Parte I: Programación en un lenguaje orientado a objetos

1. Introducción a los lenguajes de programación
2. *Datos y expresiones*
   - Tipos primitivos. Variables y constantes. Operadores y expresiones. Conversión de tipos. Uso de funciones matemáticas. Declaración de objetos. Strings. Composición de objetos. Atributos y métodos estáticos.
3. Estructuras algorítmicas
4. Datos compuestos
5. Tratamiento de errores
6. Entrada/salida
7. Herencia y polimorfismo

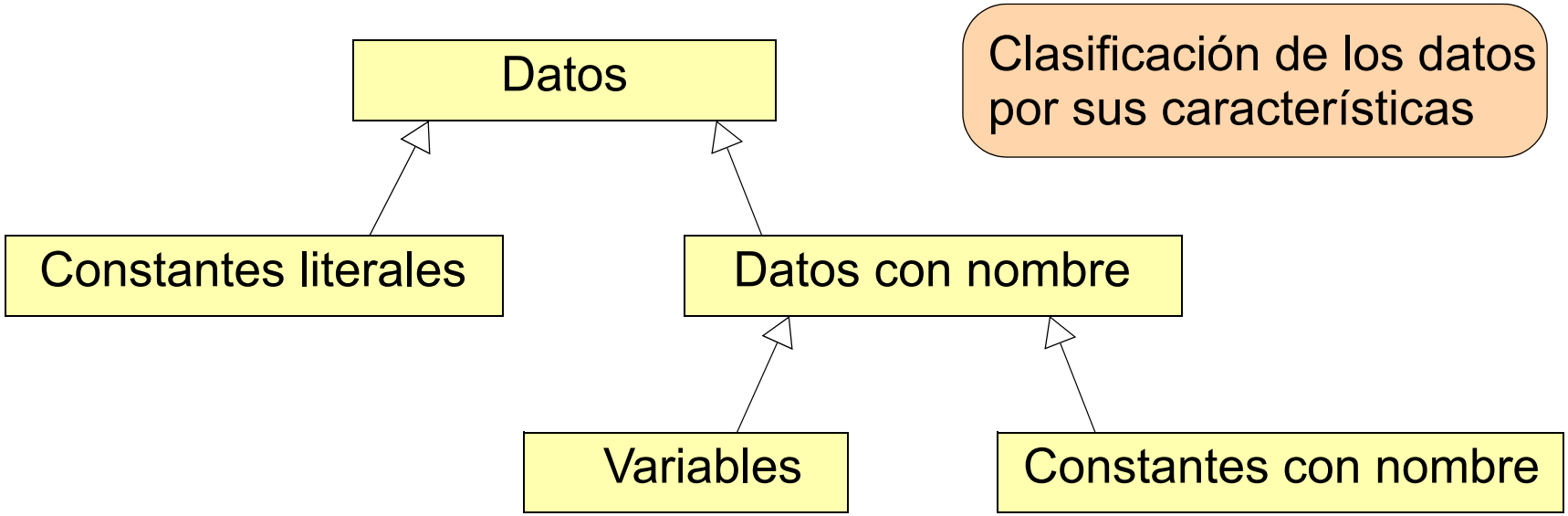
2.1. Tipos primitivos (predefinidos)

tipo	descripción	rango de valores
`boolean`	valor lógico	`true o false`
`char`	carácter unicode (16 bits)	`los caracteres internacionales`
`byte`	entero de 8 bits con signo	`-128..127`
`short`	entero de 16 bits con signo	`-32768..32767`
`int`	entero de 32 bits con signo	`-2.147.483.648.. 2.147.483.647`
`long`	entero de 64 bits con signo	`aprox. 9.0*10`^{18}
`float`	nº real de 32 bits	`unos 6 dígitos`
`double`	nº real de 64 bits	`unos 15 dígitos`

2.2. Datos variables y constantes

Los datos son de un ***tipo***, se almacenan en la ***memoria*** del computador y se pueden usar de dos formas:

- Poniendo directamente su valor: ***constantes literales***
- Usando un ***nombre*** para referirse al dato: ***datos con nombre***
  - *variables* si su valor puede variar
  - *constantes con nombre*, en caso contrario



Constantes literales

tipo	descripción	ejemplos
`boolean`	sólo uno de estos dos valores	`true false`
`char`	el carácter entre comillas simples los especiales van con `\`	`'a' 'A' '.' '6'` `'\n' '\\' '\'' '\"'`

`byte, short e int`	el número entero en decimal
	el número en octal
	el número en hexadecimal
	el número en binario
		`12  -37 1_000_000`
		`037`
		`0x2A`
		`0b1011`

`long`	el número entero con `l` o `L`	`12L  12l`
`double`	el número con parte fraccionaria notación exponencial con `e` o `E`	`0.0  13.56  -12.0` `6.023E23 1.0e-6`
`float`	como el `double`, con una `f`	`18.0f  1.23E4f`
`String`	objetos literales de la clase String, que representa texto	`"esto es un texto"`

Datos con nombre

Según dónde declaremos el nombre tendremos:

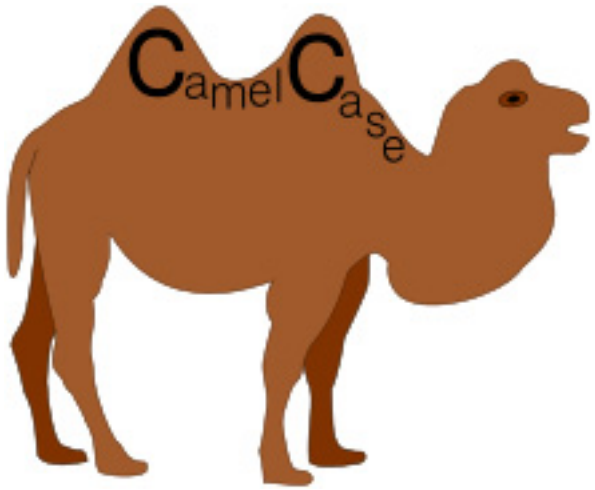
Clasificación de los datos por su rol

Rol	Lugar donde se declara	Uso	Vida
atributo	clase	datos que forman parte de los objetos de la clase	*Larga*: siempre que se pueda usar
argumento o parámetro	paréntesis del encabezamiento de un método	datos que el método necesita del exterior	*Corta*: solo mientras el método se ejecuta
variable local	contenido de un método	datos que un método calcula y usa temporalmente	*Corta*: solo mientras el método o bloque se ejecuta

Nombres o identificadores

Deben seguir estas reglas

- deben comenzar por una letra, y luego letras, dígitos, y '_'
- influyen mayúsculas y minúsculas
- estilo:
  - clases empiezan con mayúsculas
  - objetos, métodos y datos con minúsculas
  - las palabras se separan con mayúsculas (estilo "*espalda de camello*")



