

## Clases en Java



## Resumen

- Variables
- Clases y objetos – Aspectos generales
- Accesibilidad
- Atributos y métodos de instancia
- Constructor y recolector de basura
- *this*
- Sobrecarga
- Paso de parámetros
- Constantes y métodos finales
- Paquetes
- Clases de utilidad: Array, String, StringBuffer, Vector, Math



## Objetivos

- Conocer y comprender las características generales de las clases así como sus elementos principales
- Conocer y diferenciar entre atributos y métodos de instancia frente a los de clase
- Conocer y comprender el concepto de sobrecarga así como los tipos de paso de parámetros
- Conocer y aprender a utilizar distintas clases de utilidad de Java
- Diseñar e implementar pequeñas aplicaciones en Java utilizando todos los conceptos del tema



## Variables

- Nombre que contiene un valor que puede variar
- Tipos principales de variables:
  - Tipos primitivos: Están definidas por un valor único
  - Referencia: Contienen información más compleja: arrays, Strings, objetos de una clase definida por el usuario, ...
- Desde el punto de vista de su papel en el programa:
  - Miembro:
    - Se definen en una clase, fuera de los métodos
    - Pueden ser tipos primitivos o referencias
  - Locales:
    - Definidas dentro de un bloque entre llaves {}. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque
    - Pueden ser tipos primitivos o referencias



## Clases – Aspectos Generales

- Una clase es una agrupación de datos o atributos y de métodos que operan sobre ellos

```
[acceso] class NombreClase
{
    // Atributos
    // Constructor
    // Métodos
}
```

- La primera letra del nombre de la clase debe ir en mayúsculas
- Las clases se pueden agrupar en paquetes (*packages*), introduciendo una línea al comienzo del fichero (se verá más adelante)



## Clases – Aspectos Generales

```
[public] class NombreClase {
    // Definición de atributos
    // Definición de métodos
}
```

- En un fichero se pueden definir varias clases, pero sólo una de ellas puede estar declarada como pública:
  - El fichero debe llamarse como la clase pública → NombreClase.java
- Recomendaciones:
  - Escribir una sola clase por fichero (es lo habitual)



## Modificadores de Acceso

- Los modificadores de acceso se utilizan para controlar el acceso a clases, atributos y métodos
- Tipos de modificadores de acceso:
  - *public* (cualquier clase)
  - *private* (clase)
  - *protected* (clase, derivadas y paquete)
  - *package* (paquete)
- Clases:
  - *public* y *package*
- Variables y métodos:
  - *public*, *private*, *protected*, *package*



## Objetos – Aspectos Generales

- Un objeto (instancia) es un ejemplar concreto de una clase
- Visión:
  - Clase: Tipo de variable
  - Objeto: Variables concretas de un tipo determinado
- Una clase proporciona una definición para un tipo particular de objeto: define sus datos y la funcionalidad sobre dichos datos
- Un mismo programa puede manejar muchos objetos de una misma clase
- Estado: Definido por el valor de sus atributos



## Elementos de una clase - Atributos

- Los atributos definen la estructura y los datos de los objetos de una determinada clase
- Declaración de atributos:  
[acceso] [static] [final] Tipo NombreAtrib [= Valor];
- Donde:
  - Acceso: public, private, protected y package. El valor por defecto es package y puede omitirse
  - static: Define variables propias de la clase, no del objeto (más adelante)
  - final: Son constantes (más adelante)
  - Tipo:
    - Predefinidos
    - Objetos: Strings, Arrays, Objetos definidos por el usuario...
- Por defecto se inicializan a 0, 0.0, '\u0000', false o null



## Elementos de una clase - Atributos

```
class Cliente {
    String nombre;
    long DNI;
    long telefono;
}

class CuentaBancaria {
    long numero;
    Cliente titular;
    long saldo;
}

class Punto {
    public int x;
    public int y;
}
```

- Los atributos deben ocultarse → "private"



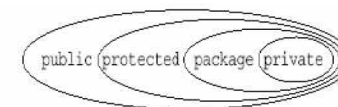
## Elementos de una clase – Atributos - Acceso

- Cada objeto que se crea de una clase tiene su propia copia de los atributos y de los métodos
- Para acceder al valor de un atributo se escribe:  
Objeto.atributo;
- Ejemplo:
  - Cada objeto de la clase Punto tiene sus propias coordenadas x e y
  - Si tenemos un objeto llamado p1 de la clase Punto:
 

```
...
p1.x=5;
p1.y=7;
System.out.println (p1.x + " , " + p1.y);
...
```



