



CAMPUS
DE EXCELENCIA
INTERNACIONAL

POLITÉCNICA

"Ingeniamos el futuro"



Universidad
Politécnica
de Madrid

**ETSI SISTEMAS
INFORMÁTICOS**

Fundamentos de Programación

Tema 4: Clases y Objetos

OBJETIVOS

- Introducir el paradigma de la programación orientada a objetos mediante el lenguaje Java, a nivel únicamente de **abstracción y encapsulamiento**.
- Conocer los conceptos de clase y objeto, y ser capaz de definir clases propias y usar objetos creados a partir de esas clases.
- Sentar las bases para que en la asignatura de *Estructuras de Datos* se puedan definir abstracciones de datos.
- Sentar las bases para que en la asignatura de *Programación Orientada a Objetos* se pueda completar dicho paradigma.
- Tomar contacto con la biblioteca Java para aprender a utilizar algunas clases importantes.

ÍNDICE

1. [Introducción a la POO.](#)
2. [Elementos de la POO.](#)
3. [Vista pública de la clase.](#)
4. [Vista privada de la clase](#)
5. [Sobrecarga de métodos.](#)
6. [Constructores](#)
7. [Instanciación de objetos.](#)
8. [Paso de mensajes.](#)
9. [La referencia this](#)
10. [Tipos enumerados \(Enum\)](#)
11. [El API de la biblioteca Java](#)
12. [Las clases de recubrimiento Java](#)

1. Introducción a la POO.

- La Programación Orientada a Objetos (POO) es un **paradigma de programación**, es decir, un estilo de programación con una serie de principios y pautas.
- Otros paradigmas de programación son: programación imperativa (lo que conocemos hasta ahora), programación funcional, programación lógica.
- Destaca la POO por su **capacidad para crear aplicaciones más robustas, flexibles y fáciles de mantener**.
- Cuando se quiere diseñar un programa en POO, en lugar de pensar en términos algorítmicos, como se hace en la programación imperativa, se piensa en las entidades (objetos) que existen en esa aplicación: Alumno, Asignatura, GrupoAlumnos, Profesor, etc.
- Cada objeto queda definido en base a una serie de **datos** y una serie de **operaciones** que definen su comportamiento.
- Una aplicación en POO se compone de una serie de objetos que se van creando, que colaboran entre sí y que van desapareciendo.

1. Introducción a la POO.

Principios de la POO:

- **Abstracción:** para usar un objeto sólo debo conocer su interfaz, no los detalles de cómo está implementado (facilidad de uso)
- **Encapsulación (o encapsulamiento):** los detalles de cómo está implementado el objeto quedan ocultos dentro del mismo. Así, si hay un error con el objeto, seguro que está en el código del objeto (facilidad para depurar errores).
- **Herencia:** queda fuera del ámbito de esta asignatura.
- **Polimorfismo:** queda fuera del ámbito de esta asignatura.

2. Elementos de la POO.

- **Clase:** descripción de los datos y operaciones que describen el comportamiento de un cierto conjunto de elementos homogéneos.

- Ej. Clase *Avión*:
 - Datos: matrícula, marca, modelo, asientos y combustible.
 - Operaciones: llenarDepósito, vaciarDepósito, gastarCombustible, etc.
- Ej. Clase *Intervalo*:
 - Datos: extremo inferior y superior.
 - Operaciones: longitud, desplazar, incluye, etc.

2. Elementos de la POO.

- **Objeto:** ejemplar concreto (instancia) de una clase, que responde al comportamiento definido por las operaciones de la clase a la que pertenece, adecuándose al estado de sus datos particulares.
- Ej. Objetos de la Clase *Avión*:
 - Variables: *boeing747*, *airbusA320*.
- Ej. Objetos de la clase Intervalo:
 - Constantes: (5, 50), (-10, 10).
 - Variables: *horarioLunes*, *temperaturasDia*.

2. Elementos de la POO.

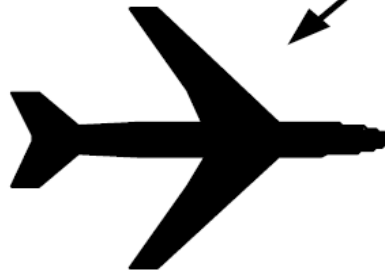


Avión del mundo real



| Clase Avión |
|---|
| String matricula String marca String modelo int numAsientos int litrosCombustible |
| llenarDeposito() vaciarDeposito() gastarCombustible(int numLitros) |

“Plantilla” que representa a
nuestro concepto de mundo real.



matrícula: 343LKD3L
marca: Boeing
modelo: 747
numAsientos: 392
litrosCombustible: 0

Objeto o instancia concreta.
Es una pieza software

2. Elementos de la POO.

- **Atributo:** cada uno de los datos de una clase, y por tanto, presente en todos los objetos de esa clase.

