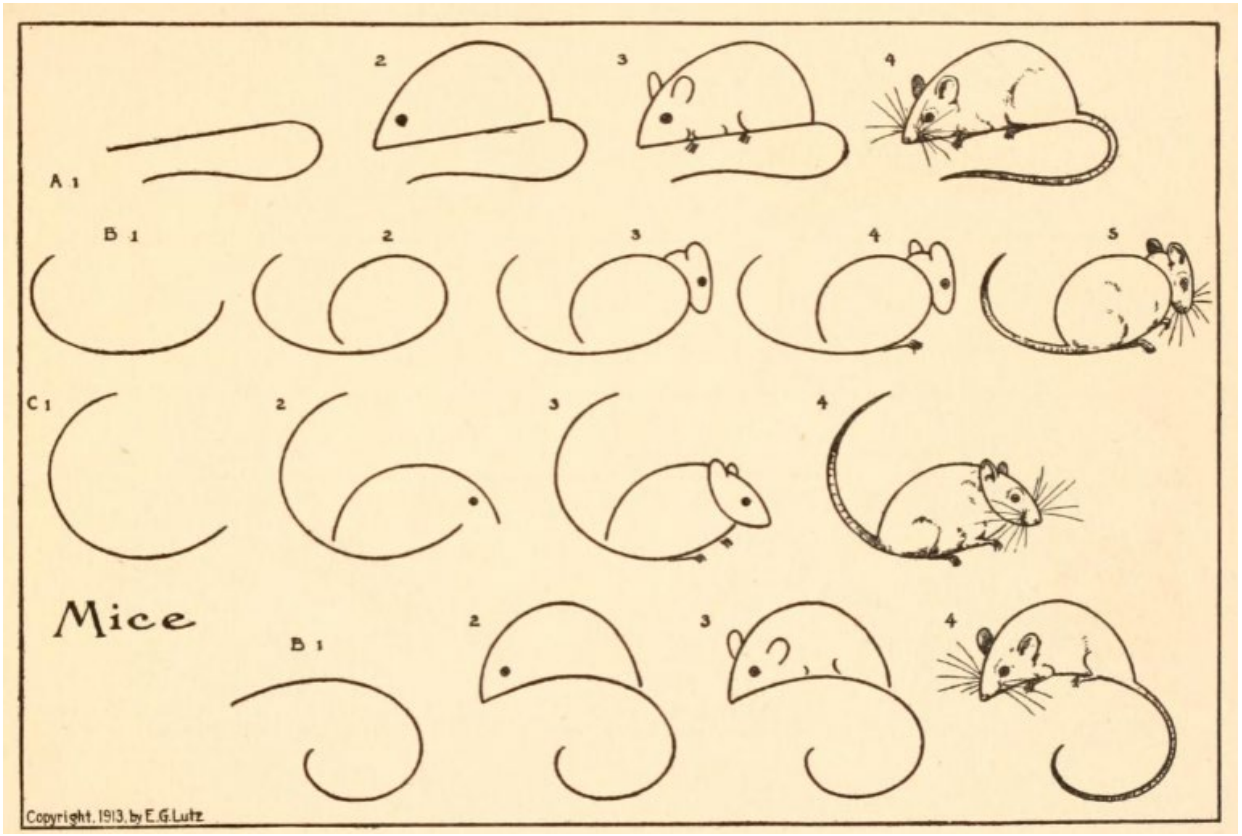
TEMA 2

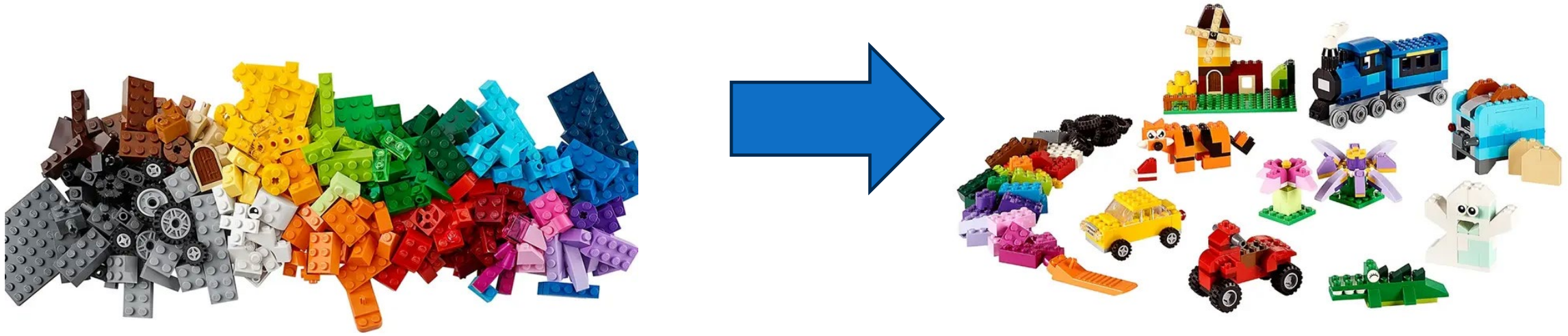
2.1 DEFINICIÓN DE CLASES Y OBJETOS

2.1 DEFINICIÓN DE CLASES Y OBJETOS



Ref: Lutz, E. G. (1913). *What to draw and how to draw it*. New York: Dodd, Mead & Company. Retrieved from https://archive.org/details/whattodrawhowtod00lutz

2.1 DEFINICIÓN DE CLASES Y OBJETOS



Ref: LEGO. (2015). *LEGO® Medium Creative Brick Box (Set 10696)*. Billund, Denmark: The LEGO Group.

2.1 DEFINICIÓN DE CLASES Y OBJETOS

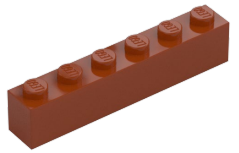
- Clase ***Bloque***
  - Color
  - Alto
  - Largo
  - Fondo

- *Objeto* ***Bloque (#123)***
  - Color : Verde
  - Alto : 3
  - Largo : 4
  - Fondo : 2

- *Objeto* ***Bloque (#126)***
  - Color : Rojo
  - Alto : 3
  - Largo : 4
  - Fondo : 2

- *Objeto* ***Bloque (#236)***
  - Color : Violeta
  - Alto : 3
  - Largo : 1
  - Fondo : 1

- *Objeto* ***Bloque (#140)***
  - Color : Azul Cielo
  - Alto : 1
  - Largo : 4
  - Fondo : 2



- *Objeto* ***Bloque (#195)***
  - Color : Marrón
  - Alto : 3
  - Largo : 6
  - Fondo : 1

- *Objeto* ***Bloque (#195)***
  - Color : Amarillo
  - Alto : 1
  - Largo : 2
  - Fondo : 2

Ref: LEGO. (2015). *LEGO® Medium Creative Brick Box (Set 10698)*. Billund, Denmark: The LEGO Group.

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Clases, Métodos y Atributos

- **Nombrado de clases**
  - El nombre de la clase tiene que comenzar con mayúscula.
  - Se declaran usando la palabra reservada **class.**
