Paradigmas de Programación

Programación Orientada a Objetos

Lenguaje Smalltalk

Smalltalk

- SMALLTALK es un lenguaje orientado a objetos puro, porque todas las entidades que maneja son objetos.
- Se basa en conceptos como objetos y mensajes, encapsulación, Interfaz, Polimorfismo, Clases, Herencia, etc.
- SMALLTALK es mucho más que un lenguaje de programación, es un ambiente completo de desarrollo de programas. Integra consistentemente un editor, un compilador, un debugger, utilitarios de impresión, un manejador de código fuente y una completa librería de clases.

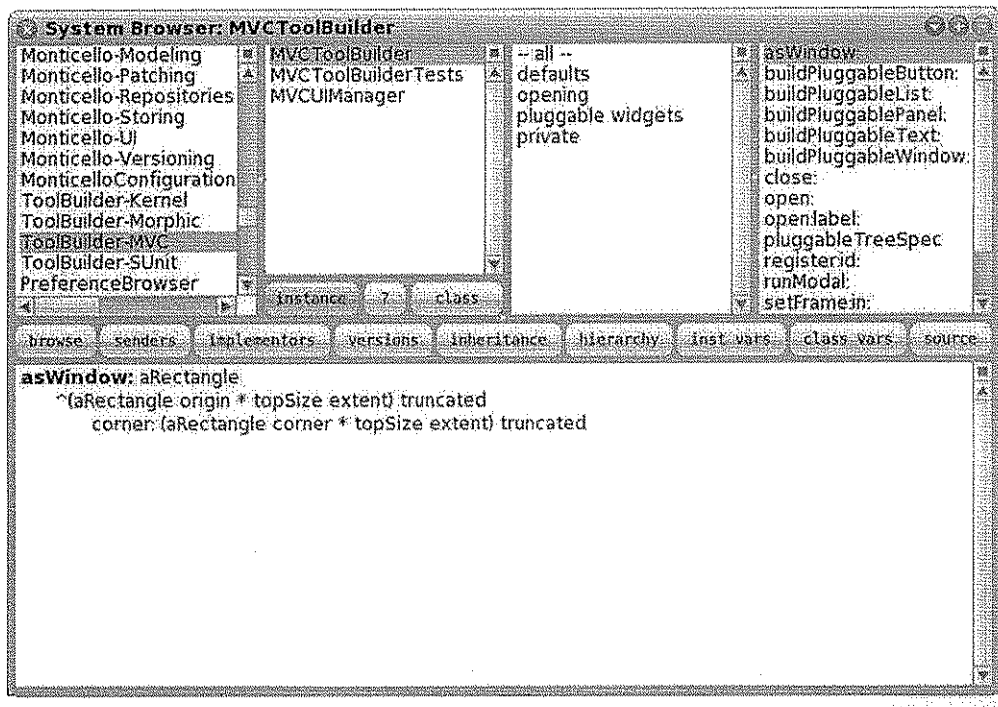
Smalltalk

- **La programación en SMALLTALK requiere de al menos los siguientes conocimientos:**
  - Diseñar nuevas aplicaciones SMALLTALK, requiere de conocimientos sobre:
    - los conceptos fundamentales del lenguaje: manejo de clases y objetos, mensajes, clases y herencia.
    - la sintaxis y la semántica del lenguaje.
    - las clases fundamentales del sistema, tales como numéricas, colecciones, gráficas y las clases de interfase del usuario.

Smalltalk

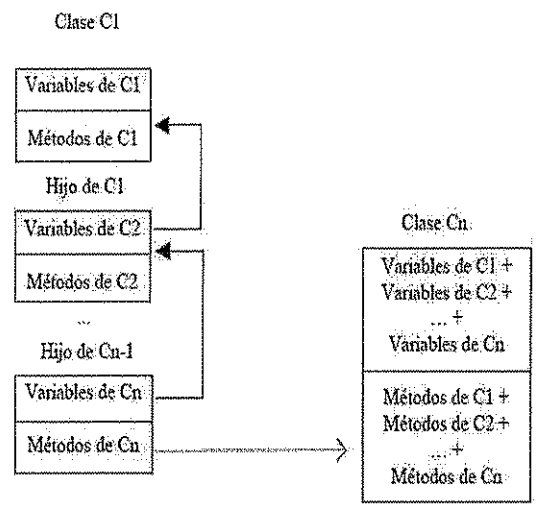
- La programación en SMALLTALK se denomina a veces **"Programación por extensión"**
- Las nuevas aplicaciones son construidas por extensión de las librerías de clases de SMALLTALK.
- La programación en SMALLTALK consiste en:
  - Crear clases.
  - Crear instancias.
  - Especificar la secuencia de mensajes entre objetos.

