3.El lenguaje Java

3.1 Identificadores

- sintaxis
- tipos primitivos

3.2 Operadores

3.3 Control de flujo

3.4 Clase `Array`

3.1 Identificadores

- Nombran variables, funciones, clases y objetos
- Comienza con una letra, un subrayado (_) o un símbolo de dólar ($). Los siguientes caracteres pueden ser letras o dígitos.
- Se distinguen las mayúsculas de las minúsculas
- No hay una longitud máxima establecida para el identificador.

Variables

- Sirven para referirse tanto a objetos como a tipos primitivos.
- Tienen que declararse antes de usarse:

```
tipo identificador;
int posicion;
```

- Se puede inicializar mediante una asignación:

```
tipo identificador = valor;
int posicion = 0;
```

- Definición de *constantes*:

```
static final float PI = 3.14159f;
```


Tipos de datos primitivos

- Se pueden utilizar valores de los siguientes tipos:
  - **byte** (entero de 8 bits)
  - **short** (entero de 16 bits)
  - **int** (entero de 32 bits)
  - **long** (entero de 64 bits)
  - **float** (decimal de 32 bits)
  - **double** (decimal de 64 bits)
  - **char** (Unicode de 16 bits)
  - **boolean** (**true**, **false**)
- No se pueden definir tipos.

3.2 Operadores

• En orden de precedencia:

Operadores	Asociatividad	Tipo
()	izquierda a derecha	paréntesis
++ -- + - !	derecha a izquierda	unarios
* / %	izquierda a derecha	multiplicativos
+ -	izquierda a derecha	aditivos
< <= > >=	izquierda a derecha	relacionales
== !=	izquierda a derecha	de igualdad
&	izquierda a derecha	AND lógico booleano
^	izquierda a derecha	OR exclusivo lógico booleano
|	izquierda a derecha	OR inclusivo lógico booleano
&&	izquierda a derecha	AND lógico
||	izquierda a derecha	OR lógico
?:	derecha a izquierda	condicional expresion ? sentencia1 : sentencia2
= += -= *= /= %=	derecha a izquierda	asignación ej. x += y ⇔ x = x + y;

3.3 Control de flujo

• Control de saltos:

```
if( expresión-booleana )
{
        sentencias;
}
 [else {
        sentencias;
}]
```

```
switch(expresión) {
    case valor1:
        sentencias;
        break;
    case valor2:
        sentencias;
        break;
    [default:
        sentencias;]
}
```

Ejemplos

```
1) int saldo;
   …
   if (saldo<0)
      System.out.println("Cuenta en números rojos");

2) int dia;
   …
   switch (dia){
      case 1: System.out.println("Lunes"); break;
      case 2: System.out.println("Martes"); break;
      …
      case 7: System.out.println("Domingo"); break;
   }
```


Bucles

```
for( inicialización; exp-booleana; iteración ) {
        sentencias;
}
```

```
[inicialización;]
do {
        sentencias;
        [iteración;]
}while(expresión-booleana );
```

```
[inicialización;]
        while( expresión-booleana ) {
                sentencias;
                [iteración;]
        }
```

Ejemplos

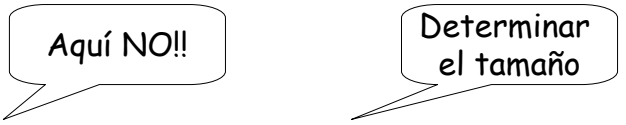
```
1. for (int index=0; i<tabla.length; index++){
        System.out.println(tabla[index]);
   };

2. int i=0;
   while (i< tabla.length){
      System.out.println(tabla[i]);
      ++i;
   };
```


3.4 Array

- Declaración:

  tipoDeElementos[] nombreDelArray = new tipoDeElemetos[tamañoDelArray];

- Creación:
  - un array vacío:



  - con valores iniciales:

```
String [] nombres = {
    "Juan","Pepe","Pedro","Maria"
};
```

