el siguiente [enlace](#) os podéis descargar el proyecto solucionado.

## 5.3 Arrays de 2 dimensiones

---

Hasta ahora todos los arrays utilizados han sido de una dimensión, unidimensionales, colecciones lineales de elementos. Basta un índice para indicar la posición del elemento dentro del array.

Los arrays pueden ser bidimensionales (en general la dimensión puede ser  $n$ ). Un array bidimensional puede representarse lógicamente de forma tabular, como una serie de filas y columnas.

	0	1	2	.....	$n - 1$	$n$
0						
1						
2						
$n - 1$						

*Si  $n = 6$ , tendremos 6 filas y 7 columnas.*

## 5.3.1 Declaración y creación de un array bidimensional

Para declarar un array bidimensional:

*tipo*[ ][ ] *nombre\_array*;

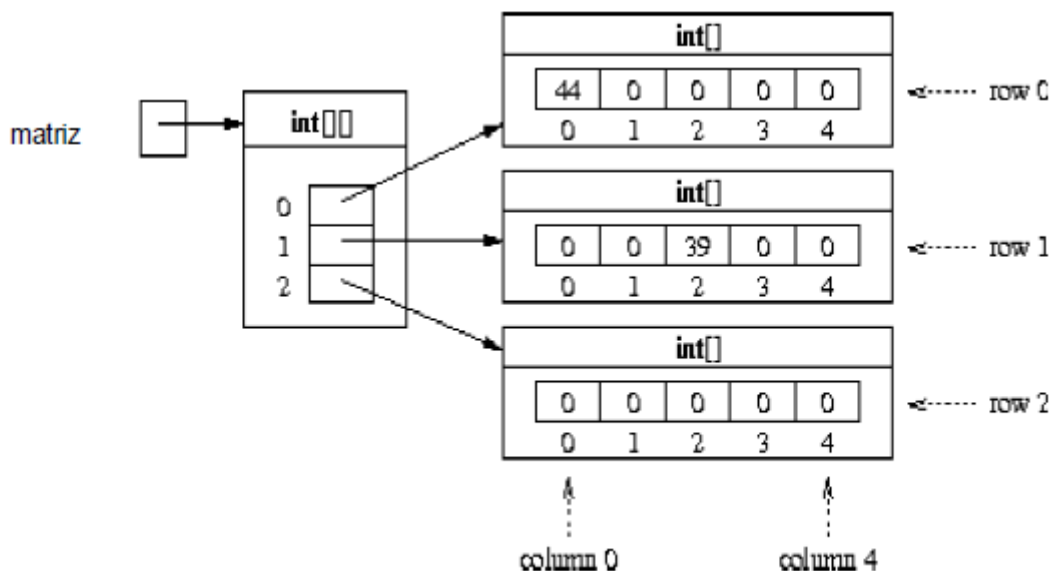
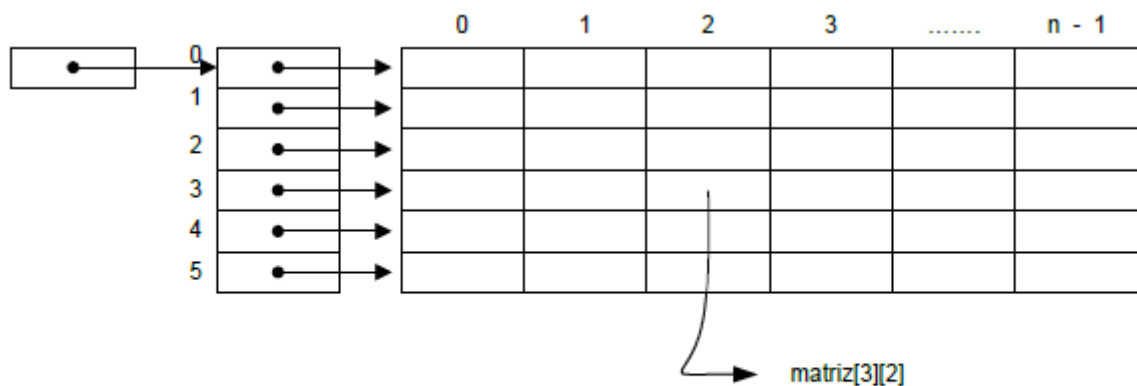
Ej.

*int*[ ][ ] *matriz*; // declara una matriz de enteros

Si queremos crear la matriz con 6 filas y 5 columnas haremos:

*matriz* = new *int*[6][5];

El primer valor entre corchetes indica las filas que tendrá el array y el segundo valor indica las columnas (un array bidimensional se puede considerar un array de arrays).



## 5.3.2 Acceso a los elementos de un array bidimensional

---

Para acceder a los elementos de un array de 2 dimensiones hay que especificar 2 índices, el 1º indica la fila dentro del array, y el 2º la columna.

**Ej.** El siguiente trozo de código inicializa la matriz anterior con valores aleatorios:

```
for (int fila = 0; fila < matriz.length; fila++)  
    for (int columna = 0; columna < matriz[fila].length; columna++)  
        matriz[fila][columna] = (int) (Math.random() * 30 + 1);
```

- `matriz.length` – devuelve el nº de elementos (nº de filas) del array `matriz`
- `matriz[fila].length` – devuelve el nº de elementos del array `matriz[fila]` (es el nº de columnas de la fila `fila`)

## 5.3.3 Declarar e inicializar un array bidimensional

---

Podemos declarar e inicializar, en un solo paso , con una serie de valores un array de dos dimensiones.

**Ej.**

```
int[] [] miArray = { {1, 2, 3, 4},  
                     {5, 6, 7, 8},  
                     {9, 10, 11, 12},  
                     {0, 6, 8, 18} };
```

Declara, crea e inicializa un array de 4 filas y 4 columnas.

También es posible:

```
int[] [] miArray;  
miArray = new int[][] { {1, 2, 3, 4},  
                        {5, 6, 7, 8},  
                        {9, 10, 11, 12},  
                        {0, 6, 8, 18} };
```

## 5.3.4 Arrays bidimensionales con filas de diferente longitud

Cada fila de un array bidimensional es un array en sí mismo. Es por ello que las filas pueden tener distinta longitud (“ragged array” – arrays desiguales).

Ej.

```
int[] [] arrayTriangular = { {1, 2, 3, 4, 5},  
                             {5, 6, 7, 8},  
                             {9, 10, 11},  
                             { 8, 18},  
                             {7} };
```

### Ejercicio 5.15

Dadas las siguientes declaraciones:

```
public class Curso{  
  
    private static final int MAX_ESTUDIANTES = 20;  
  
    private static final int MAX_ASIGNATURAS = 6;  
  
    private int[] [] notas;  
  
    public Curso(){  
  
        .....  
    }  
}
```

Un objeto de la clase Curso guarda las notas obtenidas por los estudiantes de ese curso en una serie de asignaturas.

a) Completa el constructor para que cree el array notas. Después de crearlo llama a un método privado inicializar() que toma como parámetro un array notas y lo inicializa con valores aleatorios comprendidos entre 1 y 10 (las filas representan a los estudiantes y las columnas a las asignaturas).

b) Escribe un método con la siguiente signatura:

```
public double[] calcularMedias()
```

Este método devuelve un array con las notas medias por asignatura.

