



Universidad Tecnológica Nacional
Facultad Regional Córdoba

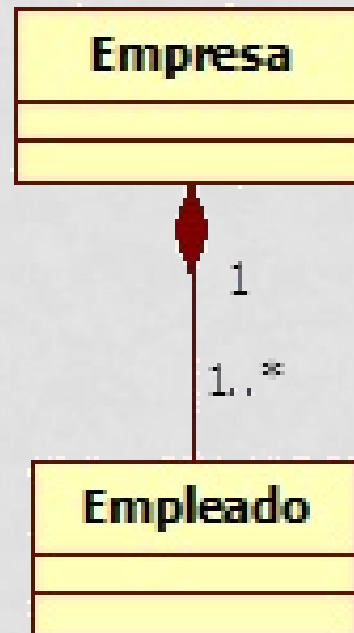
Paradigmas de Programación

UNIDAD NRO. 3
PARADIGMA ORIENTADO A OBJETOS
(PARTE III)

CONTENIDOS ABORDADOS

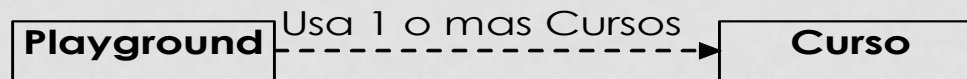
- Revisión de Relaciones entre clases
- Concepto de Herencia
- Herencia en Smalltalk
 - Introducción
 - Herencia basada en Clases
 - Modelos de Implementación
 - Implementación de Herencia en Smalltalk
 - Herencia de Variables y Métodos
 - Clases y Métodos Abstractos
- Colecciones en Smalltalk (Continuación)
 - Jerarquía de Colecciones en Smalltalk
 - Colección Set y OrderedCollection
 - Mensaje de recorrido para colecciones (Continuación)
 - select

BLOQUE 1: REPASO DE RELACIONES ENTRE CLASES

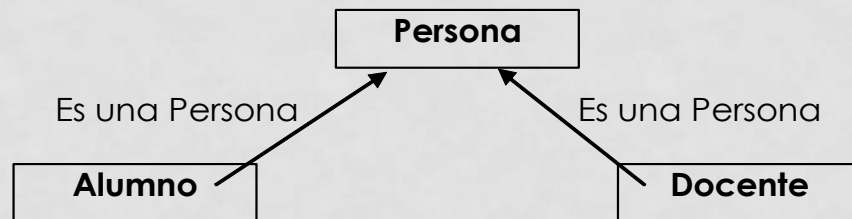


RELACIONES ENTRE CLASES

RELACION “USA” : Conexión entre clases



RELACION “ES UN” : Relaciones de Herencia



RELACION “TIENE UN”: Relaciones de Pertenencia.

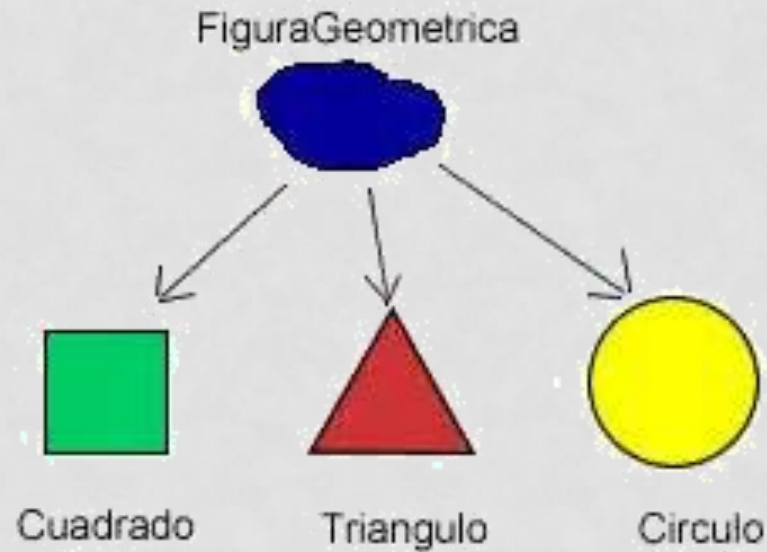


COMPOSICIÓN- AGREGACIÓN

- 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:



BLOQUE 2: HERENCIA



HERENCIA: INTRODUCCIÓN

- Es una característica clave en los sistemas orientados a objetos.
- La herencia es un mecanismo que permite la definición de una clase a partir de la definición de otra/s ya existente/s.
- Por el cual los objetos comparten conocimiento común, atributos y comportamientos.
- Permite la reusabilidad.

