



POLITÉCNICA

**UNIVERSIDAD POLITÉCNICA DE MADRID**  
**ESCUELA UNIVERSITARIA DE INFORMÁTICA**  
**LENGUAJES, PROYECTOS Y SISTEMAS INFORMÁTICOS**



**Luis Fernández Muñoz**  
**setillo@eui.upm.es**

# **Programación**

# **Orientada a Objetos**

# **en Java**

## ***Tema 3. CLASES Y OBJETOS***

1. Introducción

2. Vista Pública de las Clases

3. Vista Pública de los Objetos

4. Vista Privada de las Clases

5. Vista Privada de los Objetos

6. Miembros de Clase

# 1. INTRODUCCIÓN

---

**VISTAS:** determina el ámbito donde se puede referenciar la declaración de un miembro de la clase: atributo o método.

- *Pública*: conocido en cualquier punto de la aplicación (antes de la declaración, después y en cualquier otro fichero)
- *Privada*: conocido en cualquier punto de la clase (antes y después de la declaración pero en la implementación de la clase)

	Clases	Objetos
Vista Pública	<b>INTERFAZ:</b> Nombre de la Clase Cabecera de los Métodos	Creación de Objetos Paso de Mensajes Destrucción de Objetos
Vista Privada	<b>IMPLEMENTACIÓN:</b> Definición de Atributos Definición de Métodos	Desencadenamiento de Objetos Desencadenamiento de Mensajes

# ***1. INTRODUCCIÓN***

---

## **Dependencias:**

- *al crear objetos y lanzarles mensajes, se debe respetar la interfaz de sus clases =>*

*1º Vista Pública de las Clases*

*2º Vista Pública de los Objetos*

- *al definir atributos de una clase (generalmente, objetos de otras clases) y definir sus métodos (generalmente, lanzando mensajes a los objetos atributos), se debe respetar, transitivamente, la interfaz de otras clases =>*

*3º Vista Privada de las Clases*

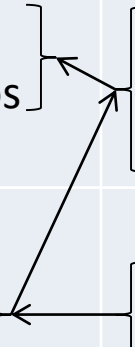
- *al crear un objeto se produce un desencadenamiento de instanciaciones y al lanzar un mensaje se produce un desencadenamiento de mensajes, acorde a la definición de atributos y de métodos de la clase del objeto =>*

*4º Vista Privada de Objetos*

# 1. INTRODUCCIÓN

## Dependencias:

	Clases	Objetos
<b>Vista Pública</b>	<b>INTERFAZ:</b> Nombre de la Clase Cabecera de los Métodos	Creación de Objetos Paso de Mensajes Destrucción de Objetos
<b>Vista Privada</b>	<b>IMPLEMENTACIÓN:</b> Definición de Atributos Definición de Métodos	Desencadenamiento de Instancias Desencadenamiento de Mensajes



*Las flechas indican las dependencias de cada apartado*

# 2. VISTA PÚBLICA DE LAS CLASES

---

## Nombre de la Clase:

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

*- donde el nombre de la clase debe comenzar con mayúscula*

```
Ej.: class Intervalo {  
}
```

*la clase Intervalo implanta el concepto de un segmento limitado por un mínimo y un máximo:*

- intervalo de horas laborales [8,15]*
- intervalo de la capacidad de memoria de los pc's de un laboratorio [1.2,4]*
- intervalo de la tensión arterial aconsejable para una persona [8.4,14.3]*
- ...*

```
Ej.: class Coordinada {  
}
```

# 2. VISTA PÚBLICA DE LAS CLASES

---

## Cabecera de Métodos de la Clase:

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

- donde el tipo<sub>1</sub> indica el tipo del valor devuelto, que puede ser:
  - **void** (nada)
  - <tipo/Clase> (un valor de tipo primitivo o una referencia a un objeto de una clase)
  - <tipo/Clase>[] (una referencia a un vector de valores de tipos primitivos o de referencias a objetos)
  - <tipo/Clase>[][] (una referencia a una matriz de valores de tipos primitivos o de referencias a objetos)
- donde el nombre del método debe comenzar con minúscula
- donde el tipo<sub>2</sub> puede ser igual que tipo<sub>1</sub> excepto **void**
- donde todos los parámetros son pasados por valor

