FIUBA - 7507

Algoritmos y programación 3

Trabajo práctico 1: Calculador de Funciones

1er cuatrimestre, 2010

(trabajo individual)

Alumno: Schenkelman Damián

Padrón: 90728

E-mail: damiansch@hotmail.com

Fecha de entrega: 21/04/2010

Corrector:

Nota Final:

Tabla de contenidos

Introducción

Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema, trabajando individualmente utilizando Smalltalk.

Consigna

El departamento de matemática ha encargado al departamento de computación el desarrollo de un software para graficación de funciones matemáticas con el fin de brindar una herramientas más a los alumnos de las cátedras de análisis matemático.
Para cumplir con dicho compromiso el departamento de computación ha decidido dividir el desarrollo en tres etapas, cada una de ellas con un entregable de código y documentación asociado. A continuación se detalla el contenido de la primera entrega. El contenido de la entregas subsiguientes se definirá a medida que se avance con el desarrollo de la primera entrega.

Entregables

Dado que para poder graficar una función es necesario previamente poder evaluarla en el intervalo que se desea graficar, se ha decidido comenzar el desarrollo por una biblioteca de evaluación de funciones.
Dicha biblioteca debe proveer la API con las siguientes capacidades:

- evaluación de funciones matemáticas de una variable [f(x)]
- soporte de las siguiente funciones básicas: suma, resta, división, producto, potencia y logaritmo
- soporte de funciones trigonométricas: seno, coseno, tangente
- soporte de funciones: factorial, arsen, arcos, fibonaci (tener en cuenta que hay puntos donde estas no pueden evaluarse)
- soporte de funciones compuestas [f(g(x))]
- evaluación de una función en un intervalo (devuelve una colección de resultados)
- evaluación (por aproximación) de la función derivada e integral
- indicar los puntos de intersección entre funciones en un determinado intervalo
- indicar los máximos y mínimo de una función en un determinado intervalo

Como es de esperar de cualquier biblioteca, la misma deberá contar con:

- conjunto de pruebas unitarias que muestren el uso de la biblioteca y su correcto funcionamiento.
- documentación completa del código fuente
- documentación completa del diseño de clases (diagramas UML de clases y secuencia)

Pruebas mínimas

Todas las clases deberán contar con sus pruebas unitarias **COMPLETAS**.

Adicionalmente, se debe entregar un clase llamada PruebaIntegracionFunciones, con las siguients pruebas:

testPruebaUno

f(x) = 5x + (x/3)

g(x) = (5 ^x) -7

comprobando (como mínimo) que:

f(9) = 48

f(0) = 0

g(2) = 18

g(0) = -6

f(g(1)) = 10.666...

testpruebaDos

f(x) = Fibonacci(x)

g(x) = Factorial(x)

comprobando (como mínimo) que:

f(3) = 2

f(8) = 21

g(3) = 6

g(10) = 3,628,800

g(-2) [responda correctamente de acuerdo a lo que han definido en su diseño]

g(f(4)) = 6

Supuestos

1. Supongo que el usuario de la librería va a utilizarla de manera correcta. Es decir, en *Smalltalk* la visibilidad de métodos y clases es pública, por lo que no hay restricciones en este aspecto respecto de los objetos a construir y el código al que llamar. Por lo tanto espero que el usuario utilice la librería de la manera que se supone, que cree los objetos como la librería lo dispone, etc.
2. Supongo que la función logaritmo pedida es logaritmo natural. De todas formas se puede llegar a otras bases haciendo divisiones.
3. Supongo no necesario proveer funcionalidad para calcular intersección de funciones discretas.

Herramientas y modo de entrega

- Informe completo (de acuerdo al checklist de corrección)

- Todos los diagramas en formato imagen incorporados a este documento y opcionalmente en el formato de la herramienta que hayan usado

- Archivo ".st" con la categoria del modelo

- Archivo ".st" con la categoria de las pruebas

- Codigo fuente impreso (si es solicitada por los ayudantes del curso)

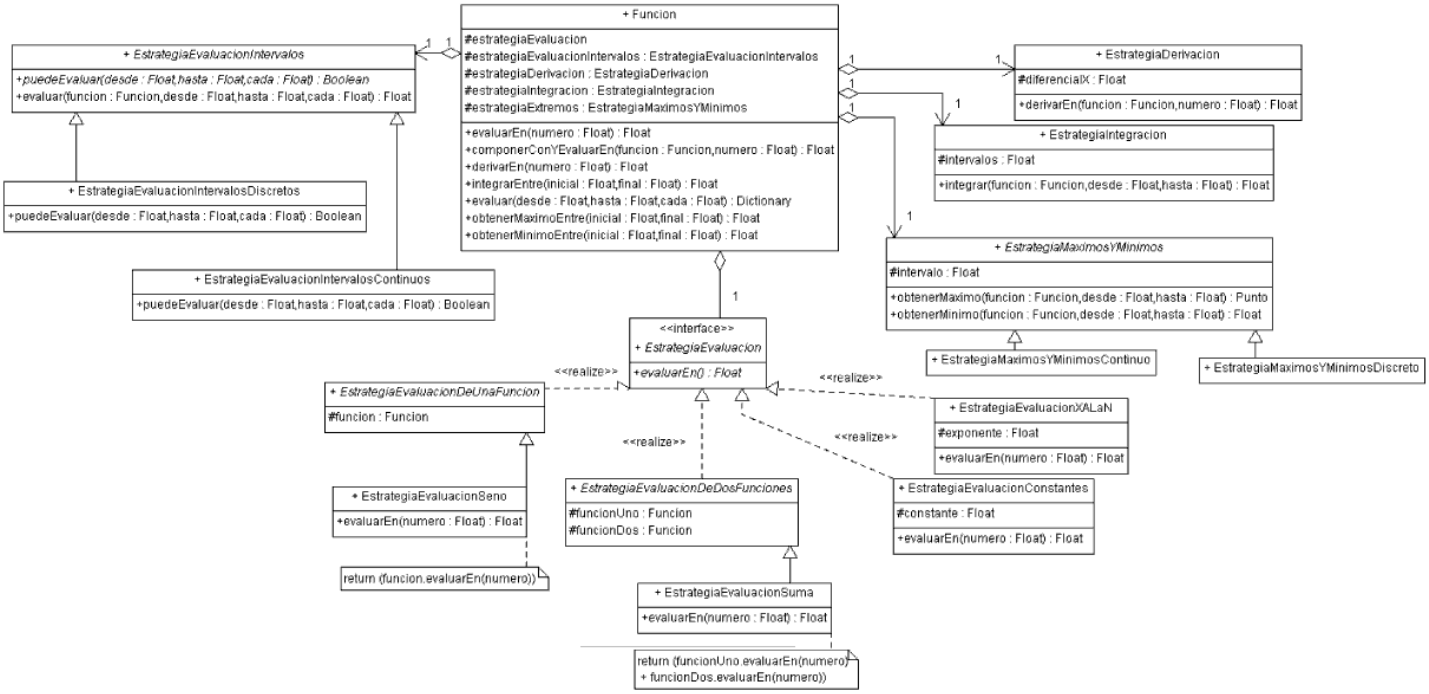
Modelo de dominio

Decidí implementar el Trabajo Práctico de una forma tal que pudiera proveer la funcionalidad requerida en la actualidad y dar la posibilidad de extender la misma fácilmente a futuro. Por eso es que determine crear una clase **Funcion** que delegue sus operaciones a otras clases. Tener la funcionalidad principal de la **Funcion** en clases separadas, permitiría variar la implementación de las operaciones fácilmente y sin necesidad de modificar en gran escala la interfaz.

Al mismo tiempo, ya que esto requeriría la creación de objetos complejos con muchas dependencias, determine que sería útil tener una fábrica de funciones. La misma podría abstraer la creación de las funciones y permitiría al usuario de la librería ignorar estos detalles complejos.

En el diagrama de clases se pueden ver la mayoría de las relaciones entre la clase función y sus dependencias

Diagramas de clases



El diagrama mostrado arriba no abarca la totalidad de las clases en la librería ni todos los métodos de las clases en el mismo. Su objetivo es mostrar como la clase función delega las diferentes operaciones y las jerarquías de clases creadas para satisfacer las diferentes operaciones. Por esa razón, por ejemplo, no se muestran todas las estrategias de evaluación creadas sino algunas que explican los casos más generales.