Smalltalk – Librería de Clases



Smalltalk - Herencia

- Permite crear nuevas clases partiendo de otras previamente definidas con características semejantes a la que se quiere crear.
- El mecanismo de herencia nos permite definir las propiedades particulares de un nuevo objeto y heredar las propiedades comunes ya existentes.
- En SMALLTALK, todas las clases heredan de una única clase llamada Object.
- SMALLTALK maneja sólo HERENCIA SIMPLE.



Smalltalk - Polimorfismo

- El mismo mensaje puede ser interpretado de diferentes formas por diferentes objetos.
- Ejemplo:
- 5 + 100
- (200 @ 200) + 100
- Ambos usan el mensaje + 100, pero los objetos receptores reaccionan al mensaje de maneras muy diferentes.
- En el primer ejemplo el receptor es el objeto **entero 5** y el selector **+** es interpretado como la **suma de enteros.**

Smalltalk - Polimorfismo

- En el segundo ejemplo el receptor es el objeto **punto (x, y)** con coordenadas (200, 200).
- En esta expresión el selector **+** es interpretado como la **suma definida sobre puntos**. Retorna el punto (300,300)
- **Es el receptor del mensaje el que determina como es interpretado el mensaje**

Smalltalk - Polimorfismo

- El método suma realmente invocado por una expresión tal como unObjeto + 100 es determinado por el tipo de objeto que recibe el mensaje en el **tiempo de ejecución**.
- Esto se denomina **vinculación dinámica**.
- Cuando el mismo selector es aceptado por clases diferentes de objetos, se dice que el selector está **sobrecargado**.

Smalltalk - Polimorfismo

- x **between:** 2 **and:** 4. mensaje **between:and:** enviado a un
- objeto carácter.
- x **between:** 2@4 **and:** 12@14. mensaje **between:and:** enviado a un
- objeto punto.
- 'carbon' **between:** 'carbohidrato'
- **and:** 'carbonato'.
- mensaje **between:and:** enviado a un
- objeto cadena.
- mensaje **between:and:** enviado a un objeto entero.

Smalltalk - Polimorfismo

x **between:** 2 **and:** 4.	mensaje **between:and:** enviado a un objeto entero.
x **between:** $a **and:** $z.	mensaje **between:and:** enviado a un objeto carácter.
x **between:** 2@4 **and:** 12@14.	mensaje **between:and:** enviado a un objeto punto.
'carbon' **between:** carbohidrato' **and:** 'carbonato'.	mensaje **between:and:** enviado a un objeto cadena.

Elementos léxicos

- **Componentes léxicos**
- **Comentarios:** Todo texto entre comillas dobles (")
- Solo hay comentarios de bloque, no existen comentarios de línea.
- **Identificadores**: En Smalltalk se puede hablar de identificadores para nombres de variables e identificadores para nombres de clases.
- Los identificadores para nombres de variables comienzan con una letra y deben ir seguidas de cero o más letras o dígitos.
- Las variables temporales y de instancia deben comenzar por una letra minúscula
- Las variables globales y compartidas deben comenzar con letra mayúscula.

Elementos léxicos

- **Componentes léxicos**
- **Pseudovariables:** Una pseudo-variable es un identificador que referencia a un objeto. La diferencia con las variables normales es que no se pueden asignar y siempre aluden al mismo objeto.
- Esto es, el valor de una pseudo-variable no puede modificarse con una expresión de asignación.

- **nil.** Referencia a un objeto usado cuando hay que representar el concepto de 'nada' o de 'vacío'. Las variables que no se asignaron nunca, referencian a nil
- **true.** Referencia a un objeto que representa el verdadero lógico.
- **false.** Referencia a un objeto que representa el falso lógico.
- **self.** Referencia al receptor del mensaje.
- **super.** Referencia al receptor del mensaje, pero indica que no debe usarse la clase del receptor en la búsqueda del método a evaluar. Se usa, sobre todo, cuando se especializa un método en una subclase y se quiere invocar el método de la superclase.
- **thisContext.** Referencia al objeto contexto-de-ejecución que tiene toda la información referente a la activación del método.