## Elementos de una clase – Atributos – Modificadores Acceso



	private	sin modificador	protected	public
misma clase	SI	SI	SI	SI
subclase del mismo paquete	NO	SI	SI	SI
otra clase del mismo paquete	NO	SI	NO	SI
subclase del distinto paquete	NO	NO	SI	SI
otra clase del distinto paquete	NO	NO	NO	SI



## Elementos de una clase - Métodos

- Los métodos definen las operaciones que se pueden realizar sobre la clase
- Declaración de métodos:  
[acceso] [static] [final] tRetorno nombreMetodo([args])  
{/\*Cuerpo del método\*/}
- Donde:
  - acceso: public, private, protected, package
  - static: Se puede acceder a los métodos sin necesidad de instanciar un objeto de la clase (más adelante)
  - final: Constantes, evita que un método sea sobrescrito (más adelante)
  - tRetorno: Tipo primitivo, referencia o void
  - La primera palabra del método debe ser un verbo
  - Lista de parámetros que recibe separados por comas



## Elementos de una clase - Métodos

- Devolución de valores:
  - En la declaración se debe especificar el tipo de valor devuelto por el método
  - Si el método no devuelve nada, el tipo de retorno será *void*
  - Si el método devuelve algo:
    - Se pueden devolver tipos primitivos u objetos, ya sean objetos predefinidos u objetos de clases creadas por el propio usuario
    - El cuerpo del método deberá contener una instrucción similar a `return valorDevuelto;`



## Elementos de una clase - Métodos - Ejemplo

```
class Punto {  
    public int x;  
    public int y;  
  
    public double calcularDistanciaCentro() {  
        double z;  
        z=Math.sqrt((x*x)+(y*y));  
        return z;  
    }  
}
```

Atributos

Método



## Elementos de una clase - Métodos

- Los métodos se aplican siempre a un objeto de la clase usando el operador punto (.) salvo los métodos declarados como *static*. Dicho objeto es su argumento implícito:

nombreObjeto.metodo([args]);

- Argumentos explícitos: Van entre paréntesis, a continuación del nombre del método. Los tipos primitivos se pasan por valor, para cambiarlos hay que encapsularlos dentro de un objeto y pasar el objeto

- Ejemplo:

...

```
double d = p1.calcularDistanciaCentro();  
System.out.println("distancia: " + d);
```



## Elementos de una clase - Métodos

- Los métodos pueden definir variables locales:
  - Visibilidad limitada al propio método
  - Variables locales no se inicializan por defecto

```
public double calcularDistanciaCentro() {  
    double z; ← Variable Local  
    z=Math.sqrt((x*x)+(y*y));  
    return z;  
}
```



## Ejercicio 1 – Clases en Java

- Realizar el código de las clases Cliente y CuentaBancaria que permita realizar lo siguiente:
  - Métodos para actualizar y devolver los valores de los atributos de las dos clases
  - Método que muestre por la salida estándar los datos de un determinado cliente
  - Métodos para ingresar y sacar una determinada cantidad de dinero de una cuenta bancaria, actualizando el saldo de la misma

```
class Cliente {  
    String nombre;  
    long DNI;  
    long telefono;  
    ...  
}
```

```
class CuentaBancaria {  
    long numero;  
    Cliente titular;  
    long saldo;  
    ...  
}
```



## Elementos de una clase – Métodos - Constructor

- Método que se llama automáticamente cada vez que se crea un objeto de una clase
- Características:
  - Reservar memoria e inicializar las variables de la clase
  - No tienen valor de retorno (ni siquiera *void*)
  - Tienen el mismo nombre que la clase
- Sobrecarga:
  - Una clase puede tener varios constructores, que se diferencian por el tipo y número de sus argumentos



## Elementos de una clase – Métodos - Constructor

- Constructor por defecto:
  - Constructor sin argumentos que inicializa:
    - Tipos primitivos a su valor por defecto
    - Strings y referencias a objetos a null
- Creación de objetos:

```
Punto p = new Punto(1,1);
```

  - Declaración: Punto p;
  - Instanciación: new Punto(1,1); crea el objeto
  - Inicialización: new Punto(1,1); llama al constructor que es el que inicializa el objeto



## Clases - Ejemplo

```
public class Punto {  
    public int x;  
    public int y;  
  
    public Punto(int a){ x=a; y=a;}  
    public Punto(int a, int b) { x=a; y=b;}  
  
    public double calcularDistancia() {  
        double z;  
        z=Math.sqrt((x*x)+(y*y));  
        return z;  
    }  
}
```