Ej. Atributos de la clase *Intervalo*:

- Extremo inferior y extremo superior

- **Estado:** conjunto de los valores de los atributos que tiene un objeto en un instante dado.

Ej. Estado del objeto *temperaturasDia*:

- 12 en extremo inferior, 21 en extremo superior.

- **Método:** definición de una operación de una clase. Actúan sobre el estado del objeto (atributos del objeto).

Ej. Métodos de la clase *Intervalo*:

- *longitud*: devuelve la diferencia entre extremo superior e inferior
- *desplazar(valor)*: el estado del intervalo cambia de manera que ambos extremos se desplazan el valor recibido como parámetro.
- *incluye(valor)*: devuelve *true* si el valor recibido se encuentra dentro del intervalo.

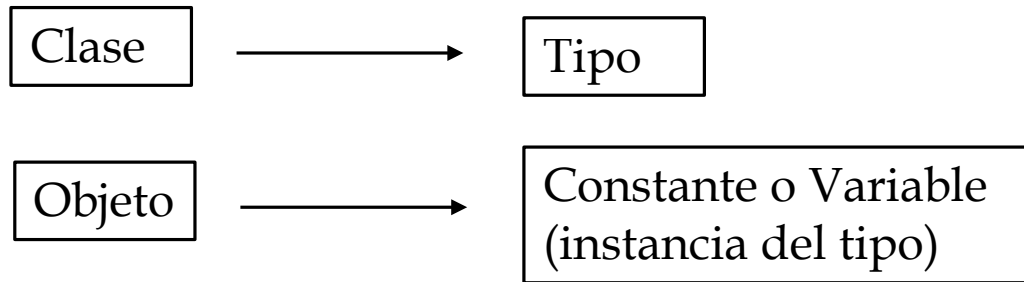
2. Elementos de la POO.

- **Mensaje:** invocación de una operación (método) sobre un objeto

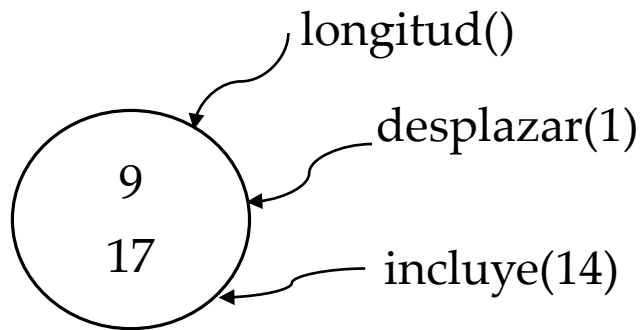
Ej. Mensajes al objeto *temperaturasDia* de la clase *Intervalo*:

- *temperaturasDia.longitud()*: devuelve 9. No cambia el estado del objeto.
- *temperaturasDia.desplazar(3)*: los extremos del intervalo pasan a ser (15, 24). Por tanto, se modifica el estado del objeto.
- *temperaturasDia.incluye(20)*: devolvería true. No cambia el estado del objeto.

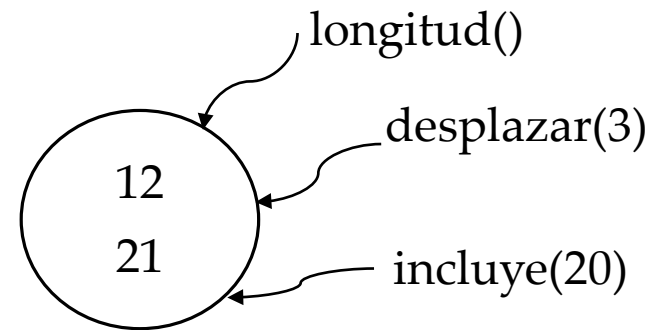
2. Elementos de la POO.



horarioLunes



temperaturasDia



2. Elementos de la POO.

Ejemplo: clase Intervalo

```
class Intervalo {  
    // ATRIBUTOS: extremo inferior y extremo superior  
    private double minimo;  
    private double maximo;  
  
    // CONSTRUCTORES: inicialización del objeto al crearlo  
    public Intervalo(double min, double max) {  
        if (min <= max) {  
            minimo = min;  
            maximo = max;  
        } else {  
            System.out.println("El mínimo no puede ser superior al máximo");  
            minimo = 0;  
            máximo = 0;  
        }  
    }  
  
    public Intervalo() {  
        minimo = 0;  
        maximo = 0;  
    }  
}
```

2. Elementos de la POO.

```
// MÉTODOS (operaciones)
public double getMinimo() {
    return minimo;
}
public double getMaximo() {
    return maximo;
}
public void setMinimo(double min) {
    if (min <= maximo) {
        minimo = min;
    } else {
        System.out.println("El mínimo no puede ser superior al máximo");
    }
}
public void setMaximo(double max) {
    if (max >= minimo) {
        maximo = max;
    } else {
        System.out.println("El máximo no puede ser inferior al mínimo");
    }
}
```