Atributos

Representan un dato con nombre, de un tipo determinado, declarado como *parte un objeto*

- su vida es larga: está disponible mientras se quiera/pueda usar

Declaración de atributos, dentro de una clase:

```
private tipo nombre;
private tipo nombre = valor;
// Tres variables del mismo tipo
private tipo nombre1, nombre2, nombre3;
```

La palabra **`private`** es un ***descriptor*** opcional

- Indica que el dato sólo se puede usar dentro de la clase
- Es habitual que los atributos sean privados

El tipo puede ser uno predefinido (int, double, ...) o una *clase*

Ejemplos de atributos

Atributos

```
private int lo = 1;
private int hi = 2;
private double x;
private Alumno al; // objeto de la clase Alumno
```

Los atributos a los que no se da valor inicial explícito se inicializan automáticamente a:

- `0` si es un número
- `false` si es un booleano
- `null` si es un objeto

Variables locales

Representan un dato con nombre, de un tipo determinado, declarado *dentro de un método*

- se destruye al finalizar el método o el bloque donde aparece

Declaración de variables locales, dentro de un método:

```
tipo nombre;
tipo nombre = valor;
// Dos variables del mismo tipo
tipo nombre1, nombre2;
```

Ejemplos de variables locales

Variables locales

```
int x1,x2;
double y=1.0e6;
Alumno pepe;
```

Si no se pone valor inicial, el valor es indefinido

- es un error usarlo antes de asignar un valor

En general no declarar las variables locales hasta que haya que usarlas

Constantes con nombre

Los atributos y variables locales se pueden definir como constantes: su valor no se puede cambiar

- declaración: atributo o variable local con descriptor `final`
  - `final` indica que el dato ya no puede cambiar de valor
- constantes locales con valor "en blanco": se les puede asignar el valor una vez
- no es necesario definirlas como `private`, ya que nadie puede "estropearlas"

Ejemplos

```
final double VOLTAJE_MAX = 5.0;
final int maxNum = 50;
final double factorEscala;
```

A veces se escriben con todas las letras mayúsculas

Argumentos o parámetros

Representan un dato con nombre, de un tipo determinado, declarado dentro de los paréntesis en el encabezamiento de un método

- se usa para pasar datos al método desde el exterior
- se les da valor al invocar el método

Ejemplo de declaración de argumentos:

```
public tipo_retornado metodo1
    (tipo1 nombre1,
     tipo2 nombre2)
{
    ...
}
```

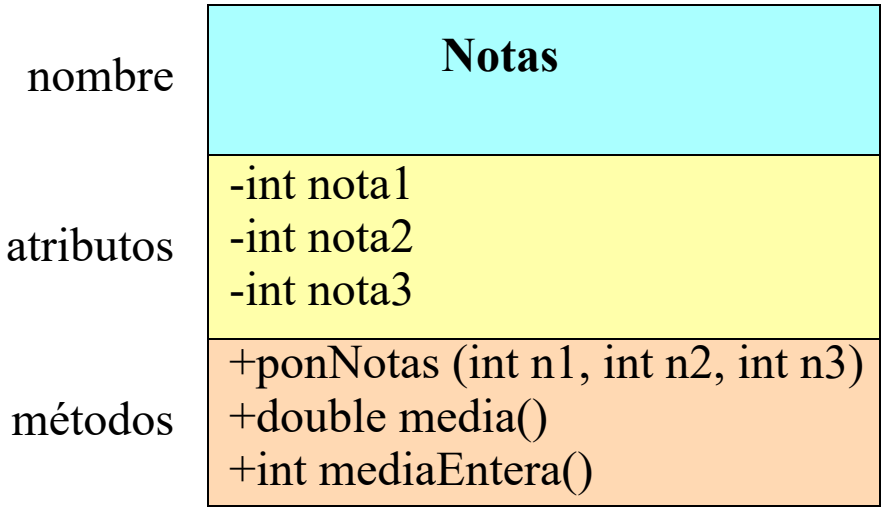
Ejemplo de programa con datos

Cálculo de la media de tres notas

La clase tendrá:

- tres atributos enteros (las tres notas)
- método para cambiar las notas
  - tres parámetros enteros (nuevos valores de las notas)
  - no retorna nada
- método para hallar la media entera: retorna la media
- método para hallar la media real: retorna la media

Diagrama de la clase



- identifica elementos privados

+ identifica elementos públicos

Ejemplo (cont)

```
/**
 * Clase que contiene tres notas de un alumno
 */
public class Notas {

    private int nota1, nota2, nota3;

    /**
     * Pone los valores de las tres notas
     */
    public void ponNotas (int n1, int n2, int n3) {
        nota1=n1;
        nota2=n2;
        nota3=n3;
    }
```

Ejemplo (cont)

```
    /**
     * Calcula la media real
     */
    public double media() {
        return (nota1+nota2+nota3)/3.0;
    }

   /**
    * Calcula la media entera
    */
    public int mediaEntera(){
        return (nota1+nota2+nota3)/3;
    }
}
```

Comentarios sobre el ejemplo

- argumentos de un método: datos del exterior que el método necesita
- valor de retorno de un método: respuesta
- operador de asignación: "="
- operador de suma: "+"
- uso de paréntesis
- expresiones reales y enteras: conversiones automáticas
  - ¡probar el cálculo como `(nota1+nota2+nota3)*(1/3)`!

Uso de un constructor

El ejemplo pone de manifiesto la necesidad de dar valor a los atributos

- hemos creado `ponNotas` para ello
- es una necesidad frecuente

Constructor

- es un ***método especial***, usado al crear el objeto (con `new`) para dar valor a los atributos
- ***sintaxis***: método de nombre igual a la clase y en el que no se pone lo que retorna
- ***ventaja***: al crear el objeto, el constructor nos obliga a poner las notas

Ejemplo con constructor

```
public class Notas {
    private int nota1, nota2, nota3;

    /** Pone los valores de las tres notas */
    public void ponNotas (int n1, int n2, int n3) {
        nota1=n1;
        nota2=n2;
        nota3=n3;
    }

    /** Constructor que pone las tres notas */
    public Notas (int n1, int n2, int n3) {
        nota1=n1;
        nota2=n2;
        nota3=n3;
    }
}
```

Notación **this**

En ocasiones los argumentos y los atributos se llaman igual

Para distinguirlos:

- **this**.nombre es el atributo
- nombre "a secas" es el argumento

```
    /** Constructor que pone las tres notas */
    public Notas (int nota1, int nota2, int nota3) {
        this.nota1=nota1;
        this.nota2=nota2;
        this.nota3=nota3;
    }
```

this representa el objeto actual

2.3. Operadores y expresiones

Las expresiones permiten transformar datos para obtener un resultado

Se construyen con operadores y operandos

Operandos:

- constantes literales
- datos simples (atributos, variables, o argumentos de tipos simples)
- funciones (métodos que retornan un valor de un tipo simple)

Los operadores más usuales

Indican la operación a realizar en una expresión

- dependen del tipo de dato
- tienen unas reglas de precedencia
- el paréntesis altera la precedencia

Operadores aritméticos: operan con números, y dan como resultado números del mismo tipo

+	-	*	/	%	++	- -
suma	resta	multipli-cación	división	módulo	incre-mento	decre-mento

- ver detalles de mezcla y conversión de tipos más adelante

Los operadores más usuales (cont.)