**Solución**

```
//Dentro del constructor
```



```

notas=new int [MAX_ESTUDIANTES] [MAX_ASIGNATURAS];
//Métodos
private void inicializar (int [] notas){
    for (int filas=0; filas<MAX_ESTUDIANTES; filas++){
        for (int columnas=0; columnas<MAX_ASIGNATURAS;
columnas++){
            notas[filas][columnas]=(int)(Math.random( ) * 10) +1;
        }
    }
}
public double [ ] calcularMedias( ){
    int [ ] medias =new int [MAX_ESTUDIANTES] [MAX_ASIGNATURAS];
    for (int filas=0; filas<MAX_ESTUDIANTES; filas++){
        for (int columnas=0; columnas<MAX_ASIGNATURAS;
columnas++){
            medias[filas][columnas]=notas[filas] [columnas]
/ MAX_ESTUDIANTES;
        }
    }
    return medias;
}

```

## Ejercicio 5.16

- Define y crea (en un solo paso) un array bidimensional, horasTrabajadas, que permita almacenar las horas que han trabajado en cada uno de los 7 días de la semana los 15 trabajadores de una empresa (las filas son los días y las columnas los trabajadores)
- Define un array de valores constantes con los nombres de los 7 días de la semana
- Inicializa el array horasTrabajadas con valores aleatorios entre 8 y 12
- Calcula las horas trabajadas en cada día de la semana por todos los trabajadores (necesitarás un array adicional)
- Escribe el nombre del día de la semana y el total de horas trabajadas calculadas en el apartado anterior

### Solución

```

int [ ] horasTrabajadas = new int[7][15];

```

```

String [ ] diasSemana = {"Lunes", "Martes", "Miércoles", "Jueves",
"Viernes", "Sábado", Domingo};
for (int i=0; i<7; i++){
    for (int j=0; j<15; j++){
        horasTrabajadas[i][j] = (int) (Math.random() * 12)+8;
    }
}
//Array auxiliar para calcular las horas trabajadas en cada día de la
semana
int [ ] horasDias=new int[7];
int horas=0;
for (int i=0; i<7; i++){
    for (int j=0; j<15; j++){
        horas+=horasTrabajadas[i][j];
    }
    horasDias[i] = horas;
}
//Mostramos el día de la semana y las horas trabajadas por semana
for (int i=0; i<horasDias.length; i++){
    System.out.println(diasSemana[i] + ": " +horasDias[i]);
}

```

## Ejercicio 5.17

Define el siguiente método:

```

public int [ ][ ] generarMatrizIdentidad(int dimension) {
    }

```

La matriz identidad tiene la diagonal principal a 1 y el resto de los elementos a 0.

### Solución

```

int [ ] matriz = new int[5];
for (int i=0; i<matriz.length; i++){
    for (j=0; j<matriz.length; j++){
        if(i==j)  matriz [i][j] = 1;
        else     matriz [i][j] = 0;
    }
}

```



## 5.4 Búsqueda en arrays

---

Una de las operaciones más frecuentes sobre un array de cualquier tipo es buscar un determinado valor.

Estudiaremos dos de los algoritmos de búsqueda más habituales: la búsqueda lineal y la búsqueda dicotómica o binaria.

## 5.4.1 Búsqueda lineal

Para efectuar una búsqueda lineal no se necesita ningún requisito, el array puede estar o no ordenado. La búsqueda lineal consiste en recorrer uno a uno, desde el principio, los elementos del array y compararlos con el valor buscado. La búsqueda termina cuando se encuentra el elemento o cuando se acaba el array sin haberlo localizado.

Ej.

```
public boolean buscarLineal(int[] números, int valorBuscado){
    boolean encontrado = false;
    int i = 0;

    while (i < números.length && ! encontrado){
        if (números[i] == valorBuscado)
            encontrado = true;
        else
            i++;
    }
    return encontrado;
}
```

### Ejercicio 5.18

Modifica el método `buscarLineal()` para que devuelva la posición donde se ha encontrado el elemento o `-1` si no está.

Solución

```
public int buscarLineal(int[] números, int valorBuscado){
    int i = 0;
    while (i < números.length){
        if (números[i] == valorBuscado)
            return i;
        else
            i++;
    }
    return -1;
}
```

## 5.4.2 Búsqueda binaria o dicotómica

Para efectuar una búsqueda binaria se requiere que el array esté ordenado.

El algoritmo de búsqueda binaria compara el valor buscado con el que ocupa la posición central del array y:

- si el valor coincide la búsqueda termina
- si el valor buscado es menor que el de la posición mitad se reduce el intervalo de búsqueda a la izquierda de este valor
- si el valor buscado es mayor que el de la posición mitad la búsqueda continúa por la derecha

Este proceso se repite hasta localizar el elemento o hasta que el intervalo de búsqueda quede anulado.

Ej.

```
public boolean buscarDicotomica(int[] números, int valorBuscado){  
  
    boolean encontrado = false;  
    int izquierda = 0;  
    int derecha = numeros.length - 1;  
    int mitad;  
  
    while (izquierda <= derecha && ! encontrado){  
        mitad = (izquierda + derecha) / 2;  
        if (numeros[mitad] == valorBuscado)  
            encontrado = true;  
        else if (numeros[mitad] > valorBuscado)  
            derecha = mitad - 1;  
        else  
            izquierda = mitad + 1;  
    }  
    return encontrado;  
}
```

### Ejercicio 5.19

- Crea un proyecto que incluya una clase **ListaSinRepetidos**
  - La clase tendrá un atributo, elementos, que será un array de nos enteros y un atributo siguiente, también entero, que indica la siguiente posición en el array a añadir un valor
  - El constructor crea el array de MAX elementos donde MAX es una constante de clase de valor 10
  - La clase incluye el método, public void insertar(int nuevoElemento), que añade el valor del parámetro al array únicamente si no existe dentro de éste y el array no está completo
  - Hay un método privado estaElemento() que toma como parámetro un nº y devuelve true si el nº está en el array y false en otro caso

- Incluye en la clase el método `estaCompleto()`, método sin parámetros que devuelve `true` si el array está lleno y `false` en otro caso
- Incluye un método `toString()` que devuelva la representación en formato cadena del array
- Prueba cada uno de los métodos de la clase anterior
- Añade al proyecto una clase **InterfazUsuario** que incluye como atributos unArray de tipo `ListaSinRepetidos` y el atributo `teclado` de la clase `Scanner`.
  - El constructor de la clase `InterfazUsuario` crea el objeto `teclado` y el objeto `unArray`
  - La clase anterior incluye un método `ejecutar()` sin parámetros y sin valor de retorno. Dentro de este método realizaremos un bucle para pedir al usuario valores que guardaremos en nuestro objeto `unArray`. Este proceso termina cuando el usuario introduzca un 0 o cuando el array se complete.
  - Prueba la clase anterior
- Por último incluye en el proyecto la clase **AplicacionSinRepetidos** que solo tendrá el método `main()`.
  - Dentro del `main()` se creará un objeto de la clase `InterfazUsuario` y se invocará el método `ejecutar()`. Testea el proyecto completo.