HERENCIA: INTRODUCCIÓN

- Las estrategias para implementarla, pueden ser:
 - **Sistemas basados en clases:** Mediante una clase (molde, plantilla) es posible definir la estructura y el comportamiento de un conjunto de objetos. Lenguajes: Smalltalk, Haskell, C++, Java, entre otros.
 - **Prototipos:** La herencia se obtiene a través de la clonación de objetos ya existentes, que sirven de prototipos, extendiendo sus funcionalidades. Lenguajes:: Self, JavaScript, entre otros.

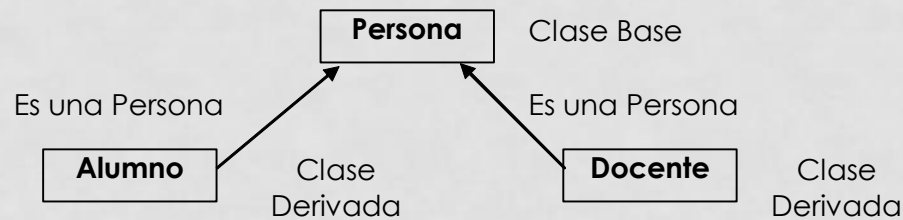
HERENCIA: BASADA EN CLASES

- Las clases se organizan jerárquicamente, y una nueva clase puede reutilizar la estructura y comportamiento de otras previamente definidas.
- Permite la reutilización de código, a través de la reutilización de los atributos y métodos de otras clases.



HERENCIA: BASADA EN CLASES

- Formada por una clase llamada base, padre o superclase y una clase llamada derivada, hija, subclase, descendiente, etc. que hereda las características y comportamientos de la clase base.

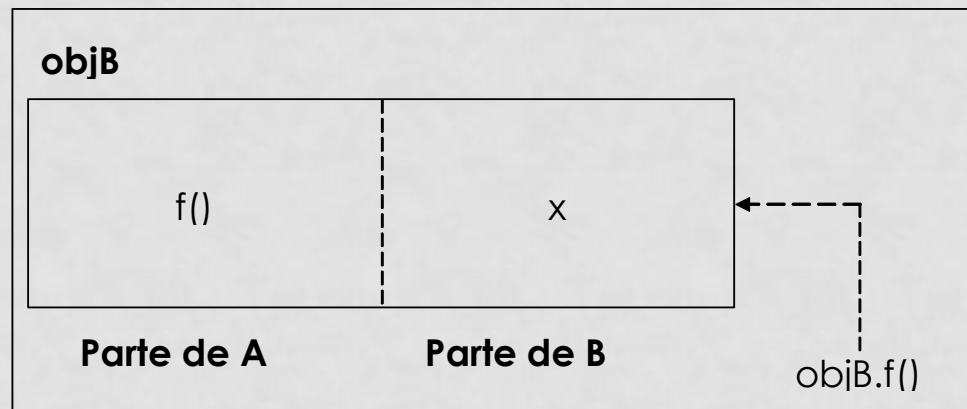


- Las clases se organizan mediante ***“jerarquías de clases”***.