Operadores relacionales: comparan dos números o caracteres y dan un resultado lógico

>	>=	<	<=	==	!=
mayor	mayor o igual	menor	menor o igual	igual	distinto

Operadores lógicos: operan con valores lógicos y dan otro valor lógico

&	|	!	&&	||
y	o	negación	y "condicional"	o "condicional"

Los operadores más usuales (cont.)

Operador de asignación simple: copia un valor en una variable

`=`

Otros operadores de asignación: opera con los operandos izquierdo y derecho y copia el resultado en el izquierdo

`+=`	`-=`	`*=`	`/=`	`%=`	`...`

Operador de concatenación: retorna la concatenación del operando izquierdo (`String`) con la conversión a `String` del operando derecho

`+`

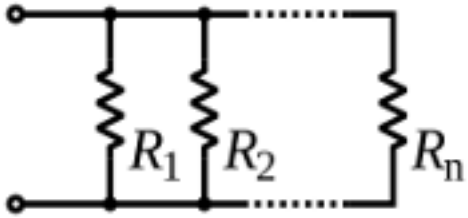
Los operadores por precedencia, de mayor a menor

++ -- ~ ! (tipo)
* / %
+ -
>> >>> <<
> >= < <=
== !=
&
^
|
&&
||
= op=

Ejemplo

Cálculo de valores resistivos en paralelo

$$Requivalente = \frac{1}{\frac{1}{r1} + \frac{1}{r2} + \frac{1}{r3}}$$



Diseño de la clase:

- atributos: las resistencias
- constructor que les da valor inicial
- un método que devuelve la resistencia equivalente

Resistencias
-double r1 -double r2 -double r3
+Resistencias(double r1, double r2, double r3) +double calculaResEquiv()

Ejemplo (cont.)

```
/**
 * Clase que contiene tres resistencias
 *  y operaciones para trabajar con ellas
 */
public class Resistencias {
    private double r1, r2, r3; //Kohms

    /**
     * Constructor al que se le pasan las
     *  resistencias en Kohms*/
    public Resistencias(double r1,
            double r2, double r3)
    {
        this.r1=r1;
        this.r2=r2;
        this.r3=r3;
    }
```

Ejemplo (cont.)

```
    /**
     * Calcula la resistencia equivalente
     *  a 3 resistencias en paralelo
     *  Retorna Kohms
     */
    public double calculaResEquiv() {
        return 1.0/(1.0/r1+1.0/r2+1.0/r3);
    }
}
```

Ejemplo: Uso desde un programa

Ahora podemos hacer una clase que usa un objeto de la clase anterior, para poder usar su operación:

```
/**
 * Programa que calcula resistencias en paralelo
 */
public class ResistenciasEnParalelo {
  public static void main(String[] args) {
      double r1=3.5; // Kohms
      double r2=5.6; // Kohms
      double r3=8.3; // Kohms
      double reff;   // Kohms
      Resistencias res=new Resistencias(r1,r2,r3);

      // alternativamente se podría hacer
      // Resistencias res=
      //      new Resistencias(3.5,5.6,8.3);
```

Ejemplo: Uso desde un programa (cont.)

```
        reff=res.calculaResEquiv();
        System.out.println
           ("Resist. en paralelo: "+r1+", "+r2+", "+
             r3+ " Kohm");
        System.out.println
           ("Resistencia efectiva = "+reff+" Kohm");
   }
}
```

A observar en este ejemplo:

- creación de un objeto de la clase `Resistencias`: declaración+`new`
- creación de variables locales
- diferencias entre crear una variable y un objeto
- uso de un método
- concatenación
- uso de la notación `this`, para referirse al objeto actual
  - permite, además, distinguir entre parámetros y atributos del mismo nombre
- comentarios de documentación con:
  - explicación general
  - explicación de los parámetros con sus unidades
  - explicación del valor retornado con sus unidades

2.4. Conversión de tipos

Compatibilidad de tipos: Java es un lenguaje con tipificación estricta.

- Los números son compatibles hacia "arriba" (promoción automática de tipos): es decir, el tipo destino tiene mayor cabida que el origen
- No son compatibles cosas de distinta naturaleza (p.e., números con `String` o `boolean`)
- También hay conversión automática al almacenar un literal entero en un `byte` o `short` (pero no en expresiones no literales)

Conversiones explícitas de tipos (cast)

Sintaxis

```
(tipo) exp
```

Hay que usarlas con precaución, porque hay truncamiento de la parte fraccionaria, y/o operación módulo. Por ejemplo:

```
int i;
double d=3.4;
i= (int) d; // valor=3
byte b;
int j=1000;
b=(byte) j; // valor=-24
```

2.5. Uso de funciones matemáticas

La clase `Math` contiene constantes y métodos estáticos. Las constantes son `E` y `PI`. Los métodos operan (casi todos) sobre datos de tipo `double` e incluyen:

`sin(a), cos(a), tan(a)`	funciones trigonométricas (radianes)

`asin(v), acos(v),`
`atan(v),`
`atan2(y,x)`
	trigonométricas inversas
	de $-\pi/2$ a $\pi/2$
	arco tangente de y/x entre $-\pi$ y π

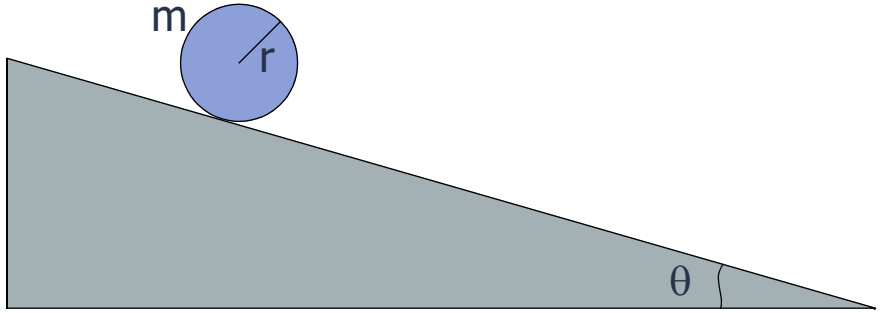
`sinh(a), cosh(a), tanh(a)`	funciones trigonométricas hiperbólicas
`exp(x), log(x), log10(x)`	e^x, logaritmo neperiano y logaritmo decimal
`pow(a,b)`	a^b
`sqrt(x)`	raíz cuadrada
`ceil(x), floor(x)`	redondeo por arriba y por abajo (retorna `double`)

Funciones matemáticas (cont.)