#### Solución

[Enlace](#) con el proyecto de la solución propuesta

## 5.5 Ordenación de arrays

---

Ordenar un array significa reagrupar sus elementos en un determinado orden:

- - **ascendente** (creciente) – de menor a mayor
- - **descendente** (decreciente) – de mayor a menor



## 5.5.1 Ordenación por inserción directa

Este método de ordenación se basa en considerar una parte del array ya ordenado y situar cada uno de los elementos restantes insertándolos en el lugar que le corresponda según su valor.

Se repite el siguiente proceso desde el 2º elemento hasta el último:

- se toma el 1er elemento de entre los que quedan sin ordenar y se inserta en el lugar que le corresponda entre los elementos situados a su izquierda (que ya están ordenados) desplazando los componentes superiores al tratado un lugar hacia la derecha

24	12	36	5	7	15	Paso 1
12	24	36	5	7	15	Paso 2
12	24	36	5	7	15	Paso 3
5	12	24	36	7	15	Paso 4
.....						

Ej.

```
/**
 * Ordenar en orden ascendente
 */
public void ordenarInsercionDirecta(int[] array)
{
    int aux, j;

    for (int i = 1; i < array.length; i++)
    {
        aux = array[i];
        j = i - 1;
        while (j >= 0 && array[j] > aux)
        {
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = aux;
    }
}
```

Buscar hueco para  
aux a la vez que se  
desplaza a la derecha

## 5.5.2 Ordenación por selección directa

El método consiste en seleccionar en cada paso de ordenación el elemento de menor valor de entre los que quedan por ordenar e intercambiarlo con el primero de la secuencia que se está tratando en ese paso. El proceso se repite desde el primero al penúltimo elemento del array.

24	12	36	<b>5</b>	7	15	Paso 1
<b>5</b>	12	36	24	<b>7</b>	15	Paso 2
5	<b>7</b>	36	24	<b>12</b>	15	Paso 3
5	7	<b>12</b>	24	36	<b>15</b>	Paso 4
.....						

Ej:

```
/**
 * Ordenar en orden ascendente
 */
public void ordenarSeleccionDirecta(int[] array,)
{
    int posmin, aux;

    for (int i = 0; i < array.length - 1; i++)
    {
        posmin = i;
        for (int j = i + 1; j < array.length; j++)
        {
            if (array[j] < array[posmin])
                posmin = j;
        }
        aux = array[posmin];
        array[posmin] = array[i];
        array[i] = aux;
    }
}
```

Selección del mínimo

### Ejercicio 5.20

- Define una clase **CursoProg** con los nombres de los alumnos del curso y las notas de cada uno en el módulo PROG.
  - El máximo nº de alumnos en el curso es MAX (constante de valor 25).
  - La clase incluye como atributo además el nº actual de alumnos en la clase.
  - El constructor crea e inicializa los atributos adecuadamente.
  - El método altaAlumno() añade el nombre de un alumno y su nota de PROG. Si el curso ya está completo no se hace nada. Cada vez que se añade un alumno se llama al método ordenarPorNombre() que ordena los alumnos alfabéticamente. Utiliza para ordenar el método de selección directa.
  - El método cursoCompleto() detecta si el curso está o no lleno.
  - El método escribirListado() escribe en pantalla un listado de los alumnos del curso y su nota.
  - El método listarPorNota() escribe en pantalla un listado pero ahora ordenado de mayor a menor calificación (no modifiques los atributos, los objetos mantienen siempre la lista ordenada por nombre). Puedes utilizar algún método privado de ayuda si lo consideras necesario. El método de ordenación que se utilizará será el de inserción directa. (Consulta la API para leer la documentación del método arraycopy() de la clase System).

Solución

[Enlace](#) al proyecto solucionado

## 5.6 Utilizando la librería de clases de Java

---

La librería de clases de Java (**Java API \_ Application Programmer's Interface**) consiste en cientos de clases útiles para nuestros proyectos. Cada una de estas clases tiene muchos métodos, con o sin parámetros, con o sin valor de retorno.

Un programador Java debe ser capaz de trabajar con la API conociendo muchas de sus clases, pero sobre todo sabiendo cómo explorar e investigar entre la documentación aquellas clases que necesite.

La API de Java está bastante bien documentada. La documentación está disponible en HTML por lo que puede ser leída en un explorador web.

### Nota

- Para utilizar una copia local de la documentación de la API de Java en BlueJ se descomprime el fichero .zip que contiene la documentación en el lugar del disco en que queramos ubicarla. Así se crea una carpeta docs.
- Abrimos el navegador y utilizando “Abrir Fichero” abrimos el fichero index.html que está en la carpeta docs/api
- Copiamos la dirección URL desde el campo dirección de nuestro navegador
- Abrimos BlueJ y en Herramientas/Preferencias/Miscellánea pegamos la URL en el campo JDK documentación URL (desactivamos la casilla Use esta URL .....)

Leer y entender la documentación es la primera tarea a realizar para poder trabajar con la API.

Más adelante aprenderemos a preparar nuestras propias clases para que otras personas puedan utilizarla de la misma manera que la librería standard.

## 5.6.1 Leyendo la documentación de una clase

Para poder leer la documentación de una clase desde BlueJ elegimos “Librería de clases Java” desde el menú “Ayuda”. Se abre el navegador y visualiza la página principal de la Java API. El explorador muestra tres partes:

- arriba a la izquierda podemos ver una lista de paquetes
- abajo una lista de todas las clases de Java
- en la ventana central aparecen los detalles de la clase o paquete seleccionado

### Ejercicio 5.21

- Localiza la clase Random en la documentación de Java. ¿En qué paquete está? ¿Cómo se crea una instancia? ¿Qué hace el método nextInt( )?
- Busca la clase Math. ¿En qué paquete está? ¿Qué hace el método round( )? ¿Qué es PI?
- Localiza el método arraycopy( ) de la clase System y el método compareTo( ) de la clase String.
- Consulta en la API la clase Arrays. ¿Qué métodos te resultan conocidos?

#### Solución

##### RANDOM

- La clase Random está en el paquete **java.util**
- Para crear una instancia tendremos que declarar una variable o atributo del tipo Random:
- **private Random numero;**
- E inicializarla en el lugar correspondiente:
- **Random ran = new Random( );**
- El método nextInt( ) genera un valor aleatorio numérico:
- **int numero=ran.nextInt( );**

##### MATH

- La clase Math está en el paquete **java.lang**
- El método **round( )** redondea un valor.
- PI es una constante que tiene el valor del numero PI

##### ARRAYCOPY

- El método arraycopy está en la Clase System que está en el paquete **java.lang**
- Permite hacer copias de arrays (completos o por rangos)

## COMPARETO

- Compara dos cadenas de caracteres e indica si la primera respecto de la segunda es menor (resultado <0), igual (resultado =0) o mayor (resultado>0)
- La comparación la realiza alfabéticamente

## ARRAY

- equals
- copyOf
- sort
- toString
- ...