- Los índices de un array siempre empiezan en 0
- Permite la ***asignación*** de un array a otro (dos ref a objetos)

4. Clases y Objetos

- Estructura de una clase
- Atributos de instancia y de clase (**`static`**)
- Control de acceso.
- Métodos
- Creación de objetos: constructores e inicialización
- **`this`**
- **`main`**
- Paquetes
- Comentarios y documentación (`javadoc`)

Estructura de una clase

```
class NombreDeLaClase {
   // declaración de las variables de instancia
   // declaración de las variables de la clase
   metodoDeInstancia() {
        // variables locales
        // código
   }
   metodoDeClase() {
        // variables locales
        // código
   }
}
```

- Todo forma parte de una clase
- Java NO soporta funciones o variables GLOBALES

Atributos

- **Atributos de clase:**
  - Común a todas las instancias de una clase
  - Sólo se inicializan una vez.
    Ej: `static public int nextCodigo;`
- **Atributos de instancia:**
  - determina el estado de los objetos
  - cada objeto reserva memoria para todas las variables de instancia

- **Declaración:**

[acceso][`static`][`final`] tipo *nombreAtributo* [= valor_inicial];

Inicialización de los atributos

- Si un atributo no se inicializa se le asigna un valor por defecto en función de su tipo:

Tipo	Valor Inicial
boolean	false
char	'\u000'
byte,short,int,long	0
float	+0.0f
double	+0.0d
Referencia a objeto	null

Control de acceso

- **public** – los miembros que se declaran como **public** son accesibles en cualquier parte donde la clase sea accesible
- **private** – miembros declarados como **private** sólo accesibles en la propia clase

```
public class A{
   public int at1;
   private int at2;
   …
}
```

```
A a;
//creamos el objeto
a.at1 = 6 //OK
a.at2 = 5 //ERROR
```

- Atributos privados y acceso a través de métodos públicos (set/get)

Métodos

- Código que "entiende" y manipula el estado de un objeto
- Pueden ser llamados dentro de la clase o por otras clases
- Es obligatorio indicar el tipo de retorno o **void**
- **Declaración:**

```
[acceso][static] tipoRetorno nomMet ([argumentos]){
  //cuerpo del metodo
  [return valorRetorno;]
 }
```

- **Invocación:** Notación punto

```
objReceptor.nomMet(argumentos);
```

Métodos de clase

- Se invocan sobre la clase, no sobre un objeto
- Sólo puede acceder a las variables y métodos de clase (**static**)
- **Ejemplos:**

```
1. int i = Integer.parseInt("123");

2. public class Ejemplo{
       static private int at;
       static public int getAt(){
           return at;
       }
   }
```

Se invoca `Ejemplo.getAt();`

Métodos

- Java soporta **sobrecarga de métodos**
  - el mismo nombre pero con **DIFERENTE** lista de argumentos
  - **SIEMPRE** devuelven el mismo tipo

```
public class ListaEnteros{
    …
  public void add (int elem){…}
  public void add (int elem, int index){…}
}
```

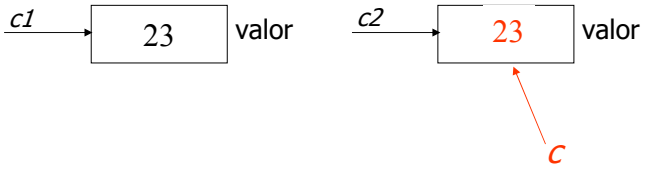
- **Paso de parámetros** siempre **por valor**
  - los valores de los parámetros son copias de los valores que especifica el que lo invoca
  - tipos primitivos no cambian
  - los objetos pueden cambiar su estado → **se pasa por valor la referencia**

Paso de parámetros

```
public class Contador {
   int valor;
   …
   public void sincronizar(Contador c){
        c.setValor(valor);
   }
}
```

```
c1.sincroniza(c2);
```

c=c2



Se puede modificar el estado del objeto pero no la referencia

Constructores

- Procedimiento con el mismo **nombre que la clase**
- Se invoca **automáticamente** cada vez que se **crea** un objeto de la clase
- **No** pueden especificar tipos ni **valores de retorno**
- Permite **sobrecarga** para especificar formas distintas de inicializar los objetos
- Si no se define, el compilador crea uno **por defecto** sin argumentos que inicializa los atributos a los valores por defecto
- El programador también puede definir un constructor sin argumentos.