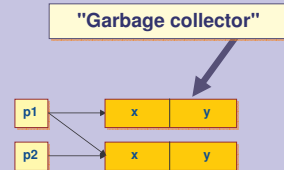
## Elementos de una clase – Constructor - Ejemplo

```
public CuentaBancaria(){}  
  
public CuentaBancaria(String n, Cliente t, long s){  
    numero = n;  
    titular = t;  
    saldo = s;  
}  
  
public CuentaBancaria(String n, Cliente t){  
    numero = n;  
    titular = t;  
}
```



## Objetos - Constructor / Destructor

```
Punto p1=new Punto(2,3);  
Punto p2=new Punto(5,7);  
p1=p2;
```



- "Garbage collector": Recolector de basura automático
- Explícitamente: objeto = null



## Ejercicio 2 – Clases en Java

- Realizar los constructores de la clase Cliente:
  - Constructor que reciba el nombre y el dni del cliente
  - Constructor que reciba el nombre, el dni y el teléfono del cliente

```
class Cliente {  
    String nombre;  
    long DNI;  
    long telefono;  
    ...  
}
```



## Variable this

- Definida implícitamente en el cuerpo de los métodos
- Dentro de un método o de un constructor, *this* hace referencia al objeto actual

```
class Vector2D{
    double x,y;
    ...
    double productoEscalar (Vector2D u) {
        return x*u.x + y * u.y;
        //return this.x*u.x + this.y * u.y;
    }
    double modulo(){
        return (double)Math.sqrt(productoEscalar(this));
    }
}
```

*this* ← v

```
//Main
Vector3D v = new Vector3D(2,-2,1);
v.modulo();
```



## Elementos de una clase – Métodos - this

- Uso de *this* con atributos:

```
public class Punto{
    public int x = 0;
    public int y = 0;
    public Punto(int a, int b){
        x = a;
        y = b;
    }
}

public class Punto{
    public int x = 0;
    public int y = 0;
    public Punto(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```



## Elementos de una clase – Métodos -this

- Un constructor puede llamar a otro constructor de su propia clase si:
  - El constructor al que se llama está definido
  - Se llama utilizando la palabra *this* en la primera sentencia



## Elementos de una clase – Métodos - this - Ejemplo

```
public Cliente(String n, long dni){
    nombre = n;
    DNI = dni;
}

public Cliente(String n, long dni, long tel){
    nombre = n;
    DNI = dni;
    telefono = tel;
}

public Cliente(String n, long dni, long tel){
    this(n,dni);
    telefono = tel;
}
```



### Ejercicio 3 - Clases en Java

- Completar el código siguiente:

```
public class Rectangulo {
    private int x, y;
    private int ancho, alto;
    public Rectangulo(int x, int y, int ancho, int alto){
        ... = x;
        ...
    }
    public Rectangulo( int ancho, int alto){ ... }
    public Rectangulo(){ ... }
}
```



### Sobrecarga de métodos

- Puede haber dos o más métodos que se llamen igual en la misma clase
- Se tienen que diferenciar en los parámetros (tipo o número)
- El tipo de retorno es insuficiente para diferenciar dos métodos



### Sobrecarga de métodos - Ejemplo

```
class Sobrecarga {
    public Sobrecarga(){}
    ① → void test() { System.out.println("vacío"); }
    ② → void test(int a) { System.out.println("a: " + a); }
    ③ → void test(int a, int b) {
        System.out.println("a y b: " + a + " " + b);
    }
    ④ → double test(double a) {
        System.out.println("double a: " + a);
        return a*a;
    }
}
```

Sobrecarga ob = new Sobrecarga();  
double res;

① → ob.test();  
② → ob.test(10);  
③ → ob.test(10, 20);  
④ → res = ob.test(5.1);



### Sobrecarga de métodos - Ambigüedad

- Se ejecuta el método que más se adecua

```
class A {
    void f (int n) { System.out.println ("Tipo int");}
    void f (float x) { System.out.println ("Tipo float");}
}
```

¿Qué ocurre?

```
...
A a = new A();
byte b = 3;
long l = 3;
double d = 3;
a.f(l);
a.f(b);
a.f(d);
```

Compiling 2 source files to D:\Estefania\08\_09\_P00\Ambigüedad\build\classes  
symbol : method f(double)  
location: class Ambigüedad.A  
a.f(d); // ERROR: necesita cast explícito  
1 error  
BUILD FAILED (total time: 0 seconds)

¿Cómo solucionarlo?

```
run:
Tipo float
Tipo int
Tipo float
```

a.f((float)d);



## Ejercicio 4 - Sobrecarga de métodos

- ¿Por qué no sería correcto el siguiente ejemplo?