En el diagrama de clases también utilice la notación de interfaces, debido a que cree una clase sin estado y con un método abstracto (por más que en *Smalltalk* no existen las interfaces).

Detalles de implementación

Las siguientes fueron algunas otras consideraciones que realice durante la implementación del trabajo práctico que en mi opinión pueden resultar interesantes:

- Utilización de Estrategias de evaluación en vez de CodeBlocks
  Como se puede ver arriba utilice estrategias para delegar la funcionalidad de evaluación de cada tipo de función. Otra opción que considere fue la utilización de CodeBlocks. Estos hubieran resultado en una menor cantidad de clases, y al mismo tiempo una menor testabilidad. Un ejemplo simple del uso de los bloques hubiera sido el siguiente:

```
crearSuma: f1 y:f2
"Metodo crearSuma en fabrica de funciones"
bloque := [:x|f1 evaluarEn:x + f2 evaluarEn:x].
función crear:bloque.
^funcion

evaluaren: numero
"Metodo evaluarEn de Funcion."
^bloque value:numero
```

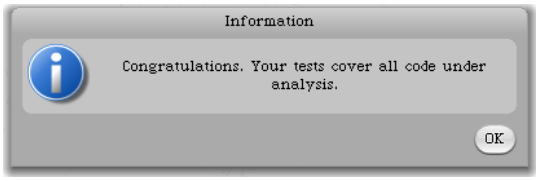
- Falta de getters en las diferentes clases. Esto se debe a que tener getters hubiera permitido realizar pruebas mas "certeras" al poder hacer cosas como:

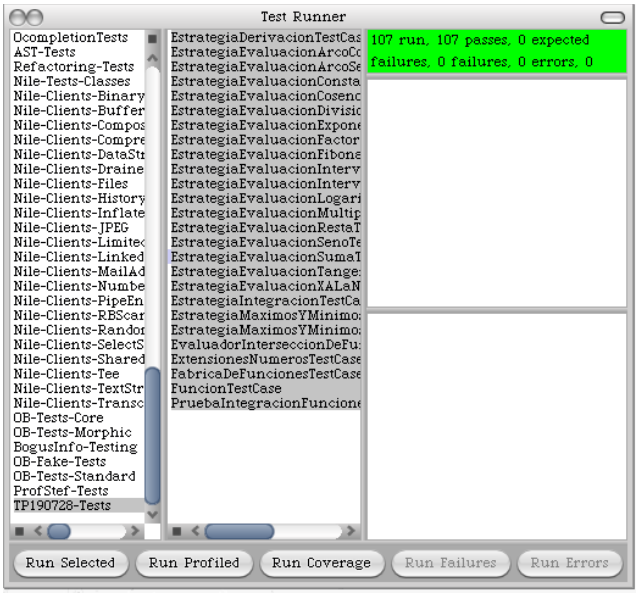
```
testCrearEstrategiaDevuelveEstrategiaNoNula
| estrategia |
estrategia := EstrategiaEvaluacionConstantes crearNueva: 5.
self deny: (estrategia == nil).
self assert: (estrategia getConstante = 5.
```

Sin embargo, tener getters hubiera permitido a usuarios de la librería tener la posibilidad de conocer el valor de los atributos internos de las funciones, por lo que decidí no tenerlos.

Algunas métricas interesantes:

- Mas de 100 pruebas unitarias
- 100% Code Coverage





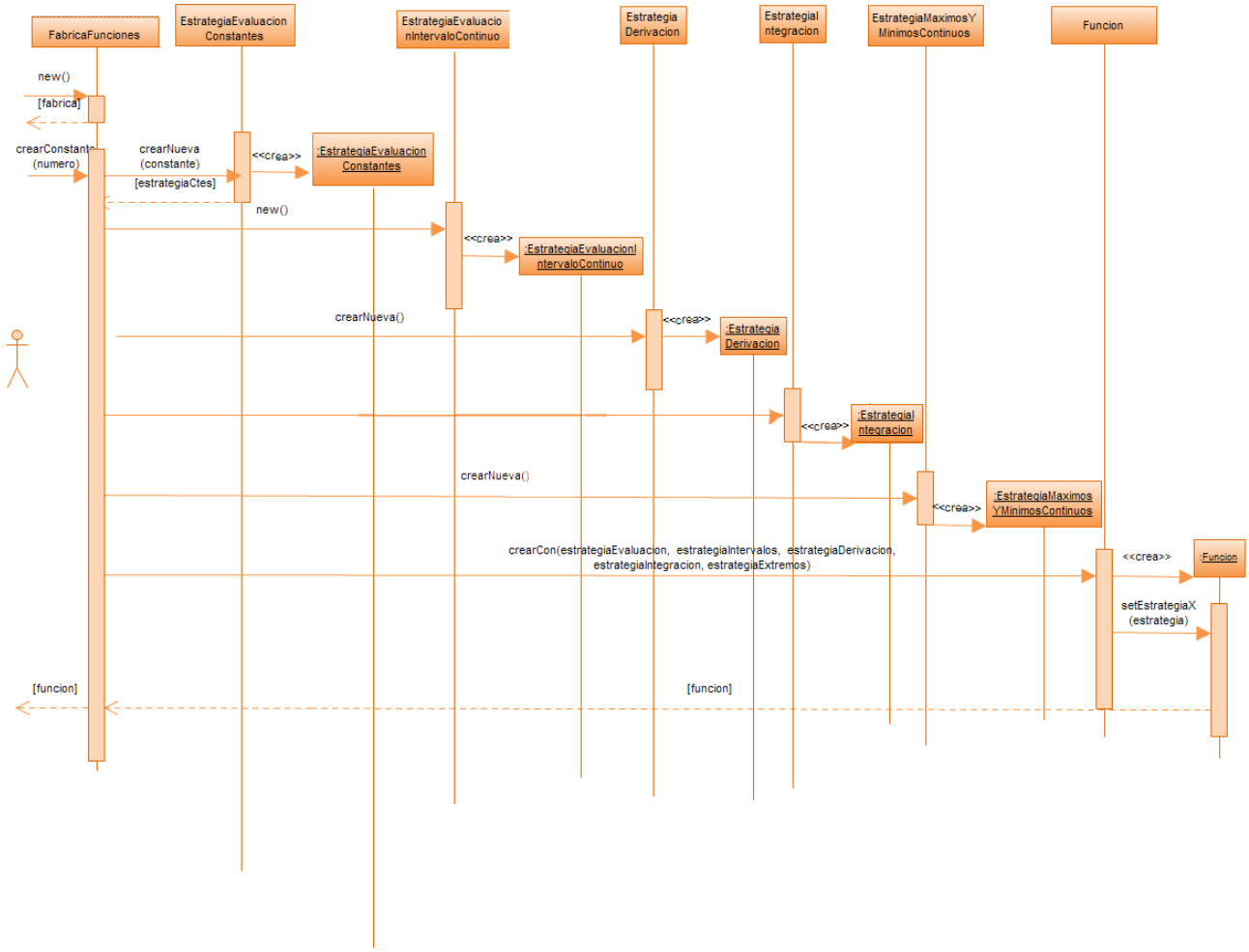
Excepciones

Las siguientes excepciones fueron creadas para el trabajo práctico:

- **ExcepcionArgumentoInvalido:** Es utilizada para avisar que un argumento recibido como parámetro no es válido en el contexto actual. Por ejemplo, el valor recibido por la función constante no es un número.
- **ExcepcionDominioFunción:** Es utilizada para avisar que una función se quiso evaluar fuera de su dominio.
- **ExcepcionEvaluacionIntervalo:** Es utilizada para avisar que el intervalo no es un intervalo válido de evaluación. Por ejemplo, si (final – inicial) < tamañoSubintervalo.
- **ExcepcionOperacionAritmeticaInvalida:** Es utilizada para avisar que una operación no puede ser realizada. Por ejemplo, derivar una función discreta.

Diagramas de secuencia

El diagrama de secuencia mostrado a continuación muestra las llamadas realizadas para crear una función constante suponiendo un cliente de la librería.