Constructor de copia

```
public class Ejemplo{
  private int at1;
  …
  public Ejemplo (Ejemplo e){
     at1 = e.at1;
  }
}
```


Ejemplo: Clase Contador v1

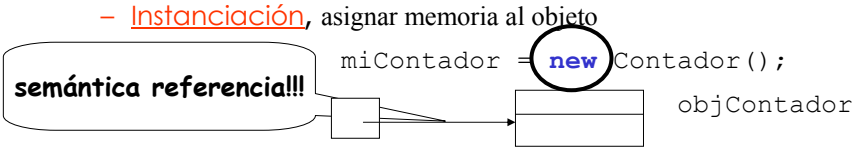
```
public class Contador{
   // Variables de instancia
  private int valor;

  // Constructores
    public Contador (){
       valor = 0;
    }
    public void incrementar(){
      valor++;
    }
    public int getValor(){
      return valor;
    }
}
```

Creación de objetos

- Un objeto es una instancia de una clase.
- La creación de un objeto se realiza en tres pasos
  - Declaración, proporcionar un nombre al objeto

```
Contador miContador;
```
(null)

  - Instanciación, asignar memoria al objeto

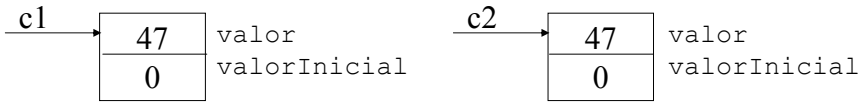


  - Inicialización, opcionalmente se pueden proporcionar valores iniciales a las variables de instancia del objeto en la declaración o mediante **CONSTRUCTORES**.

Ejemplo. Clase de prueba v1

```
public class TestContador{
  public static void main (String [] args){
       Contador c = new Contador();

       c.incrementar();
       c.incrementar();

       System.out.println("C = " + c.getValor());
   }
}
```

Igualdad vs. Identidad



¡¡OJO!!

== y != compara **REFERENCIAS** de manera que:

```
Contador c1 = new Contador();
Contador c2 = new Contador();
```

`c1 == c2` --> **`false`**, luego `c1!=c2`

`c1.equals(c2)` --> **`true`** compara el **CONTENIDO** de los objetos
si lo redefinimos en la clase
(por defecto es igual que ==)

this

- Invocación a otro de los **constructores** de la clase:

```
public Contador (int i) {
  valor=i;
  valorInicial=i;
}

public Contador(){
  this(0);
}
```

- **Instancia actual**:
  - Pasar el objeto actual como parámetro de otro método

```
servicio.añadir(this);
```

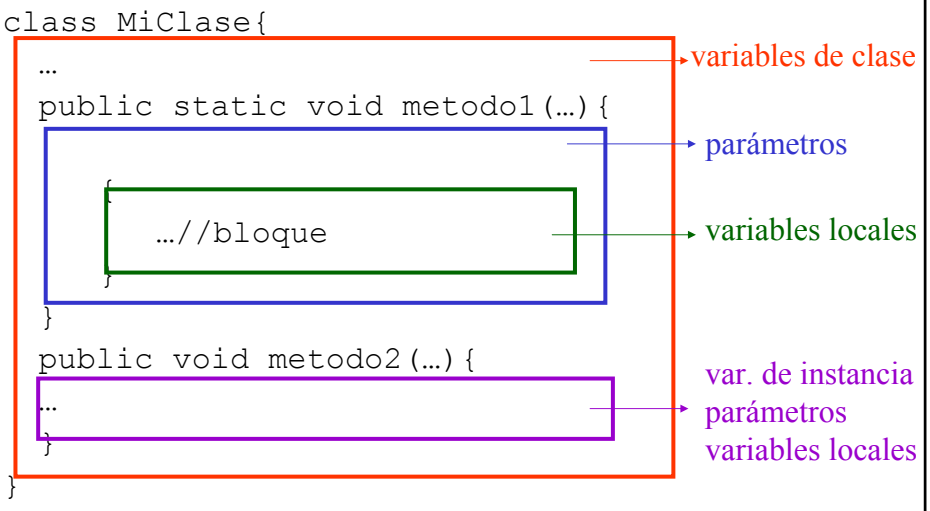
  - En cualquier método del objeto actual

```
this.otroMetodo();
```