2. Elementos de la POO

```
// MÉTODOS (operaciones)

public double longitud() {
    return maximo - minimo;
}

public double puntoMedio() {
    return (maximo + minimo) / 2;
}

public void desplazar(double cantidad) {
    minimo += cantidad;
    maximo += cantidad;
}

public String toString() {
    return "[" + minimo + ", " + maximo + "]";
}
```

2. Elementos de la POO

```
// MÉTODOS (operaciones)

public void escalar(double escala) {
    double nuevaLongitud = longitud() * escala;
    double puntoMedio = puntoMedio();
    minimo = puntoMedio - nuevaLongitud / 2;
    maximo = puntoMedio + nuevaLongitud / 2;
}

public boolean incluye(double punto) {
    return minimo <= punto && punto <= maximo;
}

} // Fin de la clase Intervalo
```

2. Elementos de la POO.

Definición de un método *main* que usa la clase Intervalo

```
class PruebasIntervalo {
    public static void main(String[] args) {
        // Crear un objeto de la clase Intervalo
        Intervalo intervalo = new Intervalo(3,8);
        // Enviar mensajes sobre el objeto creado:
        System.out.println("Intervalo: " + intervalo.toString());
        System.out.println("Longitud" + ": " + intervalo.longitud());
        System.out.println("Punto medio: " + intervalo.puntoMedio());
        intervalo.desplazar(3);
        System.out.println("Al desplazarlo 3 uds: " + intervalo.toString());
        intervalo.escalar(3);
        System.out.println("Al triplicarlo: " + intervalo.toString());
        if (intervalo.incluye(19)){
            System.out.println("El 19 está incluido");
        } else {
            System.out.println("El 19 no está incluido");
        }
    }
}
```


2. Elementos de la POO

Resultado de la ejecución:

Intervalo: [3.0, 8.0]

Longitud: 5.0

Punto medio: 5.5

Al desplazarlo 3 unidades: [6.0, 11.0]

Al triplicarlo: [1.0, 16.0]

El 19 no está incluido

3. Vista pública de la clase.

Se refiere a lo conocido en cualquier punto de la aplicación:

- La cabecera de la declaración de la clase:

```
class <NombreClase> {  
}
```

Regla de estilo: el nombre de la clase debe comenzar con mayúscula.

- La cabecera de los métodos declarados como **public**:

```
public <tipo1> <nombreMétodo>([<tipo2> <parametro>, ...])
```

- tipo₁ puede ser un tipo primitivo, una clase, o un *array* (ver próximo tema). Si el método no devuelve nada, se indica con *void*.
- tipo₂ puede ser igual que tipo₁, excepto *void*.

Regla de estilo: el nombre del método debe comenzar con minúscula.

3. Vista pública de la clase.

La vista pública de una clase representa el interfaz de la clase, la abstracción. La vista pública de la clase *Intervalo* sería:

```
class Intervalo {  
    public Intervalo(double min, double max)  
    public Intervalo()  
    public double getMinimo()  
    public double getMaximo()  
    public void setMinimo(double min)  
    public void setMaximo(double max)  
    public double longitud()  
    public double puntoMedio()  
    public void desplazar(double desplazamiento)  
    public String toString()  
    public void escalar (double escala)  
    public boolean incluye(double punto)  
}
```

4. Vista privada de la clase.

Se refiere a lo que es conocido únicamente dentro de la clase. Por tanto, no es visible en el código de otra clase.

La vista privada se asocia con el principio de encapsulación, es decir, los detalles de la clase quedarán definidos dentro de la clase, pero **ocultos** fuera de la misma.

Definición de los atributos:

Los atributos siempre deben ser privados, para lo cual se utiliza el modificador **private**.


```
Ej.: class Intervalo {  
    private double minimo;  
    private double maximo;  
    ...  
}
```

Se pueden declarar en cualquier punto de la clase.

4. Vista privada de la clase.

Los atributos siempre deben ser privados:

Si fueran públicos, podría accederse desde fuera de la clase,

```
Ej.:class Aplicacion {  
    public static void main (String[] args) {  
        Intervalo horarioLunes = new Intervalo(3,8);  
         horarioLunes.minimo = 3000;  
    }  
}
```

y no se tendrían las ventajas de la abstracción y la encapsulación:

- **Código propenso a errores**, ya que se podrían corromper los objetos. Por ejemplo podríamos poner un mínimo superior al máximo (3000, 8), lo que podría suponer problemas al ejecutar posteriormente otros métodos de la clase *Intervalo*.

Y lo que es peor, el error en el objeto se produce fuera de la clase *Intervalo*, con lo que puede ser muy complicado de depurar.