`rint(x)`	redondeo al entero más cercano (retorna `double`)
`round(x)`	redondeo al entero más cercano (retorna `long`)
`abs(x)`	valor absoluto: para cualquier valor numérico
`signum(x)`	retorna -1 si x es negativo, 0 si vale cero y +1 si es positivo
`max(x,y), min(x,y)`	máximo y mínimo: ambas para cualquier valor numérico
`random()`	número aleatorio entre 0 y 1
`toDegrees(a), toRadians(a)`	conversiones de ángulos

Ejemplo

Cálculo de movimientos de una esfera que rueda sobre un plano inclinado



Datos del ejemplo

I	momento de inercia	v	velocidad
m	masa	t	tiempo
r	radio	x	distancia
θ	ángulo del plano inclinado	E_{tras}	energía cinética de traslación
g	gravedad	E_{rot}	energía cinética de rotación
a	aceleración	ω	velocidad angular ($v=\omega r$)

Ecuaciones del movimiento

Las ecuaciones son (http://www.sc.ehu.es/sbweb/fisica/):

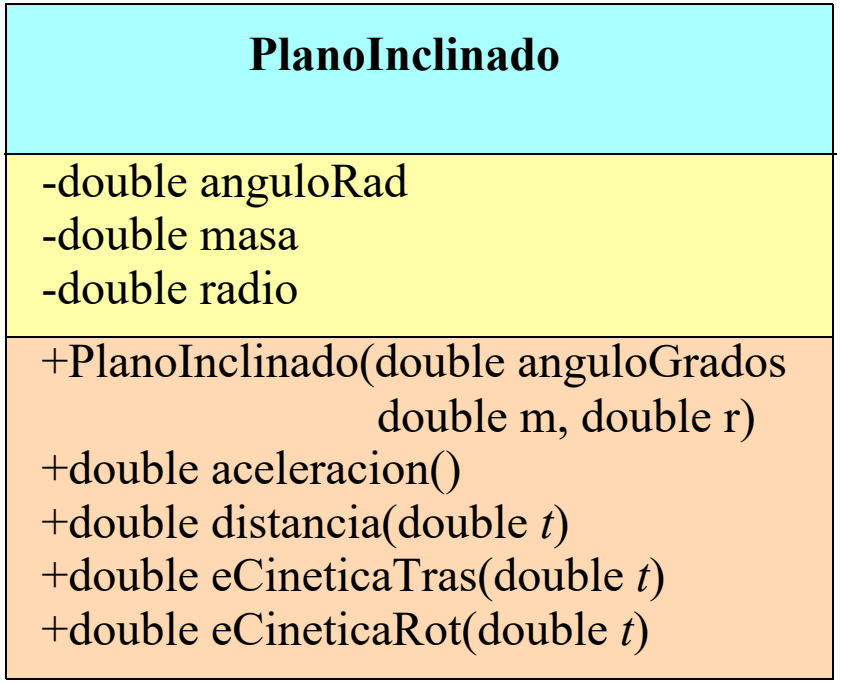
$$I = \frac{2}{5}mr^2 \qquad a = \frac{g\sin\theta}{\left(1+\frac{I}{mr^2}\right)} \qquad v = at = \omega r$$

$$x = \frac{1}{2}at^2 \qquad E_{tras} = \frac{1}{2}mv^2 \qquad E_{rot} = \frac{1}{2}I\omega^2$$

Ejemplo: Diagrama de la clase

Diseño de la clase

- atributos: los datos del plano inclinado y la esfera
- la gravedad es una constante
- métodos
  - constructor que les da valor inicial
  - cálculo de la aceleración
  - cálculo de la distancia
  - cálculo de la energía cinética de traslación
  - cálculo de la energía cinética de rotación



Ejemplo (cont.)

```
/**
 * Clase que contiene los datos de una
 * esfera que rueda por un plano inclinado
 * sin deslizar */
public class PlanoInclinado
{
    private double anguloRad; // radianes
    private double masa;      // Kg
    private double radio;     // metros

    private final double g=9.8; //metros/seg2

    /**
     * Constructor al que se le pasan el ángulo
     * en grados, la masa del objeto en Kg, y el
     * radio en metros
     */
```

Ejemplo (cont.)

```
public PlanoInclinado
    (double anguloGrados, double m, double r)
{
    anguloRad=Math.toRadians(anguloGrados);
    masa=m;
    radio=r;
}
/**
 * Calcula la aceleración lineal del objeto (m/s)
 */
public double aceleracion() {
    double momentoInercia=
      2.0*masa*radio*radio/5.0;
    return g*Math.sin(anguloRad)/
     (1+momentoInercia/(masa*radio*radio));
}
```

Ejemplo (cont.)

```
/**
 * Calcula la distancia recorrida por el
 * objeto en t segundos
 */
public double distancia (double t)
{
    return aceleracion()*t*t/2.0;
}
/**
 * Calcula la energía cinética de traslación
 * del objeto transcurridos t segundos (Jul)
 */
public double eCineticaTras (double t)
{
    double vel = aceleracion()*t;
    return masa*vel*vel/2.0;
}
```

Ejemplo (cont.)

```
    /**
     * Calcula la energía cinética de rotación del
     * objeto transcurridos t segundos (Jul)
     */
     public double eCineticaRot (double t)
     {
         double momentoInercia=
            2.0*masa*radio*radio/5.0;
         double velAngular = aceleracion()*t/radio;
         return momentoInercia*velAngular*
            velAngular/2.0;
     }
}
```

Observar que el cálculo del momento de inercia aparece repetido

- deberíamos implementarlo con un método

Ejemplo: uso de la clase

```
/**
 * Programa principal que muestra para una esfera en
 * un plano inclinado la distancia y energías a los
 * cuatro segundos */
public class CalculaMovimiento {

  public static void main(String[] args) {
    // Crear el sistema plano-esfera
    // Ángulo 30 grados, esfera de 1.5 Kg y r=0.2 m
    PlanoInclinado p=
        new PlanoInclinado(30.0,1.5,0.2);
```

Ejemplo: uso de la clase

```
        // Mostrar resultados

        System.out.println
           ("Dist. a los 4 seg: "+
                p.distancia(4.0)+" m");
        System.out.println
           ("E. C. tras. a 4 seg: "+
                p.eCineticaTras(4.0)+" J");
        System.out.println
           ("E. C. rot a 4 seg: "+
                p.eCineticaRot(4.0)+" J");
    }
}
```