Elementos léxicos y sintácticos

- **Componentes léxicos**
- **Símbolos**: Los identificadores para nombres de clases se conocen como símbolos.
- Patrón: Son una secuencia de caracteres alfanuméricos precedidos por el carácter '#'.
- #simbolo.
- #B450.
- #region.
- Algunos métodos
- #Region displayString.
- #Region hash.
- #Region hashCharacters.

Elementos léxicos y sintácticos

- **Componentes léxicos**
- **Números:** Representan valores numéricos que pueden ser instancias de las clases Number (Integer (SmallInteger, LargeInteger), Fraction o Float)
- Patrón: Secuencia de dígitos precedidos o no de un signo '-' y/o con un punto decimal.
- Los números de la clase **fraction** pueden iniciar por – o no, e ir seguido de uno o más dígitos seguido de / y seguido de uno o más dígitos.

Elementos léxicos y sintácticos

- **Componentes léxicos**
- **Números**
- 25. 27.5. -35.7. -128. 5/8. 2.3E-12.
- Algunos Métodos
- 25 factorial.
- 34 + 9.
- 45<23.

Elementos léxicos y sintácticos

- **Componentes léxicos**
- **Literales:**
- **Caracteres:** objetos que representan los símbolos que forman un alfabeto.
- Son instancias de la clase Character.
- Patrón: es una expresión precedida por el signo $ seguida por cualquier carácter
- $a. $A. $3. $+. $$
- Algunos métodos
- $a asciiValue.
- $m isVowel.

Elementos léxicos y sintácticos

- **Componentes léxicos**
- **Literales:**
- **Cadenas:** Son objetos que representan una cadena de caracteres. Responden a mensajes para acceder a caracteres individuales, sustituir secuencias, compararlas con otras secuencias y concatenarlas.
- Son instancias de la clase String
- Patrón: Secuencia de caracteres encerrados entre comillas sencillas (‘’).
- ‘Hola’. ‘secuencia de caracteres’. ‘Region 001’.
- Se pueden concatenar cadenas separándolas con coma: ‘Esto es’, ‘una tira’, ‘de caracteres’ equivale a: ‘Esto es una tira de caracteres’.
- Algunos métodos
- 'Hola' size.
- 'Mundo Maniaco' leftString:4.
- 'Mundo Maniaco' rightString:4.

Elementos léxicos y sintácticos

- **Componentes léxicos**
- **Literales:**
- **Arreglos:** Son objetos estructurados cuyos elementos son accesibles mediante un índice entero. Cada elemento del Array es un literal. Estos objetos responden a mensajes que piden acceso a su contenido.
- Son instancias de la clase ArrayedCollection (Array, ByteArray, RunArray y la propia String)).
- Patrón: Secuencia de literales separados por blancos y encerrados entre paréntesis y precedidas por el carácter '#'. Los símbolos y arreglos contenidos como literales en el arreglo, no se preceden con el carácter '#'
- #(1 2 3). " arreglo de tres elementos enteros."
- #('producto' '120' 'factura' 'cantidad'). "arreglo de cuatro elementos que son secuencias de caracteres."
- #('elemento' (1 $4 'uno') 4 $A). "arreglo de cuatro elementos: secuencia, array, entero y carácter." "
- Algunos métodos"
- #(1 2 3 4) at: 3. #(1 2 3 4) size. #(1 2 3 4) at:2 put: 10.

Elementos sintácticos

- La sintaxis de Smalltalk es verdaderamente sencilla:
  - Objeto mensaje
- **Mensajes** Los mensajes representan la interación entre los componentes de un sistema Smalltalk. Un mensaje es un encargo que un objeto le hace a otro objeto.
- Por tanto, el mensaje está compuesto por el **receptor** (el objeto al que se le da el aviso), el
- **selector** (el nombre del mensaje) y, si corresponde, por los **argumentos**.
- En esta sintaxis, el objejo que recibe el mensaje se llama ***Receptor***, el nombre del mensaje se llama ***Selector***, los mensajes a veces llevan parámetros.
- ***5 + 4.*** En este ejemplo, al objeto receptor 5 (Instancia de SmallInteger), se le invoca el método + (suma) con el parámetro 4 (Objeto 4 de la clase SmallInteger).
- ***Transcript show: 'Hola Mundo'.*** En este ejemplo, al objeto Transcript (La "Pantalla" de Smalltalk, se le invoca el método show: (que muestra algo por pantalla), con el parámetro 'Hola Mundo' (un String).

