



Universidad Tecnológica Nacional  
Facultad Regional Córdoba

Paradigmas de Programación

**UNIDAD NRO. 3**  
**PARADIGMA ORIENTADO A OBJETOS**  
**(PARTE II)**

# CONTENIDOS ABORDADOS

- Paradigma Orientado a Objetos. Revisión de conceptos principales
- Relaciones entre clases
  - Distintos tipos de relaciones
  - Relación de Pertenencia: Composición- Agregación
  - Esquema de Implementación de relación de pertenencia en Smalltalk
- Introducción a Colecciones en Smalltalk
  - Concepto de Colecciones
  - Tipos de colecciones en Smalltalk
  - Mensajes comunes para manejo de colecciones
  - Principales métodos de las colecciones
  - Introducción a mensajes de recorridos con colecciones
    - Mensaje do:

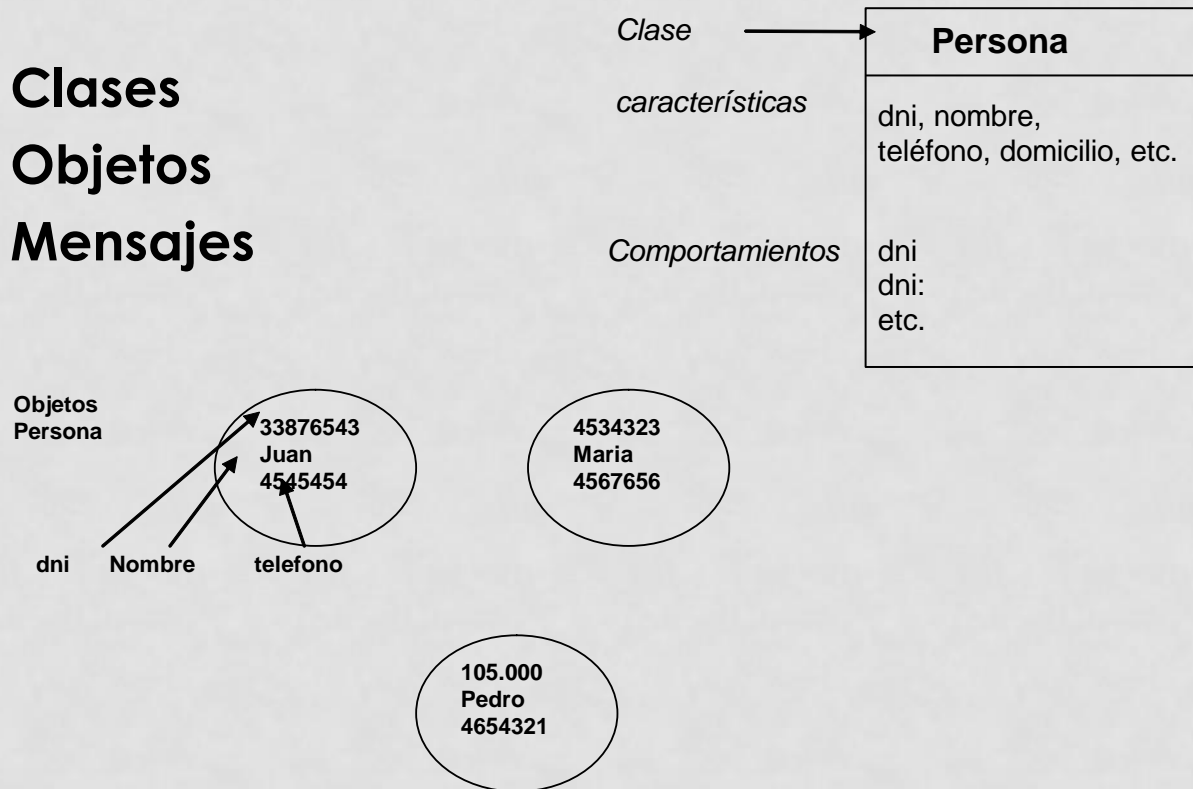
# BLOQUE 1: REVISIÓN PARADIGMA ORIENTADO A OBJETOS



# ELEMENTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

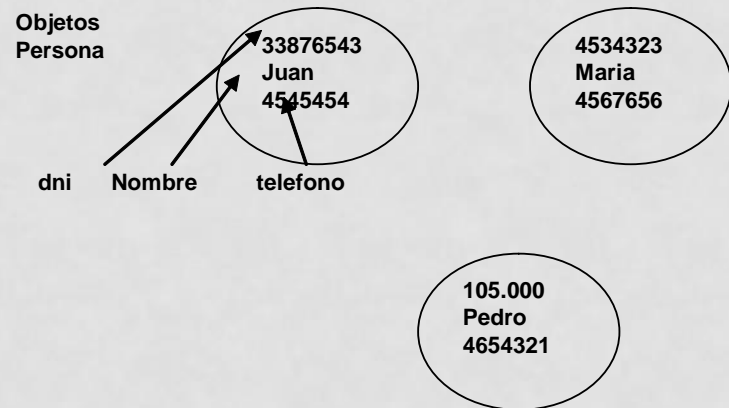
- Los elementos de la Programación Orientada a Objetos son tres:

- Clases**
- Objetos**
- Mensajes**



# CLASES Y OBJETOS

- **Clase** : Molde a partir del cual se crean los objetos; y en el que se definen los métodos y el conjunto de variables (características) que tendrán los objetos que se creen a partir del molde.
- **Objetos e Instancia** : Cada objeto es instancia de la clase que se usó como molde para crearlo. Cada objeto es instancia de exactamente una clase. Posee características y comportamientos.

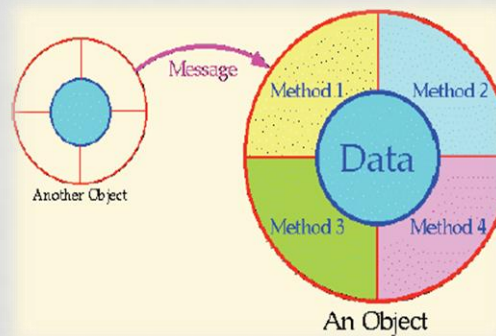


# BLOQUE 2: LENGUAJE SMALLTALK (CONTINUACIÓN)



# LENGUAJE SMALLTALK

- Es un **lenguaje orientados a objetos puro**, todas las entidades que maneja son objetos.
- La programación consiste en:
  - **Crear clases**
  - **Crear instancias**
  - **Especificar la secuencia de mensajes entre objetos**



# FLUJO DE CONTROL EN SMALLTALK

## Estructuras de control

- En Smalltalk, las estructuras de control se construyen a partir de mensajes enviados a objetos booleanos, bloques y números, y de la cooperación entre ellos.

## Lógica condicional

- La **lógica condicional** permite la ejecución del código dependiendo de un valor booleano.
- Se forman enviando mensajes a objetos booleanos y evaluando bloques si las condiciones se cumplen.



# MECANISMO DE CONTROL EN SMALLTALK

## Lógica condicional

### Ejemplos:

(  $a > b$  ) ifTrue: [  $c := a + b$  ]. “Ejecuta el bloque si la condición es verdadera”

(  $a = b$  ) ifFalse: [  $c := a + b$  ]. “Ejecuta el bloque si la condición es falsa.”

(  $a \geq b$  )  
ifTrue: [  $c := a + b$  ]  
ifFalse: [  $c := a - b$  ]. “Ejecuta el bloque [  $c := a + b$  ] si la condición es verdadera; sino ejecuta el otro bloque.”

# MECANISMO DE CONTROL EN SMALLTALK

## Iteraciones

- Smalltalk soporta cuatro tipos tradicionales de iteraciones.
- Ellos son:
  - Hacer algo “n” número de veces: **timesRepeat**.
  - Hacer algo usando un índice, comenzando con un valor inicial y finalizando en un valor final: **to:do:**
  - Hacer algo hasta que se encuentre con una condición False: **whileFalse:**
  - Hacer algo hasta que se encuentre con una condición True: **whileTrue:**

# MECANISMO DE CONTROL EN SMALLTALK

## timesRepeat

- El mensaje **timesRepeat:** ejecuta un bloque de código un número específico de veces. El formato del mensaje es:

**número timesRepeat: [código]**

donde número puede ser cualquier expresión que resulte en un entero y código es un bloque de código de cero-argumento.