Funciones de tiempo

La clase `LocalDate` (paquete `time`) tiene métodos para obtener la fecha y hora

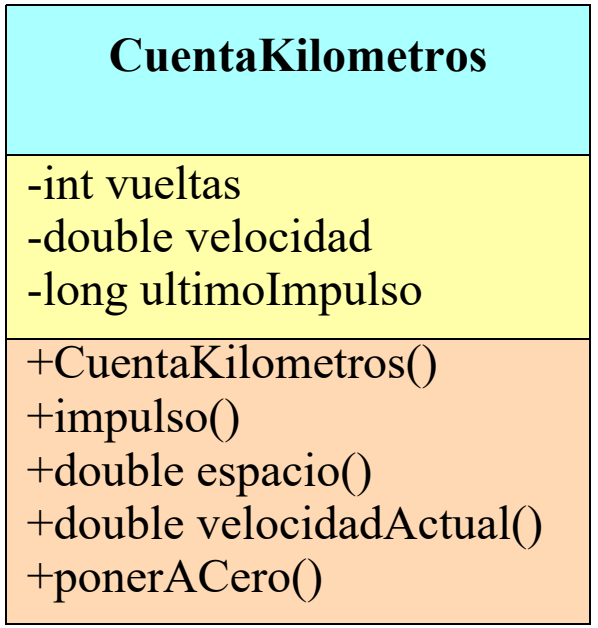
- disponible desde Java 8

Una forma sencilla de obtener medidas relativas de tiempo es con el método:

`System.currentTimeMillis()`

- retorna un `long` con el tiempo en milisegundos, medido desde el inicio de 1970

Ejemplo: Cuentakilómetros de bici



Ejemplo (cont.)

```
/**
 * Implementa funciones de un cuantakilómetros. La rueda
 * invoca al método impulso() a cada vuelta.
 */
public class CuentaKilometros
{
    /**
     * Circunferencia de la rueda en metros
     */
    public static final double CIRCUNFERENCIA=2.114;

    // Número de vueltas
    private int vueltas;
    private long ultimoImpulso; // milisegundos
    private double velocidad;   // m/s

    /**
     * Constructor que pone las vueltas y velocidad a cero,
     * y el tiempo del último impulso a la hora actual
     */
```

Ejemplo (cont.)

```
public CuentaKilometros()
{
    vueltas = 0;
    velocidad=0.0;
    ultimoImpulso=System.currentTimeMillis();
}

/**
 * Este método se invoca cuando la rueda ha dado una vuelta
 * Incrementa las vueltas y calcula la velocidad actual
 */
public void impulso()
{
    vueltas++;
    long ahora=System.currentTimeMillis();
    velocidad =
       CIRCUNFERENCIA/((ahora-ultimoImpulso)*1000.0);
    // actualiza ultimoImpulso para calcular la velocidad
    // en el proximo impulso
    ultimoImpulso=ahora;
}
```

Ejemplo (cont.)

```
/**
 * Retorna el espacio recorrido, en kilómetros
 */
public double espacio()
{
    return vueltas*CIRCUNFERENCIA/1000.0;
}

/**
 * Retorna la velocidad actual en Km/h
 */
public double velocidadActual()
{
    // multiplico por 3.6 para pasar m/s a Km/h
    return velocidad*3.6;
}
```

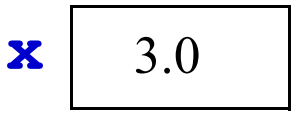
Ejemplo (cont.)

```
    /**
     * Poner a cero las vueltas y velocidad
     */
    public void ponerACero() {
        vueltas = 0;
        velocidad=0.0;
        ultimoImpulso=System.currentTimeMillis();
    }
}
```

2.6. Declaración de objetos

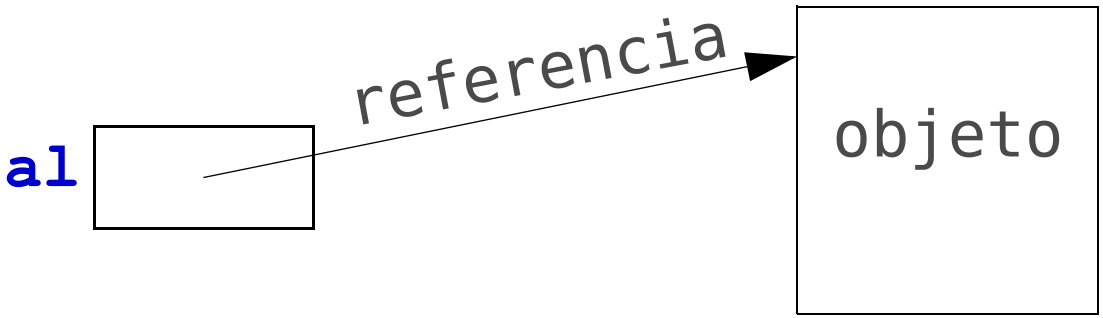
Cuando se declara una variable, ésta contiene directamente el valor del dato almacenado

```
double x=3.0;
```



Cuando se declara un objeto de una determinada clase, se hace siempre a través de una referencia

```
Alumno al=new Alumno();
```



Declaración de objetos (cont.)

Declaración de una clase que sirve como ejemplo:

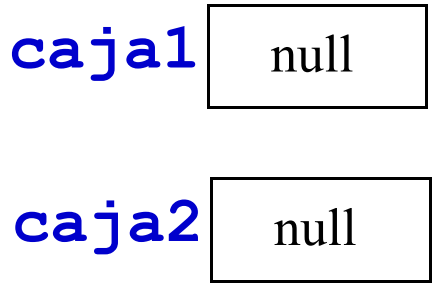
```
/**
 * Contiene las medidas de una caja
 */
public class Caja {
    private double ancho, largo, alto;

    public Caja (double ancho,
        double largo, double alto)
    {
        this.ancho=ancho;
        this.largo=largo;
        this.alto=alto;
    }
    public double volumen() {...}
    // otros métodos ...
}
```

Declaración de objetos (cont.)

Declaración de objetos de esta clase:

```
Caja caja1;
Caja caja2;
```



Las variables `caja1` y `caja2` son referencias a cajas. Inicialmente no se refieren a ninguna caja

- ello se representa con un valor llamado `null`

Invocar un método con una referencia nula es un error

```
double v=caja1.volumen();
// error: lanza NullPointerException
```

Declaración de objetos (cont.)