4. Vista privada de la clase.

Los atributos siempre deben ser privados:

Al ponerlos privados se puede permitir el acceso externo a los datos, pero a través de métodos públicos **getter** y **setter** (*getMinimo*, *getMaximo*, *setMinimo*, *setMaximo*). Estos métodos cuidan de la coherencia de los datos:

```
public void setMinimo(double min) {
    if (min <= maximo) {
        minimo = min;
    } else {
        System.out.println("El mínimo debe ser inferior al
                               máximo");
    }
}

public double getMinimo() {
    return minimo;
}
```

4. Vista privada de la clase.

Definición de los atributos:

En el caso de que los atributos sean constantes, puede postergarse su inicialización al cuerpo de los constructores. Una vez inicializados, ya no es posible la asignación de nuevos valores.

```
Ej.: class Avión {  
    private double combustible;  
    private final String MODELO;  
  
    public Avión(String modelo) {  
        combustible = 0;  
        MODELO = modelo;  
    }  
    ...  
}
```

4. Vista privada de la clase.

Definición de los métodos:

- Nos referimos aquí a los **métodos de instancia**, que son aquellos métodos que se ejecutan sobre una instancia (un objeto).
- Los métodos de instancia acceden al estado del objeto (atributos) para consultarlo y/o para modificarlo.
- Al declarar los métodos de instancia, no se utiliza el modificador *static* de Java.

```
Intervalo horarioLunes = new Intervalo(9, 15);  
horarioLunes.desplazar(1);  
if (horarioLunes.incluye(7) {
```

- Por otra parte, los **métodos de clase** son los que llevan el modificador *static*. Estos métodos se ejecutan sin crear ninguna instancia, y por tanto, no tienen acceso al estado de ningún objeto (atributos)

4. Vista privada de la clase.

Definición de los métodos:

La definición de los métodos (sentencias que los implementan) pertenece también a la vista privada de la clase.

Dentro de los métodos se tiene acceso a:

- Los atributos, es decir, estado del objeto que recibe el mensaje (sólo en los métodos de instancia)
- Los parámetros del método.
- Declaraciones locales del método.

```
public void desplazar(double cantidad) {  
    minimo += cantidad;  
    maximo += cantidad;  
}
```

4. Vista privada de la clase.

Definición de los métodos:

Desde un método no sólo se pueden manejar los atributos del objeto que recibe el mensaje, sino de cualquier otro objeto que sea de la misma clase. Para ello, se utiliza la notación punto:

`<direccion0>.<atributo>`

Por ejemplo, en la clase *Intervalo* se desea definir la operación *iguales*, que comprueba si un intervalo recibido como parámetro es igual que el intervalo que recibe el mensaje

```
Ej.: public boolean iguales(Intervalo intervalo) {  
    return minimo == intervalo.minimo  
        && maximo == intervalo.maximo;  
}
```

4. Vista privada de la clase.

Métodos declarados como privados:

En ocasiones, se considera conveniente en una clase definir métodos que no pertenecen al interfaz de la clase, pero que clarifican o facilitan la implementación de otros métodos del interfaz. Para mantener el principio de encapsulación, estos métodos auxiliares se definirán con ámbito privado.

Por ejemplo, se desea definir el método *desplazado* en el interfaz de la clase *Intervalo*. Este método devuelve un nuevo objeto de la clase *Intervalo*, sin modificar el estado del objeto que recibe el mensaje. Para implementarlo, se considera oportuno utilizar un método *copia*, que no pertenece al interfaz:

```
Ej.: private Intervalo copia() {  
    return new Intervalo(minimo, maximo);  
}  
  
public Intervalo desplazado(double cantidad) {  
    Intervalo intervalo = this.copia();  
    intervalo.desplazar(cantidad);  
    return intervalo;  
}
```

5. Sobrecarga de métodos.

En una misma clase varios métodos pueden tener el mismo nombre siempre y cuando difieran:

- En el número de parámetros.
- O bien, en el tipo de los parámetros comparados dos a dos.

Por ejemplo, en la clase *Intervalo* se definen dos operaciones *incluye*:

```
Ej.: public boolean incluye(double punto) {  
    return minimo <= punto && punto <= maximo;  
}  
  
public boolean incluye(Intervalo intervalo) {  
    return incluye(intervalo.minimo)  
        && incluye(intervalo.maximo);  
}
```

6. Constructores.

Constructores de la Clase:

Son métodos que reúnen las tareas de inicialización (no construyen) y se lanzan automáticamente en la construcción de objetos (new).