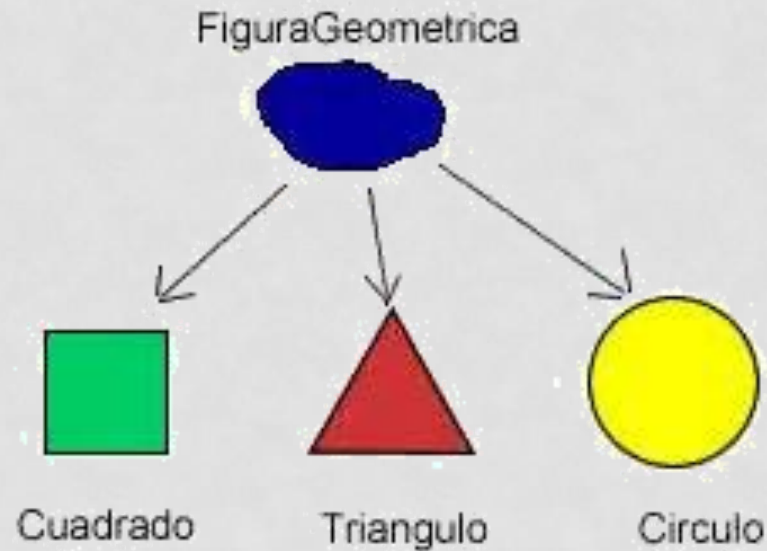


HERENCIA: MODELOS DE IMPLEMENTACIÓN

- **Herencia por Concatenación:** El objeto está compuesto por las partes que define cada una de las clases de las que hereda. Todos los atributos y métodos heredados están disponibles en el mismo objeto.

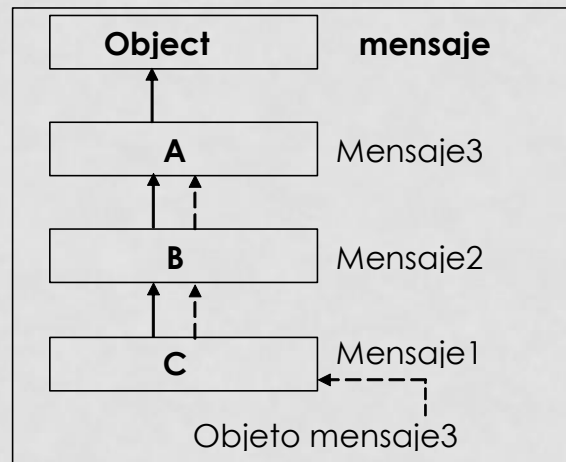


BLOQUE 3: HERENCIA EN SMALLTALK



HERENCIA EN SMALLTALK

- Las subclases heredan atributos y métodos de sus clases base o superclase.
- La subclase puede añadir más variables de instancia pero no puede eliminar ninguno de los atributos heredados.
- Implementa solo “**Herencia Simple**”.
- Utiliza el modelo de implementación de herencia por “**Delegación**”.



HERENCIA EN SMALLTALK

- Sintaxis para crear una jerarquía:

#Clase Base

Object subclass: **#Persona**

instanceVariableNames: 'dni nombre telefono'

classVariableNames: ''

poolDictionaries: ''

category: 'ClasesPPR'

#Clase Derivada

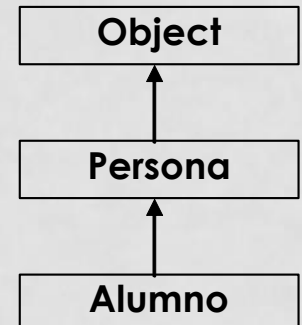
Persona subclass: **#Alumno**

instanceVariableNames: 'legajo promedio'

classVariableNames: ''

poolDictionaries: ''

category: 'ClasesPPR'



HERENCIA EN SMALLTALK

Super

- El uso de **super** provoca que la búsqueda del método comience en la superclase del objeto receptor.
- Cuando **super** es encontrado en la ejecución de un programa, Smalltalk busca el método en la superclase del receptor.
- Ejemplo:

super unMensaje. 'invoca a un mensaje de la clase base'

self unMensaje 'invoca a un mensaje de la propia clase'

HERENCIA EN SMALLTALK

Inicialización de atributos en una clase derivada

- En el método de inicialización de la clase inicializa en primer lugar los atributos heredados de su superclase y seguidamente las variables de instancia propias.
- Ejemplo método *initialize* de la clase Alumno:

initialize

“Inicializa atributos de la clase base”

super initialize.

“Inicializa atributos propios”

legajo:= 0.

promedio:= 0.

HERENCIA EN SMALLTALK

Herencia de Variables

- **Variables de Instancia:** Cada subclase tiene su propia copia, de esta forma hace a las variables de instancia propias de la instancia. Las variables de instancia de clase soportan el comportamiento de herencia, teniendo cada clase su propio estado de la variable.
- Por ejemplo: el objeto alumno tiene las variables de instancia: dni, nombre, teléfono, legajo y promedio. Estas variables incluyen las variables de instancia heredadas de Persona.