## 5.6.2 Interface versus implementación

---

La documentación de una clase incluye:

- el nombre de la clase
- una descripción general de la clase
- la lista de los constructores y métodos de la clase
- los valores de retorno y parámetros para los constructores y métodos
- una descripción del propósito de cada constructor y método

Toda esta información constituye la **interface** de la clase. La interface no muestra el código fuente de la clase, describe lo que una clase puede hacer (los servicios que proporciona) y cómo puede ser utilizada sin mostrar su implementación (cómo lo hace). La interface es la parte visible, constituye una abstracción.

La documentación no muestra:

- el código fuente
- los atributos privados
- métodos privados

El código fuente que define una clase constituye su **implementación**. Un programador trabaja en la implementación de una clase y al mismo tiempo hace uso de otras clases vía sus interfaces.

El término interfaz puede aplicarse también a un método individual. La interfaz de un método consiste en su signatura que indica:

- el modificador de acceso (public, private)
- el tipo del valor de retorno
- el nombre del método
- la lista de parámetros formales y sus tipos

**Ej.**

*public int maximo(int x, int y)*

La interfaz de un método proporciona todo lo que necesitamos para saber utilizar el método.

## 5.6.3 Public y private

---

Analicemos con mayor detalle los modificadores de acceso, public y private, que ya hemos utilizado con atributos, métodos y clases.

Los **modificadores de acceso** definen la visibilidad de un atributo, constructor, método e incluso una clase.

Si un método es público puede ser invocado desde dentro de la clase donde está definido y fuera de ella. Si el método es privado sólo puede ser llamado dentro de la clase en la que se ha declarado.

Ahora que hemos visto la diferencia entre interface e implementación de una clase se comprende mejor el propósito de los modificadores de acceso public/private.

Recordemos:

- la interfaz de una clase proporciona información sobre cómo utilizar la clase. Define lo que la clase puede hacer. El interfaz es la parte pública de una clase. Todo lo que esté declarado como public en una clase formará parte del interfaz
- la implementación define cómo trabaja una clase. El cuerpo de los métodos y los atributos (que en su mayoría son private) forman parte de la implementación. La implementación es la parte privada, todo lo que esté declarado como private. El usuario de una clase no necesita conocer nada acerca de su implementación, no le debe estar permitido acceder a la parte privada de la clase.

\* **Ocultar la información** – el principio de ocultamiento de la información asegura la modularización de la aplicación, una clase así no depende de cómo esté implementada otra, lo que conduce a un bajo acoplamiento entre las clases y, por tanto, a un más fácil mantenimiento.

\* **Métodos privados y atributos públicos** – la mayoría de los métodos que hemos incluido en las clases han sido públicos. Esto asegura que otras clases pueden utilizarlos. Hay métodos, sin embargo, que los hemos definido como privados. Estos métodos resultaban al descomponer una tarea en tareas más pequeñas y hacer así la implementación más fácil. Otra razón para hacer que un método sea privado es para describir una tarea que se necesita en varios métodos diferentes de una clase. Para evitar repetir código diseñamos un método privado que la realice.

En cuanto a los atributos deberían ser siempre privados. Declarar un atributo público rompe el principio de ocultamiento de la información y hace la clase que lo contiene vulnerable a cambios en la implementación. Además permite a los objetos mantener un control mayor sobre su estado. Si el acceso a los atributos privados se hace a través de accesorios y mutadores entonces el objeto puede asegurar que el atributo nunca contiene un valor inconsistente.



## 5.6.4 Paquetes y sentencia import

---

Las clases de la API de Java no están automáticamente disponibles para su uso como ocurre con las clases de nuestro proyecto actual. Hay que indicar en el código fuente que queremos utilizar una clase de la librería. A esto se llama **importar la clase** y se realiza con la sentencia **import**.

**import nombre\_de\_la\_clase;**

Java utiliza paquetes para organizar en grupos la multitud de clases de su librería. Un paquete es un conjunto de clases lógicamente relacionadas. Los paquetes pueden anidarse (un paquete puede contener otro paquete).

Por ejemplo, la clase *Random* está dentro del paquete `java.util`. El nombre completo o nombre calificado de la clase es el nombre del paquete que la contiene seguido por un punto y el nombre de la clase: *java.util.Random*

Java permite importar paquetes completos:

Diagrama de la sentencia `import java.util.*;`. La parte `java.util` está etiquetada como "nombre del paquete" con una flecha que apunta a ella. La parte `*` está etiquetada como "todas las clases del paquete" con una flecha que apunta a ella.

La sentencia se incluye al principio del código fuente de la clase que vaya a utilizar la librería:

*import java.util.Random;*

Una clase de un paquete se puede utilizar sin incluir una sentencia `import` si se utiliza el nombre completo (calificado) de la clase (aunque esta opción no es muy cómoda).

*private java.util.Scanner teclado = new java.util.Scanner(System.in);*

Hay algunas clases que se utilizan muy frecuentemente, como la clase **String**. Estas clases están situadas en el paquete `java.lang`. Este paquete se importa automáticamente en cada clase.

A partir de la versión Java 1.5 se puede importar elementos `static` de una clase.

*Ej. import static java.lang.Math.\*;*

## 5.7 La clase String

---

Un string es una secuencia de caracteres. En muchos lenguajes los strings se tratan como arrays de caracteres, en Java un string es un **objeto**. Java proporciona las clases **String**, **StringBuilder** y **StringTokenizer** para almacenar y procesar cadenas. En la mayoría de los casos utilizaremos la clase String para crear cadenas de caracteres.

La clase *String* se incluye en el paquete *java.lang*. Esta clase:

- a. modela una secuencia de caracteres y
- b. proporciona múltiples constructores y métodos para tratar de forma individual los caracteres, comparar cadenas, buscar subcadenas, extraer subcadenas, ....

El valor de un string se escribe entre "".

## 5.7.1 Creando un String

Como cualquier otro objeto creamos un string llamando al constructor de la clase:

- `String cadena = new String();` //crea una cadena vacía
- `String cadena = new String("Ejemplo de cadena");`

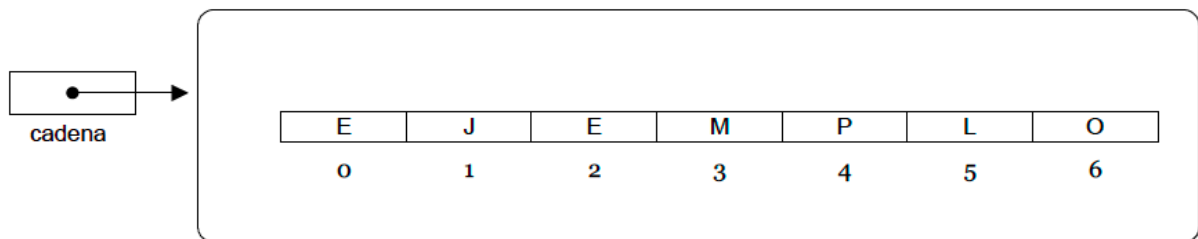
Hay una forma abreviada de crear e inicializar un String:

```
String cadena = "Bienvenido a Java";
```

Si únicamente hacemos,

```
String cadena;
```

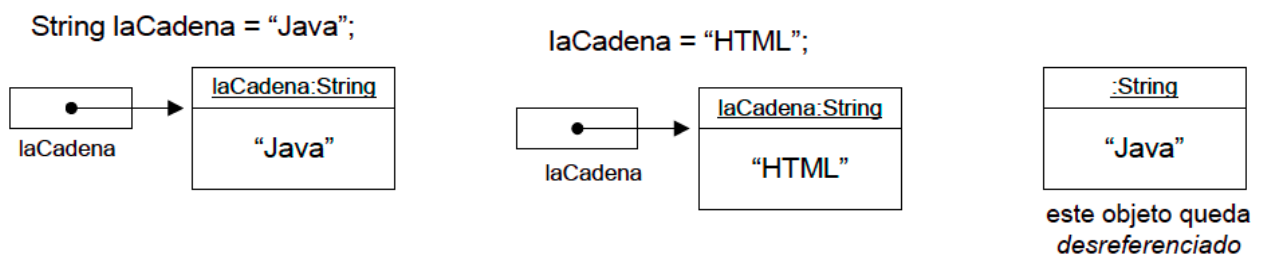
se crea una referencia que vale *null*. Puesto que un string es un objeto, una variable string almacena una referencia al objeto String que contiene la cadena.