Reglas de alcance

- Porción del programa en el que se puede hacer referencia al identificador
- Un identificador debe ser único dentro de su ámbito
- **Alcance de CLASE**:
  - los métodos de una clase pueden modificar directamente los atributos de clase y de instancia
  - Un método de clase (`static`) sólo puede acceder a atributos de clase.
- **Alcance de Bloque**:
  - variables locales y parámetros de un método
  - cuando se anidan los bloques y existen dos identificadores con nombres iguales el compilador da error de sintaxis (variable ya definida)
  - si una variable local tiene igual nombre que un atributo de la clase éste queda oculto.

Reglas de alcance



El método `main`

- Clase que conduce la aplicación
- Al ejecutar el programa se busca el método `main` que contiene dicha clase:

```
public class Eco {
  public static void main (String[] args){
     for (int i = 0; i<args.length;i++)
         System.out.print(args[i] + " ");
  }
}
```

- `String[] args`: parámetros del programa
- Ejemplo de invocación:

```
java Eco repite esto
```

- Puede haber más de un `main`, pero sólo se ejecutará uno

Normas de estilo

No es que existan reglas pero es conveniente seguir unas ***normas*** para que el código sea mas legible:

- nombre de CLASE empieza por Mayúsculas
- nombre de métodos y atributos por minúsculas
- las CONSTANTES con MAYÚSCULAS completamente
- cuando empieza una nueva palabra la ponemos en mayúsculas ej: dibujarRectangulo

Ejemplo: Clase Contador v2 (1/2)

```
public class Contador{
  // Variable de clase
    private static int nextCodigo=1;

  // Variables de instancia
    private int codigo;
    private int valor;
     private int valorInicial;
     private int step;

  // Constructores
     public Contador (int valor, int step){
         codigo =nextCodigo;
         this.valor = valor;
         valorInicial = valor;
         this.step = step;
         ++nextCodigo;
     }
      public Contador(){
        this(0,1);
      }
```


Ejemplo: Clase Contador v2 (2/2)

```
    // Métodos de instancia
  public void incrementar(){
        valor+=step;
  }
  public void decrementar(){
        if (valor>=step) valor-=step;
        else valor=0;
  }
  public void reset(){
        valor=valorInicial;
  }
  public int getValor(){
        return valor;
  }
  public int getCodigo(){
    return codigo;
 }
}
```

Ejemplo. Clase de prueba

```
public class TestContador{
  public static void main (String [] args){
       Contador c1 = new Contador();
       Contador c2 = new Contador (0, 10);

       c1.incrementar();
       c2.incrementar();

       System.out.println("C1 = " + c1.getValor());
       System.out.println("C2 = " + c2.getValor());
   }
}
```


Paquetes

- Agrupar todas las clases relacionadas

```
//MiClass.java
package miBiblioteca;
public class MiClase{ //...
```

- Se coloca al comienzo del fichero
- Indica que las **clases públicas** que contenga está accesible a cualquiera que incluya:

```
import miBiblioteca.MiClase;
```
o bien
```
import miBiblioteca.*;
```

