Orientación a Objetos 1 - 2017

Práctica 4

Ejercicio 1: El mensaje #between:and:

Ejecute paso a paso las siguientes expresiones utilizando la acción *Into* del *Debugger* para responder las preguntas que se indican debajo del código:

```
$4 between: $t and: $y.
(Date today)
      between: (Date yesterday)
      and: (Date tomorrow)
```

Responda:

1. ¿Dónde está implementado el código de #`between:and:`?
2. ¿Dónde están implementados los métodos #<= y #>=?
3. ¿Los caracteres y las fechas necesitan implementar #`between:and:`? ¿Los caracteres y las fechas necesitan implementar los métodos #<= y #>=? Justifique.
4. Busque en `Magnitude` otro ejemplo similar a `#between:and:` donde un método envía mensajes a objetos polimórficos.

Ejercicio 2: Intervalo de tiempo

En Smalltalk, las fechas se representan con instancias de la clase `Date`. Por ejemplo, el envío del mensaje #`today` a la clase `Date` retorna la fecha actual.

Tareas:

1. Investigue cómo hacer para crear una fecha determinada, por ejemplo 29/06/1986. Sugerencia: vea los mensajes de clase de la clase `Date`, en el protocolo `instance- creation`.
2. Investigue cómo hacer para ver si la fecha de hoy está entre las fechas 29/06/1986 y 29/06/2018. Sugerencia: vea los métodos de instancia que `Date` hereda de `Magnitude` (protocolo `testing`).
3. Sea un objeto `DateLapse` que representa el lapso entre dos fechas. La primera fecha se conoce como `´from'` y la segunda como `'to'`.
   Complete la implementación de la clase `DateLapse` a partir de la definición que se encuentra en el archivo `DateLapse.st` para que satisfaga el requerimiento de entender los siguientes mensajes:

```
#from
"Retorna la fecha de inicio del rango"

#to
"Retorna la fecha de fin del rango"

#from: aDateFrom to: aDateTo
"Es un método privado que asigna la fecha inicial y final de
un objeto DateLapse"

#sizeInDays
"retorna la cantidad de días entre la fecha 'from' y la fecha
'to'"

#includesDate: aDate
"recibe un objeto Date y retorna true si la fecha está entre
el 'from' y el 'to' del receptor y false en caso contrario"
```

4. Ejecute los tests provistos por la cátedra para verificar su implementación. Estos tests están definidos en la clase `DateLapseTest`, en el archivo `DateLapseTest.st`.

5. Asumiendo que implementó la clase `DateLapse` con dos variables de instancia `'from'` y `'to'`, modifique su implementación para que un `DateLapse` quede representado por los atributos `'from'` y `'sizeInDays'` (su nueva implementación debe estar basada sólo en estas variables de instancia).
   Sugerencia: si quiere conservar la versión original, antes de realizar algún cambio haga un `fileOut` de la clase `DateLapse`.

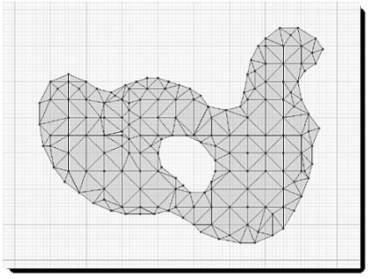
6. Vuelva a correr los tests y verifique que siguen pasando exitosamente. Los cambios en la estructura interna de un objeto sólo deben afectar a la implementación de sus métodos. Estos cambios deben ser transparentes para quien le envía mensajes, no debe notar ningún cambio y seguir usándolo de la misma forma.
7. Defina ahora los mensajes necesarios para crear una instancia de `DateLapse` a partir del envío de un mensaje a un objeto `Date`. Usted debería escribir algo de la siguiente forma para instanciar el `DateLapse`.