  - Pueden tener modificadores que restringen su vista en el sistema.

```
[Modificador] class <NombreClase> {


}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Clases, Métodos y Atributos

- **Ficheros**
  - Para cada clase se genera un código fuente coincidente en nombre con la clase y con extensión ".java".
  - Cuando se ejecute el código se generará una versión compilada del código fuente con extensión ".class"
  - Los ficheros/clases se agrupan en carpetas, que en java nombramos como "**paquetes**".
  - Existe un **paquete** por defecto en los proyectos llamado default (equivale a la carpeta raíz)
  - Si la clase no esta en el paque default deberá ponerse antes de declararla su ubicación con dirección absolutaderde la carpera raíz (default) separada con "." en lugar de "\" o "/"

```
package <nombreCarpeta.subcarpeta>;

[Modificadores] class <NombreClase> {

}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Clases, Métodos y Atributos

- **Atributos**
  - es cualquier porción de información de una clase y que sirve para definir las características de un objeto
  - Se definen en cualquier lugar de la implementación de la clase
    - Aunque suele ser al principio
  - Un atributo pertenece a un **tipo** que puede ser:
    - Tipo primitivo
    - Referencia a un objeto
    - Referencia a un vector (array de punteros a clases)

```
class <clase> {
    [Modificadores] <tipo> <atributo>;
    [Modificadores] <tipo> <atributo>;
        …
}
```