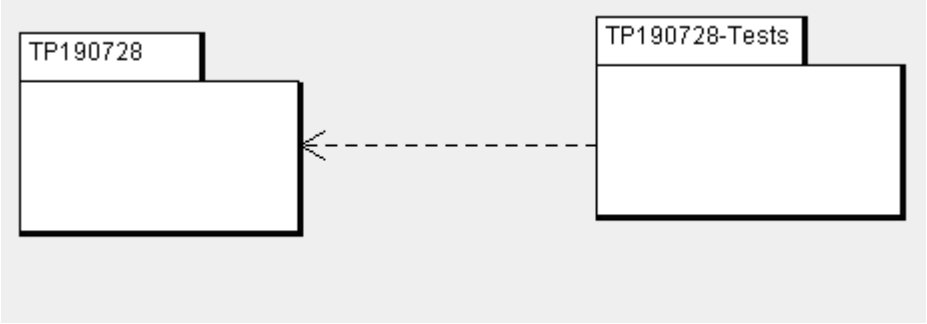


Diagramas de estado

No considero que un diagrama de estado facilitaría la comprensión del funcionamiento de la librería.

Diagramas de paquetes

A continuación se muestra el diagrama de paquetes utilizado para este trabajo. El mismo tiene dos paquetes para mantener una organización consistente con el TP0.