   ```
   unaFecha ~> otraFecha
   ```

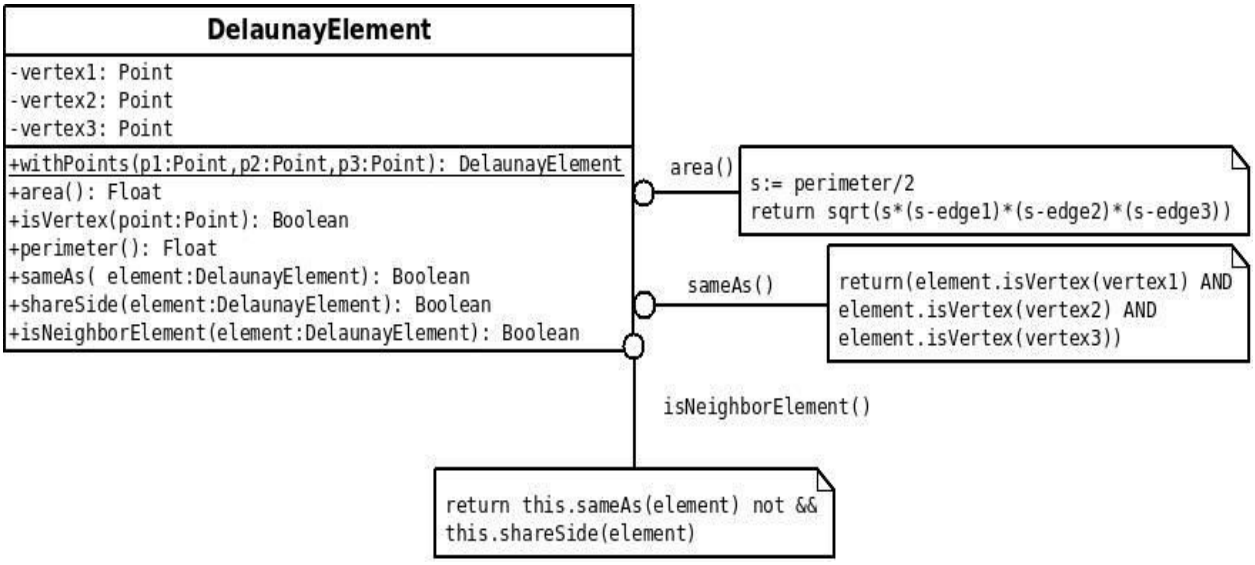
   Esto es similar a lo que ocurre con los objetos `Point` que se pueden crear a partir del envío de un mensaje @ a un objeto `Number`. La expresión `3@5` al ser evaluada retorna un objeto Point con coordenada x=3 y con coordenada y=5.

Ejercicio 3: Triangulación de Delaunay (I)

Una Triangulación de Delaunay es una red de triángulos que suele ser utilizada en aplicaciones que necesitan hacer cálculos basados en geometría, por ejemplo para calcular un área irregular como muestra la figura.



Estudie el diagrama de la clase `DelaunayElement` que se presenta abajo (una instancia de `DelanunayElement` modela un triángulo).



Implemente la clase `DelaunayElement` en Smalltalk teniendo en cuenta los siguientes puntos:

1. La operación `shareSide(:DelaunayElement)` retorna si dos triángulos comparten un lado o no
2. Hay un constructor que recibe multiples parámetros: `withPoints(p1 Point, p2 Point, p3 Point)`, ¿Cómo implementaría esto en Smalltalk?
3. Las instancias de la clase `Point` entienden el mensaje #`dist:`. Este método espera como parámetro otra instancia de `Point` y devuelve un número que representa la distancia (geométrica) entre esos dos puntos
4. En el cálculo del área se utiliza (Fórmula de Heron) el perímetro y cada uno de los lados, mientras que el `DelaunayElement` guarda los vértices del triángulo. ¿Cuál es la mejor manera de implementar el “cálculo” de cada uno de los lados?
5. La operación `sameAs()` retorna un booleano que dice si dos triángulos son iguales. Cómo podría implementar este método en Smalltalk?
6. La operación `shareSide()` retorna un booleano que dice si dos triángulos tienen un lado en común
7. La operación `isNeighborElement()` retorna un booleano que dice si dos triángulos son vecinos: no son el mismo y comparten un lado.