Un objeto String es inmutable. Una vez creado su contenido no puede cambiar, contendrá esa cadena toda su vida. Si hacemos:

- `String laCadena = "Java";`
- `laCadena = "HTML";`

La primera sentencia crea un objeto con el contenido "Java" y asigna su referencia a *laCadena*. La segunda sentencia crea un nuevo objeto con el contenido "HTML" y asigna la nueva referencia a *laCadena*.



## 5.7.2 Longitud de una cadena y acceso a caracteres individuales

---

La longitud de una cadena se obtiene a través del método `length()`. **int length()**

```
String cadena = "Hola";  
int longitud = cadena.length(); // longitud = 4
```

El acceso a cada uno de los caracteres de un string se realiza con el método `charAt()` que devuelve el carácter de la posición `pos`.

**char charAt(int posicion)** *posicion* es un valor que oscila entre 0 y `cadena.length() - 1`

```
String cadena = new String("Hola qué tal");  
char carac = cadena.charAt(5); // carac = 'q'  
int indice = 6;  
System.out.println(cadena.charAt(indice + 3)); //escribe t
```

## 5.7.3 Concatenación de cadenas

---

Para concatenar dos cadenas se utiliza el método `concat()` - **String concat(String cadena)**

```
String str1 = "Bienvenido ";  
String str2 = "al lenguaje Java ";  
String str3 = str1.concat(str2); // str3 es "Bienvenido al lenguaje Java"
```

La concatenación de strings es una operación tan común que Java proporciona un mecanismo más cómodo a través del operador `+`.

```
String str1 = "Hola";  
String str2 = " tal";  
String resul = str1 + " que" + str2;
```

Recordemos que el operador `+` también concatena un string combinado con un tipo primitivo tal como un `int`, `double` o `char`. El valor del tipo primitivo se convierte automáticamente a string y luego se concatena.

```
int valor = 3;  
String str = "Resultado = " + valor;  
System.out.println(str);
```

## 5.7.4 Extrayendo subcadenas

---

Para extraer subcadenas se utiliza el método `substring()`. Este método está sobrecargado y tiene dos versiones:

**`public String substring(int posInicial, int posFinal)`** – devuelve una subcadena (es un string) que comienza en `posInicial` y llega hasta `posFinal` (sin incluir esta posición final)

**`public String substring(int posInicial)`** – devuelve una subcadena (es un string) desde `posInicial` hasta el final de la cadena

- *`String mensaje = "Hola";`*
- *`String str1 = mensaje.substring(1); str1 vale "ola"`*
- *`String str2 = mensaje.substring(1, 3); str2 vale "ol"`*

## 5.7.5 Comparación de cadenas

Para comparar el contenido de dos objetos `String` podemos utilizar el método `equals()` - **`boolean equals(Object obj)`**

En general este método permite comparar dos objetos cualesquiera, también objetos `String`.

```
String mensaje = "Hola";  
  
if (mensaje.equals("Hola"))  
  
.....
```

También se puede utilizar el método `compareTo()` - **`int compareTo(String s)`**

Este método devuelve un valor:

```
if (str1.compareTo(str2) == 0) .....
```

- 0 si las dos cadenas que se comparan son iguales
- < 0 si `str1` es menor que `str2`
- > 0 si `str1` mayor que `str2`

Cuando se comparan dos cadenas la comparación es lexicográfica y sensible a mayúsculas (la comparación se hace carácter a carácter, por ejemplo, "abc" es menor que "abg" y teniendo en cuenta la longitud, por ejemplo, "aba" es menor que "abaaa").

Nunca hay que utilizar el operador `==` para comparar dos cadenas. *if (str1 == str2) .....* compara las referencias `str1` y `str2`, se comprueba si apuntan al mismo objeto no si se trata de la misma secuencia de caracteres. El operador `==` sirve para comparar tipos primitivos y no tipos objeto.

### Ejercicio 5.22

Comprobar lo que sucede al ejecutar las clases [TestEq](#) y [TestEqPoint](#).

Mostrar retroalimentación

**TestEq:** Lo que está sucediendo es que Java almacena en las variables referencias a memoria, y la referencia de *hello1* no es la misma que *hello2*. La forma que tiene Java de ver qué dos objetos son iguales, es decir, que sus variables contienen los mismos valores, es mediante el método `equals(Object o)`, que en la clase `String` sobreescribe al de la clase `Object` (la clase de la que heredan todas las clases en Java)

Si se realiza el test para la igualdad de dos objetos de la clase `Point` mediante la clase **`TestEqPoint`**, en el resultado se verá que no se proporciona seguridad acerca de la igualdad entre dos puntos, ya que el

método equals que tiene por defecto una clase es el mismo que proporciona la operación ==. Habrá que recurrir a la sobreescritura de métodos.

## Ejercicio 5.23

a) Dada la declaración:

```
String unaCadena = new String("Ejemplo");
```

si hacemos, `unaCadena = unaCadena.toUpperCase();`

1. ¿quién es el receptor del mensaje `toUpperCase()`?
2. ¿sería correcto hacer: `unaCadena.toUpperCase()` ?

b) Dada las declaraciones:

- *String* str = "Aprendiendo cadenas en Java";
- *String* resul, otra = "cadena de ejemPLO";
- *char* caracter;
- *int* pos;

Asigna a resul y/o a carácter y/o a pos:

- 1.- la cadena str convertida a mayúsculas
- 2.- el carácter de la posición 6 de str
- 3.- el último carácter de la cadena str
- 4.- compara str con la cadena otra sin tener en cuenta mayúsculas ni minúsculas
- 5.- pregunta si str empieza por "Ba"
- 6.- sustituye en str todas las 'e' por '\*'
- 7.- devuelve la primera aparición de la 'd' en str
- 8.- localiza la última aparición de la 'c' en str. Extrae , a partir de ahí, la subcadena existente hasta el final.
- 9.- convierte a String el valor 66
- 10.- pregunta si la cadena str es vacía (Hazlo de varias formas)

### Solución

1- la cadena str convertida a mayúsculas

```
str.toUpperCase( );
```

2 - el carácter de la posición 6 de str

```
str.charAt(6);
```

3 - el último carácter de la cadena str

```
str.charAt(str.length()-1);
```

4 -compara str con la cadena otra sin tener en cuenta mayúsculas ni minúsculas



**str.equalsIgnoreCase(otra);**

5 - pregunta si str empieza por "Ba"

**if(str.startsWith("Ba"))**

6 - sustituye en str todas las 'e' por '\*'

**str.replaceAll('e','\*');**

7 - devuelve la primera aparición de la 'd' en str

**str.indexOf('d');**

8 - localiza la última aparición de la 'c' en str. Extrae , a partir de ahí, la subcadena existente hasta el final.