  - Pueden tener modificadores que restringen su vista en el sistema y su uso.
  - Si no hay inicialización explícita, se inicializan a valores por defecto, dependiendo de su tipo (0 para tipos numéricos, false para el tipo lógico, y null para referencias);

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Tipos primitivos

Type	Size	Default Value	Minimum Value	Maximum Value
byte	8 bits	0	-128	127
short	16 bits	0	-32,768	32,767
int	32 bits	0	-2,147,483,648	2,147,483,647
long	64 bits	0L	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	32 bits	0.0f	~1.4E-45	~3.4028235E38
double	64 bits	0.0d	~4.9E-324	~1.7976931348623157E308
char	16 bits	'\u0000' (null char)	'\\u0000' (0, NUL)	'\\uFFFF' (65,535, '□')
boolean	1 bit (JVM dependent)	false	false	true

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Cabecera de métodos de la clase (todos son funciones y devuelven algo)

- El nombre del método tiene que empezar con minúscula

```
[Modificadores] <tipo1> <nombreMétodo>([<tipo2> <parametro>, …])
```

- Donde el `tipo1` 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
- donde el `tipo2` puede ser:
  - 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
- donde todos ***los parámetros primitivos son pasados por valor y los objetos por referencia***.
- Pueden tener modificadores que restringen su vista en el sistema y su uso.

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Métodos de la clase

- Los métodos pueden devolver valores mediante la sentencia

```
return <expresión>;
```

El valor devuelto por el método lo determina el resultado de la evaluación de la expresión.

- Los métodos pueden no devolver valores **public void** nombreMetodo ()
  - No es necesaria la sentencia **return**

```
public void setLongitud(double longitud)
{
     this.longitud =longitud;
 }
```

```
public double longitud() {
      return maximo - minimo;
}
public boolean valido() {
      return minimo <= maximo;
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Modificador (Vistas)

- **VISTAS**: determina el ámbito donde se puede referenciar la declaración de un miembro de la clase: atributo o método.
  - **Pública (*public*)**: conocido en cualquier punto de la aplicación (antes de la declaración, después y en cualquier otro fichero)
  - **Privada (*private*)**: conocido en cualquier punto de la clase (antes y después de la declaración, pero en la implementación de la clase)
  - **Protegida (*protected*)**: conocido en cualquier punto de la carpeta donde esta definida(**paquetes**), y ***clases derivadas (las clases podrán tener versiones)***
    - ***No subcarpetas!***
  - **Default ():** conocido en cualquier punto de la carpeta donde esta definida (**paquetes**)
    - ***No subcarpetas!***
- ***La vista siempre es el primer modificador***

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Modificador (Vistas)

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier*	✓	✓	✗	✗
private	✓	✗	✗	✗

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Constructores de la Clase

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

- Hay que tener en cuenta que:
  - no se debe especificar devolver nada (ni void) porque devuelven la propia clase.
  - deben coincidir su nombre con el de la clase.
  - pueden llamar a otros métodos internos y externos.
  - Puede haber múltiples métodos de construcción de los objetos, **sobreescritura** (construcciones diferentes)

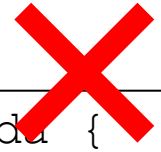
```
public class Coordenada {
   public Coordenada ();
   public Coordenada (double longitud, double latitud);
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Destructores de la Clase

- son métodos que reúnen las tareas de liberación de recursos (no destruyen) y se lanzan automáticamente en la destrucción de objetos.