Ejercicio 4: Operaciones lógicas

Implemente en Pharo las siguientes operaciones lógicas: `implies`, `nand` y `nor`. Las operaciones deben ser mensajes enviados a objetos booleanos y están definidas de acuerdo con las siguientes tablas de verdad:

Implies

a	b	a implies: b
false	false	true
false	true	true
true	false	false
true	true	true

Nand

a	b	a nand: b
false	false	true
false	true	true
true	false	true
true	true	false

Nor

a	b	a nor: b
false	false	true
false	true	false
true	false	false
true	true	false

Utilice los TestCases provistos por la cátedra en el archivo `TestBoolean.st` para probar su implementación.

Ejercicio 5: los mensajes `#and:` , `#&`

Tanto `and:` como `&` representan al “y” lógico. Investigue qué recibe como parámetro cada uno de estos mensajes y cómo influye en las operaciones lógicas.

Ejercicio 6: el mensaje `#whileTrue:`

Analice el siguiente código:

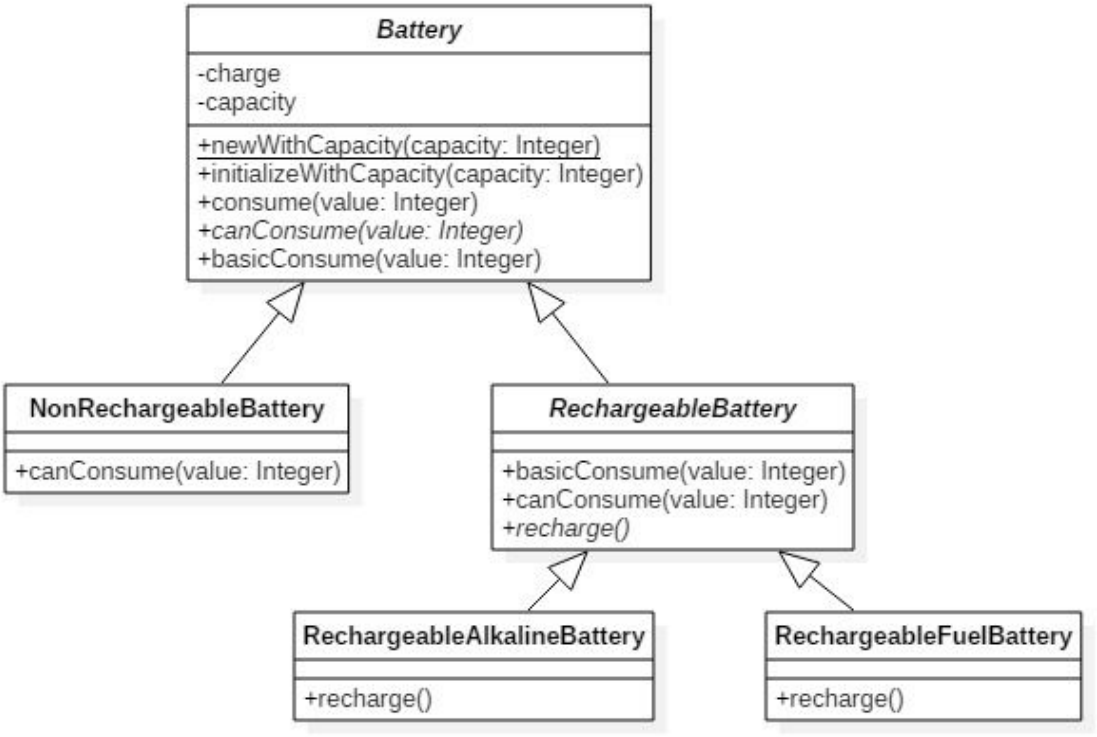
```
i := 0.
[ i < 5 ]
      whileTrue: [Transcript show: 'Que lindo es Smalltalk'.
                  i := i + 1.]
```

Tareas:

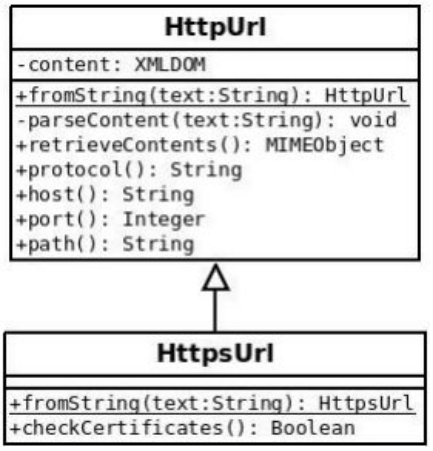
1. Indique qué resulta al evaluar esas expresiones.
2. Busque la clase donde se implementa el mensaje #`whileTrue`:
3. Conteste: ¿por qué no tiene sentido implementar este mensaje en la clase `Boolean`?.
4. Analice la implementación del mensaje #`whileTrue`:. ¿Cuál es el mensaje □que permite ejecutar un bloque?

Ejercicio 7: Diagramas de Clase UML

Analice los siguientes Diagramas de Clases de las figuras y conteste las preguntas:



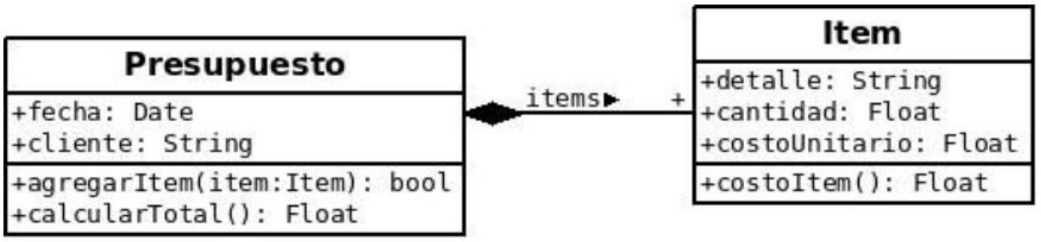
1. Liste los mensajes que entiende una instancia de `RechargeableFuelBattery`.
2. ¿A una instancia de `RechargeableAlkalineBattery` se le puede enviar el mensaje `consume:anInteger`? Justifique su respuesta.
3. ¿Qué significa un mensaje subrayado en un diagrama de clases?
4. ¿Qué significa el nombre de una clase en cursivas en un diagrama de clases? ¿y en el nombre de los métodos?
5. ¿Qué significa el simbolo - delante de las variables de instancia? y los símbolos + y - delante de los mensajes?
6. ¿Por qué el mensaje `basicConsume` está definido en Battery, en `RechargeableBattery`, pero no en `NonRechargeableBattery`? Justifique su respuesta.
7. Liste las variables que poseen las instancias de `RechargeableAlkalineBattery` y de `NonrechargeableBattery`



1. Liste los mensajes que entiende una instancia de `HttpsUrl`
2. ¿A una instancia de `HttpsUrl` se le puede enviar el mensaje `parseContent(text:String)`?
3. ¿Qué significa que un método esté subrayado en el diagrama de clases?
4. ¿Qué significan los + y - adelante de los métodos?
5. ¿Qué significa que un método esté en cursiva?
6. ¿Por qué cree que ambas clases implementan el método `fromString(:String)`?
7. ¿Qué mensajes pueden enviarse a la clase `HttpsUrl`?
8. Indique cuáles son las variables de instancia de `HttpsUrl`

Ejercicio 8: Presupuesto

Considere el diagrama de clases de la figura y responda a las preguntas.

