



Universidad Tecnológica Nacional



FACULTAD REGIONAL CORDOBA

PARADIGMAS DE PROGRAMACION

Unidad III

Paradigma Orientado a Objetos Colecciones – Parte II

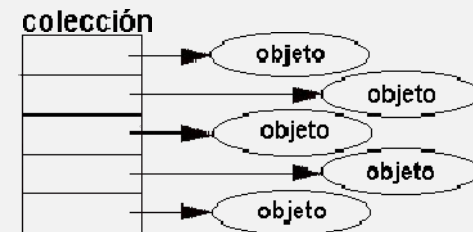


CONTENIDOS ABORDADOS

- Colecciones en Smalltalk.
 - Características de colecciones
 - Mensajes comunes a colecciones
 - Mensajes de recorridos en colecciones
 - Conversión entre colecciones
- Caso de estudio.

Colecciones en Smalltalk

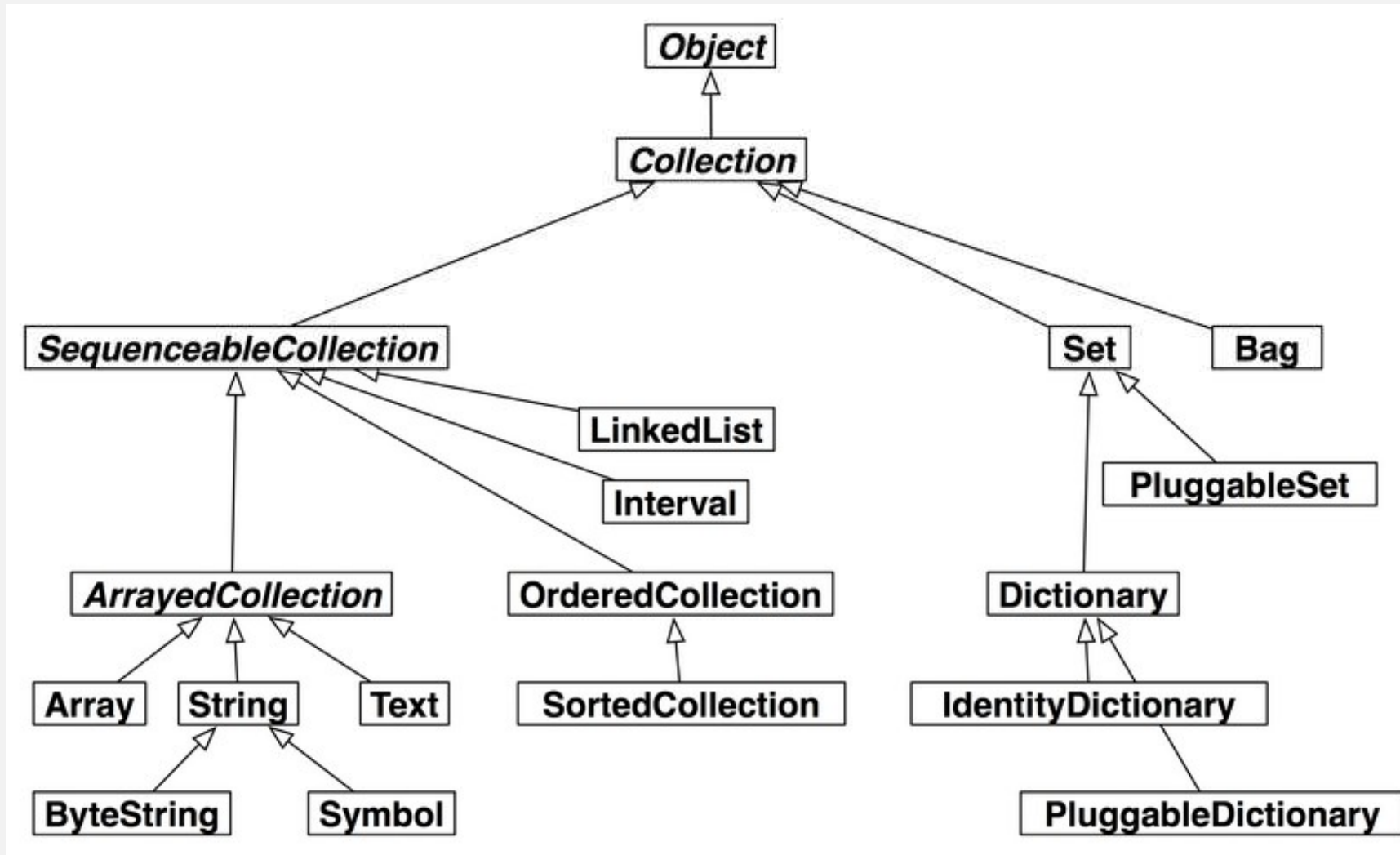
- Una colección es una estructura de datos que referencia a un conjunto de objetos.
- Una colección es un **objeto** que contiene y maneja un conjunto de objetos.
- Las operaciones lícitas que se pueden hacer con una colección:
 - Ingresos o Eliminación de elementos.
 - Recorridos, Ordenamientos.
 - Filtros de elementos.
 - Búsqueda de elementos.
 - Etc.