- Las clases que no tienen un nombre de paquete están en el paquete por defecto (**`Default Package`**)
- Soporta el concepto de **jerarquía de paquetes** (estilo directorio)

```
ej: import miPaquete.miSubPaquete.MiClase;
```

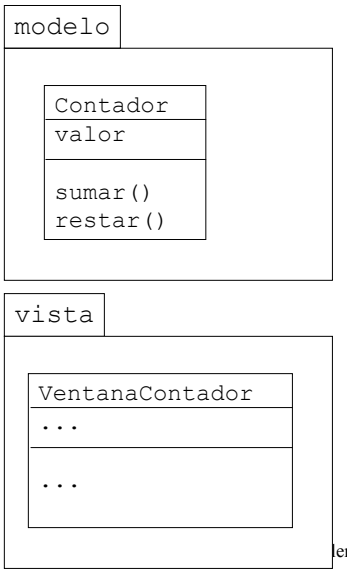
Control de acceso

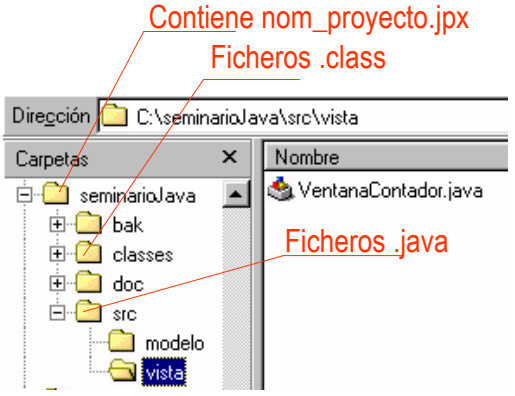
- Designar qué clases de una biblioteca (`miBiblio`) están disponibles desde fuera del paquete
- Controlan si el cliente puede crear objetos de la clase

  ```
  public class MiClase { ... }
  ```

  Entonces `import miBilio.MiClase;`
- Modificador de acceso por omisión indica **visibilidad a nivel de paquete**.
  - Tanto la clase como atributos y métodos.
- Una clase NUNCA puede ser `private`.

Estructura de un proyecto





- Una carpeta por cada paquete
- Cada carpeta contiene los ficheros correspondientes a las clases del paquete.
- Coincidencia **EXACTA** de nombres.

Paquetes básicos del sistema

- `java.lang`: para funciones del lenguaje
- `java.util`: para utilidades adicionales
- `java.io`: para manejo de ficheros
- `java.awt`: para diseño gráfico e interfaz de usuario
- `java.awt.event`: para gestionar eventos
- `javax.swing`: nuevo diseño de GUI
- `java.net`: para comunicaciones

…

Comentarios

- Hay tres **tipos** de comentarios:

```
// comentarios para una sola línea

/*
 comentarios de una o más líneas
*/

/** comentario de documentación, de una o más líneas
  justo antes del elemento (clase, var, met)
 */
```

