Bloque II. Elementos del lenguaje de programación Java

- 1.Introducción a los lenguajes de programación
- 2. Estructura de un programa
- 3. Datos y expresiones simples
- 4. Instrucciones de control
- 5. Entrada/salida simple
- 6. Arrays, secuencias y tablas
- 7. Métodos

3. Datos y expresiones simples

- 3.1. Tipos primitivos (predefinidos)
- 3.2. Los datos en Java
- 3.3. Operadores y expresiones
- 3.4. Conversión de tipos
- 3.5. Uso de funciones matemáticas
- 3.6. Declaración de clases y objetos
- 3.7. Strings
- 3.8. Composición de objetos

3.1. 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**

3.2. Los datos en Java

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**

Según dónde declaremos ese nombre tendremos

Dato	Lugar donde se declara	Uso
atributo	clase	datos que forman parte de los objetos de la clase
argumento	paréntesis del encabe-zamiento de un método	datos que el método necesita del exterior
variable	contenido de un método	datos que un método calcula y usa temporalmente

Constantes literales

tipo	descripción	ejemplos
`boolean`		**true false**
`char`	el carácter entre apóstrofes 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	**12 -37** **037** **0x2A**
`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"**

Atributos

Representan un dato de un tipo determinado, declarado como parte un objeto y alojado en la memoria

- **contiene un valor que puede cambiar**
- **se representa por un nombre**

Declaración de atributos, dentro de una clase:

```
private tipo nombre;
private tipo nombre = valor;
```

La palabra `private` es un *descriptor* opcional

- **Indica que el dato sólo se puede usar dentro de la clase**
- **Salvo para clases muy simples, es habitual que los atributos sean privados**

Variables

Representan un dato de un tipo determinado, declarado dentro de un método y alojado en la memoria

- contiene un valor que puede cambiar
- se representa por un nombre
- se destruye al finalizar el método

Declaración de variables, dentro de un método:

```
tipo nombre;
tipo nombre = valor;
//2 variab. del mismo tipo
tipo nombre1, nombre2;
```

Diseño:

```
tipo nombre;
tipo nombre:=valor;
tipo nombre1,nombre2;
```


Ejemplos

Atributos

```
private int lo = 1;
private int hi = 2;
private double x;
```

Variables

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

Si no se pone valor inicial, el valor es indefinido

- es un error usarlo

Diseño:

```
atributos
   entero lo;
   entero hi;
   real x;
fatributos
```

Diseño:

```
entero x1,x2;
real y;
```

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

Constantes

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

- declaración: atributo o variable con descriptor `final`
  - `final` indica que el dato ya no se puede cambiar de valor
- constantes 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 pi = 3.1416;
final int maxNum = 50;
final double factorEscala;
```

Diseño:

```
const real pi:=3.1416;
const entero maxNum:=50;
const real factorEscala;
```

Argumentos

Representan un dato de un tipo determinado, declarado dentro de los paréntesis en el encabezamiento de un método, y alojado en la memoria

- contiene un valor
- se representa por un nombre

Declaración de argumentos

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

```
método metodo1
 (tipo1 nombre1,
  tipo2 nombre2)
  retorna tipo_retornado
  ...
```

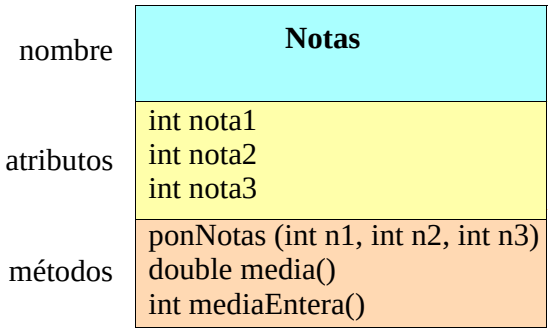

Ejemplo de programa con datos

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



Ejemplo (cont)

```
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;
    }

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

Ejemplo (cont)

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


Comentarios sobre el ejemplo

- argumentos de un método: datos 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;
    }
}
```

3.3. 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	multiplicación	división	módulo	incremento	decremento

- **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 "cortocircuitado"	o "cortocircuitado

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

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

Ejemplo

Cálculo de valores resistivos en paralelo.

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

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.)

```
public class Resistencias
{
    private double r1, r2, r3; //Kohms

    /** Constructor*/
    public Resistencias(double res1,
           double res2, double res3)
    {
        r1=res1;
        r2=res2;
        r3=res3;
    }
```


Ejemplo (cont.)

```
    /** Calcula la resistencia equivalente
     *   a 3 resistencias en paralelo */
    public double calculaResEquiv() { // Kohms
        return 1/((1/r1)+(1/r2)+(1/r3));
    }
}
```

Ejemplo: Uso desde un programa

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

```
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);
     System.out.println
        ("Resistencia efectiva = "+reff);
  }
}
```

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

3.4. 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 `char` o `boolean`)
- También hay conversión automática al almacenar un literal 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
```


3.5. 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` y son:

`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 pi
`exp(x), log(x)`	e^x y logaritmo neperiano
`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 `int`)
`abs(x), max(x,y), min(x,y)`	valor absoluto, máximo y mínimo: las tres 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

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

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

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

Ejemplo (cont.)

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 ($v=\omega r$)

Ejemplo (cont.)