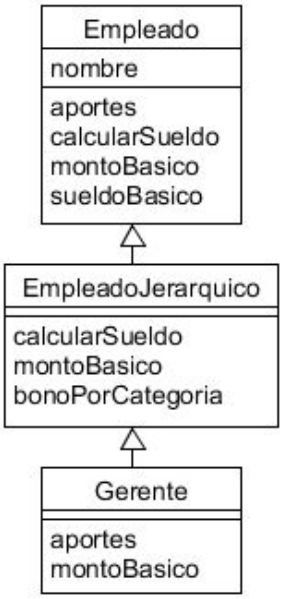


:

1. Si tuviera que implementar la clase Presupuesto en Smalltalk:
    a. ¿Cuáles serían las variables de instancia?
    b. ¿Qué variables debería inicializar y cómo lo haría?
    c. ¿Cómo haría que las variables fecha y cliente sean públicas?
2. ¿Qué significa el diamante relleno en la relación items?
    a. ¿Debería implementar código adicional?¿cuál?

Ejercicio 9: Method lookup

Sea la jerarquía de `Empleado` como muestra la figura de la izquierda, cuya implementación se incluye en la tabla de la derecha



Empleado	EmpleadoJerarquico	Gerente
sueldoBasico ^self montoBasico - self aportes	**calcularSueldo** ^super sueldoBasico + self bonoPorCategoria	**aportes** ^self montoBasico * 0.05
montoBasico ^ 700	**montoBasico** ^ 850	**montoBasico** ^1000
aportes ^10	**bonoPorCategoria** ^ 200	

Analice cada uno de los siguientes fragmentos de código y resuelva las tareas indicadas abajo:

```
| gerente |
gerente := Gerente new.
gerente aportes
```

```
| gerente |
gerente := Gerente new.
gerente calcularSueldo
```

a) Liste los métodos que son invocados como resultado del envío del último mensaje.
b) Responda qué retorna la última expresión en cada caso

Ejercicio 10: Colecciones

Dada una instancia de la clase `OrderedCollection` que recibe los siguientes mensajes:

```
#add:
#remove:
#size
#last
#first
#includes:
#at:put:.
```

a) Indique cuáles mensajes tienen por objetivo modificar al objeto receptor y cuáles buscan obtener información.
b) ¿Los mensajes que modifican el objeto receptor retornan algo?
c) Imagine dos ejemplos donde se utilicen al menos tres de los métodos listados arriba

Ejercicio 11: Más colecciones

Con el System Browser de Pharo, analice y compare el comportamiento de los mensajes enunciados abajo para cada una de estas clases: `OrderedCollection`, `SortedCollection`, `Array`, `Dictionary`, `Bag` y `Set`.

`#add:`	`#at:`	`#at:put:`
`#size`	`#do:`	`#detect:`
`#select:`	`#collect:`	`#reject:`
`#inject:into:`	`#includes:`	`#isEmpty`

Responda a las siguientes preguntas:

1. ¿Es posible que algunos mensajes no sean aplicables para algunas colecciones? Por ejemplo, ¿Se le puede enviar el mensaje #`add`: a un `Array`? ¿Y a un `Set`? ¿Se le puede enviar el mensaje #`at`: y #`at:put:` a un `Set`?

2. En respuesta al mensaje #`select`:, ¿qué retorna un `Array`? ¿Y un `Dictionary`? ¿Y una `SortedCollection`?

3. En respuesta al mensaje #`size`, ¿qué retorna un `Array` creado con `Array new:10`? ¿Qué retorna una `OrderedCollection` creada con `OrderedCollection new:10`?

4. ¿Cómo se elimina un elemento de un `Array`? ¿Es posible?

5. ¿Como se averigua la posición de un elemento en un `Array`? ¿Y la del primer elemento que cumple una condición? ¿Es posible hacerlo en un `Set`?.

6. Indique la diferencia entre #`detect`: y #`detect:ifNone:`. ¿Para qué sirve el bloque que se envía como parámetro en #`ifNone`:?

7. ¿Cómo crea una `SortedCollection` para contener instancias de `String` ordenadas por tamaño? ¿Cómo crea una `SortedCollection` para contener instancias de `String` ordenadas alfabéticamente?

8. ¿Cómo consigue los elementos en un `Array` eliminado las repeticiones?

9. ¿Cuál es el problema con la siguiente expresión si col es un `Set` con elementos? ¿Y si fuera una `OrderedCollection`?