## ***2. VISTA PÚBLICA DE LAS CLASES***

---

```
Ej.: class Intervalo {  
    public void mostrar()  
    public void recoger()  
    public double longitud ()  
    public double puntoMedio ()  
    public Intervalo simetrico ()  
    public void escalar (double factor)  
    public void desplazar (double desplazamiento)  
    public boolean incluye (double punto)  
    public boolean igual (Intervalo intervalo)  
    public boolean distinto (Intervalo intervalo)  
    public boolean intersecta (Intervalo intervalo)  
    public Intervalo desplazado (double desplazamiento)  
    public Intervalo interseccion (Intervalo intervalo)  
    public Intervalo union (Intervalo intervalo)  
    public Intervalo entre (Intervalo intervalo)  
    public Intervalo[] trozos (int trozos)  
}
```



## ***2. VISTA PÚBLICA DE LAS CLASES***

---

**Sobrecarga de Métodos de la Clase:** varios métodos pueden tener el mismo nombre con las siguientes restricciones:

- si están en la misma clase, deben diferenciarse en el número o tipo de parámetros comparados dos a dos;
- si están en distintas clases, no existe restricción;

```
Ej.: class Intervalo {  
    public boolean valido()  
    public boolean incluye(double punto)  
    public boolean incluye(Intervalo intervalo)  
    ...  
}
```

```
Ej.: class Fecha {  
    public boolean valido()  
    ...  
}
```

## ***2. VISTA PÚBLICA DE LAS CLASES***

---

**Constructores de la Clase:** son métodos que reúnen las tareas de inicialización (no construyen) y se lanzan automáticamente en la construcción de objetos. Además:

- no devuelven nada (ni **void**);
- deben coincidir su nombre con el de la clase;
- no se pueden lanzar mensajes que se correspondan con los constructores de la clase.

```
Ej.: class Intervalo {  
    public Intervalo()  
    public Intervalo(double maximo)  
    public Intervalo(double minimo, double maximo)  
    public Intervalo(Intervalo intervalo)  
    ...  
}
```

## 2. VISTA PÚBLICA DE LAS CLASES

---

**Destruectores de la Clase:** son métodos que reúnen las tareas de liberación de recursos (no destruyen) y se lanzan automáticamente en la destrucción de objetos. Además:

- su cabecera debe ser: `public void finalize()`
- no se pueden lanzar mensajes que se correspondan con los destructores de la clase.
- *muy poco frecuentes porque:*
  - *Java tiene incorporado un recolector de basura que libera automáticamente la memoria dinámica no apuntada por ninguna referencia*
  - *normalmente un objeto no “vive” con una línea de comunicación, un fichero abierto, ... que deba cerrar y, en tal caso, es preferible gestionar dichos recursos sin intervención de automatismos*

## ***2. VISTA PÚBLICA DE LAS CLASES***

---

### **CLASE: GestorIO**

```
public int inInt ()  
public float inFloat ()  
public double inDouble ()  
public long inLong ()  
public byte inByte ()  
public short inShort ()  
public char inChar ()  
public boolean inBoolean ()  
public String inString ()  
...
```

```
public void out (int salida)  
public void out (float salida)  
public void out (double salida)  
public void out (long salida)  
public void out (byte salida)  
public void out (Short salida)  
public void out (char salida)  
public void out (boolean salida)  
public void out (String salida)  
}
```

## ***2. VISTA PÚBLICA DE LAS CLASES***

---

**CADENAS DE CARACTERES**: son secuencias de caracteres de cualquier longitud.

### **Tipos:**

- **Constantes** (inmutables): cuyos caracteres no varían (ej.: el nombre de una persona, de un mes, ...).
  - Está implementadas bajo la clase **String** sin contemplar métodos que modifiquen el estado del objeto.
- **Variables** (mutables): cuyos caracteres sí varían (ej.: membrete de una carta, ...).
  - Están implementadas bajo la clase **StringBuffer** con métodos que contemplan la modificación del objeto.