- No devuelven nada, tampoco *void*.
- Debe coincidir su nombre con el de la clase.

```
Ej. public Intervalo(double min, double max) {  
    minimo = min;  
    if (min < max) {  
        maximo = max;  
    } else {  
        System.out.println("El mínimo no puede ser superior");  
        maximo = min;  
    }  
}  
  
public Intervalo() {  
    minimo = 0;  
    maximo = 0;  
}
```

6. Constructores.

Constructores de la Clase:

```
Ej. public Intervalo(Intervalo intervalo){  
    minimo = intervalo.minimo;  
    máximo = intervalo.máximo;  
}
```

Toda clase tiene un constructor por defecto, que no tiene parámetros y que no inicializa nada. Sin embargo, si se define un constructor cualquiera en la clase, ésta deja de tener constructor por defecto.

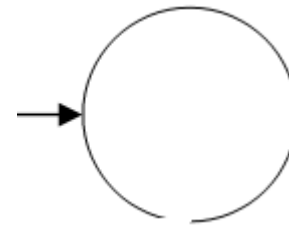
7. Instanciación de objetos.

Operador new:

Es un operador unario cuyo único operando es una clase. Crea un objeto de esa clase y devuelve la dirección de memoria donde se ha reservado espacio para dicho objeto.

Además, al crear un objeto se ejecuta el correspondiente constructor de forma automática.

```
new <Clase>([<expresion>, ...])
```

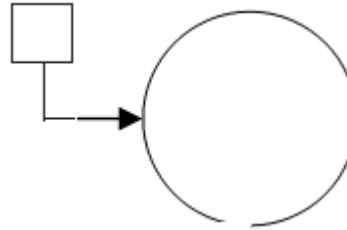


```
Ej.: new Intervalo()  
      new Intervalo(11.5, 55.1)  
      new Intervalo(new Intervalo(-1, 1))
```

7. Instanciación de objetos.

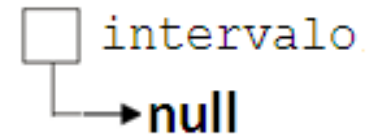
Referencia a un objeto:

Es una variable puntero que alberga la dirección de un objeto de una clase.



```
<Clase> <referencia> [ = <direccion> ];
```

```
Ej.: Intervalo intervalo;
```



Si no se inicializa, su dirección es **null**

```
Ej.: Intervalo edades = new Intervalo(0,100);  
      Intervalo años = edades;  
      Intervalo datos = new Intervalo(edades);  
      Intervalo datos2 = new Intervalo(new Intervalo(0,1));  
      boolean mismo = edades == años; // Compara referencias
```


8. Paso de mensajes.

`<referencia>.<metodo>([<expresión>, ...])`

Donde el método, sin contemplar constructores ni destructores, debe estar en la interfaz de la clase del objeto. Además, la lista de expresiones debe coincidir en número y tipos a la lista de parámetros del método.

```
Ej.: Intervalo intervalo = new Intervalo(0,100);
      Intervalo edades = intervalo.desplazado(5);
      double x = intervalo.longitud();
      double y = new Intervalo(-10,10).longitud();
      double z = intervalo.desplazado(3).puntoMedio();
      ...
```

8. Paso de mensajes.

Java utiliza siempre el paso **por valor** en los parámetros al llamar los métodos, es decir, dentro del método se trabaja con copias de los parámetros enviados:

```
Ej.: class Ejemplo {  
    public void metodo(int x) {  
        x = x * 3;  
    }  
}  
  
Ejemplo objeto = new Ejemplo();  
int valor = 10;  
objeto.metodo(valor);  
System.out.println("valor ahora es: " + valor);
```

valor ahora es: 10

8. Paso de mensajes.

Sin embargo, hay que tener en cuenta que Java maneja los objetos y los *arrays* a través de referencias. Al enviar una referencia a un método, éste toma una copia de dicha referencia, por tanto, las modificaciones que se hagan sobre el objeto **sí permanecen** después de ejecutar el método.

```
Ej.: class Ejemplo {  
    public void metodo(Intervalo distancia) {  
        distancia.desplazar(3);  
    }  
}  
  
Ejemplo objeto = new Ejemplo();  
Intervalo intervalo = new Intervalo(0,10);  
objeto.metodo(intervalo);  
System.out.println("Ahora es: " + intervalo.toString());
```

Ahora es: [3.0 , 13.0]

8. Paso de mensajes.

El valor devuelto por el método es también una copia del valor asociado a la sentencia *return*:

```
Ej.: private Intervalo copia() {  
    return new Intervalo(minimo, maximo);  
}  
  
public Intervalo desplazado(double cantidad) {  
    Intervalo intervalo = this.copia();  
    intervalo.desplazar(cantidad);  
    return intervalo;  
}
```

En ambos métodos el *return* devuelve una referencia a un objeto.

9. La referencia *this*.

this es una referencia constante que guarda la dirección del objeto que recibe el mensaje.