**Ejemplo:**



The screenshot shows a 'Playground' window with a code editor and a console. The code in the editor is:

```
"Agrega 1 a la variable x tres veces."  
|x|  
x := 2.  
3 timesRepeat: [x := x + 1].  
Transcript show:'El resultado final es:', x asString.
```

The console output shows the result of the execution:

```
El resultado final es: 5
```

A thought bubble graphic is overlaid on the console output, containing the text:

El resultado es 5

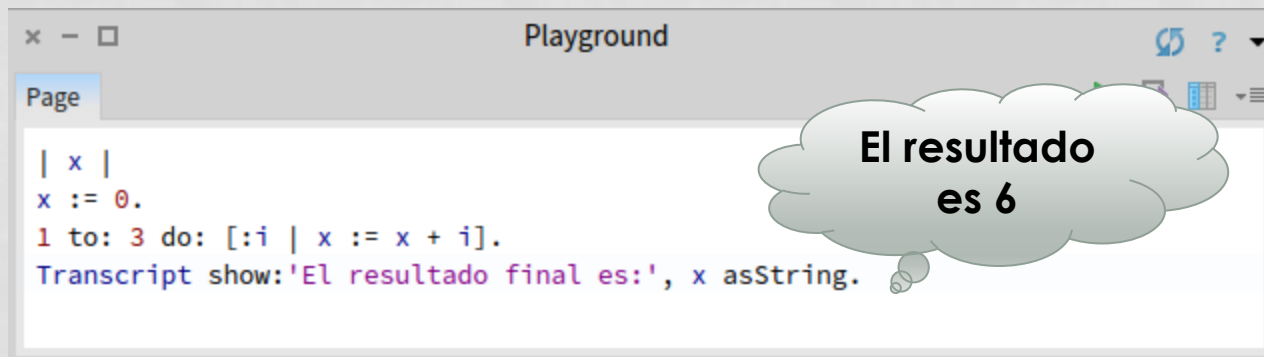
# MECANISMO DE CONTROL EN SMALLTALK

## to: do:

- El mensaje **to:do:** ejecuta un bloque múltiples veces **basado** en un valor inicial y un valor final. El formato del mensaje es:

**número1 to: número2 do: [:var | código].**

**Ejemplo:** "Ejecuta este bloque 3 veces con i referenciado a cada valor entre el rango de 1 a 3. "



The screenshot shows a 'Playground' window with a code editor and a result area. The code in the editor is:

```
| x |  
x := 0.  
1 to: 3 do: [:i | x := x + i].  
Transcript show: 'El resultado final es:', x asString.
```

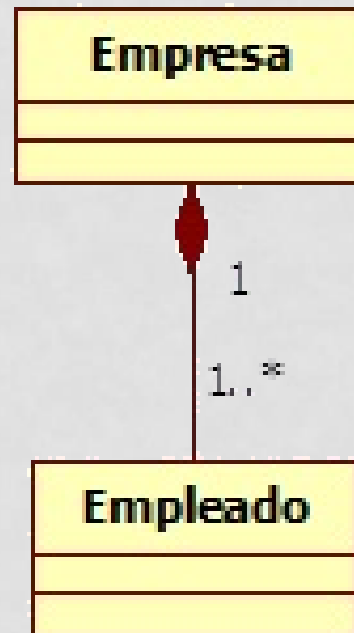
The result area on the right shows a thought bubble containing the text: "El resultado es 6".

# MECANISMO DE CONTROL EN SMALLTALK

## whileTrue: y whileFalse:

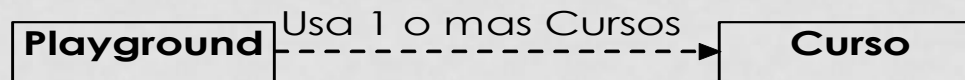
- Estos dos mensajes realizan la misma operación, excepto que uno se ejecuta por true y el otro por false.
- El formato del mensaje es:  
**[booleano] whileFalse: [código]**  
**[booleano] whileTrue: [código]**
- Un booleano puede ser cualquier expresión que resulte en un valor de true o false; debe estar encerrado en un bloque. La expresión [código] es un bloque de código de cero-argumento.