# ***2. VISTA PÚBLICA DE LAS CLASES***

---

## **CLASE: String (I)**

```
public String()  
public String(char[])  
public String(char[], int, int)  
public String(int[], int, int)  
public String(byte[], int, int, int)  
public String(byte[], int)  
public String(byte[], int, int)  
public String(byte[])  
public String(StringBuffer)  
public String(StringBuilder)  
public String(int, int, char[])  
public int length()  
public boolean isEmpty()  
public char charAt(int)
```

```
public int offsetByCodePoints(int, int)  
void getChars(char[], int)  
public void getChars(int, int, char[], int)  
public void getBytes(int, int, byte[], int)  
public byte[] getBytes()  
public boolean contentEquals(  
    StringBuffer)  
public boolean equalsIgnoreCase(String)  
public int compareTo(String)  
public int compareToIgnoreCase(String)  
public boolean regionMatches(int,  
    String, int, int)  
public boolean startsWith(String, int)  
public boolean startsWith(String)
```

# 2. VISTA PÚBLICA DE LAS CLASES

---

## CLASE: String (II)

```
public boolean endsWith(String)
public int indexOf(int)
public int indexOf(int, int)
public int lastIndexOf(int)
public int lastIndexOf(int, int)
public int indexOf(String)
public int indexOf(String, int)
public int lastIndexOf(String)
public int lastIndexOf(String, int)
public String substring(int)
public String substring(int, int)
```

```
public String concat(String)
public String replace(char, char)
public boolean matches(String)
public String replaceFirst(String, String)
public String replaceAll(String, String)
public String[] split(String, int)
public String[] split(String)
public String toLowerCase()
public String toUpperCase()
public String trim()
public String toString()
public char[] toCharArray()
```

# 2. *VISTA PÚBLICA DE LAS*

---

## *CLASES*

### CLASE: StringBuffer (I)

```
public StringBuffer()  
public StringBuffer(int)  
public StringBuffer(String)  
public int length()  
public int capacity()  
public void ensureCapacity(int)  
public void trimToSize()  
public void setLength(int)  
public char charAt(int)  
public void getChars(int, int, char[], int)  
public void setCharAt(int, char)  
public StringBuffer append(String)  
public StringBuffer append(StringBuffer)
```

```
public StringBuffer append(char[])  
public StringBuffer append(char[], int,  
                           int)  
public StringBuffer append(boolean)  
public StringBuffer append(char)  
public StringBuffer append(int)  
public StringBuffer append(long)  
public StringBuffer append(float)  
public StringBuffer append(double)  
public StringBuffer delete(int, int)  
public StringBuffer deleteCharAt(int)  
public StringBuffer replace(int, int,  
                           String)  
public String substring(int)
```



# 2. *VISTA PÚBLICA DE LAS*

---

## *CLASES*

### CLASE: StringBuffer (II)

```
public CharSequence subSequence(int,  
    int)  
public String substring(int, int)  
public StringBuffer insert(int, char[], int,  
    int)  
public StringBuffer insert(int, String)  
public StringBuffer insert(int, char[])  
public StringBuffer insert(int, boolean)  
public StringBuffer insert(int, char)  
public StringBuffer insert(int, int)  
public StringBuffer insert(int, long)  
public StringBuffer insert(int, float)  
public StringBuffer insert(int, double)
```

```
public int indexOf(String)  
public int indexOf(String, int)  
public int lastIndexOf(String)  
public int lastIndexOf(String, int)  
public StringBuffer reverse()  
public String toString()  
...
```

## 2. VISTA PÚBLICA DE LAS CLASES

---

### LITERALES STRING:

- es la única clase cuyos objetos pueden presentarse en forma literal;
- su formato es una secuencia de caracteres de cualquier longitud encerrados entre dobles comillas `"<carácter>..."`
- son objetos de la clase **String** cuya evaluación devuelve la dirección del objeto al que representan;
- además, es la única clase que disfruta de un operador (+) para la concatenación de cadenas de caracteres y combinado con cualquier tipo.