- Hay que tener en cuenta que:
  - Su cabecera debe de ser: *public void finalize();*
  - deben coincidir su nombre con el de la clase;
- Existen el garbage collector (GC) en Java que se encarga de hacer este proceso automático y desde Java 9 esta ***deprecated*** y se desacnseja su uso



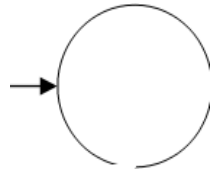
```
public class Coordenada {
   public void finalize ();
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Creación de objetos

- Los objetos se crean mediante el operador unario prefijo **new**
- El operando es una clase de objetos y devuelve la dirección de memoria donde se ha reservado el espacio para dicho objeto;

```
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.

```
new Coordenada ();
new Coordenada (10, 10))
```

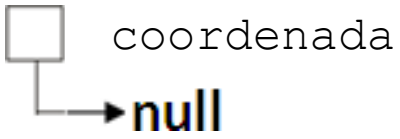
2.1 DEFINICIÓN DE CLASES Y OBJETOS

Referencia a un objeto

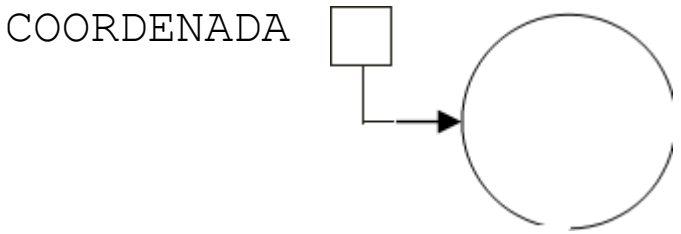
- es una variable/puntero que alberga la dirección de un objeto de una clase ***(operador "=")***.

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

```
Ej.: Coordenada coordenada;
```




```
Coordenada coordenada = new Coordenada();
```



2.1 DEFINICIÓN DE CLASES Y OBJETOS

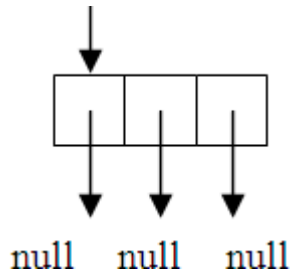
Creación de vectores de objetos

- Los vectores de objetos se crean mediante el operador unario prefijo **new**
- El 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;

```
new <Clase> [<longitud>]
```

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

```
new Coordenada [3];
```



2.1 DEFINICIÓN DE CLASES Y OBJETOS

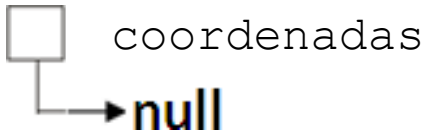
Referencia a un vector de referencias a objetos

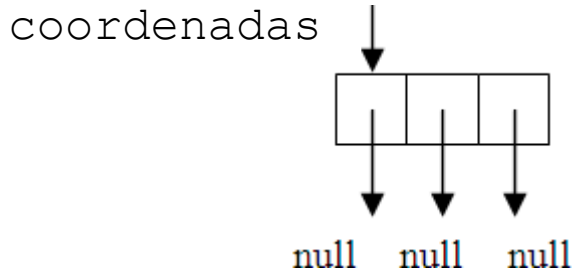
- es una variable puntero que alberga la dirección de un vector de referencias a objetos de una clase ***(operador "=")***.

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

- **final** obliga a la inicialización y fija su valor para la referencia

```
Ej.: Coordenada[] coordenadas;
```






```
Coordenada[] coordenadas = new Coordenada [3];
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

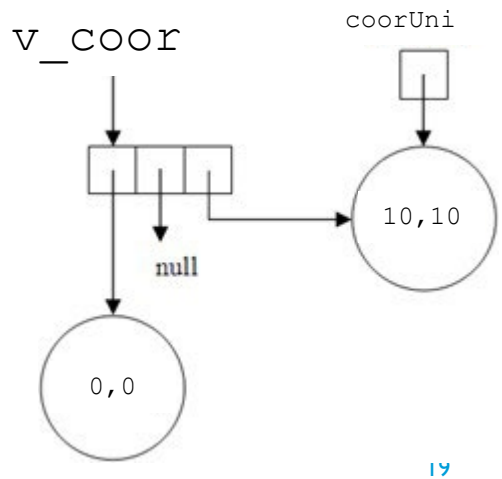
Creación de vectores de objetos (sobre objetos existentes)

- También se pueden crear con la siguiente sintaxis:

```
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;

```
Ejemplo:
   Coordenada coorUni = new Coordenada (10, 10);
   Coordenada v_coor =
new Coordenada[] {new Coordenada(), null, coorUni};
```