Elementos sintácticos

- Tipos de mensaje: Existen tres tipos de mensaje en Smalltalk. Unario, Binario y de palabra clave.
- **Mensajes Unary** Son mensajes sin argumentos. Su sintaxis es:
- **receptor selector.**

- **Ejemplos**
- **-1** abs.
- **2** class.
- **1000** factorial.
- **'aeiou'** size.
- **Date** today.
- **Time** now.
- **OrderedCollection** new.
- **#símbolo** class.
- **String** category.

Elementos sintácticos

- **Mensajes Binary:** Además del receptor y el selector, los mensajes Binary tienen un único argumento.
- Su sintaxis es:
- **receptor selector parámetro**
- Los selectores para los mensajes binarios son caracteres especiales simples o dobles.
- Los selectores de carácter simple incluyen operadores de comparación y aritméticos tales como:
- **+ - * / < > =**
- Los selectores de carácter doble incluyen operadores tales como
- **~= (not =) <= // (división entera)**

- **Ejemplos**
- **3 + 5.**
- **3 > 7.**
- **3 = 5.**
- **2 @ 10.**
- **'Un String ', 'concatenado a otro'.**
- **'Alan Kay' -> Smalltalk.**

Elementos sintácticos

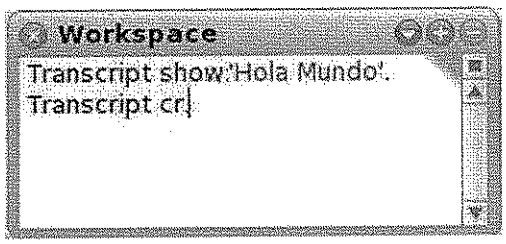
- **Mensajes Keyword:** Están formados por una o más palabras claves, con sus respectivos argumentos, con la siguiente estructura:
- **receptor selector1:arg1 selector2:arg2.. selectorn:argn.**
- **Ejemplos**
- **'Un String'** fi rst: **3**.
- **'Un String'** allButFirst: **3**.
- **'Un String'** copyFrom: **2** to: **5**.
- **5** between: **1** and: **10**.
- vector at: 2 put: 5.
- **Array** with: **1** with: **nil** with: **'string'**

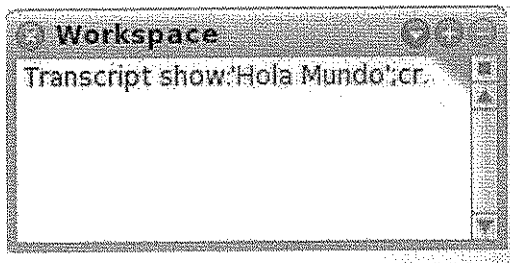
Elementos sintácticos

- Precedencia de mensajes
- El receptor o argumento de una expresión de mensajes puede ser en si mismo una expresión de mensajes.
- En Smalltalk la relación de precedencia en la evaluación de expresiones es la siguiente:
  1. **expresiones entre paréntesis.**
  2. **expresiones unarias (evaluadas de izquierda a derecha).**
  3. **expresiones binarias (evaluadas de izquierda a derecha).**
  4. **expresiones de palabra clave.**
  5. **expresiones de asignación.**
- Por ejemplo, en el siguiente mensaje contiene mensajes de los tres tipos:
- **x:=4 factorial gcd: 4 * 6.**
- Su resultado es 6

Elementos sintácticos

- Mensajes en Cascada
- Cuando se quiere enviar mas de un mensaje al mismo objeto, se puede utilizar la sintaxis para mensajes en cascada.
- Para esto se separa cada mensaje mediante punto y coma (;)
- Ejemplo: Enviar un mensaje de show al objeto Transcript y un retorno de carro al mismo objeto.





Elementos sintácticos

- Expresión de Asignación
- Como en casi todos los lenguajes, en Smalltalk existe la operación de asignación.
- El operador de asignación es := y la sintaxis es ***variable:=expresión***
- Las variables globales simplemente se usan sin declararse.
- Las variables locales se declaran encerrándose entre |. Ej: |i,j|.
- Las variables no tienen tipos, en Smalltalk no existen tipos de datos, las variables son instancias de objetos y se acomodan a la clase según el valor que se le asigne.
- x:=10.
- x:='Hola Mundo'.

