

# 1.00 Clase 7

## Cadenas (*Strings*)

### Constructores

### *this*

### Sobrecarga

## Cadenas

```
public class StringExample {
    public static void main(String[] args) {
        String first= "George ";
        String middle= "H.W. ";
        String last= "Bush";
        String full= first + middle + last;
        System.out.println(full);

        // Comprobación de igualdad en cadenas (objetos en general)
        String full2= "George H.W. Bush";
        if (full.equals(full2)) // Modo correcto
            System.out.println("Las cadenas son iguales");
        if (full == full2) // Modo incorrecto
            System.out.println(";Sería un milagro!");
        if (first == "George ") // Modo incorrecto, pero funciona
            System.out.println(";No es un milagro!");

        //La modificación de cadenas ha de ser indirectamente constante
        middle= middle.substring(0, 2) + " ";
        full= first + middle + last;
        System.out.println(full);    }    }
```

## Muelles (Springs)

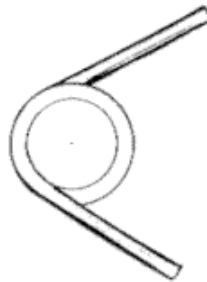
Compression



Tension



Torsion



$$F = k \, dx$$

## Constructores de la clase Spring

```
class Spring {
    private String material= "acero";           // Inicializada
    private double length;                     // Comprimi do
    private double maxDeflect;
    private double k;

    public Spring(String m, double len, double md, double k) {
        material= m;
        length= len;
        maxDeflect= md;
        this.k= k;                             // "this"
    }
    public Spring(double len, double k) {
        this("acero", len, 0.25*len, k);      // "this"
    }
    public Spring(double len) {
        this(len, 0.5*len);
    }
    public Spring() {
        this(10.0);    }
}
```

## Métodos de la clase Spring

```
public String getMaterial() { return material; }
public double getLength() { return length; }
public double getMaxDef() { return maxDeflect; }
public double getK() { return k; } // No es necesario 'this'

public void setMaterial(String m) {material= m; }
public void setLength(double len) {length= Math.abs(len);}
public void setMaxDef(double m) {maxDeflect= Math.abs(m);}
public void setK(double k) {this.k= k; } // this

public double getForce(double deflection) {
    if (deflect > maxDeflect)
        deflect= maxDeflect;
    return k*deflect;
}
public double getForce() { // Métodos sobrecargados
    return k*maxDeflect;
}
}
```

## Main de Spring

```
public class SpringExample {
    public static void main(String[] args) {
        Spring one= new Spring("aluminum", 2.0, 1.0, 5.0);
        Spring two= new Spring(5.0, 3.0);
        Spring three= new Spring(); // 3 constructores diferentes

        double f1= one.getForce(0.5);
        double f2= two.getForce(1.5);
        double f3= three.getForce(0.1);
        System.out.println("f1: " + f1 + "\nf2: " + f2 +
            "\nf3: " + f3);

        double f4= one.getForce(); // Métodos sobrecargados
        double f5= two.getForce();
        double f6= three.getForce();
        System.out.println("f4: " + f4 + "\nf5: " + f5 +
            "\nf6: " + f6);
        System.exit(0);
    }
}
```

## Diseño de la clase Spring

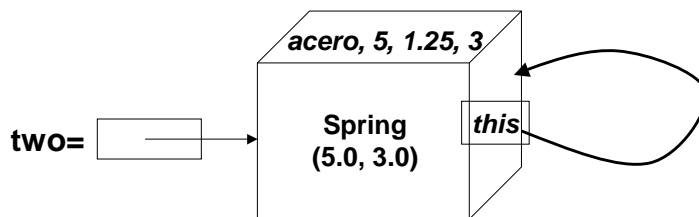
- **Todos los métodos Spring son públicos:**
  - Cualquier método de cualquier clase puede llamarlos.
  - Los métodos privados se pueden utilizar como ayuda o en cuestiones complicadas que sólo debería hacer clase.
- **Los campos de datos en Spring son privados:**
  - Sólo los métodos de la clase Spring pueden acceder a ellos.
  - Los campos de datos públicos casi nunca se utilizan.
- **El nombre del constructor debe coincidir con el nombre de la clase:**
  - Sólo se puede llamar a los constructores con *new*.
  - Los constructores no pueden tener un valor de retorno.