```
public class MiClase{
...
    public void pintaString(String s){...}
    public void pintaEntero(int n){...}
    public void pintaDecimal(double d){...}
}
```

- ¿Cómo lo mejorarías?



## Paso de parámetros - Objetos

- Se pueden pasar objetos a un método como parámetros

```
class Caja{
    double alto; double ancho; double largo;
    Caja( double a, double b, double c ){
        alto = a;    ancho = b;    largo = c;
    }
    Caja(Caja c ) {
        alto = c.alto;
        ancho = c.ancho;
        largo = c.largo;
    }
}
// En el programa principal
...
Caja c1=new Caja(10,20,15);
Caja c2=new Caja(c1);
```



## Paso de variables – Por valor

- Las variables de la lista de parámetros de un método pasarán de diferente forma dependiendo del tipo de dato de origen

- Paso por valor:

```
class Test{
    public Test(){ }
    void metodo( int i, int j ){
        i=i*2;    // Principal
        j=j/2;    Test t=new Test();
                int a=15;
                int b=20;
    }
}
Antes: 15 20    System.out.println("Antes:" + a + " " + b);
Despues: 15 20 t.metodo(a,b);
                System.out.println("Despues:" + a + " " + b);
```



## Paso de variables – Por referencia

```
public class Test {
    int a,b;
    Test (){}
    Test (int i, int j){ a = i; b = j; }
    void metodo(Test o){ o.a -=2; o.b +=2;}
    void imprime() { System.out.println("A: " + a + " B: " + b); }
}
Test o1 = new Test(15,20);
Test o2 = new Test();
A: 15 B: 20    o1.imprime();
A: 0 B: 0      o2.imprime();
A: -2 B: 2     o1.metodo(o2);
                o2.imprime();
```



## Atributos y métodos de clase - *static*

- Los elementos (atributos y métodos) definidos como *static* son independientes de los objetos de la clase
- Atributos estáticos – variables de clase:
  - Un atributo *static* es una variable global de la clase
  - Un objeto de la clase no copia los atributos *static* → todas las instancias comparten la misma variable
  - Uso de atributos estáticos:  
nombreClase.nombreAtributoEstático



## Atributos y métodos de clase - *static* - Ejemplo

```
public class MiClase {
    String nombre;
    static int contador;
    public MiClase(String n){ nombre = n; contador++;}
    public void imprimeContador()
        {System.out.println("Contador: " + contador);}
}
...
MiClase o1=new MiClase("Primero");
MiClase o2=new MiClase("Segundo");
o2.imprimeContador();           // 2
MiClase.contador = 1000;
o2.imprimeContador();           //1000
```



## Atributos y métodos de clase - *static*

- Métodos estáticos:
  - Un método *static* es un método global de la clase
  - Un objeto no hace copia de los métodos *static*
  - Suelen utilizarse para acceder a atributos estáticos
  - No pueden hacer uso de la referencia *this*
  - Llamada a métodos estáticos:  
nombreClase.nombreMetodoEstatico()
  - Ejemplo:  
public static void actualizaContador(){ contador = 0; }  
// Principal:  
...  
MiClase.actualizaContador();  
o.imprimeContador();



## Atributos y métodos de clase - *static*

- Bloques estáticos:
    - Contienen un conjunto de instrucciones que se ejecutarán una sola vez cuando la clase se cargue por primera vez
- ```
// Principal
System.out.println("Antes");
MiClase2.llamada();
System.out.println("B = " + MiClase2.b);
public class MiClase2 {
    static int a = 3;
    static int b = 5;
    static {
        System.out.println("Cargando la clase...");
        b = a * 10;
    }
    static void llamada() { System.out.println("A = " + a);
}
}
```



## Constantes - *final*

- La palabra reservada *final* indica que su valor no puede cambiar
- Si se define como *final*:
  - Una clase → No puede tener clases hijas (seguridad y eficiencia del compilador)
  - Un método → No puede ser redefinido por una subclase
  - Una variable → Tiene que ser inicializada al declararse y su valor no puede cambiarse
- Suele combinarse con el modificador *static*
- El identificador de una variable *final* debe escribirse en mayúsculas



## Constantes - *final* - Ejemplo

```
public class MiClase {  
    final int NUEVO_ARCHIVO = 3;  
    void f(){ NUEVO_ARCHIVO = 27; /* Error */ }  
}
```



## Paquetes

- Un paquete es una agrupación de clases
- La API de Java se organiza en paquetes
- Creación de paquetes: `package nombrepaquete;`
- Las clases de un paquete deben estar en el mismo directorio
- El nombre de un paquete puede constar de varios nombres unidos por puntos (p.e. `java.awt.event`), que se corresponde con la jerarquía de directorios donde se guarda la/s clase/s
- Importar paquetes (clases o todo el paquete):
  - `import java.awt.*`
  - `import paquete.usuario.aplicacion.class`



## Ejercicio 5 - Modificadores de Acceso