Código fuente

A continuación se encuentra todo el código fuente que es parte de la implementación del trabajo (no las pruebas).

```
Object subclass: #EstrategiaDerivacion
	instanceVariableNames: 'diferencialX'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaDerivacion methodsFor: 'Derivacion' stamp: 'DamianSchenkelman 4/17/2010 12:31'!
derivar: funcion en: numero
	"Recibe una funcion y un numero. Calcula la derivada numerica de la funcion en ese valor. Si la
derivacion lanza una excepcion deja que se propague."

	| derivada funcionEnNumero funcionEnNumeroMasDiferencial |
	funcionEnNumeroMasDiferencial := funcion evaluarEn: (numero + diferencialX).
	funcionEnNumero := funcion evaluarEn: (numero).
	derivada := (funcionEnNumeroMasDiferencial - funcionEnNumero / diferencialX).
	^derivada! !


!EstrategiaDerivacion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 12:10'!
setDiferencialX: numero
	"Deberia ser usado solo una vez en el momento de creacion."
	(diferencialX ~= nil)ifTrue:
	[
		Error new signal: 'No se puede volver a setear'.
	].
	diferencialX := numero.! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaDerivacion class
	instanceVariableNames: ''!

!EstrategiaDerivacion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/17/2010 12:34'!
crearNueva
	"Crea una nueva estrategia de derivacion y la devuelve."
	| estrategia |
	estrategia := EstrategiaDerivacion new.
	estrategia setDiferencialX: 0.000000000001.
	^estrategia.! !

Object subclass: #EstrategiaEvaluacion
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacion methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/10/2010 17:17'!
evaluarEn: numero
	"Metodo abstracto que debe ser implementado por todas las estrategias de evaluacion."
	self subclassResponsibility.! !

EstrategiaEvaluacion subclass: #EstrategiaEvaluacionConstantes
	instanceVariableNames: 'constante'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionConstantes methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/10/2010 16:03'!
setConstante: numero
	"Setea el atributo constante con el parametro recibido."
	constante := numero.! !


!EstrategiaEvaluacionConstantes methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/10/2010 12:57'!
evaluarEn: unNumero
	"Recibe un numero. Evalua la funcion constante en ese numero (devuelve el valor de la constante)."
	^ constante.! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionConstantes class
	instanceVariableNames: ''!
```

```
!EstrategiaEvaluacionConstantes class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/10/2010 16:04'!
crearNueva: numero
	"Crea una nueva estrategia. Le setea el valor al atributo constante y la devuelve."

	| estrategia |
	estrategia := EstrategiaEvaluacionConstantes new.
	estrategia setConstante: numero.
	^estrategia.! !

EstrategiaEvaluacion subclass: #EstrategiaEvaluacionDeDosFunciones
	instanceVariableNames: 'funcionUno funcionDos'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionDeDosFunciones methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/10/2010 17:38'!
setFuncionUno: funUno setFuncionDos: funDos
	"Recibe dos funciones y las guarda en atributos de la instancia."
	funcionUno  := funUno.
	funcionDos  := funDos.! !

EstrategiaEvaluacion subclass: #EstrategiaEvaluacionDeUnaFuncion
	instanceVariableNames: 'funcion'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionDeUnaFuncion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/10/2010 19:45'!
setFuncion: func
	"Setea la funcion, guardandola en un atributo."
	funcion := func! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionDeUnaFuncion class
	instanceVariableNames: ''!

!EstrategiaEvaluacionDeUnaFuncion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 16:41'!
crearNueva: clase conFuncion: funcion
	"comment stating purpose of message"

	| estrategia |
	estrategia := clase new.
	estrategia setFuncion: funcion.
	^estrategia.! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionArcoCoseno
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionArcoCoseno methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:22'!
evaluarEn: numero
	"Recibe un numero. Evalua la funcion en ese numero para obtener el argumento. Si el modulo del
argumento es mayor o igual a uno lanza una excepcion ExcepcionDominioFuncion."
	| argumento |
	argumento := funcion evaluarEn: numero.
	(argumento abs >= 1)ifTrue:
	[
		ExcepcionDominioFuncion new signal.
	].
	^argumento arcCos.! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionArcoCoseno class
	instanceVariableNames: ''!

!EstrategiaEvaluacionArcoCoseno class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:14'!
crearNueva: funcion
	"Recibe una funcion. Crea una nueva instancia de la estrategia y setea la funcion. Devuelve la
estrategia."
	^(super crearNueva: self conFuncion: funcion).
	! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionArcoSeno
	instanceVariableNames: ''
	classVariableNames: ''
```

```
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionArcoSeno methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 12:46'!
evaluarEn: numero
        "Recibe un numero. Evalua la funcion en ese numero para obtener el argumento. Calcula el arco seno
del argumento. Si el modulo del argumento es mayor a uno lanza una excepcion del tipo
ExcepcionDominioFuncion."
        | argumento |
        argumento := funcion evaluarEn: numero.
        (argumento abs <= 1)ifFalse:
        [
                ExcepcionDominioFuncion new signal.
        ].
        ^argumento arcSin.
        ! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionArcoSeno class
        instanceVariableNames: ''!

!EstrategiaEvaluacionArcoSeno class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:14'!
crearNueva: funcion
        "Recibe una funcion. Crea una nueva instancia de la estrategia y le setea la funcion. Devuelve la
estrategia."
        ^(super crearNueva: self conFuncion: funcion).! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionCoseno
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionCoseno methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:24'!
evaluarEn: numero
        "Recibe un numero. Evalua la funcion en ese numero y luego calcula el coseno. Devuelve el resultado."
        ^((funcion evaluarEn: numero) cos)! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionCoseno class
        instanceVariableNames: ''!

!EstrategiaEvaluacionCoseno class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:15'!
crearNueva: funcion
        "Recibe una funcion como parametro. Crea una nueva instancia de la estrategia y le setea la funcion.
Devuelve la estrategia."
        ^(super crearNueva: self conFuncion: funcion).! !

EstrategiaEvaluacionDeDosFunciones subclass: #EstrategiaEvaluacionDivision
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionDivision methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/10/2010 17:40'!
setNumerador: numerador setDenominador: denominador
        "Recibe el numerador y denominador y lo setea a los atributos correspondientes."
        super setFuncionUno: numerador  setFuncionDos: denominador.! !


!EstrategiaEvaluacionDivision methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 19:53'!
evaluarEn: numero
        "Recibe un numero. Evalua las funciones numerador y denominador en ese numero y las divide. Devuelve
el resultado."
        "La linea de abajo podria tirar una excepcion ZeroDivide. Esto es algo tenido en cuenta y he decidido
que en caso que el escenario se presente esa es la excepcion que debe ser lanzada."
        ^((funcionUno  evaluarEn: numero )/(funcionDos  evaluarEn: numero))
        ! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionDivision class
        instanceVariableNames: ''!

!EstrategiaEvaluacionDivision class methodsFor: 'as yet unclassified' stamp: 'DamianSchenkelman 4/10/2010
17:01'!
crearNuevaCon: numerador y: denominador
```

```
        "Crea una nueva instancia de la estrategia. Recibe las funciones numerador y denominador y las asigna
a los atributos respectivos. Devuelve la estrategia."

        | estrategia |
        estrategia := EstrategiaEvaluacionDivision new.
        estrategia setNumerador: numerador setDenominador: denominador.
        ^estrategia.! !

EstrategiaEvaluacionDeDosFunciones subclass: #EstrategiaEvaluacionExponencial
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionExponencial methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/10/2010 19:12'!
setBase: base setExponente: exponente
        "Recibe una base y un exponente y los guarda como atributos."
        super setFuncionUno: base setFuncionDos: exponente.! !


!EstrategiaEvaluacionExponencial methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 19:54'!
evaluarEn: numero
        "Recibe un numero. Evalua las funciones base y exponente. Eleva la base al exponente y devuelve el
valor obtenido."
        ^((funcionUno evaluarEn: numero) raisedTo: (funcionDos evaluarEn: numero))! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionExponencial class
        instanceVariableNames: ''!

!EstrategiaEvaluacionExponencial class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/10/2010 19:11'!
crearNuevaCon: base y: exponente
        "Crea una nueva instancia de la estrategia. Recibe las funciones base y exponente y las guarda en
atributos de la nuvea instancia. Devuelve la estrategia."

        | estrategia |
        estrategia := EstrategiaEvaluacionExponencial new.
        estrategia setBase: base setExponente: exponente.
        ^estrategia.! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionFactorial
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionFactorial methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:22'!
evaluarEn: numero
        "Recibe un numero. Evalua la funcion en ese numero y calcula el factorial del resultado. Devuelve el
resultado de factorial. Si el argumento del factorial no esta en el dominio lanza una excepcion
ExcepcionFactorial."
        | argumento |
        argumento := ((funcion) evaluarEn: numero).
        (ExtensionesNumeros esNaturalOCero: argumento) ifFalse:
        [
                ExcepcionDominioFuncion new signal.