Si al acceder a un atributo o invocar a un método no se especifica ningún objeto, por defecto es *this*. Por tanto, estas dos implementaciones del método *escalar* serían equivalentes:

```
Ej.: public void escalar(double escala) {  
    double nuevaLongitud = longitud() * escala;  
    double puntoMedio = puntoMedio();  
    minimo = puntoMedio - nuevaLongitud / 2;  
    maximo = puntoMedio + nuevaLongitud / 2;  
}
```

```
Ej.: public void escalar(double escala) {  
    double nuevaLongitud = this.longitud() * escala;  
    double puntoMedio = this.puntoMedio();  
    this.minimo = puntoMedio - nuevaLongitud / 2;  
    this.maximo = puntoMedio + nuevaLongitud / 2;  
}
```

9. La referencia *this*.

- Normalmente, para acceder a los atributos del objeto que recibe el mensaje no se suele utilizar *this*, mientras que sí suele usarse de forma explícita para invocar a un método del propio objeto. En cualquier caso, es el equipo de desarrollo el que debe determinar la regla de estilo sobre *this*.
- Por tanto, la implementación más habitual del método *escalar* es la siguiente:

```
Ej.: public void escalar(double escala) {  
    double nuevaLongitud = this.longitud() * escala;  
    double puntoMedio = this.puntoMedio();  
    minimo = puntoMedio - nuevaLongitud / 2;  
    maximo = puntoMedio + nuevaLongitud / 2;  
}
```

9. La referencia this.

- Sin embargo, hay ocasiones en las que no queda más remedio que utilizarlo:
 1. Para evitar ambigüedades con los parámetros del método o constructor:

Ej.:

```
public Intervalo(double minimo, double maximo) {  
    if (minimo <= maximo) {  
        this.minimo = minimo;  
        this.maximo = maximo;  
    } else {  
        System.out.println("El minimo no puede ser mayor "  
            + " que el maximo");  
        this.minimo = 0;  
        this.maximo = 0;  
    }  
}
```

9. La referencia *this*.

2. Para reutilizar constructores en la definición de nuevos constructores. La sintaxis de la sentencia es **this**([<expresión>,...]) y debe usarse como primera sentencia del constructor.

Ej.:

```
public Intervalo() {  
    this(0,0);  
}  
  
public Intervalo(Intervalo intervalo){  
    this(intervalo.minimo, intervalo.maximo);  
}
```

- En los métodos estáticos no es posible referirse al objeto. Así que no es posible utilizar atributos del objeto, ni es posible enviar mensajes sobre *this*, **porque no existe *this***.

9. La referencia *this*.

Ejemplo de uso de la referencia *this*:

Se va a definir el método público *intersección* en la clase *Intervalo*. Este método devuelve un intervalo resultante de la intersección del intervalo actual con el intervalo recibido como parámetro, sin modificar el intervalo actual ni el recibido como parámetro.

```
Ej.: public Intervalo interseccion(Intervalo otro) {  
    Intervalo res;  
    if (this.incluye(otro)) {  
        res = otro.copia();  
    } else if (otro.incluye(this)) {  
        res = this.copia();  
    } else if (this.incluye(otro.minimo)) {  
        res = new Intervalo(otro.minimo, maximo);  
    } else if (this.incluye(otro.maximo)) {  
        res = new Intervalo(minimo, otro.maximo);  
    } else {  
        res = null;  
    }  
    return res;  
}
```

10. Tipos Enumerados (Enum).

- Un **enumerado** (o **Enum**) es una clase "especial" (tanto en Java como en otros lenguajes) que limita la creación de objetos a los especificados explícitamente en la implementación de la clase.
- Una enumeración define con precisión un nuevo tipo de datos que tiene un **número fijo de valores válidos**.
- Ej: *Colores*: Rojo, Verde, Azul...
- Ej: *Naipes*: Flores, Diamantes, Corazones, Espadas.
- Ej: *Meses del año*: Enero, Febrero, ..., Noviembre, Diciembre

10. Tipos Enumerados (Enum).

- Sintaxis de la declaración de un tipo enumerado:

```
enum <NombreEnum> {  
    [<ELEMENTO1>, <ELEMENTO2>, ...]  
}
```

Reglas de estilo:

- El nombre del enumerado se escribe con la primera letra mayúscula, siguiendo la misma regla de estilo que los nombres de clases.

```
Ej: enum Color {  
    AZUL, ROJO, VERDE  
}
```

- Los nombres de los elementos se escribe en mayúsculas, siguiendo las mismas reglas de estilo que las constantes.