## ***2. VISTA PÚBLICA DE LAS CLASES***

---

```
Ej.: int longitud = "caracteres".length();
    String cadena = "caracteres";
    boolean falso = cadena == "caracteres";
    boolean cierto = cadena.equals("caracteres");
    boolean tambien = "caracteres".equals(cadena);
    StringBuffer buffer = new StringBuffer(cadena);
    buffer.insert(0, "de ").insert(0, "cadena ");
    boolean si = ("cadena de " + cadena).
        contentEquals(buffer);
    String serie = (0+1)+", " + (1+1)+", " + (2+1)+".";
```

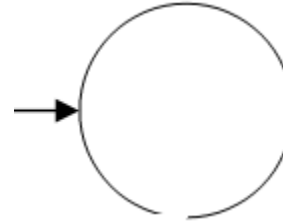
# 3. VISTA PÚBLICA DE LOS OBJETOS

---

**CREACIÓN DE OBJETOS:** **new** es un operador unario prefijo cuyo operando es una clase de objetos y devuelve la dirección de memoria donde se ha reservado el espacio para dicho objeto;

**Sintaxis f):**

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

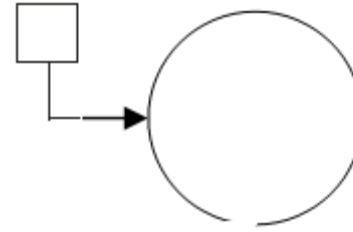


- donde la lista de expresiones debe coincidir con la lista de parámetros de alguno de los constructores de la clase; en caso de no existir, la lista debe ser vacía.

```
Ej.: new Intervalo()  
      new Intervalo(100)  
      new Intervalo(11.5, 55.1)  
      new Intervalo(new Intervalo(-1, 1))
```

# 3. VISTA PÚBLICA DE LOS OBJETOS

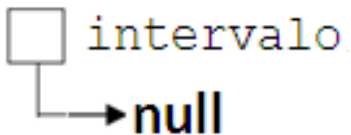
REFERENCIA a un objeto : es una variable puntero que alberga la dirección de un objeto de una clase.



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

- a falta de inicialización su dirección es **null**
- **final** obliga a la inicialización y fija su valor para la referencia

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



# ***3. VISTA PÚBLICA DE LOS OBJETOS***

---

## **Operadores:**

- *<referenciaO> = <direcciónO>*: asigna la dirección a la referencia siendo del mismo tipo;
- *<direcciónO-I> == <direcciónO-D>*: determina si dos direcciones a objetos de la misma clase son iguales;
- *<direcciónO-I> != <direcciónO-D>*: determina si dos direcciones a objetos de la misma clase son distintas;

```
Ej.: final Intervalo HORARIO = new Intervalo(7, 15);  
      Intervalo edades = new Intervalo(100);  
      Intervalo años;  
      años = edades;  
      boolean mismo = edades == años;  
      ...
```

# ***3. VISTA PÚBLICA DE LOS OBJETOS***

---

## **PASO DE MENSAJES:**

```
<direccion0>.<metodo> ([<expresión>, ...])
```

- donde el método (sin contemplar constructores ni destructores) debe estar presente en la interfaz de la clase del objeto y la lista de expresiones debe coincidir en número y tipos a la lista de parámetros del método;

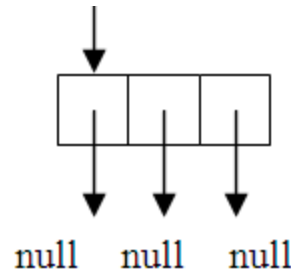
```
Ej.: intervalo.longitud()  
    new Intervalo(-100, 100).longitud()  
    edades.partido(5)  
    años.incluye(88)  
    edades.interseccion(años)  
    ...
```

# 3. VISTA PÚBLICA DE LOS OBJETOS

**CREACIÓN DE VECTORES DE OBJETOS:** **new** es operador unario prefijo cuyo operando es un vector de referencias a objetos de una clase y devuelve la dirección de memoria donde se ha reservado el espacio para dicho vector;

**Sintaxis g):**

```
new <Clase>[ <expresion >]
```



- donde la expresión debe ser de tipo entero y determina la longitud de referencias del vector inicializadas a **null**;

```
Ej.: new Intervalo[100]
```