Colecciones en Smalltalk

- Smalltalk ofrece distintos tipos de colecciones que responden a un mismo conjunto de mensajes.
- Presentan un **protocolo unificado**, por lo que todas las diferentes formas de colecciones provistas por el entorno responden a un mismo conjunto de mensajes básicos, lo que facilita su aprendizaje y el cambio de un tipo de colección a otra.
- Son **polimórficas**, lo cual significa que objetos de cualquier clase pueden ser guardados dentro de una misma colección.

Colecciones en Smalltalk



Colecciones en Smalltalk: Clases

Nombre Colección	Características
Set(Conjunto) y Bag (Bolsa)	Son colecciones desordenadas. Las instancias Bag admiten duplicados, mientras que las de Set no. (Tamaño variable)
OrderedCollection	Es una colección indexada en la que se mantiene el orden en que los objetos son insertados. (Tamaño variable)
SortedCollection	Provee colecciones ordenadas según un criterio específico.
Dictionary	Es una colección que mantiene asociaciones entre objetos llamados claves (key) y valores (values)
Array (Vector o Arreglo)	Es una colección de “tamaño fijo” que es indexable.
String	Es similar al vector excepto que contiene caracteres.
Symbol	Es similar a una cadena excepto en que no permite objetos Symbol duplicados con el mismo valor.

Colecciones en Smalltalk

Colección	Indexación	Tam. Variable	Duplicados	Orden	Contenidos
Bag	N	S	S	N	Cualquier objeto, pero no nil.
Set	N	S	N	N	. Cualquier objeto, pero no nil.
OrderedCollection	S	S	S	N	Cualquier objeto.
SortedCollection	S	S	S	S	Cualquier objeto.
Array	S	N	S	N	Cualquier objeto.
String	S	N	S	N	Caracteres
Symbol	S	N	N	N	Caracteres
Dictionary	N	S	N	N	Clave+Cualquier objeto

Colecciones en Smalltalk: Mensajes

Nombre Mensaje	Tipo de Mensaje	Descripción
new	Creación de instancia	Instancia una colección, sin especificar un tamaño inicial.
new: tamaño inicial	Creación de instancia	Instancia una colección especificando un tamaño inicial. Válido para colecciones de tamaño fijo.
add: unObjeto	Agregación	Agrega un elemento en la colección.
remove: unObjeto	Eliminación	Elimina el primer <i>unObjeto</i> de la colección. Si no se encuentra genera error.
size	Consulta	Retorna la cantidad de elementos de una colección.
isEmpty	Consulta	Retorna "true" si la colección esta vacía.

Colecciones en Smalltalk: Mensajes

Nombre Mensaje	Tipo de Mensaje	Descripción
do: unBloque	Consulta	Recorre la colección completa elemento por elemento sin retornar ningún objeto, ejecuta el bloque de código de un argumento unBloque
select: unBloque	Consulta	Retorna una colección conteniendo todos los elementos para los cuales la evaluación del argumento unBloque es verdadera (filtro)
includes: unObjeto	Consulta	Retorna true si el objeto está contenido en la colección.
detect: unBloque1 ifNone: :unbloque2	Consulta	Retorna el primer objeto que cumple la condición de unBloque1, sino encuentra coincidencia retorna el contenido de unBloque2