# BLOQUE 3: RELACIONES ENTRE CLASES

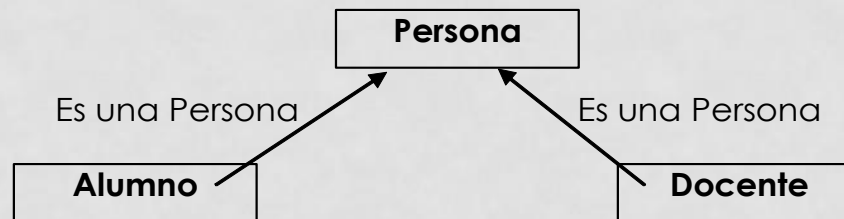


# RELACIONES ENTRE CLASES

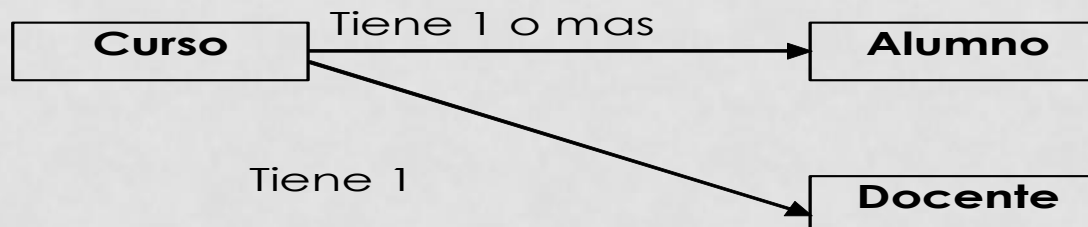
## RELACION “USA” : Conexión entre clases



## RELACION “ES UN” : Relaciones de Herencia

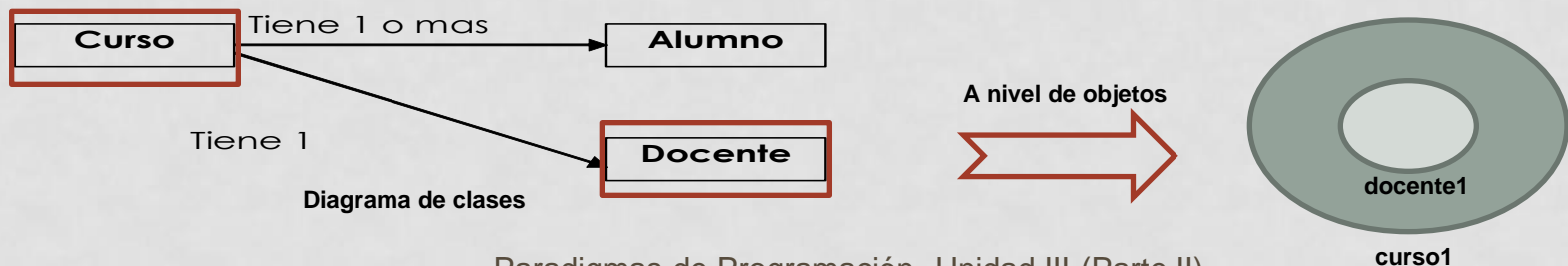


## RELACION “TIENE UN”: Relaciones de Pertenencia.



# COMPOSICIÓN- AGREGACIÓN

- Es una relación que es utilizada cuando una clase se compone de otras clases.
- Se presenta entre una clase **TODO** y una clase **PARTE**, que es componente de un **TODO**. La implementación se logra definiendo como atributo un objeto de la otra clase que es parte-de.
- Los objetos de la clase **TODO** son objetos contenedores, es decir que puede contener otros objetos. Este tipo de relación se caracteriza por la frase “**Tiene un**”. Por ejemplo:





# COMPOSICIÓN- AGREGACIÓN

- Existen dos tipos de especializaciones de esta relación entre clases:
  - Agregación:** implica una “composición débil”, si una clase se compone de otras y quitamos algunas de ellas, entonces la primera sigue funcionando correctamente. Ejemplo:



- Composición:** implica una “composición fuerte”, donde la vida de la clase contenida debe coincidir con la vida de la clase contenedora.