```
• ¿Cuál es la salida del siguiente programa?  
... /* Principal */  
Test o = new Test();  
o.a = 10;  
o.b = 20;  
o.c = 100;  
System.out.println(o.a + " " + o.b + " " + o.getc());  
  
class Test {  
    int a;  
    public int b;  
    private int c;  
    void setc(int i) { c = i;}  
    int getc() { return c;}  
}
```



## Ejercicio 6 - POO y Programación Estructurada

- Completar la siguiente tabla:

| POO | Prog. Estructurada          |
|-----|-----------------------------|
|     | Tipo Registro               |
|     | Variable Registro           |
|     | Subprograma                 |
|     | Llamada                     |
|     | Campo de registro           |
|     | Valor del campo de registro |



## POO y TADs

|                | Programación con TAD | POO       |
|----------------|----------------------|-----------|
| Definición     | TAD                  | Clase     |
| Estado         | Datos                | Atributos |
| Comportamiento | Operaciones          | Métodos   |
| Ejemplares     | Variables            | Objetos   |



## Clases de Utilidad

- Arrays
- String y StringBuffer
- Vector
- Math



## Clases útiles – Array – Aspectos generales

- Se pueden crear *arrays* de objetos de cualquier tipo
- Creación de *arrays*:
  - operador new + tipo + número de elementos máximo
- Acceso a los elementos:
  - nombreArray[posicion] donde  $0 \leq \text{posicion} < \text{length}$
- Número de elementos máximo:
  - nombre.length
- Inicialización a valores por defecto: 0 para numéricos, carácter nulo para char, false para boolean, y null para Strings y referencias
- Se pasan a los métodos por referencia



## Clases útiles – Arrays – Definición y creación

- Definición de un array: `double[] x;`
- Creación del array: `x = new double[100];`
- Definición y creación del array:  
`double[] x = new double[100];`
- Inicialización:
  - Con valores entre llaves {...} separados por comas
  - Con varias llamadas a new dentro de unas llaves {...}
- Si se igualan dos referencias a un array **no se copia el array**: es un único objeto array con dos nombres



## Clases útiles – Arrays - Ejemplos

```
int[] miarray1 = new int[10];

int[] miarray2 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

int[] miarray1 = new int[10];
for (int i = 0; i < 10; i++)    miarray1[i] = i;
System.out.println("Elementos: " + miarray1.length);

String[] dias = {"lunes", "martes", "miercoles", "jueves",
                "viernes", "sabado", "domingo"};
for (int i = 0; i < dias.length; i++)
    System.out.println("Posicion: " + i + " contenido: " + dias[i]);
```



## Clases útiles – Arrays - Ejemplos

```
public class MiClase {
    private int identificador;
    private static int contador = 1;
    public MiClase(){
        identificador = contador;
        contador++;
    }
    public int getId(){ return identificador;}
}

MiClase[] arrayObjetos = new MiClase[5];
for( int i = 0 ; i < 5; i++) {
    arrayObjetos[i] = new MiClase();
    System.out.println(i + " ID: " + arrayObjetos[i].getId());
}
```



## Clases útiles – Arrays - Ejemplos

```
public class Matriz {
    int[][] datos;
    int dim;
    public Matriz(int num)
    {
        dim = num;
        Random rnd = new Random();
        datos = new int[num][num];
        for (int i = 0; i < num; i++){
            for (int j = 0; j < num; j++){
                datos[i][j] = rnd.nextInt();
            }
        }
    }
}
```



## Ejercicio 7 – Clases en Java

- Hacer una clase que gestione el comportamiento de una pila de enteros en la que se pueden:
  - Añadir nuevos elementos
  - Sacar de la pila elementos hasta el último ingresado
- La pila será un array de enteros de 10 posiciones
- Desde el programa principal se crearán objetos de tipo Pila y se manejarán sus métodos



## Clases útiles – *String* y *StringBuffer* – Aspectos generales

- Clase *String*:
  - Cadena de caracteres no modificable
  - Longitud fija
- Clase *StringBuffer*:
  - Cadena de caracteres que se puede manipular después de crearse
  - Longitud variable



## Clases útiles – *String* – Constructores

- String vacío:  
`String s = new String();`
- String inicializado con caracteres  
`char chars[] = {'a','b','c'};`  
`String s = new String(chars);`  
`String s1 = "Hello";`  
`String s2 = new String(s1);`
- Usar subrango de vector de caracteres  
`char chars[] = {'a','b','c','d','e','f'};`  
`String s = new String(chars,2,3);`
- Forma abreviada:  
`String s = "abc";`



## Clases útiles – *String* – Concatenación (+)