        ].
        ^(argumento factorial).! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionFactorial class
        instanceVariableNames: ''!

!EstrategiaEvaluacionFactorial class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:15'!
crearNueva: funcion
        "Recibe una funcion como parametro. Crea una nueva instancia de la estrategia y le setea la funcion.
Devuelve la estrategia."
        ^(super crearNueva: self conFuncion: funcion).! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionFibonacci
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionFibonacci methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:22'!
evaluarEn: numero
```

```
	"Recibe un numero. Evalua la funcion en el numero para obtener el numero de la serie a obtener. Si no
es natural o cero lanza una excepcion ExcepcionDominioFuncion."
	| argumento num1 num2 res |
	argumento := funcion evaluarEn: numero.
	(ExtensionesNumeros esNaturalOCero: argumento)ifFalse:
	[
		ExcepcionDominioFuncion new signal.
	].
	(argumento = 0)ifTrue:
	[
		^0
	].
	(argumento = 1)ifTrue:
	[
		^1
	].
	num1 := 0.
	num2 := 1.
	argumento timesRepeat:
	[
		res := num1 + num2.
		num2 := num1.
		num1 := res
	].
	^res! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionFibonacci class
	instanceVariableNames: ''!

!EstrategiaEvaluacionFibonacci class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:15'!
crearNueva: funcion
	"Recibe una funcion. Crea una nueva instancia de la estrategia y le setea la funcion. Devuelve la
estrategia."
	^(super crearNueva: self conFuncion: funcion).! !

Object subclass: #EstrategiaEvaluacionIntervalo
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionIntervalo methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 11:07'!
evaluar: funcion desde: inicial  hasta: final  cada: intervalo
	"Recibe una funcion, un valor inicial, un final y el intervalo. Evalua la funcion entre los valores
iniciales y finales en intervalos definidos por intervalo. Devuelve un diccionario con las claves del valor
de x y valores el valor de y. No devuelve una coleccion de puntos, porque considero que es mas usable un
diccionario en este caso."

	| resultados resultado |
	resultados := Dictionary new.
	inicial to: final by: intervalo do:
	[:x|
		resultado := funcion evaluarEn: x.
		resultados at: x put: resultado.
	].
	^resultados. ! !

!EstrategiaEvaluacionIntervalo methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/12/2010 22:56'!
puedeEvaluarDesde: inicial hasta: final cada: intervalo
	self subclassResponsibility! !

EstrategiaEvaluacionIntervalo subclass: #EstrategiaEvaluacionIntervaloContinuo
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionIntervaloContinuo methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/13/2010 00:59'!
evaluar: funcion desde: inicial  hasta: final  cada: intervalo
	"Recibe una funcion, un valor inicial, un final y el intervalo. Evalua la funcion entre los valores
iniciales y finales en intervalos definidos por intervalo. Devuelve un diccionario con las claves del valor
de x y valores el valor de y. Si la evaluacion en el intervalo lanza una excepcion del tipo
ExcepcionDominioFuncion, lanza una nueva excepcion del tipo ExcepcionEvaluacionIntervalo."
	[^super evaluar: funcion desde: inicial  hasta: final cada: intervalo] on: ExcepcionDominioFuncion
do:[ExcepcionEvaluacionIntervalo new signal].! !

!EstrategiaEvaluacionIntervaloContinuo methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 10:47'!
```

```
puedeEvaluarDesde: inicial hasta: final cada: intervalo
	"Recibe un valor inicial, uno final y un intervalo. Devuelve false si el intervalo es menor a cero,
si el valor inicial es mayor o igual al final o s¡ la diferencia entre el valor final e inicial es menor al
intervalo. Devuelve false en caso contrario."
	^ExtensionesNumeros esCrecienteYNoVacioDesde: inicial  hasta: final cada: intervalo.! !

EstrategiaEvaluacionIntervalo subclass: #EstrategiaEvaluacionIntervaloDiscreto
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionIntervaloDiscreto methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/12/2010 23:40'!
evaluar: funcion desde: inicial  hasta: final  cada: intervalo
	"Recibe una funcion, un valor inicial, un final y el intervalo. Evalua la funcion entre los valores
iniciales y finales en intervalos definidos por intervalo. Devuelve un diccionario con las claves del valor
de x y valores el valor de y."
	^super evaluar: funcion desde: inicial  hasta: final cada: intervalo.! !

!EstrategiaEvaluacionIntervaloDiscreto methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 10:47'!
puedeEvaluarDesde: inicial hasta: final cada: intervalo
	"Recibe un valor inicial, uno final y un intervalo. Devuelve false si el intervalo es menor a cero,
si el valor inicial es mayor o igual al final o s¡ la diferencia entre el valor final e inicial es menor al
intervalo. Devuelve false en caso contrario."
	| puedeEvaluar |
	puedeEvaluar := ExtensionesNumeros esCrecienteYNoVacioDesde: inicial  hasta: final cada: intervalo.
	(puedeEvaluar)ifTrue:
	[
		^((ExtensionesNumeros esNaturalOCero: inicial) & (ExtensionesNumeros esNatural: final) &
(ExtensionesNumeros esNatural: intervalo))
	]
	ifFalse:
	[
		^false.
	].! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionLogaritmoNatural
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionLogaritmoNatural methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:23'!
evaluarEn: numero
	"Recibe un numero. Evalua la funcion en ese numero para obtener el argumento  y luego calcula el
logaritmo natural. Devuelve el resultado. Si el argumento es menor o igual a cero lanza una excepcion del
tipo ExcepcionDominioFuncion."
	| argumento |
	argumento := funcion evaluarEn: numero.
	(argumento <= 0)ifTrue:
	[
		ExcepcionDominioFuncion new signal.
	].
	^((argumento) log: Float e).! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionLogaritmoNatural class
	instanceVariableNames: ''!

!EstrategiaEvaluacionLogaritmoNatural class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010
17:15'!
crearNueva: funcion
	"Recibe una funcion. Crea una nueva instancia de la estrategia y le setea la funcion. Devuelve la
estrategia."
	^(super crearNueva: self conFuncion: funcion).! !

EstrategiaEvaluacionDeDosFunciones subclass: #EstrategiaEvaluacionMultiplicacion
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionMultiplicacion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/10/2010 17:40'!
setProductoUno: factorUno setProductoDos: factorDos
	"Recibe dos funciones como parametros y las guarda en atributos."
	super setFuncionUno: factorUno setFuncionDos: factorDos.! !
```

```
!EstrategiaEvaluacionMultiplicacion methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 19:54'!
evaluarEn: numero
	"Recibe un numero. Evalua ambas funciones (factores) guardadas y multiplica los resultados. Devuelve
el valor obtenido."
	^((funcionUno  evaluarEn: numero)*(funcionDos evaluarEn: numero))! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionMultiplicacion class
	instanceVariableNames: ''!

!EstrategiaEvaluacionMultiplicacion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/10/2010 16:39'!
crearNuevaCon: productoUno y: productoDos
	"Crea una nueva estrategia. Recibe los dos producos que son funciones y los guarda."

	| estrategia |
	estrategia := EstrategiaEvaluacionMultiplicacion new.
	estrategia setProductoUno: productoUno setProductoDos: productoDos.
	^estrategia! !

EstrategiaEvaluacionDeDosFunciones subclass: #EstrategiaEvaluacionResta
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionResta methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/10/2010 19:15'!
setFuncionInicial: funcionInicial setFuncionARestar: funcionARestar
	"Recibe una funcion inicial y una funcion a restar y las guarda en atributos."
	super setFuncionUno: funcionInicial setFuncionDos: funcionARestar! !


!EstrategiaEvaluacionResta methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 19:54'!
evaluarEn: numero
	"Recibe un numero y evaluar las dos funciones en ese valor. Devuelve el resultado."
	^((funcionUno  evaluarEn: numero)-(funcionDos evaluarEn: numero)).! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionResta class
	instanceVariableNames: 'fInicial'!

!EstrategiaEvaluacionResta class methodsFor: 'as yet unclassified' stamp: 'DamianSchenkelman 4/10/2010
16:03'!
crearNuevaCon: funcionInicial y: funcionARestar
	"Recibe dos funciones. Crea una nueva instancia de la estrategia y guarda las funcion en atributos.
La primera es la funcion inicial la segunda es la funcion a restar. Devuelve la estrategia."

	| estrategia |
	estrategia := EstrategiaEvaluacionResta new.
	estrategia setFuncionInicial: funcionInicial setFuncionARestar:funcionARestar.
	^estrategia.! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionSeno
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaEvaluacionSeno methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:23'!
evaluarEn: numero
	"Recibe un numero y evalua la funcion en ese numero y luego calcula el seno. Devuelve el valor
obtenido."
	^((funcion evaluarEn: numero) sin)! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionSeno class
	instanceVariableNames: ''!

!EstrategiaEvaluacionSeno class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:15'!
crearNueva: funcion
	"Recibe una funcion. Crea una nueva instanci de la estrategia y le asigna la funcion. Devuelve la
estrategia."
	^(super crearNueva: self conFuncion: funcion).! !

EstrategiaEvaluacionDeDosFunciones subclass: #EstrategiaEvaluacionSuma
	instanceVariableNames: ''
	classVariableNames: ''
```

```
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionSuma methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/15/2010 20:12'!
setFuncionUno: fUno setFuncionDos: fDos
        "Recibe como parametros dos funciones y las guarda en atributos."
        funcionUno := fUno.
        funcionDos  := fDos.! !


!EstrategiaEvaluacionSuma methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 19:54'!
evaluarEn: numero
        "Evalua la funcion en el valor recibido. Llama al metodo evaluar de las dos funciones que tiene la
instancia."