2.1 DEFINICIÓN DE CLASES Y OBJETOS

Operadores

- Asignación
  - <referenciaO> = <direcciónO>: asigna la dirección a la referencia siendo del mismo tipo; ✓
- Igualdad
  - <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;

```
Coordenada coordenada = new Coordenada (10, 10);
Coordenada madrid = new Coordenada (40.4167754, -3.7037902);
Coordenada capital;
capital = madrid;
boolean mismo = madrid == capital;
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Operadores

- Diferencia
  - <referenciaO> > <direcciónO>: Determina si el objeto/primitivo es superior según el orden natural;
  - <referenciaO> < <direcciónO>: Determina si el objeto/primitivo es inferior según el orden natural;
  - <referenciaO> <= <direcciónO>: Determina si el objeto/primitivo es inferior o igual según el orden natural;
  - <referenciaO> <= <direcciónO>: Determina si el objeto/primitivo es superior o igual según el orden natural;

```
boolean uno = 2 > 2;  //False
boolean dos = 2 > 1;  //True
boolean tres = 2 >= 2;  //True
boolean cuatro = 1 >= 2;  //False
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Operadores

- Acceso en vectores
  - Mediante el índice, **siempre empezando por 0**, puede sobrescribir la posición o almacenarla en una variable. Posiciones de 0 a n-1 siendo n el tamaño del vector.
  - *Si se accede a una posición invalida, por ejemplo, mayor que el tamaño salta un error de java critico, comúnmente llamados Excepcion (en este caso ArrayIndexOutOfBounds)*

```
Coordenada coordenadas[] = new Coordenada [3];
Coordenadas[1] = new Coordenada()  // Posición 1 pasa de null a objeto nuevo
Coordenadas[2] = null // colocamos un null en la posición 2
Coordenadas[3] = new Coordenada() // ArrayIndexOutOfBounds Error critico en ejecución.
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Sobrecarga de Métodos de la Clase

- varios métodos pueden tener el mismo nombre (y se consideran iguales) con las siguientes restricciones:
  - si están en la misma clase, deben diferenciarse en el número o tipo de parámetros de entrada comparados dos a dos.
  - Si cumple la primera norma también pueden diferenciarse en la salida.

```
class Coordenada {
   public boolean valido();
   public void escribir (String texto);
   public void escribir (int escala);
}
```

```
class Intervalo {
   public boolean valido();
   public int valido(int i);

}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Sobrecarga de Métodos de la Clase

Método 1	Método 2	¿Sobrecarga?	Motivo
`void m(int a)`	`void m(double a)`	✓	Cambia el tipo del parámetro.
`void m(int a)`	`void m(int a, int b)`	✓	Cambia el número de parámetros.
`void m(int a, String b)`	`void m(String b, int a)`	✓	Cambia el orden de parámetros (tipos distintos).
`int m(int a, double b)`	`double m(String s)`	✓	Diferentes parámetros de entrada y diferente retorno.
`void m(int a, int b)`	`void m(int x, int y)`	✗	Solo cambian nombres de variables.
`void m(int a)`	`int m(int a)`	✗	El tipo de retorno no cuenta (mismos parámetros).

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Acceso a atributos.

- Se realiza mediante expresiones que tienen la siguiente sintaxis:

```
<direccionO>.<atributo>
```

- Donde el atributo debe estar presente en la clase del objeto y ser accesible (modificadores);
- Los atributos se conectan con los objetos mediante un punto (.)

```
class Persona {
   public int edad;
   public Persona(int edad){…}
…
}
```

```
Persona persona = new Persona (25);
int edadget= persona.edad
persona.edad = 75
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Paso de mensajes

- Se realiza mediante expresiones que tienen la siguiente sintaxis:

```
<direccionO>.<metodo>([<expresión>,…])
```

- Donde el método (sin contemplar constructores ni destructores) debe estar presente en la clase del objeto y ser accesible (modificadores) y la lista de expresiones debe coincidir en número y tipos a la lista de parámetros del método;
- Los métodos se conectan con los objetos mediante un punto (.)