**str.substring(str.lastIndexOf('c'));**

9 - convierte a String el valor 66

**Integer.toString(66);**

10 - pregunta si la cadena str es vacía (Hazlo de varias formas)

**if(str.length() == 0)**

**if(str.isEmpty())**

## Ejercicio 5.24

Dada la clase:

```
public class Palabra {  
    private String palabra;  
  
    public Palabra(String palabra) {  
        this.palabra = palabra;  
    }  
    .....  
}
```

Construye los siguientes métodos:

- public String invertir()** – obtiene la palabra invertida
- public int contarVocales()** – devuelve el nº de vocales de la palabra
- public boolean mayorQue(String palabra)** – devuelve true si la palabra es mayor que la recibida como parámetro, false en otro caso
- public boolean esPalindromo()** – devuelve true si es un palíndromo (se lee igual de izquierda a derecha que de derecha a izquierda)
- public boolean esPalindromoRecursiva()** - versión recursiva del método anterior. Llama desde aquí al método privado boolean esPalin(String p)
- public String invertirConPila(Pila p)** – devuelve la palabra invertida con ayuda de la pila p (la pila es de caracteres)

[Enlace](#) de la solución del proyecto

## Ejercicio 5.25

Abre el proyecto Lista de Nombres ([Enlace al proyecto](#)) que se te proporciona. El proyecto incluye una clase ListaNombres que mantiene una lista de nombres ordenada alfabéticamente. Los nombres en la lista se guardan siempre en minúsculas. La clase define como atributo un array lista de tipo String, la posición del último nombre que está en la lista y una constante MAX\_NOMBRES de valor 10. El constructor crea la lista vacía e inicializa la posición a -1. ¿Por qué se inicializa a -1?

Los métodos que incluye son:

- **insertarNombre()** – dado un nombre lo inserta en orden en la lista únicamente si no está. Si la lista está llena no se insertará. El método devuelve true si se ha podido realizar la inserción y false en otro caso. Para ver si el nombre está en la lista se utilizará el método privado estaNombre() que tomando como parámetro un nombre devuelve true si está en la lista y false en otro caso. La búsqueda será dicotómica.
- **nombreMasLargo()** – devuelve el nombre de mayor longitud en la lista o null si la lista está vacía
- **borrarLetra()** – borra de la lista los nombres que empiezan por la letra indicada como parámetro (el parámetro es de tipo char). Utiliza el método privado borrarDePosicion()
- **empiezanPorLetra()** – devuelve una lista de String que contiene los nombres que empiezan por la letra indicada como parámetro al método (la letra es de tipo char). Da igual que sea minúscula o mayúscula.
- **empiezanPor()** – devuelve cuantos nombres empiezan por la cadena indicada como parámetro al método. Da igual que sea minúscula o mayúscula
- **listaVacía()** – devuelve true si la lista está vacía
- **listaLlena()** – devuelve true si la lista está llena
- **toString()** – devuelve una representación textual de la lista

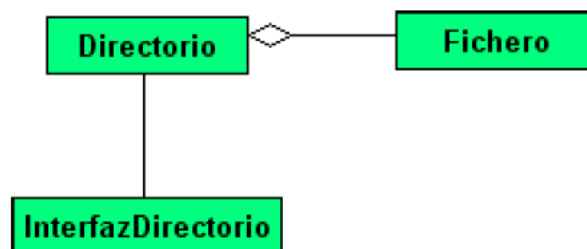
Añade al proyecto una clase **DemoLista** que:

- define un atributo lista de tipo ListaNombres y define los siguientes métodos:
- en el constructor crea la lista y añade una serie de nombres concretos (unos 7)
- **testInsercion()** – añade cuatro nombres más, algunos estarán repetidos
- **testBorrado()** – borra los nombres que empiezan por 'm'
- **testEmpiezan()** – escribe los nombres que empiezan por "a" y cuántos empiezan por "al"
- **testMasLargo()** – escribe el nombre más largo de la lista
- **testEscribir()** – escribe la lista completa (con ayuda del método toString())

## Ejercicio 5.26

Un directorio de un disco contiene ficheros de un determinado tamaño y extensión. Sobre un directorio, además de añadir un nuevo fichero podremos ordenarlo por nombre y cambiar de nombre a un fichero. También podemos obtener el fichero de mayor tamaño y el tamaño total del directorio. Crearemos un proyecto que incluya las clases necesarias para realizar las tareas indicadas.

El diagrama de clases muestra la siguientes clases en el proyecto:



La clase **Fichero** incluye como atributos el nombre del fichero (el nombre supondremos es un nombre que incluye la extensión, por ejemplo, .exe, o .java, o .pas, o .class) y el tamaño en Kbytes.

- El constructor tiene dos parámetros, nombre y tamaño, para crear los objetos.
- Incluye además un accesor para el tamaño, un accesor `getNombre()` que devuelve el nombre sin extensión, otro `getNombreCompleto()` que devuelve el nombre completo del fichero – con la extensión - y `getExtension()` que devuelve la extensión del fichero.
- Hay un mutador `cambiarNombre()` que permite cambiar el nombre al fichero y el método `toString()`

La clase **Directorio** define:

- constantes estáticas:
  - `MAX_FICHEROS` que indica el nº máximo de ficheros que puede almacenar el directorio (es el mismo para todos los directorios por eso definimos la constante como static)
  - `LLENO` con el valor 1
  - `REPETIDO` con el valor 2
  - `ÉXITO` con el valor 3
- un atributo `ficheros` que es un array de objetos **Fichero**
- un atributo entero siguiente que indica la posición siguiente a añadir el fichero

- el constructor inicializa el atributo siguiente y el array adecuadamente
- un método public int añadirFichero(String queNombre, int queTamaño) que dados como parámetros un nombre de fichero y su tamaño lo añade al directorio. Los ficheros se añaden siempre al final (no hay orden en el directorio). Solo se podrá añadir si hay espacio en el directorio. Si no se puede añadir no se hace nada (se puede crear un método privado estaLleno() que devuelve true si el directorio está completo). Tampoco se puede añadir si ya hay un fichero con el mismo nombre. Hay un método privado int existeFichero(String nombre) que indica si ya existe un fichero con ese nombre en el directorio. Si existe devuelve la posición en la que está y si no existe devuelve -1. El método añadirFichero() devuelve un entero que indica el resultado de la operación (si ha habido éxito o estaba completo .....)
- un método getTamañoDirectorio() que devuelve un entero que es el tamaño total en KBytes del directorio
- el método getFicheroMayor() devuelve un array de String que es el nombre/s de/l os fichero/s de mayor tamaño
- el método ordenarPorNombre() devuelve un array de ficheros ordenados por nombre (el array original no se modifica)
- el método public boolean renombrarFicheroString (String queFichero, String nuevoNombre) cambia de nombre a un fichero. Solo se puede renombrar si existe el fichero en el directorio y si no hay otro fichero con el mismo nombre (se puede utilizar aquí de nuevo el método existeFichero())
- incluye un método toString() con la representación textual del directorio