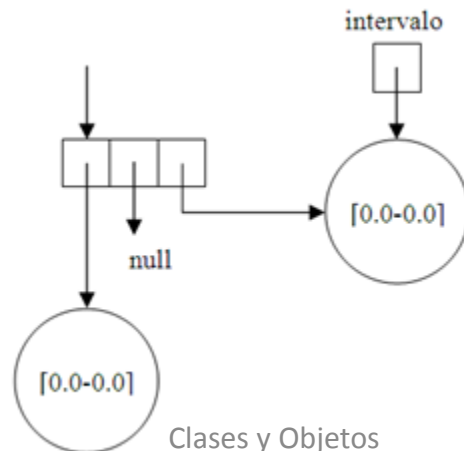
# 3. VISTA PÚBLICA DE LOS OBJETOS

## Sintaxis h):

```
new <Clase>[] { <expresion>, ..., <expresion> }
```

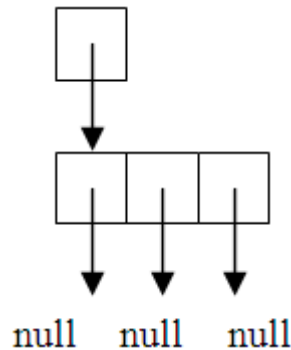
- donde cada expresión debe ser una dirección a un objeto de la clase que inicializan las referencias del vector creado de longitud igual al número de expresiones;

```
Ej.: Intervalo intervalo = new Intervalo();  
    new Intervalo[] {new Intervalo(), null, intervalo}
```



# 3. VISTA PÚBLICA DE LOS OBJETOS

REFERENCIA a un vector de referencias a objetos: es una variable puntero que alberga la dirección de un vector de referencias a objetos de una clase.



```
[ final ] <Clase>[] <referencia> [ = <direccionVR> ];
```

- a falta de inicialización su dirección es **null**
- **final** obliga a la inicialización y fija su valor para la referencia

```
Ej.: Intervalo[] intervalos;
```

intervalos



# ***3. VISTA PÚBLICA DE LOS OBJETOS***

---

```
Ej.:  
Intervalo[] intervalos = new Intervalo[10];  
intervalos[0] = new Intervalo();  
intervalos[1] = intervalos[0].desplazado(1);  
intervalos[intervalos.length-1] = new Intervalo(2,2);  
Intervalo intervalo = intervalos[1];  
intervalos = null1;  
...
```

*Se libera automáticamente toda la memoria de las referencias del vector y de cada objeto excepto del segundo intervalo porque nadie está apuntando a dichos elementos;*

# 4. VISTA PRIVADA DE LAS CLASES

---

**DEFINICIÓN DE ATRIBUTOS**: en cualquier punto de la implementación de la clase, se declaran variables de tipos primitivos, referencias a objetos o vectores de éstos anteponiendo la palabra **private** o, excepcionalmente para constantes globales<sup>1</sup>, **public**;

```
class <clase> {  
    private <atributo>  
    private <atributo>  
    ...  
}
```

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

<sup>1</sup> Se tratarán posteriormente;

# ***4. VISTA PRIVADA DE LAS CLASES***

---

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

```
Ej.: class Persona {  
    private byte edad;  
    private final int dni;  
  
    public Persona(int dni) {  
        edad = 0;  
        this.dni = dni;  
    }  
    ...  
}
```

# 4. VISTA PRIVADA DE LAS CLASES

---

**DEFINICIÓN DE METODOS:** en cualquier punto de la implementación de la clase, se define el cuerpo de las cabeceras de los métodos acompañándolos de una sentencia secuencial que contiene las declaraciones locales y sentencias que se consideren oportunas;

```
class <clase> {  
    public <cabeceraMetodo>  
        <sentenciaSecuencial>  
    public <cabeceraMetodo>  
        <sentenciaSecuencial>  
    ...  
}
```

# 4. VISTA PRIVADA DE LAS CLASES

---

- dentro del cuerpo del método se tiene acceso a los atributos, los parámetros del método y a las declaraciones locales (*ley estricta de Demeter*)

```
Ej.: class Intervalo {  
  
    private double minimo;  
    private double maximo;  
  
    public void desplazar(double cantidad) {  
        minimo += cantidad;  
        maximo += cantidad;  
    }  
    ...  
}
```

# 4. VISTA PRIVADA DE LAS CLASES

---

- si el tipo devuelto no es **void**, la sentencia determina el valor devuelto por el método;

**return** <expresion>;

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



# ***4. VISTA PRIVADA DE LAS CLASES***

---

- dado que los atributos son privados a la clase, se puede acceder a los atributos de un objeto de la clase que se está implementando mediante la notación punto: `<direccion0>.<atributo>`