# IMPLEMENTACIÓN DE COMPOSICIÓN

**Object** subclass: **#Docente**

instanceVariableNames: 'legajo apellido cargo'

classVariableNames: ''

poolDictionaries: ''

category: 'ClasesPPR'

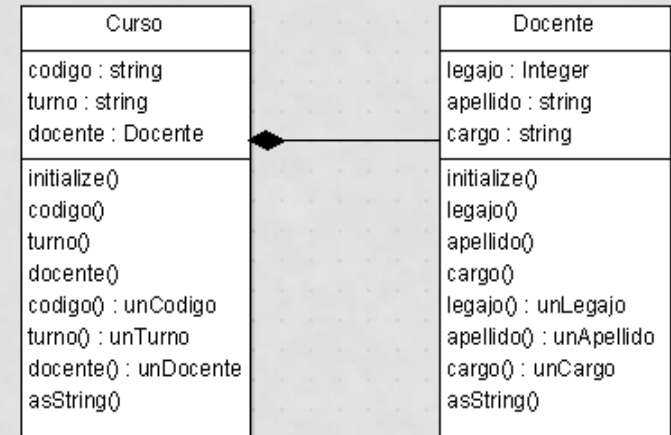
**Object** subclass: **#Curso**

instanceVariableNames: 'codigo turno **docente**'

classVariableNames: ''

poolDictionaries: ''

category: 'ClasesPPR'



# IMPLEMENTACIÓN DE COMPOSICIÓN

- Implementación de ciertos métodos en clase Curso.

**initialize**

```
codigo := ''.  
turno := ''.  
docente := nil.
```

**docente**

```
^ docente.
```

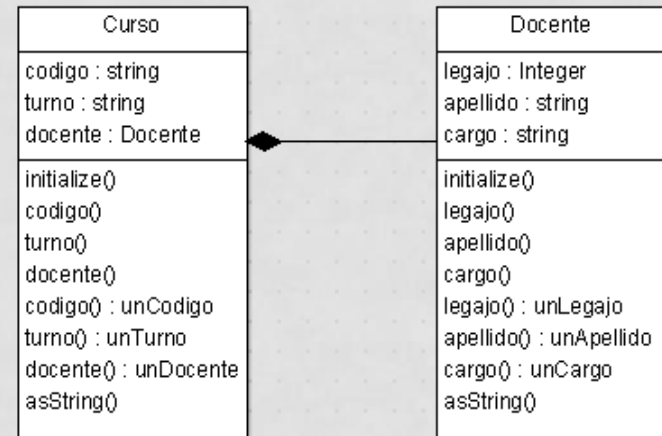
**docente:unDocente**

```
docente := unDocente.
```

**asString**

```
|datos|
```

```
datos := 'Codigo curso:', codigo asString, 'Turno:', turno asString,  
'Docente: ', docente asString.  
^ datos.
```



Se invoca el  
asString de la  
clase Docente.

# IMPLEMENTACIÓN DE COMPOSICIÓN

- Esquema de **creación** de objetos usando Composición.

#En la ventana de Playground

```
|docente1 curso1|
```

#creo objeto Docente para asignárselo a objeto curso

```
docente1 := Docente new.
```

```
docente1 legajo:12; apellido:'Pérez';cargo:'JTP'.
```

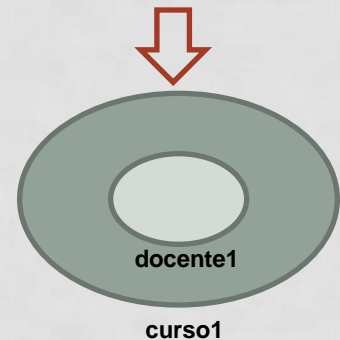
#creo una instancia de Curso

```
curso1:= Curso new.
```

```
curso1 codigo:'2K7'; turno:'Tarde';docente:docente1.
```

```
Transcript show: 'El docente del curso:', curso1 docente asString.
```