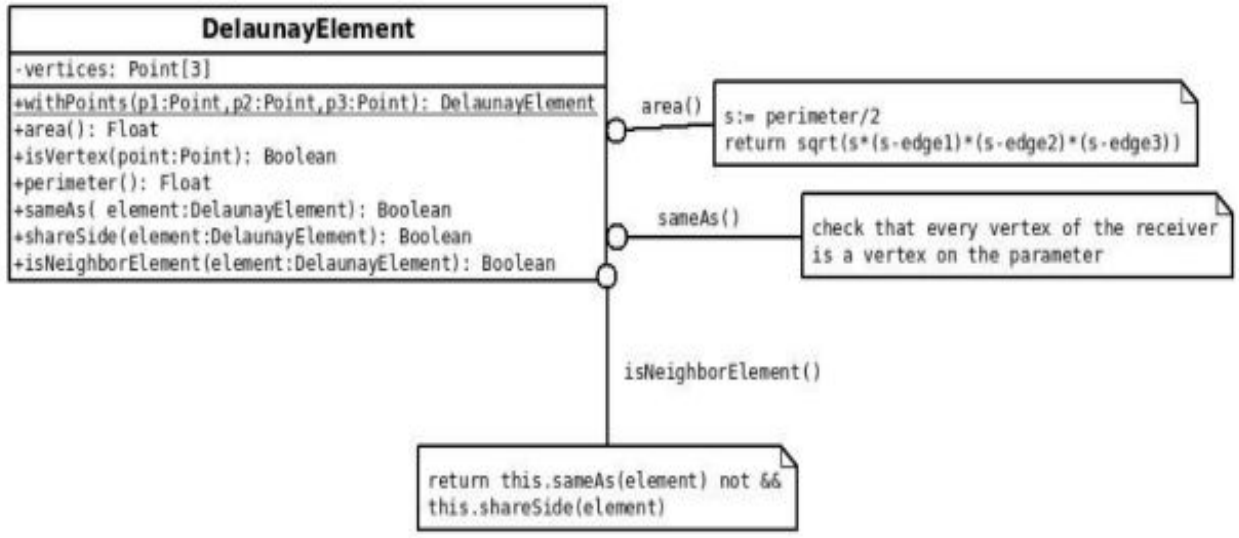
   a. `col do: [ :each | col remove: each]`

10. ¿Cuál es el efecto de enviar el mensaje #`add`: con nil como parámetro a una `OrderedCollection`? ¿Por qué?

11. ¿Qué diferencia hay entre los mensajes #`includes`: y #`contains`:?

Ejercicio 12: Triangulación de Delaunay (II)

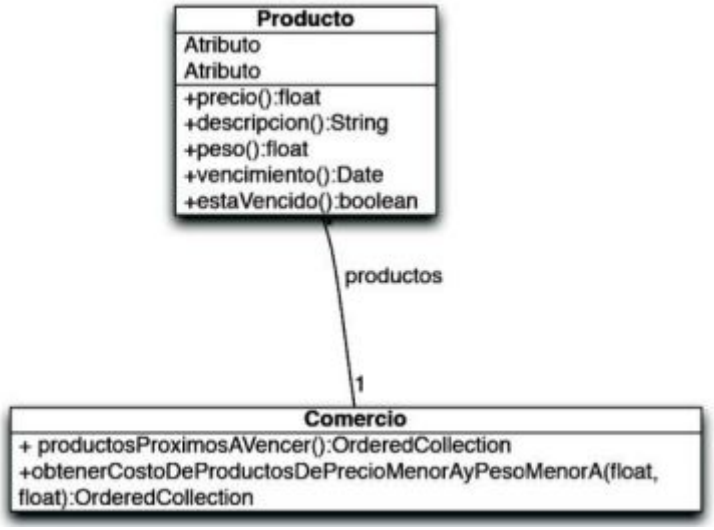
Reimplemente la clase `DelaunayElement` (presentado en el ejercicio 3 de esta práctica) según se documenta en el siguiente diagrama.