```
String s = "He is " + age + " years old.";
```

- Es equivalente a:  
`String s = new StringBuffer("He is ")`  
`.append(age)`  
`.append(" years old.")`  
`.toString();`



### Clases útiles – *String* – Extracción de caracteres

- Extraer un carácter de una determinada posición - *charAt*  
"abc".charAt(1) ---- 'b'
- Extraer más de un carácter - *getChars*

```
class getCharsDemo {
    public static void main(String args[]) {
        String s = "Esto es una demo del método getChars.";
        int start = 11;
        int end = 15;
        char buf[] = new char[end - start];
        s.getChars(start,end,buf,0);
        System.out.println(buf);
    }
}
```
- *toCharArray* - Devuelve vector de *char* para el *String*



### Clases útiles – *String* – Comparación de caracteres

- *equals* → Devuelve *true* si el parámetro es igual que el objeto sobre el que se llama
- *equalsIgnoreCase* → Ignora mayúsculas y minúsculas



### Clases útiles – *String* – Comparación de caracteres - Ejemplo

```
class equalsDemo {
    public static void main(String args[]) {
        String s1 = "Hello";
        String s2 = "Hello";
        String s3 = "Good-bye";
        String s4 = "HELLO";
        S.O.P(s1 + " equals " + s2 + " -> " + s1.equals(s2));
        S.O.P(s1 + " equals " + s3 + " -> " + s1.equals(s3));
        S.O.P(s1 + " equals " + s4 + " -> " + s1.equals(s4));
        S.O.P(s1 + " equalsIgnoreCase " + s4 + " -> " +
            s1.equalsIgnoreCase(s4));
    }
}
```



### Clases útiles – *String* – Comparación de caracteres – Equals & ==

- Si comparamos dos strings con:
  - El operador == → Devuelve *true* si las dos variables se refieren al mismo objeto
  - El operador *equals* → Devuelve *true* si los strings son iguales

```
class EqualsNotEqualTo {
    public static void main(String args[]) {
        String s1 = "Hello";
        String s2 = new String(s1);
        S.O.P(s1 + " equals " + s2 + " -> " + s1.equals(s2));
        S.O.P(s1 + " == " + s2 + " -> " + (s1 == s2));
    }
}
```



### Clases útiles - *String* - Comparación de caracteres - Otros métodos

- *startsWith*: Comprueba si un *String* es el inicio de otro  
"Estefania".startsWith("Este") → true
- *endsWith*: Indica si un *String* es el final de otro  
"Estefania".endsWith("nia") → true
- *compareTo(arg1)*: Compara dos *Strings* y devuelve:
  - < 0 → si String < arg1
  - = 0 → si String = arg1
  - > 0 → si String > arg1



### Ejercicio 8 - Clases útiles - *String* - Comparación de caracteres - Otros métodos

```
class QueHago {
    static String arr[] = {"un ", "ejemplo", "Esto ", "es "};
    public static void main(String args[]) {
        for (int j = 0; j < arr.length; j++) {
            for (int i = j + 1; i < arr.length; i++) {
                if (arr[i].compareTo(arr[j]) < 0) {
                    String t = arr[j];
                    arr[j] = arr[i];
                    arr[i] = t;
                }
            }
            S.O.P.(arr[j]);
        }
    }
}
```



### Clases útiles - *String* - Búsqueda de caracteres

- Índice de la primera / última ocurrencia del carácter 'c'  
int indexOf(int c)  
int lastIndexOf(int c)
- Índice del primer carácter de la primera / última ocurrencia del substring str  
int indexOf(String str)  
int lastIndexOf(String str)
- Índice del primer carácter de la 1ª ocurrencia después (o antes) de pos del carácter 'c'  
int indexOf(int c, int pos)  
int lastIndexOf(int c, int pos)
- Índice del primer carácter de la 1ª ocurrencia después (o antes) de pos del substring str  
int indexOf(String str, int pos)  
int lastIndexOf(String str, int pos)



### Ejercicio 9 - Clases útiles - *String* - Búsqueda de caracteres