A nivel de objeto

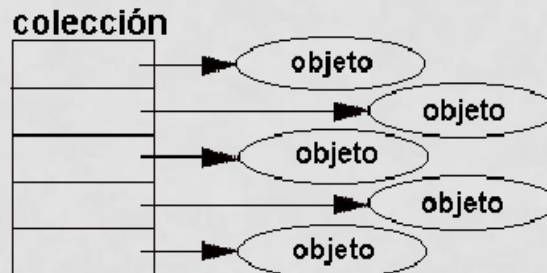


# BLOQUE 4: INTRODUCCIÓN A COLECCIONES EN SMALLTALK



# COLECCIONES EN SMALLTALK

- Una colección es una **estructura de datos** que referencia a un conjunto de objetos.
- Las colecciones **son objetos** que pueden contener un número arbitrario de objetos.
- Presentan un **protocolo unificado**.



Representación gráfica de una Colección

# TIPOS DE COLECCIONES EN SMALLTALK

- Smalltalk dispone de una variedad de colecciones:
- **Colecciones no Indexadas (no ordenadas)**
  - Tamaño Variable
    - Ejemplos: **Bag**, Set, Dictionary.
- **Colecciones Indexadas**
  - Tamaño Fijo
    - Ejemplos: Array, Symbol, Interval.
  - Tamaño Variable
    - Colecciones Ordenadas: Ejemplos: **OrderedCollection**, **SortedCollection**.

# COLECCIONES EN SMALLTALK

## Similitud en el comportamiento de las Colecciones

Las clases que representan colecciones tienen comportamientos similares, como:

- Representan un **grupo de objetos** llamados **elementos**.
- Proveen un estructura de datos básica para la programación.
- Reemplazan construcciones iterativas en los lenguajes tradicionales con los mensajes a las colecciones **do: , detect:, select: , y collect:.**
- Soportan cuatro categorías de mensajes:
  - **Agregan nuevos elementos**
  - **Eliminan nuevos elementos**
  - **Determinan ocurrencias de los elementos**
  - **Enumeran elementos**



# PRINCIPALES MÉTODOS DE LAS COLECCIONES

Nombre Mensaje	Tipo de Mensaje	Descripción
<b>new</b>	Creación de instancia	Instancia una colección, sin especificar un tamaño inicial.
<b>new: tamaño inicial</b>	Creación de instancia	Instancia una colección especificando un tamaño inicial. Válido para colecciones de tamaño fijo.
<b>with:</b>	Creación de instancia	Instancia una colección conteniendo inicialmente un objeto.
<b>add: unObjeto</b>	Incorporación de elementos	Agrega unObjeto a la colección
<b>size</b>	Consulta	Devuelve la cantidad de elementos de una colección.
<b>isEmpty</b>	Consulta	Retorna "true" si la colección esta vacía.
<b>notEmpty</b>	Consulta	Retorna "true" la colección tiene algún objeto.

# PRINCIPALES MÉTODOS DE LAS COLECCIONES

Nombre Mensaje	Tipo de Mensaje	Descripción
<b>remove: unObjeto</b>	Eliminación	Elimina el primer <i>unObjeto</i> de la colección. Si no se encuentra genera error. Válido colecciones de tamaño no fijo.
<b>remove:unObjeto</b> <b>ifAbsent: unBloque</b>	Eliminación	Elimina <i>unObjeto</i> de la colección. Si <i>unObjeto</i> no esta incluido en la colección, ejecuta unBloque.
<b>includes: unObjeto</b>	Consulta	Retorna true si la colección incluye a unObjeto
<b>ocurrencesOf: unObjeto</b>	Consulta	Retorna la cantidad de elementos de la colección que son iguales a unObjeto

# BOLSAS (BAG)

- Una Bag es una colección desorganizada de elementos.
- Guardan sus elementos en un orden al azar y **no son indexadas**.
- Puede incrementar o decrementar su tamaño (**dinámicas**).
- **Acepta elementos duplicados** (puede contener el mismo objeto varias veces).