## Métodos de la clase Spring

- **¿Por qué disponer de todos estos métodos?**
  - Los métodos "get" son el único acceso a los datos de Spring desde otras clases:
    - Nos permiten volver a implementar Spring en caso necesario
  - Los métodos "set" deberían comprobar errores:
    - Nuestro método debería servir para asegurarnos de que longitud>0, refracción máx < longitud, etc.
- **Los métodos sobrecargados deben tener diferentes firmas (argumentos):**
  - Distintos tipos de retorno no permiten distinguir entre dos métodos por lo demás idénticos
    - `int getForce(int a)`      **y**
    - `double getForce(int b)`    **no compilarán**

## This

```
Spring two= new Spring(5.0, 3.0);
```

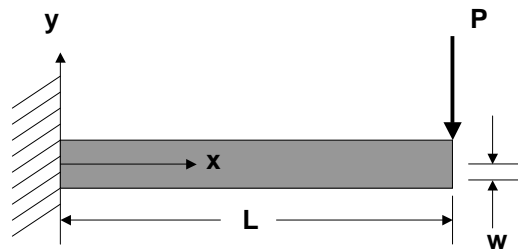


“this” también se emplea como abreviatura para hacer referencia a otros constructores de la clase

## Destrucción de objetos

- **Java recupera automáticamente la memoria del objeto mediante el "recolector de basura" cuando no hay referencias activas a ese objeto:**
  - En el lenguaje C++ el programador debe recuperar la memoria manualmente. El uso de "new" en C++ es limitado, ya que es necesario utilizar "delete" al terminar para evitar fugas de memoria
  - En C++, sólo se utiliza 'new' en constructores, nunca en `main()` u otros métodos, muy al contrario que en Java.
- **Java dispone de finalizadores que limpian otros recursos (archivos, dispositivos bloqueados, etc.) cuando se destruye un objeto:**
  - Un consejo informal: nunca utilice finalizadores
  - Pueden invocar a cualquier objeto, lo que reduce drásticamente la eficacia del recolector de basura

## Ejercicio del haz de luz



Escriba una clase en Java para modelar este haz de luz y calcular su refracción máxima

$$w = - \frac{P L^3}{3 E I}$$

donde:

P= carga en el extremo (1200 N)

L= longitud del haz (20 m)

E= coeficiente de elasticidad (30000 N/ m<sup>2</sup>)

I= momento de inercia (1000 m<sup>4</sup>)

w= refracción (m)

## Ejercicio del haz de luz (2)

- Campos de datos:
  - ¿Qué características tiene el haz de luz?
  - ¿Cuáles de ellas son externas?
- Escriba dos constructores:
  - Uno con todos los campos como argumentos
    - Utilice los mismos nombres para argumentos y campos
  - Uno con argumento de longitud únicamente
    - Otros campos por defecto, como en la diapositiva anterior
    - Utilice 'this' para invocar a otro constructor o basarse en inicialización
  - Emplee inicialización en su clase: suponga que tiene que emplear varios haces de luz como el del ejemplo
- Escriba dos métodos para devolver la refracción w:
  - Utilice el mismo nombre para ambos métodos (sobrecarga)
  - Uno recibe la carga como argumento; otro, carga y unidades (metros o pies) como String. Conversión del resultado: 1 metro = 3,3 pies
- No escriba métodos get y set
- Escriba una clase con un método main() que cree un haz de luz, calcule su refracción y muestre el resultado por pantalla

## Ejercicio del haz de luz (3)

- **Opcional, fase avanzada:**
  - **Añada un análisis dimensional:**
    - Almacene las unidades para cada variable de la clase
      - Decida cómo codificarlas (exponentes, etc.)
    - **Adapte los constructores para que admitan argumentos de esas unidades**
    - **Convierta las unidades si es necesario(N/lbf, m/pies)**
      - 1 lbf = 4,4 N; 1 pie = 0,3 m
    - **Compruebe que las unidades se adaptan al cálculo**
    - **Muestre las unidades con el resultado del método**

## Clase Beam

```
class Beam {
    private double L;
    private double E= 30000.0;           // Inicialización
    private double I= 1000.0;
    public Beam(double L, double E, double I) {
        this.L= L;                       // this para acceder a los campos
        this.E= E;                         // de objetos
        this.I= I;
    }
    public Beam(double L) {
        this.L= L;                         // Utilice la inicialización para otros
    }                                     // Podría usar this(L, 30000., 1000.);
    public double getDeflection(double P) {
        return -P*L*L*L/(3.0*E*I);
    }
    public double getDeflection(double P, String u) {
        if (u.equals("ft"))                // no utilice ==
            return -3.3*P*L*L*L/(3.0*E*I);
        else
            return -P*L*L*L/(3.0*E*I);
    }
}
```

## main() de Beam

```
public class BeamExample {
    public static void main(String[] args) {
        Beam one= new Beam(20.0);
        double w1= one.getDeflection(1200.0);
        System.out.println("w1: " + w1);
        double w2= one.getDeflection(1200.0, "ft");
        System.out.println("w2: " + w2);
        System.exit(0);
    }
}
```