La clase InterfazDirectorio incluye los métodos necesarios para interactuar con el usuario y poder efectuar operaciones sobre el directorio. Entre otras cosas la clase incluye:

- el directorio sobre el que se va a actuar (será un atributo)
- un método menu() que presenta al usuario las operaciones que puede realizar sobre el directorio y devuelve la opción elegida
- métodos adicionales para tratar todas las opciones
- el método iniciar() que es el método principal que controla toda la lógica de la clase

#### Solución

[Enlace](#) con la solución del ejercicio

## 5.8 Las clases **StringBuilder** y **StringTokenizer**.

---

### La clase **StringBuilder**

Los objetos *String* son inmutables, una vez asignado un valor éste no puede modificarse. Java crea un nuevo objeto *String* cuando se le asigna un nuevo valor.

La clase *StringBuilder* modela cadenas que pueden ser modificadas (la clase *StringBuffer* ofrecía la misma funcionalidad hasta la aparición de *StringBuilder* con Java 1.5. *StringBuffer* se mantiene en Java 5 pero al ser sincronizada es menos eficiente que la clase *StringBuilder* que no lo es).

La clase *StringBuilder* está en el paquete *java.lang*. Los objetos *StringBuilder* son más lentos. Los métodos de *StringBuilder* no pueden ser usados con *String*. Las operaciones principales con objetos de este tipo son las de añadir e insertar.

La clase incluye, entre otros, los siguientes constructores y métodos:

- *new StringBuilder()* – construye una cadena vacía con una capacidad inicial de 16 caracteres
- *new StringBuilder(String s)* – construye un objeto *StringBuilder* a partir del contenido del *String s*
- *int length()* – devuelve el nº de caracteres actualmente almacenados en la cadena
- *char charAt(int indice)* – devuelve el carácter de la posición marcada por índice (el primer carácter tiene índice 0)
- *void setCharAt(int indice, char c)* – se asigna a la posición índice el carácter *c*
- *StringBuilder append(String s)* – añade la cadena *s* al final del objeto *StringBuilder*
- *StringBuilder append(StringBuilder s)* – añade el objeto *StringBuilder s* al objeto actual
- *StringBuilder delete(int inicio, int final)* – borra los caracteres entre inicio y final – 1
- *StringBuilder deleteCharAt(int indice)* – borra el carácter de la posición índice
- *StringBuilder reverse()* - invierte la cadena
- *StringBuilder insert(int posicion, String s)* – inserta la cadena *s* a partir de posición
- *String toString()* - devuelve un *String* a partir de un *StringBuilder*

Ej:

```
public String borrarLaE(String original) {
    StringBuilder resul = new StringBuilder(original);
    int i = 0;
    while (i < resul.length()) {
        if (resul.charAt(i) == 'E')
            resul.deleteCharAt(i);
        else
            i++;
    }
    return resul.toString();
}
```

### La clase **StringTokenizer**

La clase *StringTokenizer* permite “partir” una cadena en subcadenas individuales según un criterio de separación. Por ejemplo, se pueden extraer las palabras de una frase teniendo en cuenta que están separadas por blancos.

El delimitador se utiliza para indicar el carácter / s separadores de las “palabras” o tokens. El delimitador por defecto es el blanco (o tabulador o salto de línea) pero se pueden especificar otros como la ‘,’ o ‘:’ o ‘//’

La clase se encuentra en el paquete *java.util*.

- ***new StringTokenizer(String s)*** –construye un StringTokenizer a partir de la cadena s. En este caso el delimitador es el espacio en blanco
- ***new StringTokenizer(String s, String delimitador)*** – construye un StringTokenizer a partir de la cadena s indicando que el delimitador es el especificado. El delimitador puede ser un String de varios caracteres, en ese caso, cada uno de ellos es un delimitador
- ***new StringTokenizer(String s, String delimitador, boolean flag)*** – si flag está a true el delimitador se devuelve como un token más
- ***int countTokens()*** – devuelve el nº de tokens de la cadena
- ***String nextToken()*** – devuelve el String que representa el siguiente token
- ***boolean hasMoreTokens()*** – devuelve true si hay más tokens en la cadena

**Ej:**

```
import java.util.StringTokenizer;
.....

public void ejemploStringTokenizer() {
    final String SEPARADOR = “,”;
    String frase = “Esto, es, un, ejemplo”;
    StringTokenizer palabras = new StringTokenizer(frase, SEPARADOR);
    while (palabras.hasMoreTokens()) {
        String unaPalabra = palabras.nextToken();
        System.out.println(unaPalabra);
    }
}
```

## 5.8.1 La clase Scanner como tokenizer

---

La clase Scanner (a partir de Java 5) puede utilizarse para separar tokens de un String. Además si queremos que el delimitador sea una palabra completa hay que utilizar esta clase (el delimitador en un StringTokenizer es un char).

Los métodos de esta clase para separar y obtener los diferentes tokens son, entre otros: (consulta la API)

- new Scanner(String s)
- boolean hasNext()
- boolean hasNextInt()
- boolean hasNextFloat()
- String next()
- String nextLine()
- int nextInt()
- float nextFloat()
- Scanner useDelimiter(String s)

**Ej:**

```
import java.util.Scanner;
.....

public void ejemplo01Scanner() {

    String s = "Ejemplo de Java";
    Scanner sc = new Scanner(s);
    sc = sc.useDelimiter("de");
    while (sc.hasNext())
        System.out.println(sc.next());

}
```

**Ej:**

```
import java.util.Scanner;
.....

public void ejemplo02Scanner()v{

    String s = "1 2 3 4 5 6 7";
    Scanner sc = new Scanner(s); //por defecto el delimitador es el espacio
    int suma = 0;
    while (sc.hasNextInt())
        suma += sc.nextInt();
    System.out.println("La suma es " + suma);

}
```

## 5.9 Signatura del método main: argumentos en la línea de comandos

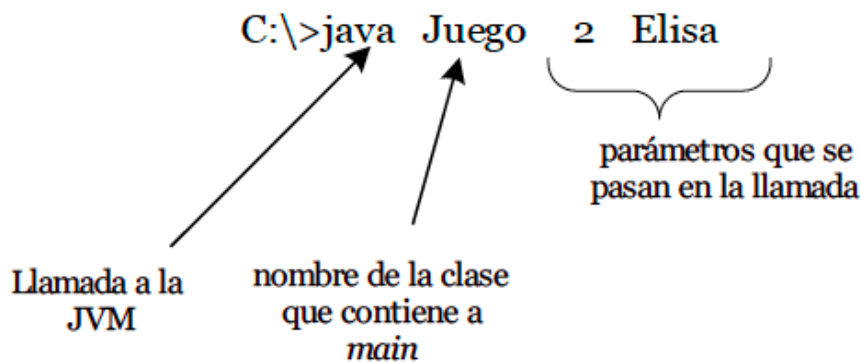
Ya sabemos que para iniciar una aplicación Java fuera del entorno BlueJ necesitamos invocar a un método sin que exista ningún objeto aún. Este método es un método de clase (estático) denominado `main()`. El usuario indica la clase que debe iniciarse (desde la línea de comandos llamando a, `java nombre_clase`) y esta clase es la que contendrá a `main()`.