```
Coordenada madrid = new Coordenada (40.4167754, -3.7037902);
double longitud = madrid.getLongitud(); // getLongitud() en Coordenada
double latitud = madrid.getLatitud(); // getLatitud() en Coordenada
```

2.1 PROGRAMACIÓN BASADA EN OBJETOS

La variable this

- Todas las clases tienen una variable por defecto e interna que es **this.** Mientras que desde fuera de la clase un atributo o un método de un objeto debe accederse como:

```
<direccionObjeto>.<atributo>
```

- **this** es una referencia constante que guarda la dirección del objeto que recibe el mensaje correspondiente al método que se está definiendo y para uso interno dentro del objeto. se representa como:

```
public Intervalo(double minimo, double maximo) {
    this.minimo = minimo;
    this.maximo = maximo;
    this.imprimeintervalo(this.mínimo, this.máximo)
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Uso de this

- resolución de ambigüedades en la colisión de nombres 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;
- ***Puede omitirse si no hay ambigüedad***

```
public Intervalo(double minimo, double maximo) {
     this.minimo = minimo; //minimo != this.minimo
     this.maximo = maximo; //maximo != this. maximo
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Vista Privada en las Clases. Atributos y constructores y métodos

- La vista privada es muy útil para poder tener elementos internos de la clase y métodos.

```
class Persona {
  private int edad;
  private String DNI;

  public Persona(String dni) {
       this.edad = 0;
       this.DNI = dni;
  }
```

```
  private boolean revisaEdad(int e){
       if (e>=18) return true;
       else return false;
    }

  public setEdad(int edadin){
       if (this.revisaEdad(edadin))
           this.edad=edadin;
    }
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Constantes en JAVA

- En el caso de que los atributos sean inmutables, puede postergarse la inicialización al cuerpo de los constructores mediante el modificardor **final, siempre posterior del modificar de vistas** . Una vez inicializadas, ya no es posible la asignación de nuevos valores.

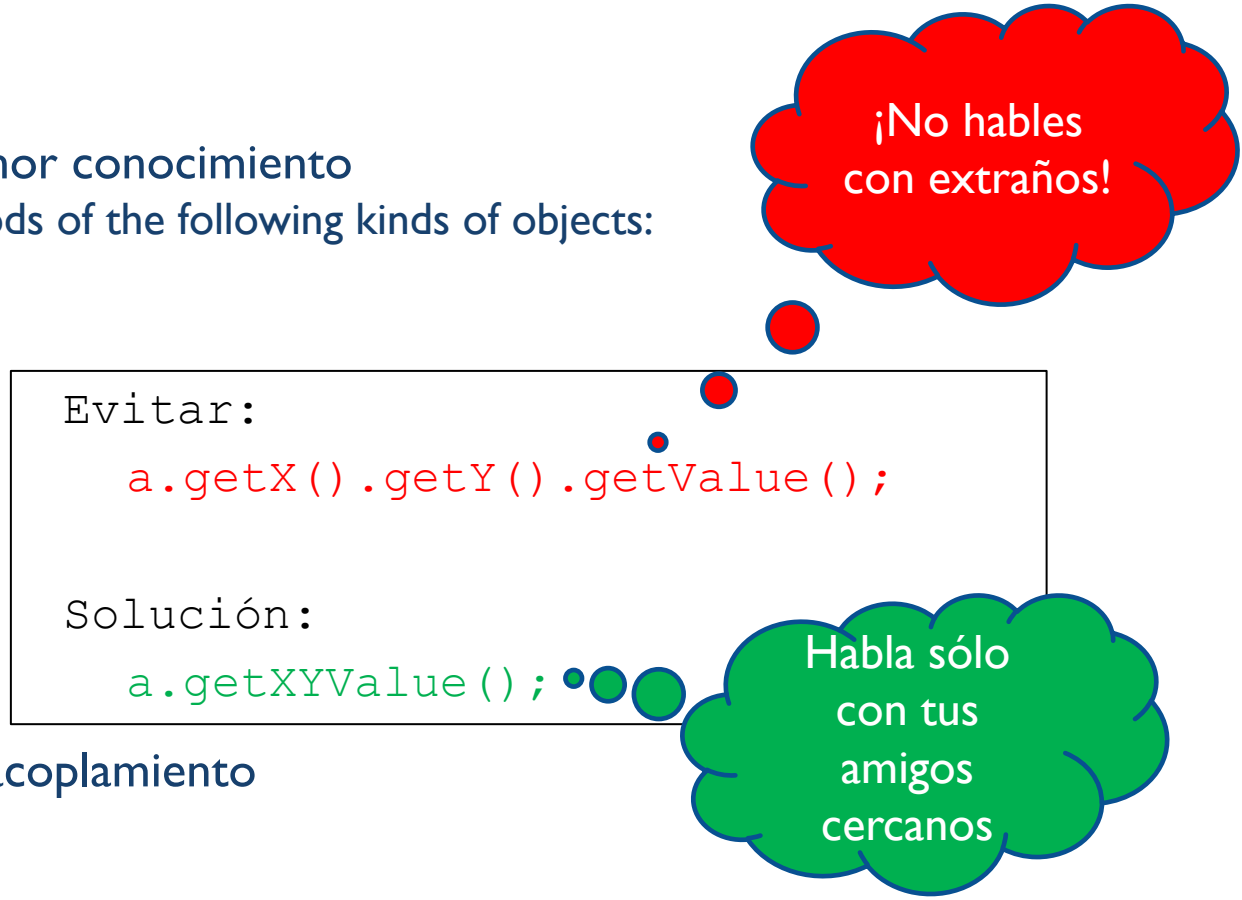
```
class Persona {
    private byte edad;
    private final String DNI;

    public Persona(String dni) {
        edad = 0;
        this.DNI = dni;
    }
    …
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Llamadas encadenadas

- Ley estricta de Demeter (LoD) - Principio del menor conocimiento
  - "A method of an object should invoke only the methods of the following kinds of objects:
    - itself
    - its parameters
    - any objects it creates/instantiates
    - its direct component objects"
- Dentro del cuerpo del método se tiene acceso a:
  - Los atributos,
  - Los parámetros del método y
  - Las declaraciones locales
- Objetivo: mejorar el encapsulamiento y reducir el acoplamiento



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

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
- Recursivamente se 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.

Desencadenamiento de Mensajes

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