10. Tipos Enumerados (Enum).

Para declarar una referencia tipo enumerado se hace de la misma manera que para cualquier otro objeto:

```
<NombreEnum> <referencia> [ = <NombreEnum>.<ELEMENTO> ] ;
```

```
Ej.: Color color;
```

La diferencia es que ahora no hay que crear el objeto con new, ya que los posibles objetos ya se encuentran creados:

```
Ej.: color = Color.AZUL;
```

10. Tipos Enumerados (Enum).

Puedo comparar si dos referencias a enumerados contienen el mismo enumerado de la siguiente manera:

```
Ej.: Color color1 = Color.AZUL;  
     Color color2 = color1;  
     Color color3 = Color.ROJO;  
     Color color4 = Color.AZUL;  
     boolean mismo = color1 == color4;
```

En este caso el resultado sería true, ya que tanto color1 como color 2 están referenciando al mismo valor enumerado.

10. Tipos Enumerados (Enum).

Estructuras de control y tipos enumerados:

En los condicionales con *if* para comprobar el valor de un tipo enumerado es necesario invocar el tipo enumerado y el valor:

```
Ej.:  if (color1 == Color.AZUL){  
        ...  
    }
```

Sin embargo, el condicional *switch* no requiere invocar el tipo enumerado, únicamente el valor a comprobar:

```
Ej.:  switch (color1){  
        case AZUL:  
            System.out.println("El color es Azul");  
            break;  
        case ROJO:  
            System.out.println("El color es Rojo");  
            break;  
        case VERDE:  
            System.out.println("El color es Verde");  
            break;  
    }
```

10. Tipos Enumerados (Enum).

Funciones nativas para tipos enumerados:

```
public String name()
```

Devuelve el nombre del enumerado en forma de cadena de texto

```
public int ordinal()
```

Devuelve la posición del enumerado dentro de la secuencia

```
public int compareTo(<enumType> other)
```

Devuelve la diferencia de posición entre el enumerado actual y el enumerado other.

```
public static <EnumType> valueOf(String value)
```

A partir de una cadena de texto que representa a un valor enumerado, te devuelve una referencia al objeto correspondiente.

```
public static <EnumType>[] values()
```

Devuelve un *array* con todos los valores de los enumerados.

10. Tipos Enumerados (Enum).

Funciones nativas para tipos enumerados:

```
Color color5 = Color.valueOf("VERDE");  
Color color3 = Color.AZUL;  
System.out.println("Nombre de color5: " + color5.name());  
System.out.println("Ordinal de color3: " + color3.ordinal());  
System.out.println("Ordinal de color5: " + color5.ordinal());  
System.out.println("color3 - color5: " + color3.compareTo(color5));  
System.out.println("color5 - color3: " + color5.compareTo(color3));
```

Resultado:

```
Nombre de color5: VERDE  
Ordinal de color3: 0  
Ordinal de color5: 2  
color3 - color5: -2  
color5 - color3: 2
```


11. El API de la biblioteca de Java.

La biblioteca de Java:

- JSE (Java Standard Edition) ofrece una gran cantidad de clases en su biblioteca para que los desarrolladores puedan utilizarlas en sus aplicaciones. Además, este código se encuentra ya probado, con lo que reducimos la posibilidad de cometer errores.
- Estas bibliotecas se encuentran organizadas en **paquetes**. Cada paquete contiene una serie de clases.
 - Paquete **java.lang**: Clases básicas del lenguaje.
 - Paquete **java.util**: Utilidades y estructuras de datos
 - Paquete **java.io**: Acceso a entrada/salida.
 - Paquete **javax.swing**: Interfaz gráfica de usuario.
 - Paquete **java.net**: Acceso a redes.
 - Paquete **java.sql**: Acceso a base de datos.
 - ...

11. El API de la biblioteca de Java.

La biblioteca de Java:

- Para utilizar una clase de la biblioteca en una aplicación se utiliza la sentencia *import*

```
import <paquete>.<clase>;
```

Ej.:

```
import java.util.Date;

class Aplicación {
    public static void main(String[] args){
        Date fecha = new Date();
        ...
    }
}
```

- También se puede utilizar la notación **import** <paquete>.* cuando son muchas las clases las que se quieren importar del mismo paquete.
- Las clases del paquete **java.lang** se importan de forma implícita, por lo que no necesitan sentencia *import*.

11. El API de la biblioteca de Java.

Para utilizar la biblioteca, Java nos proporciona un API que contiene información de los paquetes y de las clases. Concretamente, contiene el interfaz de cada una de las clases para poder utilizarlas correctamente.

<https://docs.oracle.com/javase/8/docs/api/>

Ejemplo: la clase String (java.lang.String):

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

Algunos constructores:

```
public String()  
public String(String original)  
...
```

- Es la única clase cuyos objetos se pueden representar de forma literal. Por ejemplo, `String saludo = "hola";`
- Además, es la única clase que tiene un operador (+), que sirve para la concatenación de cadenas de caracteres.