Instancia de Alumno



HERENCIA EN SMALLTALK

Herencia de Variables

- **Variables de Clase:** Una variable de clase es una variable que es compartida por todos los objetos de la clase.
- Sólo existe una copia de la variable de clase en la jerarquía local de clases. Todos los objetos de la clase referencian a esta única copia de la variable.
- Las variables de clase también permiten compartir la información a través de todas las subclases de la clase en que fueron declaradas.

HERENCIA EN SMALLTALK

Herencia de Métodos

- La herencia de métodos es útil para permitir a una clase modificar su comportamiento respecto de su superclase. Esto puede ser hecho “agregando nuevos métodos”, o *“redefiniendo los métodos heredados”*.
- **Agregando Métodos:**
Se puede agregar métodos de manera muy simple, incorporándolos en la definición de la clase. Todo objeto de la subclase soporta los métodos de su superclase, más los nuevos métodos.

HERENCIA EN SMALLTALK

Herencia de Métodos

- **Redefinición de Métodos**

- Una subclase puede redefinir (volver a definir) algún método existente en la superclase, con el objeto de proveer una implementación diferente, por ejemplo: método **asString**.
- Para redefinir un método en la subclase, se tiene que declarar con la misma signatura (nombre y parámetros).
- En la invocación de métodos, si existen dos métodos con la misma signatura, uno en la subclase y otro en la superclase, se ejecutará siempre el de la subclase.

HERENCIA EN SMALLTALK

Clases Abstractas

- Son clases genéricas que sirven para agrupar clases del mismo género.
- Definen comportamientos que se implementarán en las subclases.
- No se pueden instanciar, es decir no se pueden crear objetos a partir de ellas, ya que representan conceptos tan generales que no se puede definir como será la implementación de los mismos.
- Smalltalk no posee mecanismos formales para implementarlas, pero el desarrollador puede hacerlo definiendo métodos sin implementación (abstractos).

HERENCIA EN SMALLTALK

Métodos abstractos. Forma de Implementación en Smalltalk

- **Ejemplo:**

Clase abstracta - Método abstracto:

size

self subclassResponsability.

o

size

“sin implementación”

Clase derivada - Método implementado:

size

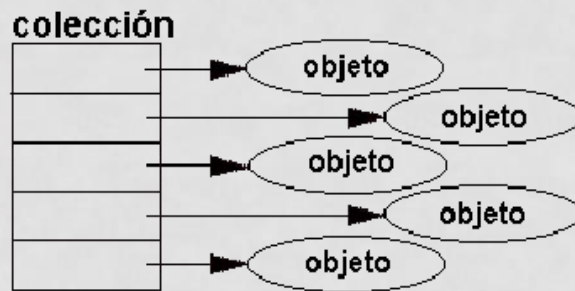
^size

BLOQUE 4: COLECCIONES EN SMALLTALK (CONTINUACIÓN)