        ^((funcionUno  evaluarEn: numero) +(funcionDos  evaluarEn: numero)). ! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionSuma class
        instanceVariableNames: ''!

!EstrategiaEvaluacionSuma class methodsFor: 'as yet unclassified' stamp: 'DamianSchenkelman 4/10/2010 16:04'!
crearNuevaCon: funcionUno y: funcionDos
        "Recibe dos funciones. Crea una nueva instancia de la estrategia y asigna los valores de los
atributos f1 = funcfuncionUno, f2 = funcionDos con los parametros recibidos. Devuelve la estrategia."

        | estrategia |
        estrategia := EstrategiaEvaluacionSuma new.
        estrategia setFuncionUno: funcionUno setFuncionDos: funcionDos.
        ^estrategia.
        ! !

EstrategiaEvaluacionDeUnaFuncion subclass: #EstrategiaEvaluacionTangente
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionTangente methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:25'!
evaluarEn: numero
        "Recibe un numero y evalua la funcion en ese numero y luego calcula la tangente. Devuelve el valor
obtenido."
        | argumento |
        argumento := (funcion evaluarEn: numero).
        (((argumento \\ (Float pi / 2)) = 0) & ((argumento \\ (Float pi) ~= 0)))ifTrue:
        [
                ExcepcionDominioFuncion  new signal.
        ].
        ^(argumento tan)! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaEvaluacionTangente class
        instanceVariableNames: ''!

!EstrategiaEvaluacionTangente class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:15'!
crearNueva: funcion
        "Recibe una funcion. Crea una nueva instancia de la estrategia y le setea la funcion. Devuelve la
estrategia."
        ^(super crearNueva: self conFuncion: funcion).! !

EstrategiaEvaluacion subclass: #EstrategiaEvaluacionXALaN
        instanceVariableNames: 'exponente'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!EstrategiaEvaluacionXALaN methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/11/2010 20:01'!
setExponente: numero
        exponente := numero.! !


!EstrategiaEvaluacionXALaN methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:20'!
evaluarEn: numero
        "Recibe un numero. Eleva el numero al exponente. Devuelve el resultado."
        ^(numero raisedTo: (exponente))! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!
```

```
EstrategiaEvaluacionXALaN class
	instanceVariableNames: ''!

!EstrategiaEvaluacionXALaN class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 20:00'!
crearConExponente: numero
	"Recibe un numero. Crea una nueva instancia de la estrategia seteando como exponente al numero
recibido. Devuelve la estrategia."

	| estrategia |
	estrategia := EstrategiaEvaluacionXALaN new.
	estrategia setExponente: numero.
	^estrategia.
	! !

Object subclass: #EstrategiaIntegracion
	instanceVariableNames: 'subIntervalos'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaIntegracion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 15:23'!
setSubIntervalos: numero
	"Deberia ser usado solo una vez en el momento de creacion."
	(subIntervalos ~= nil)ifTrue:
	[
		Error new signal: 'No se puede volver a setear'.
	].
	subIntervalos := numero.! !


!EstrategiaIntegracion methodsFor: 'Integracion' stamp: 'DamianSchenkelman 4/17/2010 17:32'!
integrar: f entre: a y: b
	"Devuelve la integral definida de la funcion recibida como parametro entre los valores a y b."
	| sumatoria multiplicadorSentido inicial final fDeInicial fDeFinal |
	"De la implementacion en Python de la regla de integraciontrapezoidal de
http://en.wikipedia.org/wiki/Trapezoidal_rule"
	(b < a)ifTrue:
	[
		inicial := b.
		final := a.
		multiplicadorSentido := -1.
	]
	ifFalse:
	[
		inicial := a.
		final := b.
		multiplicadorSentido := 1.
	].
	fDeInicial := f evaluarEn: inicial.
	fDeFinal := f evaluarEn: final.
	sumatoria := ExtensionesNumeros obtenerSumatoriaDe: ([:x|f evaluarEn: (inicial + ((final-inicial) *
(x/subIntervalos)))]) desde: 1 hasta: subIntervalos.
	^((final - inicial) * (fDeInicial*0.5 + fDeFinal*0.5 + sumatoria) / subIntervalos) *
multiplicadorSentido.! !


"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaIntegracion class
	instanceVariableNames: ''!

!EstrategiaIntegracion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/17/2010 14:38'!
crearNueva
	"Crea una nueva estrategia de integracion y la devuelve."
	| estrategia |
	estrategia := EstrategiaIntegracion new.
	estrategia setSubIntervalos:30000.
	^estrategia.! !
```

```
Object subclass: #EstrategiaMaximosYMinimos
	instanceVariableNames: 'intervalo'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EstrategiaMaximosYMinimos methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 10:48'!
obtenerMaximoDe: funcion desde: inicial hasta: final
	"Devuelve el maximo valor en el intervalo inicial a final."
	| resultados resultado punto |
	(ExtensionesNumeros esCrecienteYNoVacioDesde: inicial hasta: final cada: intervalo)ifFalse:
	[
		ExcepcionEvaluacionIntervalo new signal.
	].
	^self obtenerExtremoDe: funcion desde: inicial hasta:final  bloqueComparacion: [:p :y| p getY < y]
	! !

!EstrategiaMaximosYMinimos methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 10:48'!
obtenerMinimoDe: funcion desde: inicial hasta: final
	"Devuelve el minimo valor en el intervalo inicial a final."
	| resultados resultado punto |
	(ExtensionesNumeros esCrecienteYNoVacioDesde: inicial hasta: final cada: intervalo)ifFalse:
	[
		ExcepcionEvaluacionIntervalo new signal.
	].
	^self obtenerExtremoDe: funcion desde: inicial hasta:final  bloqueComparacion: [:p :y| p getY > y]! !


!EstrategiaMaximosYMinimos methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 16:41'!
setIntervaloEvaluacion: numero
	intervalo := numero! !


!EstrategiaMaximosYMinimos methodsFor: 'as yet unclassified' stamp: 'DamianSchenkelman 4/17/2010 17:57'!
obtenerExtremoDe: funcion desde: inicial hasta:final bloqueComparacion: bloque
	"Obtiene un extremo de la funcion en el intervalo inicial a final usando el bloque de comparacion
para determinar si se alcanzo un nuevo extremo."
	| resultados resultado punto y |
	punto := Punto new.
	punto setX: inicial.
	punto setY: (funcion evaluarEn: inicial).
	inicial to: final by: intervalo do:
	[:x|
		y := funcion evaluarEn: x.
		(bloque value: punto value:y)ifTrue:
		[
			punto setY: y.
			punto setX: x.
		]
	].
	^punto. ! !

EstrategiaMaximosYMinimos subclass: #EstrategiaMaximosYMinimosContinuos
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EstrategiaMaximosYMinimosContinuos class
	instanceVariableNames: 'estrategia'!

!EstrategiaMaximosYMinimosContinuos class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/17/2010 16:49'!
crearNueva
	"Crea una nueva estrategia y la devuelve."
	estrategia := EstrategiaMaximosYMinimosContinuos new.
	estrategia setIntervaloEvaluacion: 0.0001.
	^estrategia! !

EstrategiaMaximosYMinimos subclass: #EstrategiaMaximosYMinimosDiscretos
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!
```

```
EstrategiaMaximosYMinimosDiscretos class
	instanceVariableNames: ''!

!EstrategiaMaximosYMinimosDiscretos class methodsFor: 'as yet unclassified' stamp: 'DamianSchenkelman
4/17/2010 17:40'!
crearNueva
	"Crea una nueva estrategia y la devuelve."
	| estrategia |
	estrategia := EstrategiaMaximosYMinimosDiscretos new.
	estrategia setIntervaloEvaluacion: 1.
	^estrategia! !

Object subclass: #EvaluadorInterseccionFuncionesContinuas
	instanceVariableNames: 'subIntervalos'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!EvaluadorInterseccionFuncionesContinuas methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/18/2010
10:22'!
setSubIntervalos: numero
	subIntervalos := numero.! !


!EvaluadorInterseccionFuncionesContinuas methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 10:48'!
obtenerInterseccionDe: f y: g  desde: inicial hasta: final
	"Recibe dos funciones f y g. Calcula los puntos de interseccion en el intervalo [inicial;final] y los
devuelve en una coleccion de puntos."
	| paso puntos fDeX gDeX punto |
	paso := (final - inicial) abs / subIntervalos.
	(ExtensionesNumeros esCrecienteYNoVacioDesde: inicial  hasta: final  cada: paso)ifFalse:
	[
		ExcepcionEvaluacionIntervalo new signal.
	].
	puntos := OrderedCollection new.
	inicial to: final by: paso do:
	[
		:x|
		fDeX := f evaluarEn: x.
		gDeX := g evaluarEn: x.
		(fDeX closeTo: gDeX)ifTrue:
		[
			punto := Punto new.
			punto setX: x.
			punto setY: fDeX.
			puntos add: punto.
		].
	].
	^puntos.

! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

EvaluadorInterseccionFuncionesContinuas class
	instanceVariableNames: ''!

!EvaluadorInterseccionFuncionesContinuas class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010
11:03'!
crearNuevo
	"Crea un nuevo evaluador de interseccion de funciones y lo devuelve."
	| evaluador |
	evaluador := EvaluadorInterseccionFuncionesContinuas new.
	evaluador setSubIntervalos: 100000.
	^evaluador.! !

Error subclass: #ExcepcionArgumentoInvalido
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

ArithmeticError subclass: #ExcepcionDominioFuncion
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

ArithmeticError subclass: #ExcepcionEvaluacionIntervalo
```

```
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

ArithmeticError subclass: #ExcepcionOperacionAritmeticaInvalida
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

Object subclass: #ExtensionesNumeros
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

ExtensionesNumeros class
        instanceVariableNames: ''!

!ExtensionesNumeros class methodsFor: 'Conjuntos' stamp: 'DamianSchenkelman 4/13/2010 00:06'!
esNatural: numero
        "Recibe un numero y devuelve True si es natural o cero y false en caso contrario."
        ^((numero > 0) & (numero - (numero asInteger) = 0))! !

!ExtensionesNumeros class methodsFor: 'Conjuntos' stamp: 'DamianSchenkelman 4/13/2010 00:07'!
esNaturalOCero: numero
        "Recibe un numero y devuelve True si es natural o cero y false en caso contrario."
        ^((ExtensionesNumeros esNatural: numero) | (numero = 0))! !


!ExtensionesNumeros class methodsFor: 'Operaciones' stamp: 'DamianSchenkelman 4/17/2010 16:14'!
obtenerSumatoriaDe: expresion desde: inicial hasta: final