```
class indexOfDemo {
    public static void main(String args[]) {
        String s = "Esto es un ejemplo. ";
        S.O.P.(s);
        S.O.P. (" indexOf(s) = " + s.indexOf('s'));
        S.O.P. (" lastIndexOf(s) = " + s.lastIndexOf('s'));
        S.O.P. (" indexOf(es) = " + s.indexOf("es"));
        S.O.P. (" lastIndexOf(es) = " + s.lastIndexOf("es"));
        S.O.P. (" indexOf(s,2) = " + s.indexOf('s',2));
        S.O.P. (" lastIndexOf(s,10) = " + s.lastIndexOf('s',10));
        S.O.P. (" indexOf(es,2) = " + s.indexOf("es",2));
        S.O.P. (" lastIndexOf(es,10) = " + s.lastIndexOf("es",10));
    }
}
```



### Clases útiles – *String* – Otras operaciones

- `substring` – Saca parte de una cadena de caracteres  
`"Hello World".substring(6) --- "World"`  
`"Hello World".substring(3,8) --- "lo Wo"`
- `concat` – Concatena dos cadenas  
`"Hello".concat(" World") --- "Hello World"`
- `replace` – Reemplaza un caracter por otro  
`"Hello".replace('l','w') --- "Hewwo"`
- `toLowerCase` / `toUpperCase` – Convierten a mayúscula / minúscula  
`"Hello".toLowerCase() --- "hello"`  
`"Hello".toUpperCase() --- "HELLO"`
- `trim` – Elimina los espacios del principio y del final  
`" Hello World ".trim() --- "Hello World"`
- `valueOf` – Convierte cualquier tipo a un string  
`StringSystem.out.println(String.valueOf(Math.PI));`



### Clases útiles – *StringBuffer* – Constructores y Métodos

- Constructores:
  - Sin parámetros → Reserva espacio para 16 caracteres
  - Con un argumento entero → Tamaño inicial del buffer
  - Con un argumento de cadena de caracteres → Cadena de caracteres + 16 caracteres adicionales
- Métodos:
  - `length` → Devuelve la longitud actual de la cadena  
`StringBuffer sb = new StringBuffer("Hello");`  
`sb.length();`
  - `capacity` → Devuelve la capacidad reservada  
`sb.capacity();`
  - `setLength` → Especifica la longitud de la cadena



### Ejercicio 10 – *StringBuffer* – Constructores y métodos

- ¿Cuál sería la salida del siguiente código?  

```
StringBuffer sb = new StringBuffer("Hello");
System.out.println("Longitud: " + sb.length());
System.out.println("Capacidad: " + sb.capacity());
sb.setLength(3);
System.out.println("Cadena: " + sb);
```



### Ejercicio 10 – *StringBuffer* – Constructores y métodos Solución

- ¿Cuál sería la salida del siguiente código?  

```
StringBuffer sb = new StringBuffer("Hello");
System.out.println("Longitud: " + sb.length());
System.out.println("Capacidad: " + sb.capacity());
sb.setLength(3);
System.out.println("Cadena: " + sb);
```

```
Longitud: 5
Capacidad: 21
Cadena: Hel
```



### Clases útiles – *StringBuffer* – Métodos

- `charAt` y `setCharAt` – Extraen o modifican un carácter específico

```
class setCharAtDemo {  
    public static void main(String args[]) {  
        StringBuffer sb = new StringBuffer("Hello");  
        System.out.println("buffer antes = " + sb);  
        System.out.println("charAt(1) antes = " + sb.charAt(1));  
        sb.setCharAt(1,'i');  
        sb.setLength(2);  
        System.out.println("buffer después = " + sb);  
        System.out.println("charAt(1) después = " +  
        sb.charAt(1));  
    }  
}
```

buffer antes = Hello  
charAt(1) antes = e  
buffer después = Hi  
charAt(1) después = i



### Clases útiles – *StringBuffer* – Métodos

- `append` – Concatena a un *StringBuffer*
- `insert` – Inserta un cadena de caracteres en una determinada posición



### Ejercicio 11 – *StringBuffer* – Métodos

- ¿Cuál sería la salida del siguiente código?

```
String s;  
int a = 42;  
StringBuffer sb = new StringBuffer(40);  
s = sb.append("a =  
").append(a).append("!").toString();  
System.out.println(s);  
StringBuffer sb2 = new StringBuffer("hello world!");  
sb.insert(6, "there ");  
System.out.println(sb2);
```



### Clases útiles – *Vector* – Aspectos generales

- Representa *array* variable de objetos
- Es una alternativa al uso de *arrays*
- Elementos principales:
  - Número de elementos del vector
  - Capacidad: Capacidad actual que puede variar con el tiempo (creciente o decreciente)
  - Incremento: Número de elementos que creará la capacidad (fijo)
- Constructores:
  - `Vector(int capacidadInicial, int incremento)`
  - `Vector(int capacidadInicial) → Incremento = 0`
  - `Vector() → Capacidad inicial = 10 e incremento = 0`
- Paquete: `java.util.Vector`;



## Clases útiles – *Vector* – Métodos

- Métodos:
  - size(): Número de elementos almacenados en el vector
  - capacity(): Capacidad actual
  - capacityIncrement: Incremento de capacidad. Si es 0, se doblará la capacidad
  - add(Object o): Añade un objeto al final del vector
  - add(int pos, Object o): Añade un objeto al vector en una determinada posición, desplazando el resto de los elementos
  - firstElement() / lastElement(): Devuelve el primer / último elemento del vector
  - get(int posicion): Devuelve el elemento que esté en una determinada posición



## Clases útiles – *Vector* – Métodos

- Métodos:
  - contains(Object o): Devuelve *true* si el vector contiene un determinado objeto
  - elements(): Devuelve un objeto de tipo *Enumeration* con todos los elementos del vector
  - indexOf(Object o): Devuelve el índice de un determinado objeto en el vector. Si no lo encuentra devuelve -1
  - setElementAt(Object o, int pos): Inserta un objeto en una determinada posición del vector
  - remove(int pos): Borra un elemento de una determinada posición, devolviendo el elemento borrado



## Clases útiles – *Vector* – Ejemplo