Para que `caja1` se refiera a un objeto de la clase `Caja`, éste debe crearse:

```
caja1 = new Caja(2.3,1.0,3.2);
caja2 = caja1; //copia la referencia
```

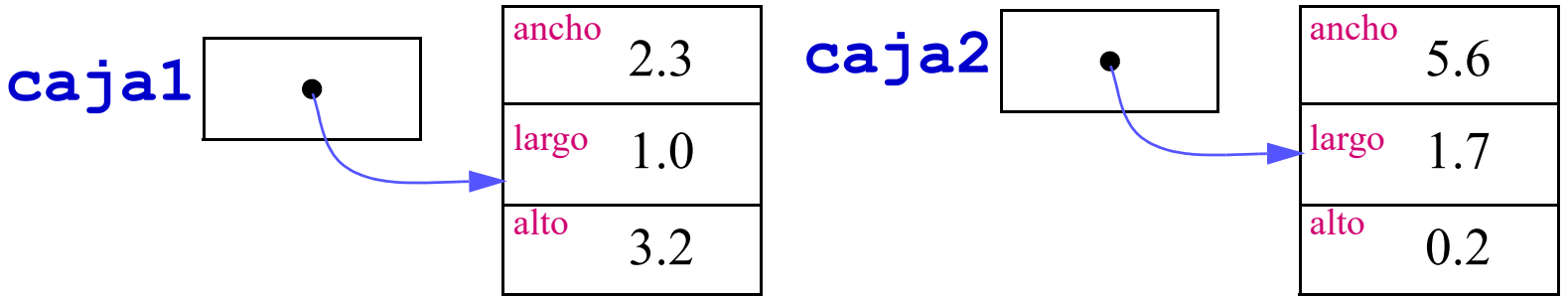
Ahora es posible obtener el volumen

```
double v=caja2.volumen();
// mismo volumen que caja1
```

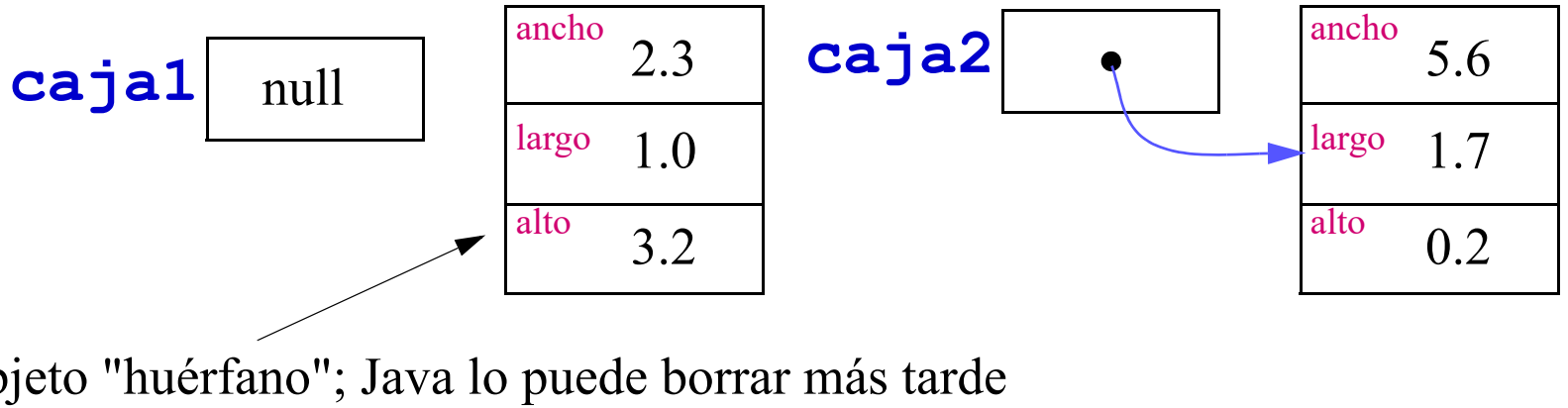
Declaración de objetos (cont.)

Si cambiamos una referencia, se pierde el enlace a la anterior

```
caja2 = new Caja(5.6,1.7,0.2);
```




```
caja1 = null;
```



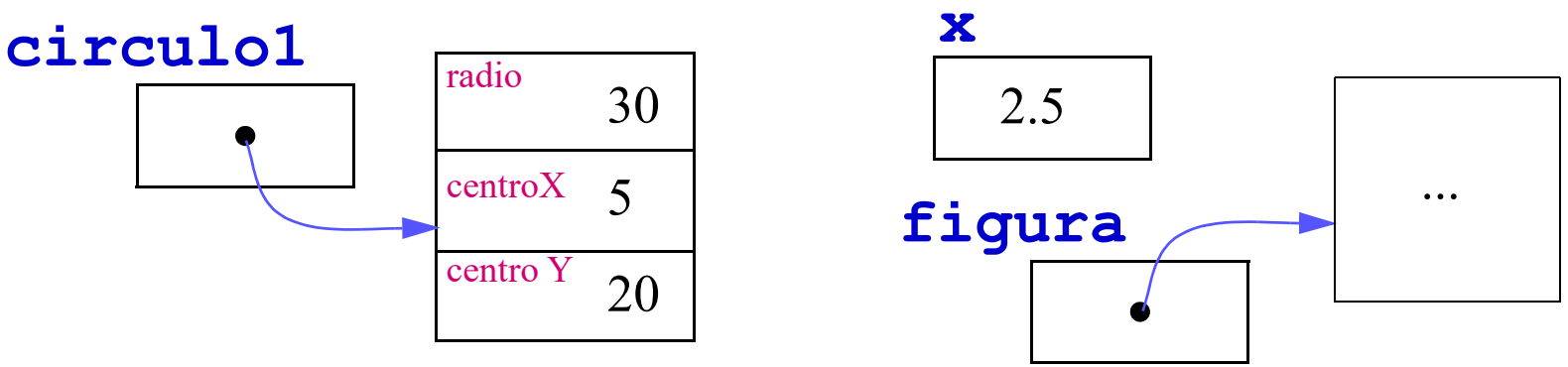
Objetos como parámetros

Las variables y objetos se comportan también diferente cuando se pasan como parámetros a un método

- ***variables***: el método recibe una copia
  - si se modifica la copia, el original no cambia
- ***objetos***: el método recibe una referencia
  - si se modifica el objeto, el cambio se refleja en el objeto original

Ejemplo de paso de variables y objetos como parámetros

Disponemos de esta variable y objetos



Ahora invocamos un método de la figura, que tiene esta cabecera:

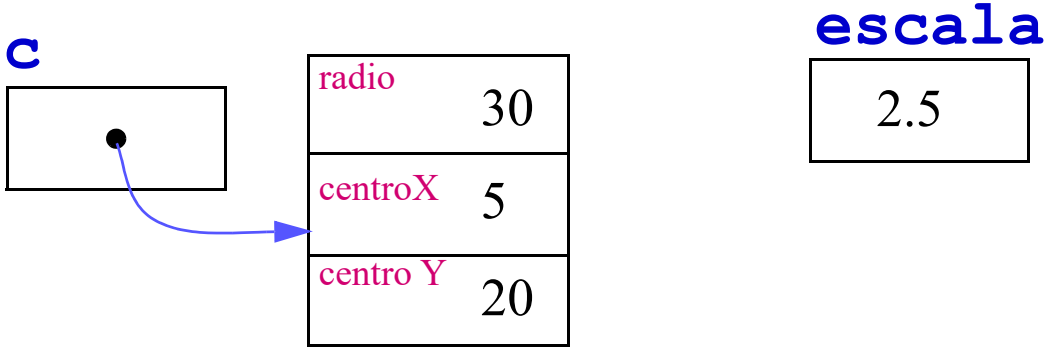
```
public void engloba(Circulo c, double escala)
```

Ejemplo (cont.)