```
public Intervalo() {
      this(0, 0);
}
```

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

```
private Intervalo copiar() {
      return new Intervalo (this);
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Miembros de Clase. Atributos, métodos y código estático

- **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

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Miembros de Clase. Atributos, métodos y código estático

- **Atributos estáticos**:
  - caracterizados por el modificador **static** siempre posterior al modificador de 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>`
- **Métodos estáticos (static binding):**
  - caracterizados por la palabra reservada **static** siempre posterior al modificador de visibilidad;
  - no se permite el acceso a **this** ni a los atributos de instancia; ni la sobreescritura (override).
  - ***se permite el acceso a los atributos estáticos;***
  - la notación sintáctica para el lanzamiento del mensaje:

```
<Clase>.<metodoEstatico> (<argumento>, … );
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Miembros de Clase. Atributos, métodos y código estático

- **Atributos estáticos (usos)**:
  - Constantes relativas al código de la clase y no la instancia (mensajes, números).
    - ***Nos permiten evitar "NUMEROS MAGICOS", "TEXTOS REPETIDOS" en el sistema***
  - Permiten manejar atributos relativos a todos los objetos en simultaneo. Texto variable para todos o el conteo (índices y ids)
- **Métodos estáticos (usos):**
  - Permiten manejar los atributos staticos
  - Permiten operaciones generales independientes del objeto y sus atributos. (como una conexión a un fichero o fuente externa)
- **Tanto los métodos como los atributos se ponen en MAYUSCULAS**

Miembros de Clase. Atributos, métodos y código estático

- Ejemplo (I)

```
class Fecha {
    private int dia;
    private int mes;
    private int anio;
    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.anio == fecha.anio)
            if (this.mes == fecha.mes)
                resultado = this.dia > fecha.dia;
            else
                resultado = this.mes > fecha.mes;
        else
            resultado = this.anio > fecha.anio;
        return resultado;
    }
```

Miembros de Clase. Atributos, métodos y código estático

- Ejemplo (II)