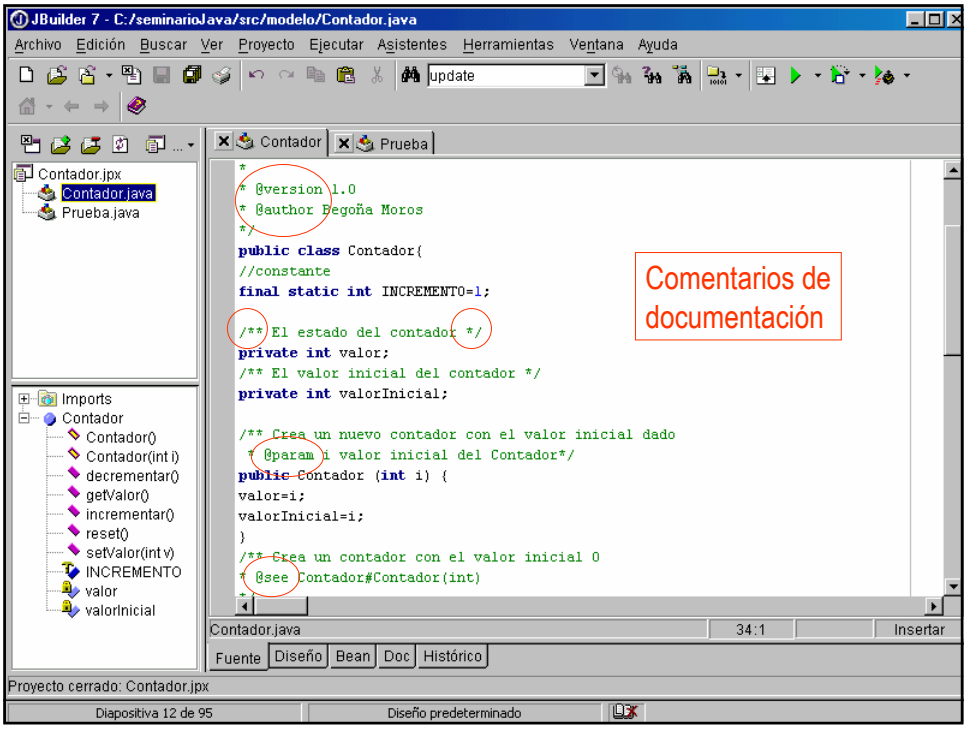
- Los comentarios de documentación pueden llevar **etiquetas**
- La documentación se genera con `javadoc` en formato html

Las etiquetas pueden ser:
@see <referencia a otra clase>

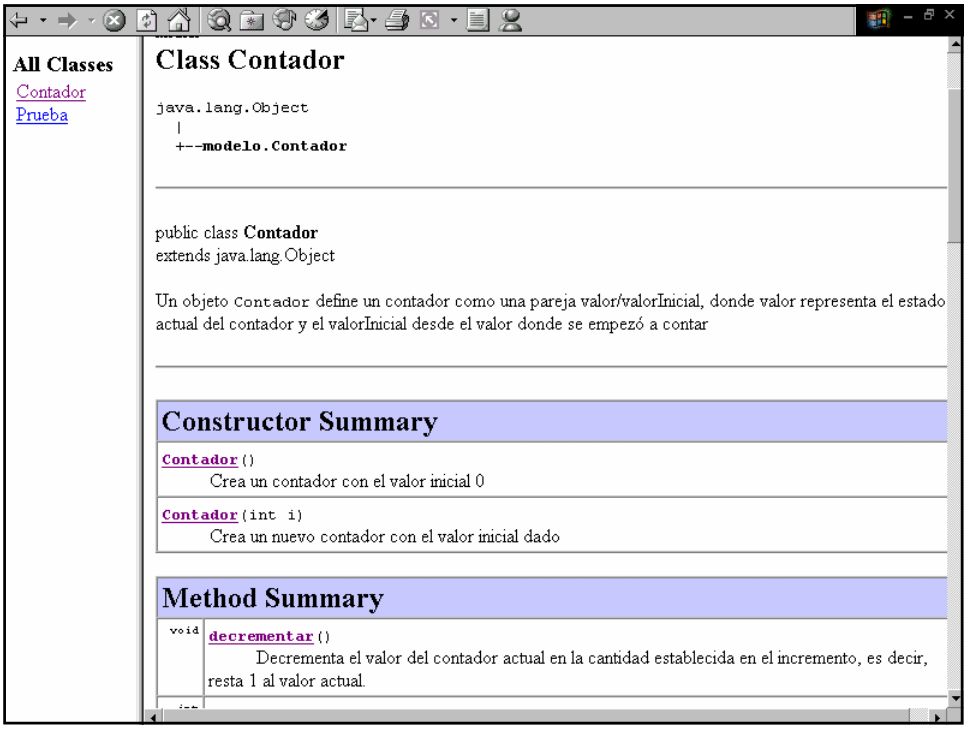
- Etiquetas de documentación de clases:
@version <información sobre la versión>
@author <nombre autor>