La invocación se hace así:

```
figura.engloba(circulo1,x)
```

Los parámetros quedan así, durante la ejecución del método engloba:



`escala` es una copia de `x`

`c` es una referencia al mismo objeto al que apunta `circulo1`

Uso de atributos y métodos públicos de un objeto

Para usar un atributo o un método (públicos) de un objeto, se hace

```
nombre_ref.nombre_atributo
nombre_ref.nombre_metodo()

//ejemplo, si alto fuese público
caja1.alto=3.2; // si alto es privado, error

// ejemplo de invocar volumen()
// usamos el valor retornado en una expresión
double v=caja1.volumen();

// ejemplo, si aumentar() fuese un método público
// que no retorna nada
caja1.aumentar();
```

2.7. Lectura de datos desde una ventana

Vamos a usar una clase llamada `Lectura`, que contiene operaciones para leer datos por teclado

Está en el paquete `fundamentos`

- ver la documentación de este paquete

Pasos a realizar:

1. Crear el objeto de la clase `Lectura`
2. Crear en él entradas, una para cada dato
3. Esperar a que el usuario tenga tiempo de teclear
4. Leer desde el programa los datos que el usuario tecleó

Queremos modificar el programa del plano inclinado para que acepte datos de entrada metidos con `Lectura`

Lectura de datos de una ventana (cont.)

```
import fundamentos.*;

/**
 * Programa principal que pide datos por teclado para
 *  una esfera en un plano inclinado y muestra la
 *  distancia y energías a los cuatro segundos */
public class CalculaMovimiento2 {
  public static void main(String[] args)
  {
    double angulo, masa, radio, tiempo;
    Lectura l=
       new Lectura("Datos para el plano inclinado");
    l.creaEntrada("Angulo (grados)",0.0);
    l.creaEntrada("Masa (Kg)",0.0);
    l.creaEntrada("Radio (m)",0.0);
    l.creaEntrada("Tiempo (s)",0.0);
    l.espera();
```

Lectura de datos de una ventana (cont.)

```
        angulo=l.leeDouble("Angulo (grados)");
        masa=l.leeDouble("Masa (Kg)");
        radio=l.leeDouble("Radio (m)");
        tiempo=l.leeDouble("Tiempo (s)");
        PlanoInclinado p=
            new PlanoInclinado(angulo,masa,radio);
        System.out.println
            ("Distancia a los"+tiempo+" seg: "+
             p.distancia(tiempo)+" m");
        System.out.println
            ("E. C. de tras. a los "+tiempo+" seg: "+
             p.eCineticaTras(tiempo)+" J");
        System.out.println
            ("E. C. de rot. a los "+tiempo+" seg: "+
             p.eCineticaRot(tiempo)+" J");
    }
}
```

Lectura de datos del terminal

```
import java.util.Scanner;

/**
 *Prueba a usar un Scanner sobre la entrada de teclado (System.in)
 */
public class PruebaLecturaTerminal {
   public static void main(String[]args){
      // Variable Scanner usada para leer
      Scanner scan = new Scanner(System.in);

      System.out.println("Escribe un número real: ");
      double p= scan.nextDouble();
      System.out.println("Escribe un entero: ");
      int i1= scan.nextInt();
      System.out.println("Escribe una palabra: ");
      String t=scan.next();
   }
}
```

Los números reales pueden requerir notación en español. Ej: 2,34

2.8. Strings

La clase predefinida `String` permite manipular textos en Java

Igual que con las demás clases, al declarar un objeto de tipo `String` lo que hacemos es declarar una referencia al objeto.

```
String str1;
```

La clase `String` es especial porque admite literales de string

```
String str2="Hola soy un string";
str1=str2; // ambos se refieren al mismo string
```

La operación de concatenación de strings ("+") crea un nuevo string a partir de otros dos.

```
str1=str2+" y yo soy pepe";
```

Ejemplo de programa con variables de texto

```
import fundamentos.*;
/**
 * Lee dos nombres y pone un mensaje en la pantalla
 */
public class Nombres {
  public static void main(String[] args) {
     Lectura pantalla = new Lectura ("Nombres");
     String nombre,padre;

     pantalla.creaEntrada("tu nombre","");
     pantalla.creaEntrada("nombre de tu padre","");

     pantalla.espera
        ("Introduce los nombres y pulsa Aceptar");
```

Ejemplo de programa con variables de texto (cont.)

```
        nombre=pantalla.leeString("tu nombre");
        padre=pantalla.leeString("nombre de tu padre");

        System.out.println
          ("El padre de "+nombre+" es "+padre);
    }
}
```