```
public static boolean bisiesto(int anio) {
    return ((anio % 4 == 0) && ((anio % 100 != 0) || (anio % 400 == 0)))
}
public boolean enBisiesto() {
    return Fecha.bisiesto(anio);
}
public int diasTranscurridosAnio() {
    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;
}
```

Miembros de Clase. Atributos, métodos y código estático

- Ejemplo (III)

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

Miembros de Clase. Atributos, métodos y código estático

- 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 cuando se carga la clase;
  - Sintaxis:

```
<atributoEstatico>
static {
    <sentencia/declaración>
            …
    <sentencia/declaración>
}
```

2.1 DEFINICIÓN DE CLASES Y OBJETOS

Miembros de Clase. Atributos, métodos y código estático

```
public class Configuracion {
    public static Map<String, String> parametros;
    static {
        parametros = new HashMap<>();
        parametros.put("BD", "localhost");
        parametros.put("PORT", "3306");
        System.out.println("⚡ Clase Configuracion cargada e inicializada");
    }
}
public class Main {
    public static void main(String[] args) {
        System.out.println(Configuracion.parametros.get("BD"));
    }
}
```

2.1 PROGRAMACIÓN BASADA EN OBJETOS

Miembros de Clase. Atributos, métodos y código estático

- 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
- CLASE:Math

EJERCICIOS BÁSICOS

2.1 DEFINICIÓN DE CLASES Y OBJETOS

CLASES Y OBJETOS BÁSICOS

Ejercicio:

1. Declara una clase llamada Coche.
2. Añade tres atributos públicos: marca (String), modelo (String) y año (int).
3. Crea dos objetos de tipo Coche en un método main.
4. Imprime en pantalla los valores de sus atributos.

CONSTRUCTORES

Ejercicio:

1. Modifica la clase Coche para que sus atributos sean privados.
2. Implementa dos constructores:
    1. Uno vacío que inicialice los atributos con valores por defecto.
    2. Uno con parámetros (marca, modelo, año).
3. Crea un objeto con cada constructor y muestra sus valores por pantalla.

MÉTODOS

Ejercicio:

1. Añade en la clase Coche un método público arrancar() que imprima: "El coche ha arrancado".
2. Añade un método público edadDelCoche(int añoActual) que devuelva la edad del coche.
3. Crea un objeto y llama a ambos métodos desde el main.

MODIFICADORES DE VISIBILIDAD

Ejercicio:

1. Mantén los atributos de Coche como privados.
2. Implementa getters y setters para cada atributo.
3. Desde el main, cambia la marca de un coche usando el setter e imprime el resultado usando el getter.
4. Reflexiona: ¿por qué es más seguro este enfoque que tener atributos públicos?

ATRIBUTOS Y MÉTODOS ESTÁTICOS

Ejercicio:

1. Añade en la clase Coche el atributo estático NUMERO_RUEDAS = 4.
2. Añade un método estático descripcionGeneral() que imprima: "Todos los coches tienen 4 ruedas" utilizando el atributo estático NUMERO_RUEDAS.
3. Llama a este método desde el main sin crear un objeto.

CONSTANTES CON 'FINAL'

Ejercicio:

1. Crea una clase Fecha.
2. Declara en ella el array constante:

   ```
   private static final int[] DIAS_MESES = new int[]{31,28,31,30,31,30,31,31,30,31,30,31};
   ```

3. Añade un método diasDelMes(int mes) que devuelva el número de días del mes recibido.
4. Desde el main, prueba con al menos dos meses distintos.

SOBRECARGA DE MÉTODOS

Ejercicio:

1. Crea una clase Calculadora.
2. Implementa dos métodos llamados suma:
    1. int suma(int a, int b)
    2. double suma(double a, double b)
3. Desde el main, llama a ambos métodos y muestra el resultado.
4. Analiza cuál de los dos métodos se ejecuta en cada llamada y por qué.

VECTORES DE OBJETOS

Ejercicio:

1. Declara un array de 3 coches:
   Coche[] coches = new Coche[3];
2. Inicializa cada posición del array con un objeto distinto.
3. Recorre el array con un bucle e imprime la marca de cada coche.