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

# 4. VISTA PRIVADA DE LAS CLASES

---

- **this** es una referencia constante que guarda la dirección del objeto que recibe el mensaje correspondiente al método que se está definiendo; por tanto: `private final <Clase> this;`

## USOS:

- resolución de ambigüedades en la colisión de parámetros o declaraciones locales con el mismo nombre que los atributos;
- reutilización de métodos en la codificación de otros métodos;
- reutilización de constructores en la codificación de otros constructores;

# ***4. VISTA PRIVADA DE LAS CLASES***

---

- Resolución de ambigüedades en la colisión de parámetros o declaraciones locales con el mismo nombre que los atributos;

```
Ej.: public Intervalo(double minimo, double maximo) {  
    this.minimo = minimo;  
    this.maximo = maximo;  
}
```

# ***4. VISTA PRIVADA DE LAS CLASES***

---

- Reutilización de métodos en la codificación de otros métodos;

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

# 4. VISTA PRIVADA DE LAS CLASES

---

- Reutilización de constructores en la definición de otros constructores mediante la sintaxis: `this ( [ <expresion>, ... ] );`
- Debe ser la primera sentencia del constructor;

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

# 4. VISTA PRIVADA DE LAS CLASES

---

- dado que puede ser conveniente disponer de métodos, que no han sido solicitados, para implementar otros métodos, cabe la posibilidad de definir métodos de ámbito privado que sólo se pueden usar en la implementación de la clase:

```
class <clase> {  
    private <cabeceraMetodo>  
    <sentenciaSecuencial>  
    private <cabeceraMetodo>  
    <sentenciaSecuencial>  
    ...  
}
```

# ***4. VISTA PRIVADA DE LAS CLASES***

---

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

# ***4. VISTA PRIVADA DE LAS CLASES***

---

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



# ***5. VISTA PRIVADA DE LOS OBJETOS***

---

**DESENCADENAMIENTO DE INSTANCIACIONES**: cuando se crea un objeto (instancia):

- se crean los atributos definidos en la clase;
- se ejecuta la inicialización de los atributos;
- se ejecuta el constructor;

Donde, en particular:

- pueden declararse referencias a objetos de otras clases y, por tanto,
- crearse nuevos objetos en su inicialización
- o en su constructor

Donde, recursivamente, pueden crear nuevos objetos de otras clases hasta llegar, de esta manera, a la creación de objetos que se basan directamente en tipos primitivos.

# ***5. VISTA PRIVADA DE LOS OBJETOS***

---

**DESENCADENAMIENTO DE MENSAJES**: cuando se lanza un mensaje a un objeto:

- se crean las declaraciones locales con su inicialización y
- se ejecuta el cuerpo del método correspondiente

Donde, en particular:

- pueden lanzarse nuevos mensajes a objetos que sean atributos de su clase,
- a objetos que sean argumentos del mensaje o
- a objetos que se crean en su ejecución

Donde, recursivamente, pueden lanzar nuevos mensajes en la definición de sus respectivos métodos hasta llegar, de esta manera, a la definición de métodos que se basan directamente en tipos primitivos.

# ***5. VISTA PRIVADA DE LOS OBJETOS***

---

- el desencadenamiento de instanciaciones puede provocar un desencadenamiento de mensajes a través de la ejecución de los constructores que pueden lanzar mensajes;

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

- el desencadenamiento de mensajes puede provocar un desencadenamiento de instanciaciones a través de la creación de objetos en la definición de los métodos.

```
Ej.: private Intervalo copia() {  
        return new Intervalo(this);  
    }  
    ...
```

# ***6. MIEMBROS DE CLASE***

---

## **MIEMBROS DE INSTANCIA:**

- atributos presentes en cada uno de los objetos de la clase;  
*Ej.: día, mes y año de una fecha concreta.*
- métodos cuyos mensajes se lanzan sobre un objetos particular.  
*Ej.: si una fecha concreta se encuentra en una año bisiesto.*

## **MIEMBROS DE CLASE (miembros estáticos):**

- atributos compartidos por la globalidad de objetos de la clase;  
*Ej.: los nombres de los meses de cualquier año.*
- métodos cuyos mensajes NO se lanzan sobre un objetos particular.  
*Ej.: si un año (no una fecha particular) es bisiesto.*