Elementos sintácticos

- Expresiones de Bloque
- Son instancias de la clase BlockContext y representan una secuencia de expresiones separadas por puntos y encerradas entre corchetes '[]' que serán evaluadas sólo cuando el mensaje value adecuado sea enviado al bloque.
- La evaluación de un bloque retorna el resultado de la evaluación de la última expresión del mismo.
- Sintaxis: [objeto*1* mensaje*1*. objeto*2* mensaje*2*. .. Objeto*n* mensaje*n*].
- [Date today dayName. 'cadena a convertir' asUpperCase.] value.

Métodos de control

- Son métodos de clases booleanas y otras que se asemejan a las estructuras de control de lenguajes imperativos.
- Selección o alternativa:se implementa por medio del envío de mensajes a los objetos
  - true
  - false.
- Iteración o Repetición: se implementa por medio del envío de mensajes a objetos de las clases:
  - Number.
  - BlockContext.
  - Collection.

Selección

- Existen dos objetos de tipo BOOLEAN en el lenguaje y algunos mensajes enviados a estos objetos permiten la implementación de la selección en Smalltalk.
- Los mensajes que se envían a los objetos true y false llevan como parámetro un bloque sin argumentos.
- La evaluación de dicho bloque depende del mensaje y del objeto al que se lo envía.
- Los mensajes provistos por Smalltalk permiten el testeo y control condicional
  - simple
  - compuesto

Selección Simple

- Sintaxis ifTrue:
- objetoBooleano ifTrue: bloque
- SMALLTALK evalúa una vez el bloque si el mensaje fue enviado al objeto true. En cambio si fue enviado al objeto false no realiza ninguna acción.
- Sintaxis ifFalse:
- objetoBooleano ifFalse: bloque
- y el análisis es el inverso.
- 'cama' < 'casa' **ifTrue**: [^ 'cama'].
- 'cama' > 'casa' **ifTrue**: [^ 'cama'].

Selección Simple

- Sintaxis ifTrue:ifFalse:
- objetoBooleano ifTrue: bloqueVerdadero ifFalse: bloqueFalso
- cadena1 < cadena2 **ifTrue**: [^cadena1] **ifFalse**: [^cadena2]
- numero :=numero < 0 **ifTrue**: [^(numero **negated)**] **ifFalse**: [^numero]

Iteración

- La iteración en SMALLTALK es soportada mediante el envío de mensajes a objetos de las clases Number, BlockContext y Collection.
- La clase Collection tiene varias subclases, como por ejemplo Array.
- Ciclos equivalentes al For
- to:do: Y to:by:do:
- timesRepeat:
- Ciclo equivalente al While
- whileTrue:
- whileFalse:
- Coleciones
- do:
- collect:
- select:
- reject:
- detect:

to:do: y to:by:do

- **CICLOS to:do: Y to:by:do:**
- Un ciclo iterativo simple con un número específico de repeticiones se puede llevar a cabo usando mensajes de la clase Integer.
- **Sintaxis:**
- enteroInicial **to**: enteroFinal **do**: [:parámetro | s1. s2. ... sn]
- El método **to:** enteroFinal **by:** incremento **do:** unBloque es una variación del anterior que especifica la cantidad en la cual el argumento del bloque debe incrementarse en cada evaluación. **Sintaxis:**
- enteroInicial **to**: enteroFinal **by**: incremento **do**: [:parámetro | s1. s2. ... sn]

to:do: y to:by:do

- |unArreglo|
- unArreglo := Array **new**: 10.
- 1 **to**: 10 **do**: [:j |unArreglo **at**: j **put**: 0.0].
- 1 **to**: 10 **by**: 2 **do**: [:j |unArreglo **at**: j **put**: 3.0].

timesRepeat

- **CICLO timesRepeat:**
- Otro ciclo iterativo simple con un número específico de repeticiones se puede llevar a cabo usando el mensaje timesRepeat de la clase Integer. Este mensaje lleva como parámetro un bloque sin argumentos que se evaluará tantas veces como lo indique el entero receptor del mismo.
- **Sintaxis:**
- entero **timesRepeat**: [s1. s2. ... sn]
- |suma|
- suma := 0.
- 4 **timesRepeat**: [suma. suma + 5].
- ^suma

whileTrue: whileFalse:

- **CICLOS whileTrue: whileFalse**
- Existen dos mensajes a objetos de la clase BlockContext que proveen un equivalente funcional de los ciclos while de otros lenguajes. Estos mensajes son **whileTrue**: y **whileFalse**:.
- **Sintaxis:**
- [bloqueDeTesteo] **whileTrue**: [bloqueEjecutable]
- [bloqueDeTesteo] **whileFalse**: [bloqueEjecutable]

whileTrue: whileFalse:

- **CICLOS whileTrue: whileFalse**
- "Contar las vocales de una string"
- |vocales string indice|
- vocales := 0.
- indice := 0.
- string := 'String con varias vocales".
- [indice < = string **size**]
- **whileTrue**: [(string **at**: indice) **isVowel**
- **ifTrue**: [vocales . vocales + 1].
- indice . indice +1].
- ^vocales

Colecciones

- La clase Collection provee una variedad de esquemas de iteración. Todos los mensajes llevan como argumento un bloque con un parámetro que es evaluado una vez por cada elemento en el receptor (cada carácter de una string, cada elemento de un arreglo, etc.)
- **Sintaxis:**
- objetoColección mensaje: [:parámetro | s1. s2. ... sn]

Colecciones

- **do**: se ejecuta el bloque argumento tantas veces como elementos aparezcan en la colección, asignando al parámetro un elemento de la lista en cada ejecución.
- **collect**: para cada uno de los elementos del receptor se evalúa el bloque con ese elemento como parámetro, y devuelve una nueva colección formada por el resultado de esas evaluaciones.
- **select**: crea y retorna una subcolección de los elementos del receptor, que verifican la expresión booleana final del bloque. Es decir que la expresión sn debe ser booleana.
- **reject**: crea y retorna una subcolección de los elementos del receptor que no verifiquen la expresión booleana final del bloque. Es decir que la expresión sn debe ser booleana.
- **detect**: devuelve el primer elemento de la lista que verifique la expresión booleana final del bloque.

Colecciones

- |string cuenta|
- cuenta := 0.
- string := 'Orientacion a Objetos'.
- string **do**: [:caracter | caracter **isUppercase**
  ifTrue: [cuenta := cuenta + 1]].
- string **collect**: [:caracter | caracter **asUppercase**].
- string **select**: [:caracter | caracter **isUppercase**].
- string **reject**: [:caracter | caracter **isUppercase**].
- string **detect**: [:caracter | caracter **isUppercase**].

COLECCIONES: esquemas de REPETICIÓN

- La clase Collection provee una variedad de esquemas de iteración.
- Todos los mensajes llevan como argumento un bloque con un parámetro que es evaluado una vez por cada elemento en el receptor (cada carácter de un string, cada elemento de un arreglo, etc.)
- **Sintaxis:**

```
objetoColección mensaje: [ :parámetro | s1. s2. ... sn ]
```

- El mensaje puede ser alguno de los siguientes:

COLECCIONES: esquemas de REPETICIÓN

do: se ejecuta el bloque argumento tantas veces como elementos aparezcan en la colección, asignando al parámetro un elemento de la lista en cada ejecución.

collect: para cada uno de los elementos del receptor se evalúa el bloque con ese elemento como parámetro, y devuelve una nueva colección formada por el resultado de esas evaluaciones.

select: crea y retorna una subcolección de los elementos del receptor, que verifican la expresión booleana final del bloque. Es decir que la expresión sn debe ser booleana.

reject: crea y retorna una subcolección de los elementos del receptor
que no verifiquen la expresión booleana final del bloque. Es decir que
la expresión sn debe ser booleana.

detect: devuelve el primer elemento de la lista que verifique la
expresión booleana final del bloque.

Colecciones

- |string cuenta|cuenta := 0.
- string := 'Orientacion a Objetos'.
- string **do**: [:caracter | caracter **isUppercase**
  ifTrue: [cuenta := cuenta + 1]].
- string **collect**: [:caracter | caracter **asUppercase**].
- string **select**: [:caracter | caracter **isUppercase**].
- string **reject**: [:caracter | caracter **isUppercase**].
- string **detect**: [:caracter | caracter **isUppercase**].