Note que los vértices se guardan ahora en una colección en lugar de tres variables.

Ejercicio 13

Analice el siguiente Diagrama de Clases



Como puede apreciarse, la clase `Producto` posee el siguiente protocolo: `#precio`, `#descripcion`, `#peso`, `#vencimiento`, `#estaVencido`.

Se desea modificar el código de los métodos de la clase Comercio de forma tal que utilicen métodos de colecciones de mayor nivel de abstracción sin alterar el comportamiento original. La implementación actual de estos métodos es la siguiente:

`#productosProximosAVencer`

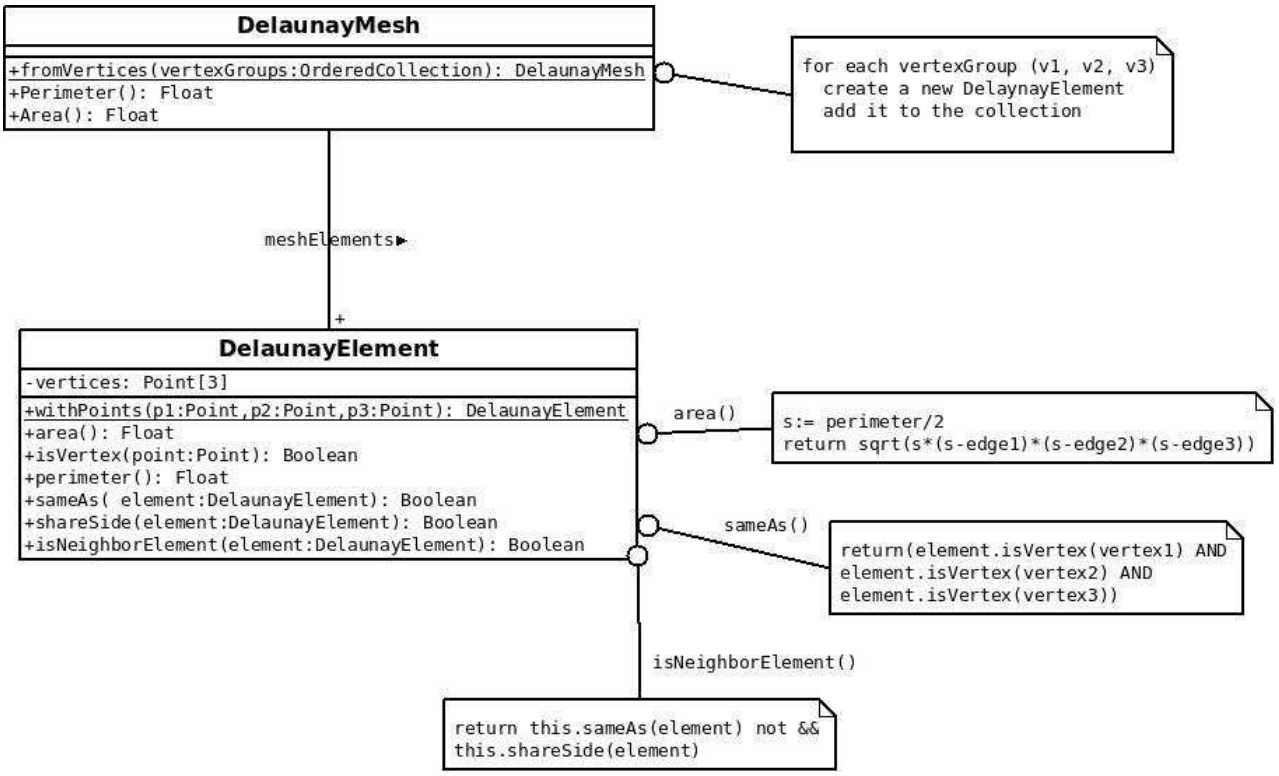
```
"Retorna los productos que vencen en siete días"
|nextWeek results|
nextWeek:= Date today addDays:7.
results:= OrderedCollection new.
self productos do:[ :producto | producto vencimiento = nextWeek
        ifTrue:[ results add:producto] ].
^results

#obtenerCostoDeProductosDePrecioMenorA: unPrecio
yPesoMenorA:unPeso
"retorna la suma del precio de todos los productos recibidos como
parámetros que su valor individual sea menor a unPrecio y su peso
individual sea menor a unPeso"
|results total|
total:=0.
self productos do:[ :p | p peso < unPeso
     ifTrue:[ p precio < unPrecio ifTrue:[ total:= total + p
precio]. ].
].
^total
```