11. El API de la biblioteca de Java.

Ejemplo: la clase String (java.lang.String):

Algunos Métodos:

```
public int charAt(int index)
public int compareTo(String anotherString)
public boolean endsWith(String suffix)
public int length()
public boolean isEmpty()
public int indexOf(String)
public String substring(int, int)
public String substring(int)
public String trim()
public boolean equals(Object anObject)
...
```

11. El API de la biblioteca de Java.

Ejemplo: la clase String (java.lang.String):

```
// No necesario importar java.lang.String (pertenece a java.lang)
class Pruebas {
    public static void main(String[] args){
        String cadena = "competencia";
        String otraCadena = new String("desleal");
        String otraCadenaMas = cadena + " " + otraCadena;
        System.out.print(otraCadenaMas + ": " + otraCadenaMas.length() +
                           " caracteres");

        System.out.println(", comienza por " + otraCadenaMas.charAt(0);
        System.out.println("Posicion 'pete' en " + cadena + ": " +
                           cadena.indexOf("pete"));
        System.out.println("Caracteres 5 al 8 en " + cadena + ": " +
                           cadena.substring(5,8));
    }
}
```

competencia desleal: 19 caracteres, comienza por c
Posicion 'pete' en competencia: 3
Caracteres 5 al 8 en competencia: ten

11. El API de la biblioteca de Java.

Ejemplo: la clase Math (java.lang.Math):

<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

- Contiene un conjunto de métodos **estáticos** o de clase. Dicho de otra manera, para utilizar estos métodos no hay que instanciar ningún objeto, ni se lanza ningún mensaje sobre ningún objeto. La sintaxis para invocar los métodos estáticos es la siguiente:

```
<Clase>.<metodoEstatico>([<argumento>, ... ])
```

11. El API de la biblioteca de Java.

Ejemplo: la clase Math (java.lang.Math):

Métodos:

```
public static double sin(double a);  
public static double round(double a);  
public static double log(double a)  
public static double random();  
public static double pow(double a, double b)  
...
```

```
Ej.: System.out.println("8 al cubo: " + Math.pow(8,3));  
      double x = Math.pow(Math.sin(1),2) + Math.pow(Math.cos(1),2);  
      System.out.println("Ecuación fundamental trigonometría: " + x);
```

8 al cubo: 512.0

Ecuación fundamental trigonometría: 1.0

11. El API de la biblioteca de Java.

- La clase *Math* contiene también dos atributos **estáticos** o globales. Se trata también de atributos que no están asociados a un objeto, son únicos para la clase. En este caso, el valor de la constante **PI** y el número **E**.

Atributos:

```
public static final double E;  
public static final double PI
```

- Las constantes globales es el único tipo de atributo que puede definirse como público, ya que son servicios de la clase.

```
Ej.:   int radio = 7;  
       System.out.println("Circunferencia: " + 2 * Math.PI * radio);  
       System.out.println("Logaritmo natural: " + Math.log(Math.E));
```

Circunferencia: 43.982297150257104

Logaritmo natural: 1.0

- El atributo *out* en la clase *System* es otro ejemplo de constante global.

12. Las clases de recubrimiento de Java.

- Los tipos primitivos son los únicos tipos que no son clases en Java.
- Sin embargo, Java nos proporciona una serie de clases que engloban a los tipos primitivos: se manejan igual, pero amplían su funcionalidad.
- CLASES: **Byte, Short, Integer, Long, Float, Double, Boolean.**

La clase Integer (java.lang.Integer):

Constantes globales:

```
public static final MAX_VALUE;
```

Constructores:

```
public Integer(int value);
```

```
public Integer(String s);
```

Métodos:

```
public static int parseInt(String s);
```

Si en la cadena de texto hay un entero, obtiene dicho entero.

```
public static int max(int a, int b);
```

```
...
```

12. Las clases de recubrimiento de Java.

La clase Integer (java.lang.Integer):

```
Ej.: Integer numero = new Integer(5);  
    // Se puede hacer lo mismo que con un int  
    numero+= 3;  
    System.out.println ("numero es " + numero);  
    // Uso de métodos estáticos  
    int valor = Integer.parseInt("27");  
    int minimo = Integer.min(5,3);  
    System.out.println("valor es " + valor + " y minimo es " +  
                        minimo);
```

numero es 8

valor es 27 y minimo es 3

12. Las clases de recubrimiento de Java.

Ejercicio:

Defina un nuevo constructor para la clase *Intervalo* que reciba un parámetro de tipo *String*, con formato “[*minimo,máximo*]” . Por ejemplo, “[1.3, 5]”

```
Ej.: public Intervalo(String cadena) {  
    minimo = Double.parseDouble(cadena.substring(  
        1, cadena.indexOf(", ")));  
    cadena = cadena.substring(cadena.indexOf(", ") + 1,  
        cadena.length() - 1);  
    maximo = Double.parseDouble(cadena);  
    if (minimo > maximo){  
        System.out.println("El mínimo no puede ser superior");  
        minimo = 0;  
        maximo = 0;  
    }  
}
```