Colecciones en Smalltalk

- **Ejemplo1:**

```
|unaColeccion|
```

```
unaColeccion := OrderedCollection new.
```

```
unaColeccion isEmpty           respuesta: true.
```

```
unaColeccion size              respuesta: 0
```

```
"agrega 3 elementos"
```

```
unaColeccion add:1; add:5; add:10.
```

```
unaColeccion isEmpty           respuesta: false.
```

```
unaColeccion size              respuesta: 3
```

```
unaColeccion includes:5        respuesta: true
```

```
unaColeccion includes:2        respuesta: false
```

Colecciones en Smalltalk: Recorridos

- **do: unBloque** : Es una iteración general que recorre cada elemento de la colección y ejecuta un bloque de código de un argumento, especificado por el argumento unBloque, con cada elemento de la colección como argumento de este bloque. Por ejemplo:

```
"Calcula el total de la suma de los números en  
numeros "
```

```
|numeros suma|
```

```
numeros := #(1 2 3 4 5).
```

```
suma := 0.
```

```
numeros do: [ :unNumero | suma := suma +  
unNumero].
```

```
^suma
```

```
resultado: 15
```

Colecciones en Smalltalk: Recorridos

- **detect: unBloque** : Devuelve el primer elemento en el receptor para el cual la evaluación del argumento unBloque es verdadera.

```
unObjeto:= #( 4 7 10 3 7) detect: [ :nro | nro > 7]  
ifNone: [nil].
```

Si encuentra, asigna en unObjeto, el número 10, sino nil.

- **select: unBloque** : Devuelve un subconjunto del receptor conteniendo todos aquellos elementos para los cuales la evaluación del argumento unBloque es verdadera.

```
unaColeccion := 'ahora es el momento' select:  
[:letra | letra isVowel ]
```

Devuelve la colección formada por: a o a e e o e o

Colecciones en Smalltalk: Recorridos

- **reject: unBloque** : Devuelve un subconjunto del receptor conteniendo aquellos elementos para los cuales la evaluación del argumento unBloque es falsa.

```
unaColeccion := 'ahora es el momento' reject:  
[:letra | letra isVowel ]
```

Devuelve la colección formada por: h r s l m m n t

- **collect: unBloque** : Crea y devuelve una nueva colección del mismo tamaño que la receptora. Los elementos de la nueva colección son el resultado de ejecutar el argumento unBloque en cada uno de los elementos del receptor.

```
unaColeccion := #('ahora' 'es' 'el' 'momento' 123)  
collect: [:elemento | elemento isString ]
```

*Devuelve la colección formada por: true true true
true false*

Vector (Array)

- Un Array es una colección, de tamaño fijo, de elementos que pueden ser indexados por claves de números enteros que comienzan en 1 y se incrementan.
- No puede crecer o disminuir.
- Los elementos de un Vector puede ser cualquier objeto, los objetos duplicados están permitidos, y los elementos se guardan según la posición.
- Es útil cuando se conoce el tamaño de la colección y ese tamaño cambia raramente.

```
unArray := # (1 2 3 3 3 3) .
```

ó

```
unArray := Array new: 4.
```

```
unArray add: 2; add:3; add:5; add:4.
```

Bolsas (Bag)

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

- Ejemplo :

```
unaBag := Bag new.  
unaBag add:3; add:6;add:7.  
unaBag size -> 3
```

- Ejemplo a partir de un vector:

```
unaBag := #(1 2 3 3 3 3) asBag.  
unaBag size -> 6  
unaBag occurrencesOf: 3           resultado: 4
```

Conjuntos (Set)

- Un Set es similar a una Bolsa excepto porque no permite objetos duplicados.
- Ignora cualquier petición que pudiera agregar un elemento duplicado a la colección.
- Asegurarse que no hay duplicados agrega un costo cada vez que un objeto es añadido a la lista.
- Por ejemplo:

```
unSet := #(1 2 3 3 3 3) asSet.
```

```
unSet size -> 3
```

```
unSet occurrencesOf: 3      Resultado: 1
```


Colección Organizada (OrderedCollection)

- Ofrece el protocolo más completo de cualquiera de las clases de colecciones.
- Puede ser indexada por una clave de números enteros que comienzan en 1 y crecen.
- Los elementos pueden ser objetos de cualquier clase.
- Permite objetos duplicados y *mantiene el orden* en que los objetos son insertados.
- Esta colección es principalmente utilizada para listas de tamaño variable que requieren de un control sobre la ubicación de los elementos de la colección.