El método `main()` tenía una signatura específica:

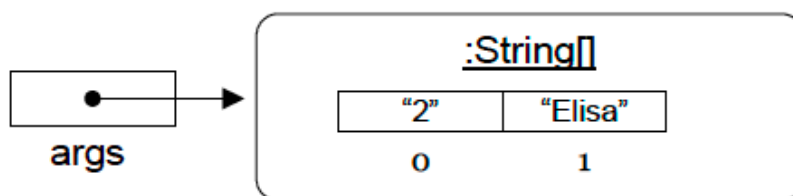
```
public static void main(String[ ] args)
```

El parámetro del método `main()` es un array de tipo `String`: **`String[ ] args`**. Esto permite al usuario pasar argumentos adicionales en la línea de comandos cuando se invoca a una clase Java para su ejecución.

Ej:



Cada valor incluido en la línea de comandos después del nombre de la clase se lee como un `String` separado y se pasa al método `main()` como un elemento del array `args`. En nuestro ejemplo, el array `args` contendrá 2 elementos, las cadenas `"2"` y `"Elisa"`.



Cuando se invoca al método `main()` el intérprete Java crea un array para guardar los argumentos de la línea de comandos y pasa la referencia de ese array a `args` (`args = new String[n]` siendo `n` el nº de argumentos. Si `n = 0`, no hay argumentos, entonces `args = new String[0]`. El array está vacío, tiene longitud 0 pero `args` no es null aunque `args.length = 0`).

Ej:

```
public class PruebaMain {  
    public static void main(String[ ] args) {  
        if (args.length == 2) {
```



```
        int edad = Integer.parseInt(args[0]);
        String nombre = args[1];
        Persona p = new Persona(nombre, edad);
    } else
    } .....
}
```

Desde BlueJ podemos invocar al método `main()` de una clase y pasar un array de `String` como argumentos del método pasando un array de constantes: `{"2", "Elisa"}`. A través de los argumentos del `main()` se puede pasar a un programa datos tales como el nombre de un fichero, la capacidad de una lista, el nº de jugadores en un juego, ...

## 5.10 El tipo Enum

---

Java 1.5 introduce un nuevo tipo denominado tipo enumerado. La sintaxis completa de este nuevo tipo está muy cerca de la definición de una clase. Nosotros solo veremos una simplificación del tipo.

El siguiente ejemplo muestra una manera habitual de asociar nombres simbólicos a valores numéricos a través de constantes.

```
/**  
* Los objetos de esta clase mantienen un atributo nivel  
*/  
  
public class Control {  
  
    private static final int BAJO = 0;  
    private static final int MEDIO = 1;  
    private static final int ALTO = 2;  
    private int nivel;  
  
    /**  
* Inicializa nivel a BAJO  
*/  
    public Control() {  
        nivel = BAJO;  
    }  
  
    /**  
* devuelve el valor actual de nivel  
*/  
    public int getNivel() {  
        return nivel;  
    }  
  
    /**  
* establece el nivel actual  
*/  
    public void setNivel(int nivel) {  
        this.nivel = nivel;  
    }  
}
```

El mayor problema con este código es que no hay garantía de que en el método `setNivel()` el valor pasado sea un entero válido, coincida con uno de los tres elegidos para las constantes (a menos que se testee el valor del parámetro pasado).

El tipo enumerado proporciona una buena solución a este problema.

```
public class Control {  
  
    private Nivel nivel;  
  
    /*  
* Inicializa nivel a BAJO  
*/
```

```

    public Control() {
        nivel = Nivel.BAJO;
    }

    /**
     * devuelve el valor actual de nivel
     */
    public Nivel getNivel() {
        return nivel;
    }

    /**
     * establece el nivel actual
     */
    public void setNivel(Nivel nivel) {
        this.nivel = nivel;
    }
}

/**
 * Enumera el conjunto de valores
 * disponibles para un objeto Control
 */
public enum Nivel {
    BAJO, MEDIO, ALTO;
}

```

En su forma más simple un tipo enumerado permite crear un tipo dedicado para un conjunto de nombres, como BAJO, MEDIO, ALTO. Los valores de este tipo se refieren ahora como Nivel.BAJO, Nivel.MEDIO, Nivel.ALTO.

El tipo del atributo nivel es ahora Nivel y no un int. El método setNivel() está ahora protegido de recibir un parámetro de valor inadecuado pues el argumento actual debe coincidir ahora con uno de los tres valores posibles del tipo. De la misma manera un cliente que llame al método getNivel() no podrá tratar el valor devuelto como un entero ordinario.

Para crear, por tanto, un tipo enumerado:

- especificaremos la cláusula enum
- indicaremos el nombre del tipo
- enumeraremos la lista de valores del tipo (por convención se escriben en mayúsculas)

Desde BlueJ podremos crear un tipo enumerado Nueva Clase / Enum (New Class / Enum)

Algunas consideraciones acerca de los tipos enumerados:

- son clases
- no son enteros
- no tienen constructor público
- los valores del tipo son public, static, final
- heredan el método toString() – Nivel.MEDIO.toString() devuelve la cadena "MEDIO". Este método se puede redefinir dentro del tipo enumerado
- proporcionan un método estático valueOf() – complementa a toString(). Nivel.valueOf("MEDIO") devuelve Nivel.MEDIO
- cada enumerado define un método público estático values() que devuelve un array cuyos elementos contienen los valores del tipo

**Ej:**

```
Nivel[ ] niveles = Nivel.values();
for (int i = 0; i < niveles.length; i++)
    System.out.println(niveles[i].toString());
```

## Ejercicio 5.27

Define los siguientes tipos enumerados:

- Palo que define los palos que puede tomar una carta de la baraja española
- Estacion – define las estaciones del año
- Nota – define las notas posibles de un estudiante (notable, suficiente, ....)

### Solución

```
public enum Palo {
    OROS, COPAS, ESPADAS, BASTOS;
}

public enum Estaciones {
    PRIMAVERA, VERANO, OTOÑO, INVIERNO;
}

public enum Notas {
    INSUFICIENTE, SUFICIENTE, BIEN, NOTABLE, SOBRESALIENTE;
}
```

## Ejercicio 5.28

Crea una clase **Semáforo** con un atributo estado de tipo **Estado**. El tipo Estado es un enumerado con los colores posibles de un semáforo. Incluye un accesor para el atributo estado y un mutador `cambiarEstado()` que cambia el estado del semáforo. (La secuencia de cambio de un semáforo es: verde - amarillo – rojo – verde – amarillo ....). Incluye un método `toString()` que devuelva el valor actual del semáforo. Añade el método `String getColoresSemaforo()` que devuelve una cadena con la relación de todos los colores posibles de un semáforo.

### Solución

[Enlace](#) al proyecto