        "Recibe una expresion. Devuelve la sumatoria desde el valor inicial al final de la expresion."
        | sumatoria auxiliar |
        sumatoria := 0.
        inicial to: final by: 1 do:
        [
                :x|
                auxiliar := expresion value: x.
                sumatoria := sumatoria + auxiliar.
        ].
        ^sumatoria! !


!ExtensionesNumeros class methodsFor: 'Intervalos' stamp: 'DamianSchenkelman 4/18/2010 10:47'!
esCrecienteYNoVacioDesde: inicial hasta: final cada: intervalo
        "Recibe un valor inicial, uno final y un intervalo. Devuelve false si el intervalo es menor a cero,
si el valor inicial es mayor o igual al final o si la diferencia entre el valor final e inicial es menor al
intervalo. Devuelve true en caso contrario."
        (intervalo <= 0)ifTrue:[^false].
        (inicial >= final)ifTrue:[^false].
        ((final - inicial) < intervalo)ifTrue:[^false.].
        ^true.! !

Object subclass: #FabricaFunciones
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'TP190728'!

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:33'!
crearArcoCoseno: funcion
        "Recibe una funcion. Crea una nueva funcion ArcoCoseno con la funcion de argumento y la devuelve."
        ^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionArcoCoseno
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion:
EstrategiaDerivacion crearNueva estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcion: funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:34'!
crearArcoSeno: funcion
        "Recibe una funcion. Crea una nueva funcion ArcoSeno con la funcion de argumento y la devuelve."
        ^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionArcoSeno
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion:
EstrategiaDerivacion crearNueva estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcion: funcion! !
```

```
!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:34'!
crearConstante: numero
	"Recibe un numero. Crea una nueva funcion con ese numero como constante. Si el parametro recibido no
es un numero lanza una excepcion ExcepcionArgumentoInvalido."
	| estrategiaEvaluacion funcion |
	(numero isKindOf: Number)ifFalse:
	[
		ExcepcionArgumentoInvalido new signal.
	].
	estrategiaEvaluacion := EstrategiaEvaluacionConstantes crearNueva: numero.
	funcion := Funcion crearConEstrategiaEvaluacion: estrategiaEvaluacion estrategiaEvaluacionIntervalos:
EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion: EstrategiaDerivacion  crearNueva
estrategiaIntegracion: EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva.
	^funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:34'!
crearCoseno: funcion
	"Recibe una funcion. Crea una nueva funcion Coseno con la funcion de argumento y la devuelve."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionCoseno estrategiaEvaluacionIntervalos:
EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion: EstrategiaDerivacion crearNueva
estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcion: funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:34'!
crearDivision: funcionUno y: funcionDos
	"Recibe dos funciones. Crea una nueva funcion representando el cociente de funcionUno sobre
funfuncionDos. Devuelve la funcion."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionDivision
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion:
EstrategiaDerivacion crearNueva estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcionUno: funcionUno funcionDos: funcionDos! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:35'!
crearExponencial: funcionUno y: funcionDos
	"Recibe dos funciones. Crea una nueva funcion representando funcionUno elevado a funcionDos. Devuelve
la funcion."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionExponencial
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion:
EstrategiaDerivacion crearNueva estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcionUno: funcionUno funcionDos: funcionDos! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:35'!
crearFactorial: funcion
	"Recibe una funcion. Crea una nueva funcion representando el factorial de la funcion recibida como
parametro. Devuelve la funcion nueva."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionFactorial
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloDiscreto new estrategiaDerivacion: nil
estrategiaIntegracion: nil estrategiaExtremos: EstrategiaMaximosYMinimosDiscretos crearNueva funcion:
funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:35'!
crearFibonacci: funcion
	"Recibe una funcion. Crea una nueva funcion representando el numero de la serie de fibonacci de la
funcion recibida como parametro. Devuelve la funcion nueva."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionFibonacci
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloDiscreto new estrategiaDerivacion: nil
estrategiaIntegracion: nil estrategiaExtremos: EstrategiaMaximosYMinimosDiscretos crearNueva funcion:
funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 11:48'!
crearFuncionConEstrategiaEvaluacion: estrategiaEvaluacion estrategiaEvaluacionIntervalos:
estrategiaIntervalos estrategiaDerivacion: estrategiaDerivacion estrategiaIntegracion: estrategiaIntegracion
estrategiaExtremos: estrategiaExtremos funcion: funcion
	"Recibe un tipo de estrategia de evaluacion, una estrategia de evaluacion de intervalos, una de
derivacion, una de integracion, una de extremos y dos funciones. Crea una funcion con las estrategias de
evaluacion y usa las funciones para crear la estrategia de evaluacion."
	| estEvaluacion |
	estEvaluacion := estrategiaEvaluacion crearNueva: funcion.
	^(Funcion crearConEstrategiaEvaluacion: estEvaluacion estrategiaEvaluacionIntervalos:
estrategiaIntervalos estrategiaDerivacion: estrategiaDerivacion estrategiaIntegracion: estrategiaIntegracion
estrategiaExtremos: estrategiaExtremos)! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 11:47'!
crearFuncionConEstrategiaEvaluacion: estrategiaEvaluacion estrategiaEvaluacionIntervalos:
estrategiaIntervalos estrategiaDerivacion: estrategiaDerivacion estrategiaIntegracion: estrategiaIntegracion
estrategiaExtremos: estrategiaExtremos funcionUno: funcionUno  funcionDos: funcionDos
	"Recibe un tipo de estrategia de evaluacion, una estrategia de evaluacion de intervalos, una de
derivacion, una de integracion, una de extremos y dos funciones. Crea una funcion con las estrategias de
```

```
evaluacion y usa las funciones para crear la estrategia de evaluacion."
	| funcion estEvaluacion |
	estEvaluacion := estrategiaEvaluacion crearNuevaCon: funcionUno y: funcionDos.
	^(Funcion crearConEstrategiaEvaluacion: estEvaluacion estrategiaEvaluacionIntervalos:
estrategiaIntervalos estrategiaDerivacion: estrategiaDerivacion estrategiaIntegracion: estrategiaIntegracion
estrategiaExtremos: estrategiaExtremos)! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:36'!
crearLogaritmoNatural: funcion
	"Recibe una funcion. Crea una nueva funcion representando el logaritmo de la funcion recibida como
parametro como argumento. Devuelve la funcion nueva."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionLogaritmoNatural
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion:
EstrategiaDerivacion crearNueva estrategiaIntegracion: EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcion: funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:36'!
crearMultiplicacion: funcionUno y: funcionDos
	"Recibe dos funciones. Crea una nueva representando el producto de ambas y la devuelve."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionMultiplicacion
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion:
EstrategiaDerivacion crearNueva estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcionUno: funcionUno funcionDos: funcionDos! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:36'!
crearResta: funcionUno y: funcionDos
	"Recibe dos funciones. Crea una nueva funcion representando la resta de la primera menos la segunda y
la devuelve."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionResta estrategiaEvaluacionIntervalos:
EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion: EstrategiaDerivacion crearNueva
estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcionUno: funcionUno funcionDos: funcionDos! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:36'!
crearSeno: funcion
	"Recibe una funcion. Crea una nueva funcion Seno con la funcion de argumento y la devuelve."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionSeno estrategiaEvaluacionIntervalos:
EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion: EstrategiaDerivacion crearNueva
estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcion: funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:36'!
crearSuma: funcionUno y: funcionDos
	"Recibe dos funciones. Crea una nueva funcion representando la suma de ambas y la devuelve."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionSuma estrategiaEvaluacionIntervalos:
EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion: EstrategiaDerivacion crearNueva
estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcionUno: funcionUno funcionDos: funcionDos
	! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:37'!
crearTangente: funcion
	"Recibe una funcion. Crea una nueva funcion Tangente con la funcion de argumento y la devuelve."
	^self crearFuncionConEstrategiaEvaluacion: EstrategiaEvaluacionTangente
estrategiaEvaluacionIntervalos: EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion:
EstrategiaDerivacion crearNueva estrategiaIntegracion:  EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva funcion: funcion! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 20:25'!
crearTerminoPolinomialConMultiplicador: multiplicador exponente: exponente
	"Recibe un multiplicador y un exponente. Crea una multiplicacion entre una constante con el
multiplicador y un termino X a la N con el exponente y la devuelve."
	^(self crearMultiplicacion: (self crearConstante: multiplicador)  y: (self crearXALaNConExponente:
exponente)).! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 20:52'!
crearX
	"Crea un termino polinomial con multiplicador 1 y exponente 1 (es decir x) y lo devuelve."