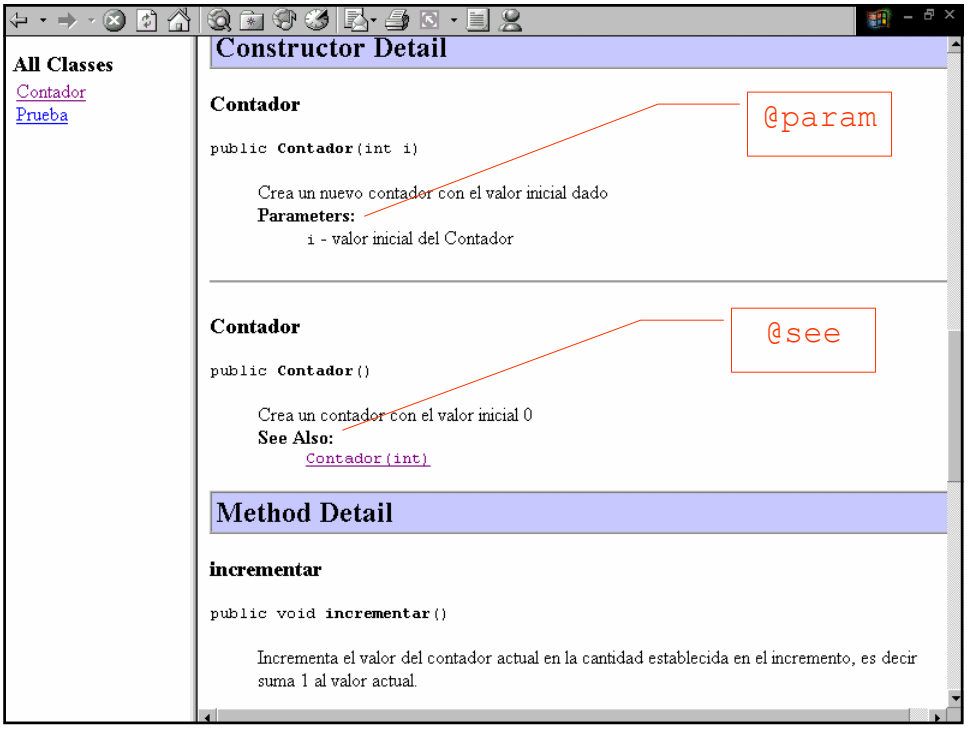
- Etiquetas de documentación de métodos:
@param <nombre argumento><descripción>
@return <descripción>
@exception <excepción>

- Otras: (no aparece en la documentación)
@todo <comentario sobre algún tema pendiente>



```
C:\seminarioJava>javadoc -sourcepath src/ modelo -d doc/
Loading source files for package modelo...
Constructing Javadoc information...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating doc/\overview-tree.html...
Generating doc/\index-all.html...
Generating doc/\deprecated-list.html...
Building index for all classes...
Generating doc/\allclasses-frame.html...
Generating doc/\index.html...
Generating doc/\packages.html...
Generating doc/\modelo\package-summary.html...
Generating doc/\modelo\package-tree.html...
Generating doc/\modelo\package-frame.html...
Generating doc/\modelo\Contador.html...
Generating doc/\modelo\Prueba.html...
Generating doc/\serialized-form.html...
Generating doc/\package-list...
Generating doc/\help-doc.html...
Generating doc/\stylesheet.css...
```




All Classes
Contador
Prueba
Constructor Detail
Contador
public Contador(int i)
Crea un nuevo contador con el valor inicial dado
Parameters:
i - valor inicial del Contador
@param
Contador
public Contador()
Crea un contador con el valor inicial 0
See Also:
Contador(int)
@see
Method Detail
incrementar
public void incrementar()
Incrementa el valor del contador actual en la cantidad establecida en el incremento, es decir suma 1 al valor actual.