Los miembros de clase o estáticos  
asumen los aspectos globales  
de la comunidad de objetos de la clase

# 6. MIEMBROS DE CLASE

---

## ATRIBUTOS ESTÁTICOS:

- caracterizados por la palabra reservada **static** tras su visibilidad;
- su reserva de memoria e inicialización obligatoria se realiza al principio de la ejecución del programa, incluso ante la ausencia de objetos de la clase;
- accesibles desde cualquier método de la clase;
- la notación sintáctica para el acceso:

```
<Clase>.<atributoEstatico>
```

# 6. MIEMBROS DE CLASE

---

## MÉTODOS ESTÁTICOS:

- caracterizados por la palabra reservada **static** tras su visibilidad;
- no se permite el acceso a **this** ni a los atributos de instancia;
- se permite el acceso a los atributos estáticos;
- la notación sintáctica para el lanzamiento del mensaje:

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

# 6. MIEMBROS DE CLASE

---

```
Ej.: class Fecha {
    private int dia;
    private int mes;
    private int año;
    private static final int[] DIAS_MESES =
        {31,28,31,30,31,30,31,31,30,31,30,31};

    public boolean posterior(Fecha fecha) {
        boolean resultado;
        if (this.año == fecha.año)
            if (this.mes == fecha.mes)
                resultado = this.dia > fecha.dia;
            else
                resultado = this.mes > fecha.mes;
        else
            resultado = this.año > fecha.año;
        return resultado;
    }
}
```

# 6. MIEMBROS DE CLASE

---

```
Ej.: public static boolean bisiesto(int año) {  
    return año%4 == 0;  
}  
  
public boolean enBisiesto() {  
    return Fecha.bisiesto(año);  
}  
  
public int diasTranscurridosAño() {  
    int dias = dia - 1;  
    for (int i = 1; i < mes; i++)  
        dias += Fecha.DIAS_MESES[i - 1];  
    if (this.enBisiesto() &&  
        this.posterior(new Fecha(29, 2, año)))  
        dias++;  
    return dias;  
}  
...
```



# 6. MIEMBROS DE CLASE

---

```
Ej.: public static int diasAño(int año) {  
    int dias = 0;  
    for (int i = 0; i < Fecha.DIAS_MESES.length; i++)  
        dias += Fecha.DIAS_MESES[i];  
    if (Fecha.bisiesto(año))  
        dias++;  
    return dias;  
}  
...
```

# 6. MIEMBROS DE CLASE

---

## CÓDIGO ESTÁTICO:

- sirve para la inicialización de los atributos estáticos cuando la inicialización posible no alcanza sus objetivos;
- se ejecutan en cualquier orden en el comienzo de la ejecución del programa;
- sintaxis:

```
<atributoEstático>  
static {  
    <sentencia/declaración>  
    ...  
    <sentencia/declaración>  
}
```

```
Ej.: private static int[] coeficientes = null;  
    static {  
        // lectura del fichero "coeficientes.sys"  
    }
```

# ***6. MIEMBROS DE CLASE***

---

## **Clases de Utilidad:**

- son clases que reúnen un conjunto cohesivo de métodos estáticos
- suelen no ser instanciables porque no responden al concepto habitual de clases de objetos (se consigue a través constructores privados que eviten la creación de objetos de esa clase de utilidad)

## **CLASE: System**

```
public static long currentTimeMillis()  
public static long nanotime()  
public static void exit(int)  
public static void gc()
```

# 6. MIEMBROS DE CLASE

---

## CLASE: Math (I)

```
public static final double E;  
public static final double PI;  
public static double sin(double)  
public static double cos(double)  
public static double tan(double)  
public static double asin(double)  
public static double acos(double)  
public static double atan(double)  
public static double toRadians(double)  
public static double toDegrees(double)  
public static double exp(double)  
public static double log(double)  
public static double log10(double)  
public static double sqrt(double)  
public static double cbrt(double)  
public static double IEEERemainder(  
    double, double)
```

```
public static double ceil(double)  
public static double floor(double)  
public static double rint(double)  
public static double atan2(double,  
    double)  
public static double pow(double,  
    double)  
public static int round(float)  
public static long round(double)  
public static double random()  
public static int abs(int)  
public static long abs(long)  
public static float abs(float)  
public static double abs(double)  
public static int max(int, int)  
public static long max(long, long)
```