Reimplemente utilizando métodos de colecciones de mayor abstracción.

Ejercicio 14

Complete el modelo del ejercicio 13 con la clase `DelaunayMesh`. `DelaunayMesh` está compuesto por un conjunto de triángulos (`DelaunayElement`) y puede calcular el área y el perímetro. El área es la sumatoria de los triángulos que conforman el DelaunayMesh. El perímetro resulta de la sumatoria de los “lados externos” o los “lados no compartidos” de los triángulos que conforman el `DelaunayMesh`. `DelaunayMesh` tiene un constructor #`fromVertices`: que recibe los vértices de lostriángulos. El siguiente diagrama de clases muestra una vista preliminar que puede utilizar para diseñar y programar su solución.



Ejercicio 15

Utilizando el entorno de simulación de robots mejore la implementación del robot para que tenga una nueva manera de moverse que le permita sortear obstáculos como paredes u otros robots mientras no haya alcanzado la distancia solicitada.

Ejercicio 16

Considerando las descripciones de los algoritmos de cifrado Vigenère y Rail Fence cumpla con las siguientes consignas

1) Documente con un Diagrama de Clases
2) Implemente en Smalltalk
3) Verifique su implementación utilizando los test cases provistos por la clase

a) Cifrado por sustitución

Vigenère es un cifrado por sustitución “polialfabetico”. Utiliza una “palabra” como clave y un alfabeto de referencia (como lo hace un Shift Cipher). En Vigenère cada letra de la clave se utiliza para definir un mapeo de sustitución.

Para usar el cifrado de Vigenère al codificar un mensaje, primero debemos tomar la palabra clave y repetirla hasta que haya cubierto el mensaje (que queremos cifrar) entero.
Entonces, cada letra el mensaje original es cifrado con el “mapeo de sustitución” que le corresponde según la letra de la clave.

Ejemplo, se desea cifrar el mensaje “I LOVE CRYPTOGRAPHY” utilizando Vigenère con clave “WORD” y el alfabeto tradicional “ABC...XYZ”. El siguiente cuadro presenta el mensaje a cifrar, la clave desplegada tantas veces como letras tiene el mensaje original y por último el mensaje cifrado resultante .

Texto	I L O V E C R Y P T O G R A P H Y
Clave	W O R D W O R D W O R D W O R D W
Cifrado	E Z F Y A Q I B L H F J N O G K U

Note que los espacios no son considerados en el proceso de cifrado.

b) Cifrado por transposición

Un cifrado por transposición es un tipo de cifrado en el que unidades de texto plano se cambian de posición siguiendo un esquema bien definido; las 'unidades de texto' pueden ser de una sola letra (el caso más común), pares de letras, tríos de letras, mezclas de lo anterior.

Rail Fence

En Rail Fence, el texto plano se escribe hacia abajo y diagonalmente en sucesivos "carriles" de una cerca imaginaria, y luego se mueve hacia arriba cuando llegamos al riel inferior. Cuando llegamos al riel superior, el mensaje se escribe de nuevo hacia abajo hasta que se escribe todo el texto plano. El mensaje se lee entonces en filas. Por ejemplo, si tenemos 3 "carriles" y un mensaje original: “WE ARE DISCOVERED. FLEE AT ONCE”, el cifrado se representa como:

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

Y el texto cifrado resultante será “WECRLTEERDSOEEFEAOCAIVDEN”. Note que se han perdido los espacios.