COLECCIONES EN SMALLTALK

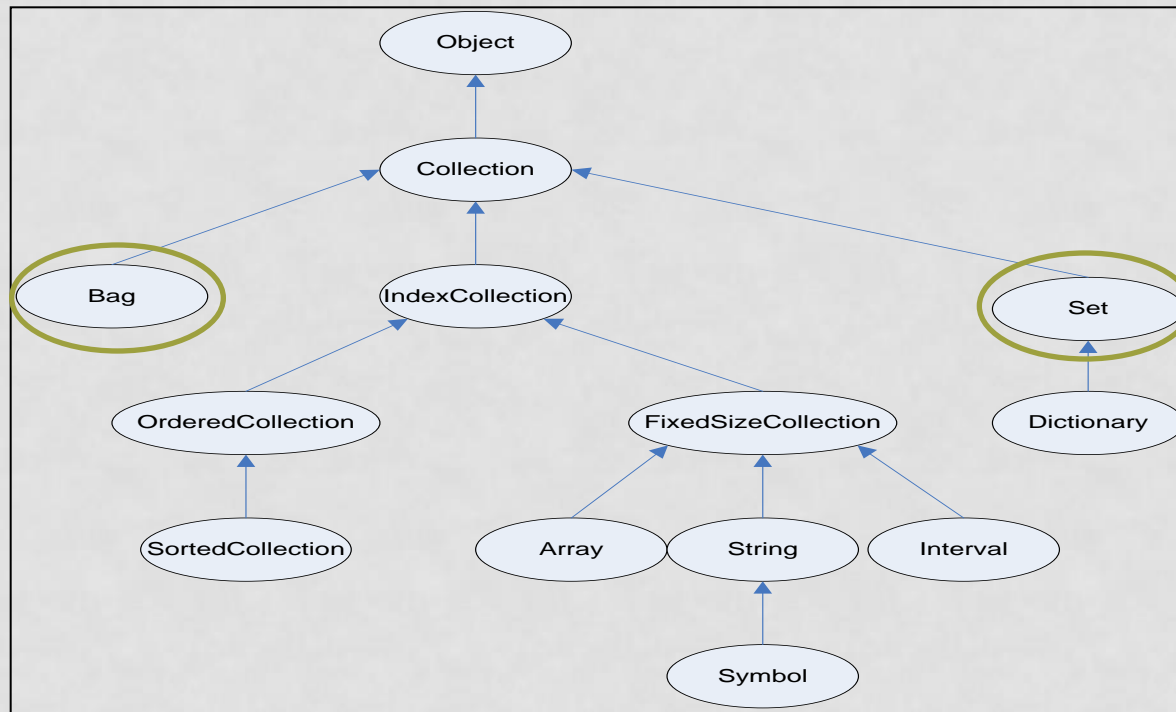
- Una colección es una **estructura de datos** que referencia a un conjunto de objetos.
- Las colecciones **son objetos** que pueden contener un número arbitrario de objetos.
- Presentan un **protocolo unificado**.



Representación gráfica de una Colección

COLECCIONES EN SMALLTALK

El siguiente cuadro muestra la jerarquía de colecciones disponibles en Smalltalk:



En esta jerarquía existen clases abstractas y concretas.

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). EJEMPLOS DE MANEJO

```
Playground
Page
|z encontro vacio|
z := Bag new.
z add:1.
z add:3.
z add:1.
"remove: elimina el primer objeto que encuentra en la colección"
z remove:1 ifAbsent:[Transcript show:'No se encontro elemento a borrar..'].
Transcript show:'La cantidad de elementos de la bolsa es:', z size asString, String cr.
encontro := z includes:3. "Verifica si 3 esta incluida en la bolsa"
Transcript show:'Resultado retornado por includes:', encontro asString, String cr.
vacio := z isEmpty. "Verifica si la bolsa esta vacía"
Transcript show:'Resultado de isEmpty:', vacio asString. |
```

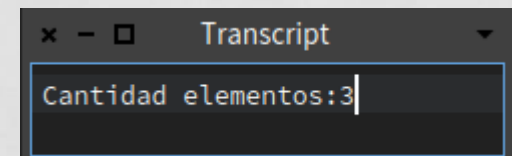
```
Transcript
La cantidad de elementos de la bolsa es:2
Resultado retornado por includes:true
Resultado de isEmpty:false|
```

CONJUNTOS (SET)

- Un Set es similar a una Bolsa (Bag) 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. Ejemplo:



```
|cant unSet|
unSet := Set new.
unSet add:1; add:2; add:3;add:1.
cant:=unSet size.
"Retornaría: 3 elementos"
Transcript show:'Cantidad elementos:', cant asString.
```



```
Cantidad elementos:3
```

MENSAJES DE RECORRIDOS EN COLECCIONES

- **do:**
- detect:
- **select:**
- collect:
- reject:
- inject:

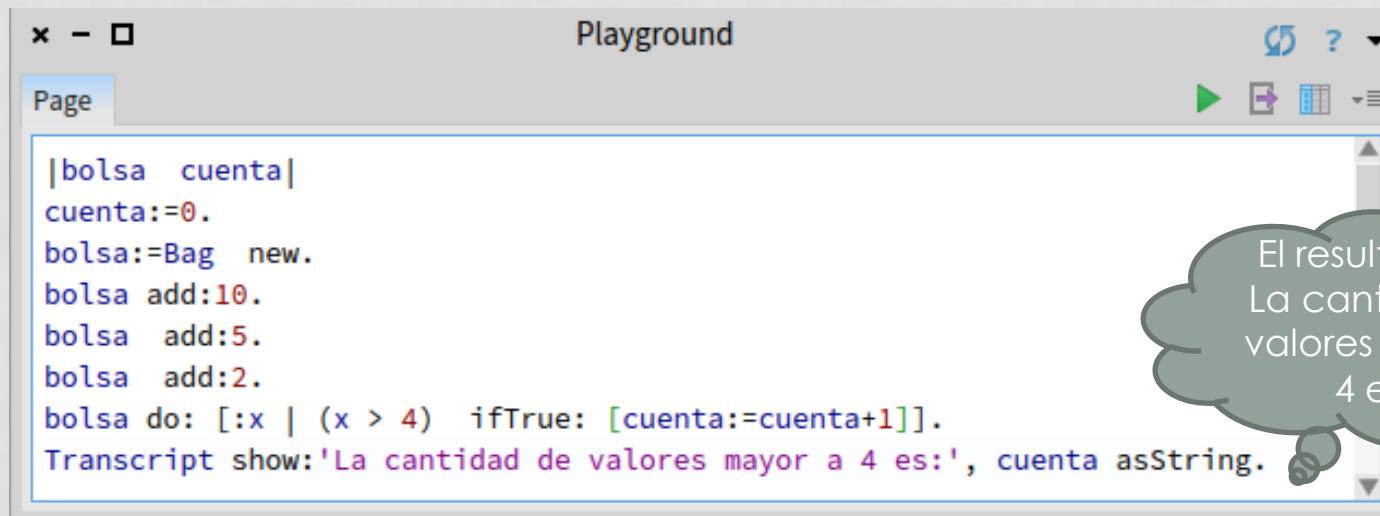
COLECCIONES: MENSAJES RECORRIDOS

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.
```

El resultado es:
La cantidad de
valores mayor a
4 es: **2**

COLECCIONES: MENSAJES RECORRIDOS

Mensaje

select: unBloque

Evalúa un bloque para cada elemento de la colección, retornando una nueva colección con los elementos que hicieron unBloque evaluará en true.

Sintaxis: `retorno:=nombre_coleccion select:[x | expresiones].`

```
Playground
Page
|conjunto1 retorno|
retorno:=nil.
conjunto1:=Set new.
conjunto1 add:1;add:5;add:7;add:8.
"Filtrar los numeros(objetos) que sean pares"
"Y retorna una colección (Set) del mismo tipo de colección que recibe el
mensaje, con los objetos que cumplen condición"
retorno:=conjunto1 select:[x|x even].
Transcript show:'Objetos pares de la colección par:', retorno asString.
```

El resultado es:
Objetos pares
de la colección
par: **a Set(8)**

“TIPS PARA USO ADECUADO DE MENSAJES RECORRIDOS”

- El mensaje **do:** se utiliza mayormente cuando necesitamos que “todos los elementos” de la colección se genere un determinado efecto.
 - El mensaje do: **NO tiene retorno.**
 - Dentro del bloque del mensaje do: (sección de expresiones) puede haber mas de una expresión y el uso de condicionales en caso necesario.

“TIPS PARA USO ADECUADO DE MENSAJES RECORRIDOS”

- El mensaje **select**: se utiliza mayormente cuando necesitamos buscar elementos (“más de uno”) que cumplen un determinado criterio.
- **Recordar:** El mensaje select: *“siempre tiene retorno”* y es una colección del mismo tipo que la colección receptora del mensaje de recorrido. En caso de que ningún objeto cumpla la condición lo retornado es “nil”.
- Dentro del select: solo se expresa la condición o criterio por el cual se efectúa la búsqueda.

“TIPS PARA USO ADECUADO DE MENSAJES RECORRIDOS”

- El mensaje **select**: se utiliza mayormente cuando necesitamos buscar elementos (“más de uno”) que cumplen un determinado criterio.
- **Recordar:** El mensaje select: *“siempre tiene retorno”* y es una colección del mismo tipo que la colección receptora del mensaje de recorrido. En caso de que ningún objeto cumpla la condición, lo retornado es una colección del mismo tipo de la colección receptora del mensaje select:, pero vacía.
- Dentro del select: solo se expresa la condición o criterio por el cual se efectúa la búsqueda.