2.9. Composición de objetos

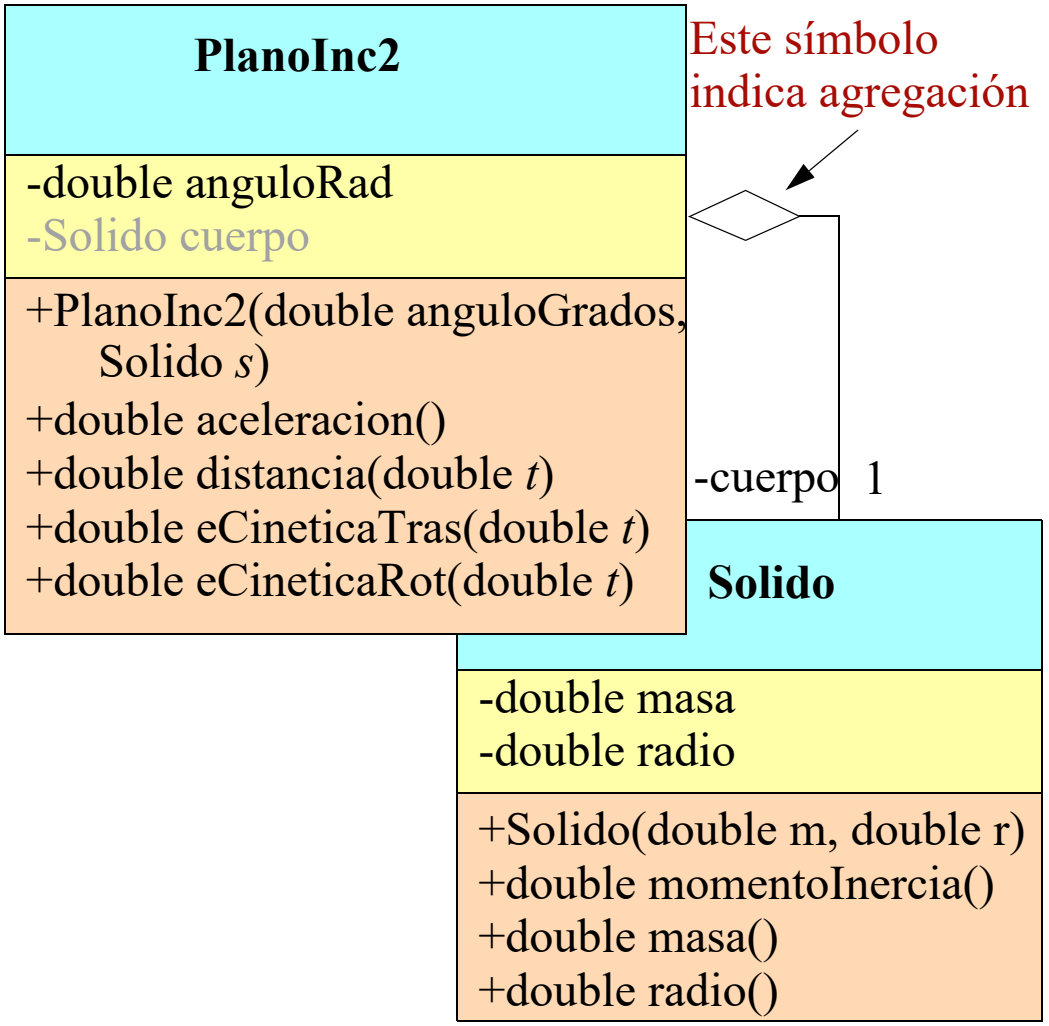
Un atributo puede ser un objeto de otra clase

- se dice que está *agregado* a la clase

Por ejemplo, vamos a modificar el programa del plano inclinado

- para que el objeto que rueda sea otra clase
- y así poder cambiar su momento de inercia

$$I = \frac{2}{5}mr^2 \quad esfera \qquad I = mr^2 \quad aro \qquad I = \frac{1}{2}mr^2 \quad cilindro$$



Ejemplo: clase Solido

```
/**
 * Clase que contiene los datos de un sólido que
 * puede rotar
 */
public class Solido
{
   private double masa;        // Kg
   private double radio;       // metros

   /**
    * Constructor al que se le pasan
    * la masa en Kg, y el radio en metros
    */
   public Solido(double masa, double radio) {
       this.masa=masa;
       this.radio=radio;
   }
```

Ejemplo: clase Solido (cont.)

```
/**
 * Momento de inercia, en Kg*m^2
 */
public double momentoInercia()
{
    return 2.0*masa*radio*radio/5.0;
}

/**
 * Masa, en Kg
 */
public double masa() {
    return masa;
}
```

Ejemplo: clase Solido (cont.)

```
    /**
     * radio, en metros
     */
    public double radio() {
        return radio;
    }
}
```

Ejemplo: clase PlanoInc2

```
/**
 * Clase que contiene los datos de un
 * sólido que rueda por un plano inclinado */
public class PlanoInc2 {
  private double anguloRad; // radianes
  private Solido cuerpo;
  private final double g=9.8; //metros/seg2

  /**
   * Constructor al que se le pasan el angulo en
   * grados, y el solido
   */
  public PlanoInc2 (double anguloGrados,Solido s)
  {
    anguloRad=Math.toRadians(anguloGrados);
    cuerpo=s;
  }
```

Ejemplo: clase PlanoInc2

```
/**
 * Calcula la aceleración lineal del objeto (m^2)
 */
public double aceleracion()
{
  return g*Math.sin(anguloRad)/
    (1+cuerpo.momentoInercia()/
    (cuerpo.masa()*cuerpo.radio()*
     cuerpo.radio()));
}

/**
 * Distancia recorrida en t segundos (en m)
 */
public double distancia (double t) {
  return aceleracion()*t*t/2.0;
}
```

Ejemplo: clase PlanoInc2

```
/**
 * Calcula la energía cinética de traslación del
 * objeto transcurridos t segundos (J)
 */
public double eCineticaTras (double t) {
   double vel = aceleracion()*t;
   return cuerpo.masa()*vel*vel/2.0;
}
```

Ejemplo: clase PlanoInc2

```
 /**
  * Calcula la energía cinética de rotación del
  * objeto transcurridos t segundos (J)
  */
  public double eCineticaRot (double t) {
    double velAngular =
       aceleracion()*t/cuerpo.radio();
    return cuerpo.momentoInercia()*velAngular*
       velAngular/2.0;
  }
}
```

2.10. Atributos y métodos estáticos

Una clase compilada separadamente puede tener atributos y métodos estáticos:

- no pertenecen a los objetos de la clase, sino a la clase misma
- si son atributos, sólo existe un dato por clase, no uno por objeto
  - la vida es la de la clase, y no la del objeto
- si son métodos, sólo pueden usar atributos estáticos
- se llaman también atributos y métodos ***de clase***

En el ejemplo siguiente se usa un atributo de clase para asignar a cada objeto un número de serie diferente

Ejemplo: números de serie

```
/**
 * Clase que contiene los datos de un coche
 */
public class Coche {

   // atributo estático
   private static int ultimoNumSerie=0;

   // atributos normales
   private int numSerie;
   public  String color;
   public  String nombre;
```

Ejemplo (cont.)

```
/**
 * Constructor; se le pasan los datos del coche
 */
public Coche (String nombre, String color) {
    ultimoNumSerie++;
    this.numSerie=ultimoNumSerie;
    this.color=color;
    this.nombre=nombre;
}
```

Ejemplo (cont.)

```
   /**
    * Muestra los datos en pantalla
    */
   public void muestra() {
      System.out.println("Vehiculo : "+nombre);
      System.out.println("Num serie: "+numSerie);
      System.out.println("Color    : "+color);
   }
}
```

Ejemplo (cont.)

```
/**
 * Programa de prueba de la clase Coche
 */
public class PruebaVehiculo {

   public static void main(String[] args) {

      Coche V=new Coche("coche1","rojo");
      Coche W=new Coche("coche2","verde");

      V.muestra();
      W.muestra();

   }
}
```