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

PlanoInclinado
double anguloRad double masa double radio
PlanoInclinado(double θ, double m, double r) double aceleracion() double distancia(double t) double eCineticaTras(double t) double eCineticaRot(double t)

Ejemplo (cont.)

```
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 angulo
     *  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 aceleracion lineal del objeto
     */
    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 cinetica de traslacion del
    *  objeto transcurridos t segundos
    */
    public double eCineticaTras (double t)
    {
        double vel = aceleracion()*t;
        return masa*vel*vel/2.0;
    }
```


Ejemplo (cont.)

```
    /** Calcula la energia cinetica de rotacion del
     *  objeto transcurridos t segundos
     */
    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

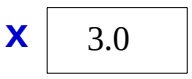
Programa principal que muestra la distancia y energías a los cuatro segundos

```
public class CalculaMovimiento {
  public static void main(String[] args) {
    PlanoInclinado p=
        new PlanoInclinado(30.0,1.5,0.2);
    System.out.println
       ("Dist. a los 4 seg: "+p.distancia(4.0));
    System.out.println
      ("E. C. tras. a 4 seg: "+p.eCineticaTras(4.0));
    System.out.println
       ("E. C. rot a 4 seg: "+p.eCineticaRot(4.0));
     }
}
```


3.6. Declaración de clases y objetos

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

```
double x=3.0;
```

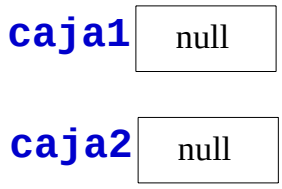


Cuando se declara un objeto de una determinada clase, se hace siempre a través de una referencia

Declaración de clases y objetos

Ejemplo: Declaración de una clase:

```
public class Caja {
   double ancho, largo, alto;
}
```



Declaración de objetos de esta clase:

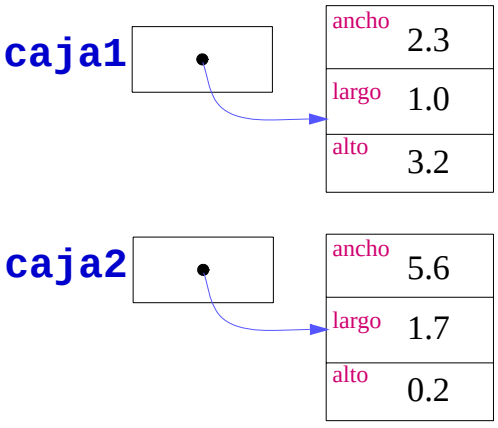
```
   Caja caja1;
   Caja caja2;
```

Las variables `caja1` y `caja2` son referencias a cajas. Inicialmente no se refieren a ninguna caja (ello se representa con un valor llamado `null`)

Declaración de clases y objetos (cont.)

Para que `caja1` y `caja2` se refieran a un objeto de la clase `Caja`, éste debe crearse:

```
   caja1 = new Caja();
   caja2 = caja1; //copia la ref.
   caja2 = new Caja();
```



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

```
   nombre_ref.nombre_atributo
   nombre_ref.nombre_metodo()
   //ejemplo
   caja1.alto=3.2;
```

Ejemplo

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

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

Ejemplo

```
import fundamentos.*;

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();
    angulo=l.leeDouble("Angulo (grados)");
    masa=l.leeDouble("Masa (Kg)");
```

Ejemplo (cont.)

```
      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));
      System.out.println
         ("E. C. de tras. a los "+tiempo+" seg: "+
          p.eCineticaTras(tiempo));
      System.out.println
         ("E. C. de rot. a los "+tiempo+" seg: "+
          p.eCineticaRot(tiempo));
      }
}
```


3.7. 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);
  }
}
```

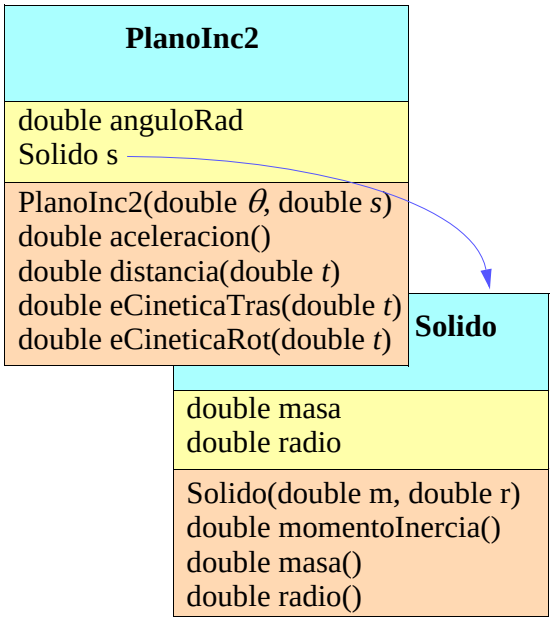
3.8. Composición de objetos

Un atributo puede ser un objeto de otra 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

```
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 m, double r) {
      masa=m;
      radio=r;
  }
```

Ejemplo: clase Solido (cont.)

```
   /** Momento de inercia
    */
   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

```
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
   */
  public double aceleracion()
  {
    return g*Math.sin(anguloRad)/
      (1+cuerpo.momentoInercia()/
      (cuerpo.masa()*cuerpo.radio()*
       cuerpo.radio()));
  }

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

Ejemplo: clase PlanoInc2

```
  /** Calcula la energia cinetica de traslacion del
   *  objeto transcurridos t segundos
   */
  public double eCineticaTras (double t) {
    double vel = aceleracion()*t;
    return cuerpo.masa()*vel*vel/2.0;
  }
```


Ejemplo: clase PlanoInc2

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