	^(self crearTerminoPolinomialConMultiplicador: 1 exponente: 1).! !

!FabricaFunciones methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 12:37'!
crearXALaNConExponente: numero
	"Recibe un numero. Crea una funcion representando X elevado a la N y la devuelve."
	| estrategia funcion |
	(numero isKindOf: Number)ifFalse:
	[
		ExcepcionArgumentoInvalido new signal.
	].
	estrategia := EstrategiaEvaluacionXALaN crearConExponente: numero.
```

```
	funcion := Funcion crearConEstrategiaEvaluacion: estrategia estrategiaEvaluacionIntervalos:
EstrategiaEvaluacionIntervaloContinuo new estrategiaDerivacion: EstrategiaDerivacion crearNueva
estrategiaIntegracion: EstrategiaIntegracion crearNueva estrategiaExtremos:
EstrategiaMaximosYMinimosContinuos crearNueva.
	^funcion! !

Object subclass: #Funcion
	instanceVariableNames: 'estrategiaEvaluacion estrategiaEvaluacionIntervalos estrategiaDerivacion
estrategiaIntegracion estrategiaExtremos'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!Funcion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 13:19'!
setEstrategiaDerivacion: estrategia
	(estrategiaDerivacion ~=  nil)ifTrue:
	[
		Error new signal: 'No se puede volver a setear'.
	].
	estrategiaDerivacion := estrategia.! !

!Funcion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 12:13'!
setEstrategiaEvaluacion: estrategia
	(estrategiaEvaluacion ~=  nil)ifTrue:
	[
		Error new signal: 'No se puede volver a setear'.
	].
	estrategiaEvaluacion := estrategia.! !

!Funcion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 12:13'!
setEstrategiaEvaluacionIntervalos: estrategiaIntervalos
	(estrategiaEvaluacionIntervalos ~=  nil)ifTrue:
	[
		Error new signal: 'No se puede volver a setear'.
	].
	estrategiaEvaluacionIntervalos := estrategiaIntervalos! !

!Funcion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 18:16'!
setEstrategiaExtremos: estrategia
	(estrategiaExtremos ~=  nil)ifTrue:
	[
		Error new signal: 'No se puede volver a setear'.
	].
	estrategiaExtremos := estrategia.! !

!Funcion methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 15:37'!
setEstrategiaIntegracion: estrategia
	(estrategiaIntegracion ~=  nil)ifTrue:
	[
		Error new signal: 'No se puede volver a setear'.
	].
	estrategiaIntegracion := estrategia.! !


!Funcion methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/11/2010 21:26'!
componerCon: funcion evaluarEn: numero
	"Recibe una funcion y un numero. Evalua la funcion en ese numero. Evalua la funcion actual en el
resultado. Devuelve el resultado final."
	| resultado |
	resultado :=    funcion evaluarEn: numero.
	^self evaluarEn: resultado.! !

!Funcion methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/12/2010 23:49'!
evaluarDesde: inicial hasta: final cada: intervalo
	"Recibe un valor inical uno final y un intervalo. Devuelve "
	(estrategiaEvaluacionIntervalos puedeEvaluarDesde: inicial hasta: final  cada: intervalo)ifFalse:
	[
		^ExcepcionEvaluacionIntervalo new signal.
	].
	^estrategiaEvaluacionIntervalos evaluar: self desde: inicial  hasta: final  cada: intervalo.! !

!Funcion methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/15/2010 20:40'!
evaluarEn: numero
	"Recibe un numero. Llama a la estrategia de evaluacion para que evalue en ese numero."
	^estrategiaEvaluacion evaluarEn: numero.! !

!Funcion methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/17/2010 18:23'!
obtenerMaximoEntre: inicial y: final
	"Obtiene el punto maximo de la funcion entre el intervalo inicial y final."
```

```
        ^estrategiaExtremos obtenerMaximoDe: self desde: inicial hasta: final.! !

!Funcion methodsFor: 'Evaluacion' stamp: 'DamianSchenkelman 4/18/2010 10:06'!
obtenerMinimoEntre: inicial y: final
        "Obtiene el punto maximo de la funcion entre el intervalo inicial y final."
        ^estrategiaExtremos obtenerMinimoDe: self desde: inicial hasta: final.! !


!Funcion methodsFor: 'Derivacion' stamp: 'DamianSchenkelman 4/18/2010 12:28'!
derivarEn: numero
        "Recibe un numero. Devuelve la derivada numerica de la funcion en el numero. Si la funcion no es
derivable lanza una excepcion del tipo ExcepcionOperacionAritmeticaInvalida."
        (estrategiaDerivacion == nil)ifTrue:
        [
                ExcepcionOperacionAritmeticaInvalida new signal.
        ].
        ^estrategiaDerivacion derivar: self en: numero.! !


!Funcion methodsFor: 'Integracion' stamp: 'DamianSchenkelman 4/18/2010 13:03'!
integrarEntre: inicial y: final
        "Devuelve la integral definida de la funcion entre el valor inicial y final. Si la funcion no es
integrable lanza una excepcion del tipo ExcepcionOperacionAritmeticaInvalida."
        (estrategiaIntegracion == nil)ifTrue:
        [
                ExcepcionOperacionAritmeticaInvalida new signal.
        ].
        ^estrategiaIntegracion integrar: self entre: inicial  y: final.! !


!Funcion methodsFor: 'Operaciones' stamp: 'DamianSchenkelman 4/18/2010 11:26'!
* unaFuncion
        "Recibe una funcion y devuelve una nueva funcion representando el producto de la actual y la
recibida. Se puede hacer de otra forma, pero es una suerte de azucar sintactico."
        ^((FabricaFunciones new) crearMultiplicacion: self y: unaFuncion).! !

!Funcion methodsFor: 'Operaciones' stamp: 'DamianSchenkelman 4/18/2010 11:20'!
+ unaFuncion
        "Recibe una funcion y devuelve una nueva funcion representando la suma de la actual y la recibida. Se
puede hacer de otra forma, pero es una suerte de azucar sintactico."
        ^((FabricaFunciones new) crearSuma: self y: unaFuncion).! !

!Funcion methodsFor: 'Operaciones' stamp: 'DamianSchenkelman 4/18/2010 11:23'!
- unaFuncion
        "Recibe una funcion y devuelve una nueva funcion representando la resta de la actual menos la
recibida. Se puede hacer de otra forma, pero es una suerte de azucar sintactico."
        ^((FabricaFunciones new) crearResta: self y: unaFuncion).! !

!Funcion methodsFor: 'Operaciones' stamp: 'DamianSchenkelman 4/18/2010 11:30'!
/ unaFuncion
        "Recibe una funcion y devuelve una nueva funcion representando el cociente de la actual sobre la
recibida. Se puede hacer de otra forma, pero es una suerte de azucar sintactico."
        ^((FabricaFunciones new) crearDivision: self y: unaFuncion).! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Funcion class
        instanceVariableNames: ''!

!Funcion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/11/2010 17:57'!
crearConEstrategiaEvaluacion: estrategia
        "Recibe una estrategia de evaluacion. Crea una funcion y le setea esta estrategia de evaluacion.
Devuelve la funcion."
        | funcion |
        funcion := self new.
        funcion setEstrategiaEvaluacion: estrategia.
        ^funcion! !

!Funcion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/12/2010 23:15'!
crearConEstrategiaEvaluacion: estrategiaEvaluacion estrategiaEvaluacionIntervalos: estrategiaIntervalos
        "Recibe una estrategia de evaluaciony una de evaluacion de intervalos. Crea una funcion con ambas
estrategias y la devuelve."
        | funcion |
        funcion := Funcion crearConEstrategiaEvaluacion: estrategiaEvaluacion.
        funcion setEstrategiaEvaluacionIntervalos: estrategiaIntervalos.
        ^funcion.! !

!Funcion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/17/2010 15:37'!
crearConEstrategiaEvaluacion: estrategiaEvaluacion estrategiaEvaluacionIntervalos: estrategiaIntervalos
```

```
estrategiaDerivacion: estrategiaDerivacion estrategiaIntegracion: estrategiaIntegracion
	"Recibe una estrategia de evaluacion, una de evaluacion de intervalos, una de derivacion y una de
integracion. Crea una funcion con las estrategias y la devuelve."
	| funcion |
	funcion := Funcion crearConEstrategiaEvaluacion: estrategiaEvaluacion estrategiaEvaluacionIntervalos:
estrategiaIntervalos.
	funcion setEstrategiaDerivacion: estrategiaDerivacion.
	funcion setEstrategiaIntegracion: estrategiaIntegracion.
	^funcion.! !

!Funcion class methodsFor: 'Creacion' stamp: 'DamianSchenkelman 4/18/2010 11:35'!
crearConEstrategiaEvaluacion: estrategiaEvaluacion estrategiaEvaluacionIntervalos: estrategiaIntervalos
estrategiaDerivacion: estrategiaDerivacion estrategiaIntegracion: estrategiaIntegracion estrategiaExtremos:
estrategiaExtremos
	"Recibe una estrategia de evaluacion, una de evaluacion de intervalos, una de derivacion, una de
integracion y una de evaluacion extremos. Crea una funcion con las estrategias y la devuelve."
	| funcion |
	funcion := self new.
	funcion setEstrategiaEvaluacion: estrategiaEvaluacion.
	funcion setEstrategiaEvaluacionIntervalos: estrategiaIntervalos.
	funcion setEstrategiaDerivacion: estrategiaDerivacion.
	funcion setEstrategiaIntegracion: estrategiaIntegracion.
	funcion setEstrategiaExtremos: estrategiaExtremos.
	^funcion.! !

Object subclass: #Punto
	instanceVariableNames: 'x y'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'TP190728'!

!Punto methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 16:37'!
getX
	^x! !

!Punto methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 16:37'!
getY
	 ^y! !

!Punto methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 16:36'!
setX: valor
	x := valor.! !

!Punto methodsFor: 'Propiedades' stamp: 'DamianSchenkelman 4/17/2010 16:37'!
setY: valor
	y:= valor! !
```

Checklist de corrección

Esta sección es para uso exclusivo de la cátedra, por favor no modificar.

Carpeta

Generalidades

- ¿Son correctos los supuestos y extensiones?
- ¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

Modelo

- ¿Está completo?¿Contempla la totalidad del problema?
- ¿Respeta encapsulamiento?
- ¿Hace un buen uso de excepciones?
- ¿Utiliza polimorfismo en las situaciones esperadas?

Diagramas

Diagrama de clases

- ¿Está completo?
- ¿Está bien utilizada la notación?

Diagramas de secuencia

- ¿Está completo?
- ¿Es consistente con el diagrama de clases?
- ¿Está bien utilizada la notación?

Diagrama de estados

- ¿Está completo?
- ¿Está bien utilizada la notación?

Diagrama de paquetes

- ¿Está completo?
- ¿Está bien utilizada la notación?

Código

Generalidades

- ¿Respeta estándares de codificación?
- ¿Está correctamente documentado?