Colección Ordenada (SortedCollection)

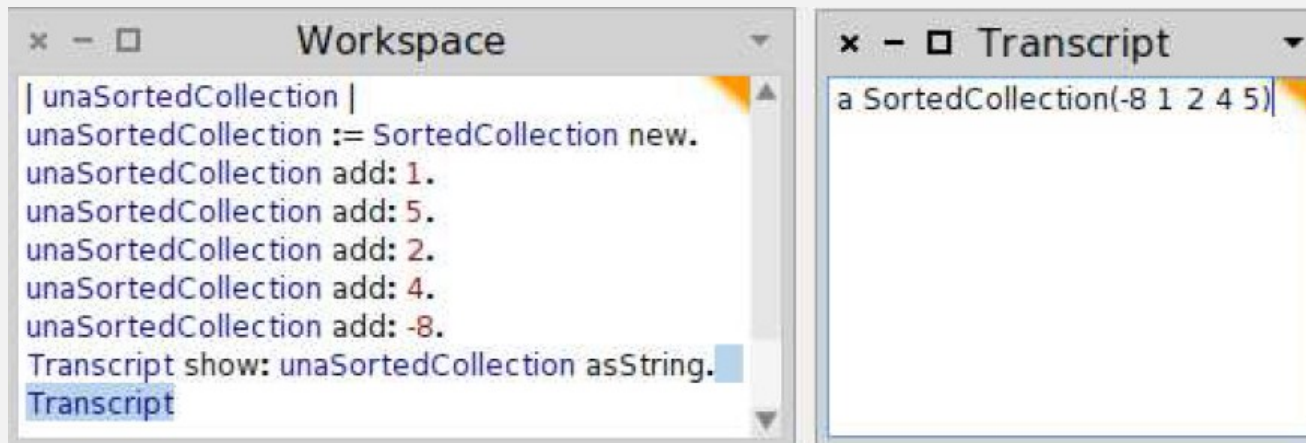
- Guarda objetos en el orden especificado por un bloque de código llamado **sortBlock**.
- El bloque sortBlock es un bloque de dos argumentos que indica el orden en que los dos argumentos debieran ser almacenados en la colección respecto uno del otro.
- La colección guarda el primer argumento adelante del segundo argumento en la colección cuando la evaluación del bloque sortBlock es verdadera (true).
- El bloque sortBlock puede contener múltiples sentencias, pero debe retornar un valor de verdad: true o false.

Colección Ordenada (SortedCollection)

- Si se instancia una **SortedCollection** con la siguiente expresión:

`SortedCollection new.`

- Se obtiene una colección que ordena sus elementos de **menor a mayor**. Implica que los objetos contenidos utiliza el criterio de ordenamiento por defecto. Ejemplo:



The image shows two side-by-side windows from a software development environment. The left window, titled 'Workspace', contains a sequence of commands: creating a variable 'unaSortedCollection' as a new 'SortedCollection', adding elements 1, 5, 2, 4, and -8 to it, and then displaying the collection as a string in the transcript. The right window, titled 'Transcript', shows the output of the last command: 'a SortedCollection(-8 1 2 4 5)', where the elements are sorted in ascending order.

```
Workspace
| unaSortedCollection |
unaSortedCollection := SortedCollection new.
unaSortedCollection add: 1.
unaSortedCollection add: 5.
unaSortedCollection add: 2.
unaSortedCollection add: 4.
unaSortedCollection add: -8.
Transcript show: unaSortedCollection asString.
Transcript

Transcript
a SortedCollection(-8 1 2 4 5)
```

Colección Ordenada (SortedCollection)

- Para ordenar los elementos de la colección con un criterio en particular, se necesita el mensaje “**sortBlock**”, que es un bloque (objeto de la clase BlockClosure).
- Para especificar un algoritmo de ordenamiento diferente, la colección debe instanciarse con la siguiente expresión:

SortedCollection sortBlock: unBloqueBinario



The image shows two windows from a Smalltalk environment. The 'Workspace' window on the left contains the following code:

```
[unBloque unaColeccion]
unBloque := [:x :y | x >= y ].
unaColeccion := SortedCollection sortBlock: unBloque.
unaColeccion add:1.
unaColeccion add:5.
unaColeccion add:2.
unaColeccion add:4.
unaColeccion add:-8.
Transcript show: unaColeccion asString. Transcript
```

The 'Transcript' window on the right shows the output of the last line of code:

```
a SortedCollection(5 4 2 1 -8)
```

Colección Ordenada (SortedCollection)

- Para crear una colección ordenada que posea objetos de alguna clase en particular, se deberá establecer el criterio de orden en el mensaje **sortBlock**, a través:
 - Mensajes que los objetos puedan responder
 - Sobrecarga de los operadores en los objetos
- Por ejemplo si se quiere almacenar los productos ordenados por precio de mayor a menor, los mismos deben responder al mensaje precio para poder hacer la comparación:

```
coleccionOrdenada := SortedCollection sortBlock:  
  [:unProd :otroProd | unProd precio > otroProd  
    precio].
```

- Con el mismo ejemplo, pero teniendo los operadores > y <= sobrecargados con el criterio de precio en la clase, se puede hacer lo siguiente:

```
coleccionOrdenada := SortedCollection sortBlock:  
  [:unProd :otroProd | unProd > otroProd].
```

Diccionarios (Dictionary)

- Sus elementos están formados por pares de claves y valores.
- Los objetos que referencia esta colección son instancias de una clase particular denominada ***Association***.
- En una *Association* se tienen dos objetos, uno que asume el rol de clave y otro de valor.
- Un diccionario es similar a una lista enlazada, donde el acceso de elementos no se hace por un índice entero, sino por un objeto.
- Tienen mensajes de recorridos especiales:

associationsDo: unBloque

associationsSelect: unBloque

keysDo: unBloque.. **queBinario**

Diccionarios: Métodos principales

	Mensaje
at:unaClave put:unValor	Inserta <i>unvalor</i> en el diccionario y lo asocia a una <i>unaClave</i> .
removeKey:unaClave	Elimina un valor asociado a <i>unaClave</i> . Si una clave no figura en el diccionario produce un error.
removeKey:unaClave ifAbsent:unBloque	Elimina un valor asociado a <i>unaClave</i> . Si una clave no figura en el diccionario ejecuta <i>unBloque</i> .
at:unaClave ifAbsent:unBloque	Devuelve el valor asociado a unaClave. Si la clave no se encuentra ejecuta unBloque.

Diccionarios: Ejemplo

```
Playground

| unDiccionario |
unDiccionario := Dictionary new.
unDiccionario at: #Auto put: #Car;
at: #Mesa put: #Table;
at: #Perro put: #Dog;
at: #Gato put: #Cat;
at: #Azul put: #Blue.
Transcript show: 'Elementos de la colección: ', String cr, unDiccionario asString.
```

```
Transcript

Elementos de la colección:
a Dictionary(#Auto->#Car #Azul->#Blue #Gato->#Cat #Mesa->#Table #Perro->#Dog )
```


Conversiones entre colecciones

- Todas las colecciones entienden una serie de mensajes que permiten obtener distintos tipos de colecciones con los mismos elementos que la colección receptora. Estos mensajes son de la forma “**as{ColeccionQueQuiero}**”.
- Ejemplos:
- Si tuviese una colección de la clase Bag, y se quiere sacarle los repetidos:

```
sinRepetidos := miBag asSet.
```

- Si tuviese un array, y se quiere convertir en una colección de tamaño variable, se podría hacer:

```
coleccionVariable := miVector asOrderedCollection.
```

Conversiones entre colecciones

- Si quisiera ordenar mi carrito de compras del producto más caro al más barato, haría algo como:

```
ordenadosPorPrecio := miCarrito asSortedCollection:  
[:unProd :otroProd | unProd precio > otroProd  
  precio].
```

- También está el mensaje `asSortedCollection` sin parámetro que ordenará los elementos por el “orden natural”, dado por el mensaje `<`. En este caso, se debe tener en cuenta que todos los objetos que se agreguen a esta colección deberán entender el mensaje `<`, sino se produciría un error de mensaje no entendido.