# BOLSAS (BAG)

```
Playground

|z encontro vacio|
z := Bag new. "crea una nueva instancia de la clase Bag denominada z"
z add: 1. "Agrega el objeto uno a la bolsa z"
z add: 3. "Agrega el objeto tres a la bolsa z"
z remove: 1 ifAbsent:[Transcript show:'No se encontró'] . "Remueve el objeto 1 de la bolsa z"
Transcript show: 'Cantidad de elementos de la bolsa: ', z size asString, String cr.
encontro := z includes: 3. "Verifica la existencia del objeto 3 en bolsa, en este caso devuelve True"
Transcript show: 'Resultado de includes ', encontro asString, String cr.
vacio := z isEmpty. "devuelve false porque la bolsa no esta vacía"
Transcript show: 'Resultado de isEmpty ', encontro asString.
```

```
Transcript

Cantidad de elementos de la bolsa: 1
Resultado de includes true
Resultado de isEmpty true
```

# COLECCIONES. MENSAJES DE RECORRIDOS

- Smalltalk provee distintos mensajes para recorrer las diferentes colecciones que soporta. Algunos de ellos son:
  - **do:**
  - detect:
  - select:
  - collect:
  - reject:
  - inject:

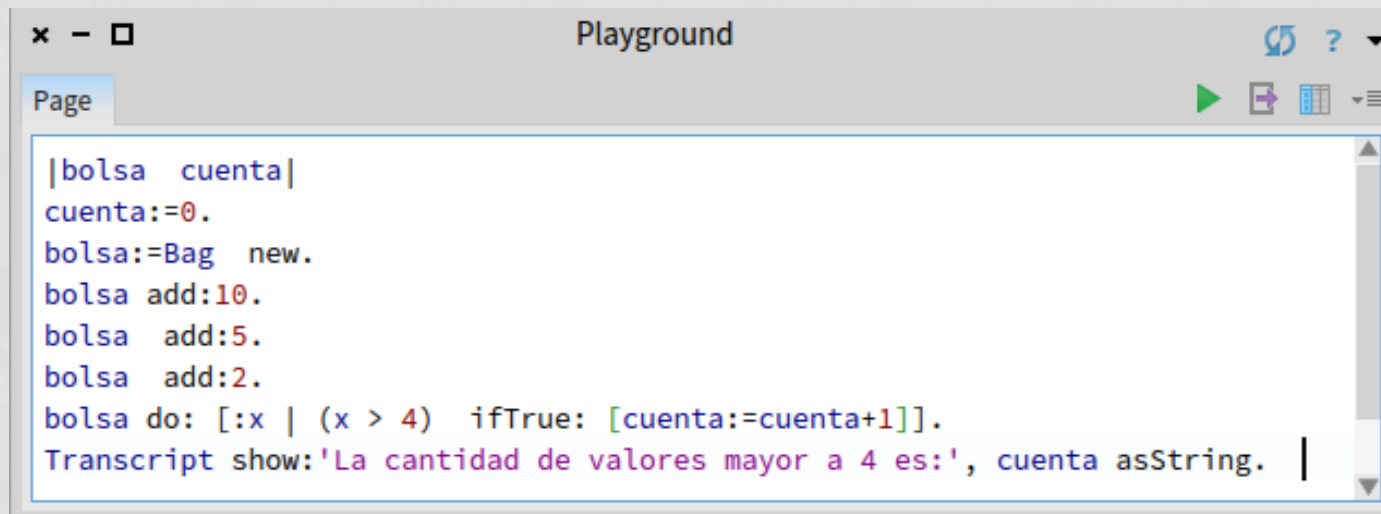
# COLECCIONES: MENSAJES RECORRIDOS

## Mensaje

### do: unBloque

Evalúa un bloque para cada elemento de la colección, pasándole a cada elemento de la colección como parámetro un bloque.

**Sintaxis:** *nombre\_coleccion do:[:x | expresiones.].*



```
|bolsa cuenta|
cuenta:=0.
bolsa:=Bag new.
bolsa add:10.
bolsa add:5.
bolsa add:2.
bolsa do: [:x | (x > 4) ifTrue: [cuenta:=cuenta+1]].
Transcript show:'La cantidad de valores mayor a 4 es:', cuenta asString.
```