# 6. MIEMBROS DE CLASE

---

## CLASE: Math (II)

```
public static float max(float, float)
public static double max(double,
    double)
public static int min(int, int)
public static long min(long, long)
public static float min(float, float)
public static double min(double,
    double)
public static double ulp(double)
public static float ulp(float)
public static double signum(double)
public static float signum(float)
public static double sinh(double)
public static double cosh(double)
public static double tanh(double)
public static double hypot(double,
    double)
```

```
public static double expm1(double)
public static double log1p(double)
public static double copySign(double,
    double)
public static float copySign(float, float)
public static int getExponent(float)
public static int getExponent(double)
public static double nextAfter(double,
    double)
public static float nextAfter(float,
    double)
public static double nextUp(double)
public static float nextUp(float)
public static double scalb(double, int)
public static float scalb(float, int)
...
```

# 6. MIEMBROS DE CLASE

---

## CLASES DE RECUBRIMIENTO:

para los tipos primitivos sirven para aglutinar funciones de conversión entre ellos y las correspondientes cadenas de caracteres junto con algunas otras funciones auxiliares particulares de cada clase.

CLASE: Byte

CLASE: Short

CLASE: Integer

CLASE: Long

CLASE: Float

CLASE: Double

CLASE: Boolean

## CLASE: Character

```
public static String toString(char)
public static char[] toChars(int)
public static boolean isLowerCase(char)
public static boolean isUpperCase(char)
public static boolean isTitleCase(char)
public static boolean isDigit(char)
public static boolean isDefined(char)
public static boolean isLetter(char)
public static boolean isLetterOrDigit(char)
public static char toLowerCase(char)
public static char toUpperCase(char)
public static char toTitleCase(char)
public static int getNumericValue(char)
public static boolean isSpace(char)
public static boolean isSpaceChar(char)
public static boolean isWhitespace(char)
```

# 6. MIEMBROS DE CLASE

---

## CLASE: Integer

```
public static final int MIN_VALUE;  
public static final int MAX_VALUE;  
public static final int SIZE;  
public static String toString(int, int)  
public static String toHexString(int)  
public static String toOctalString(int)  
public static String toBinaryString(int)  
public static String toString(int)  
static void getChars(int, int, char[])  
static int stringSize(int)  
public static int parseInt(String, int)  
public static int parseInt(String)  
public static Integer valueOf(String, int)  
public static Integer valueOf(String)  
public static Integer valueOf(int)  
public Integer(int)  
public Integer(String)
```

```
public byte byteValue()  
public short shortValue()  
public int intValue()  
public long longValue()  
public float floatValue()  
public double doubleValue()  
public String toString()  
public int hashCode()  
public static Integer getInteger(String)  
public static Integer getInteger(String,  
                                int)  
public static Integer getInteger(String,  
                                Integer)  
public static Integer decode(String)  
public int compareTo(Integer)  
...
```

# 6. MIEMBROS DE CLASE

---

```
Ej.: public Fecha(String cadena) {
    dia = Integer.parseInt(
        cadena.substring(0, cadena.indexOf("/")));
    cadena = cadena.substring(
        cadena.indexOf("/") + 1, cadena.length());
    mes = Integer.parseInt(
        cadena.substring(0, cadena.indexOf("/")));
    cadena = cadena.substring(
        cadena.indexOf("/") + 1, cadena.length());
    año = Integer.parseInt(cadena);
}

public String toString() {
    return dia + "/" + mes + "/" + año;
}

...
```



# ***6. MIEMBROS DE CLASE***

---

**PRUEBAS DE CLASE**: cada clase debe ser probada por separado antes de integrarse en las pruebas de otras clases que se apoyan en ésta (método bottom-up). Para ello puede incorporarse el método **main** en cada clase:

```
public static void main(String[] args) {  
    ...  
}
```

- donde el parámetro **args** contiene las cadenas de caracteres que acompañan a la ejecución de las pruebas de la clase en la línea de comandos.