```
String s1 = "elemento1", s2 = "elemento2", s3 = "elemento3";
String[] array = {s1, s2, s3};
Vector v = new Vector();
v.addElement(s1); v.addElement(s2); v.addElement(s3);
System.out.println("Las componentes del array son: ");
for (int i=0; i<3; i++) { System.out.println(array[i]); }

System.out.println("Las componentes del vector son: ");
// Enumeration es una interfaz definida en java.util.Enumeration
for (Enumeration e = v.elements(); e.hasMoreElements() ; ) {
    System.out.println(e.nextElement());
}
```



## Ejercicio 12 – *Vector*

- ¿Cuál sería la salida del siguiente código?  
String s1 = "elemento1", s2 = "elemento2", s3 = "elemento3";  
Vector v = new Vector();  
v.addElement(s1);  
v.addElement(s2);  
v.addElement(s3);  
System.out.println("Tam: " + v.size() + " Cap: " + v.capacity());  
System.out.println("Ultimo elemento: " + v.lastElement());  
v.add(0,"Primero");  
v.add("Ultimo");  
for (Enumeration e = v.elements(); e.hasMoreElements() ; ) {  
 System.out.println(e.nextElement());  
}



### Ejercicio 13 – Vector

- Realizar las clases relacionadas con el siguiente programa:

```
import java.util.*;
public class Ejemplo {
    public static void main(String args[]) {
        ClaseAlumnos c1=new ClaseAlumnos("Tercero"),
            c2=new ClaseAlumnos("Segundo");
        Alumno a1=new Alumno("Juan", 21), a2=new Alumno("Ana", 22),
            a3=new Alumno("Luis", 23), a4=new Alumno("Sonia", 23);
        c1.maticular(a1); c1.maticular(a2);
        c2.maticular(a3); c2.maticular(a4);
        ...
    }
}
```



### Ejercicio 13 – Vector

```
...
System.out.println("Numero de clases de alumnos: "
    + ClaseAlumnos.numeroDeClasesCreadas());
System.out.println(c1.descripcion());
System.out.println(c2.descripcion());
}}
Salida:
```

```
Numero de clases de alumnos: 2
Clase: Tercero (alumnos: 2)
  Alumno: [Juan (21 años, clase Tercero)]
  Alumno: [Ana (22 años, clase Tercero)]
Clase: Segundo (alumnos: 2)
  Alumno: [Luis (23 años, clase Segundo)]
  Alumno: [Sonia (23 años, clase Segundo)]
```



### Clases útiles – Math – Aspectos generales

- La clase *Math* (*java.lang.Math*) contiene métodos que realizan operaciones numéricas básicas tales como las funciones trigonométricas, logarítmicas, exponenciales...
- Constantes:
  - public static final double E // Número de Euler e
  - public static final double PI // Número Pi
- Funciones:
  - public static double sqrt (double a) // Raíz cuadrada
  - public static double cbrt (double a) // Raíz cúbica
  - public static double pow (double a, double b) // Devuelve a<sup>b</sup>



### Clases útiles – Math – Funciones

- public static double abs (double a) (\*)
  - public static double max (double a, double b) (\*)
  - public static double min (double a, double b) (\*)
  - public static double floor (double a)
  - public static int round (double a)
  - public static double random () // 1.0 > n >= 0.0
- (\*) Admiten int, long y float
- Funciones logarítmicas:
    - public static double exp (double a)
    - public static double log (double a)
    - public static double log10 (double a)



## Clases útiles – *Math* – Funciones

- Funciones trigonométricas

- `public static double sin (double a)`
- `public static double cos (double a)`
- `public static double tan (double a)`
- `public static double asin (double a)`
- `public static double acos (double a)`
- `public static double atan( double a)`
- `public static double toRadians (double angdeg)`
- `public static double toDegrees (double angrad)`

