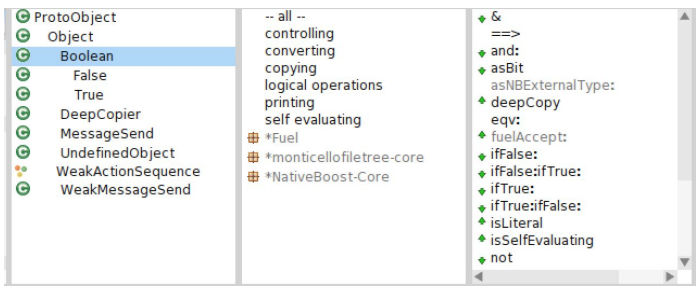
Orientación a Objetos 1 - 2016
Práctica 3

Ejercicio 1: Explorando Pharo

Abra el Browser (World Menu → System Browser) y desarrolle las siguientes tareas:

1. Busque la clases `ProtoObject y Object`.
2. Lea los comentarios de los métodos `#=` (en Object) y `#==` `(en ProtoObject)` y explique la diferencia entre ambos. Verifique con un ayudante.
3. Encuentre todas las implementaciones del método `#=` e `#==`
   a. *Una manera de hacerlo es seleccionar el método y utilizar el menu contextual: "Implementors of..."*
4. Encuentre todos los métodos donde a algún objeto se le envía el mensaje `#==` `y #=`.
   a. *Una manera de hacerlo es seleccionar cada uno de los métodos y utilizar el menú contextual: "Senders of"*
   b. ¿Puede asegurar para cada caso qué método **#==** y **#=** será ejecutado?
   c. ¿Por qué?
5. Encuentre todas las implementaciones del método `#size`.
   a. *Una manera de hacerlo es seleccionar el método y utilizar el menu contextual: "Implementors of.."*
6. ¿Qué uso se da al protocolo `private`? ¿Qué implica que un método esté en el protocolo `private`?
7. Busque la implementación por default de #initialize. Discuta con un ayudante el porqué de dicha implementación.
8. Si inspecciona el protocolo de Boolean, en browser, al lado de los métodos, verá flechas verde. ¿Qué indican las flechas hacia arriba? ¿Y hacia abajo?



Ejercicio 2: Clases Vs. Instancias

En un procesador de texto la clase `Document` representa un documento de texto que debe tener un título (`title`) y un texto principal (`body`). `Document` debe responder a los siguientes mensajes:

```
Document>> size
  "retornar el tamaño del documento que es la suma del tamaño del título, el
tamaño del body y el overhead del documento que siempre es el 10% del body"

Document>> addLine: aString
 "agrega (concatena) al body la linea aString"

Document class>> titled: aString
 "Retorna una nueva instancia de Document, cuyo título  es aString"
```

Tareas:
1.- Complete la implementación en Smalltalk.
2.- Instancie la clase `Document` en un Playground y verifique el funcionamiento.
3.- Considerando el fragmento de código (los números a la izquierda indican número de línea de código):

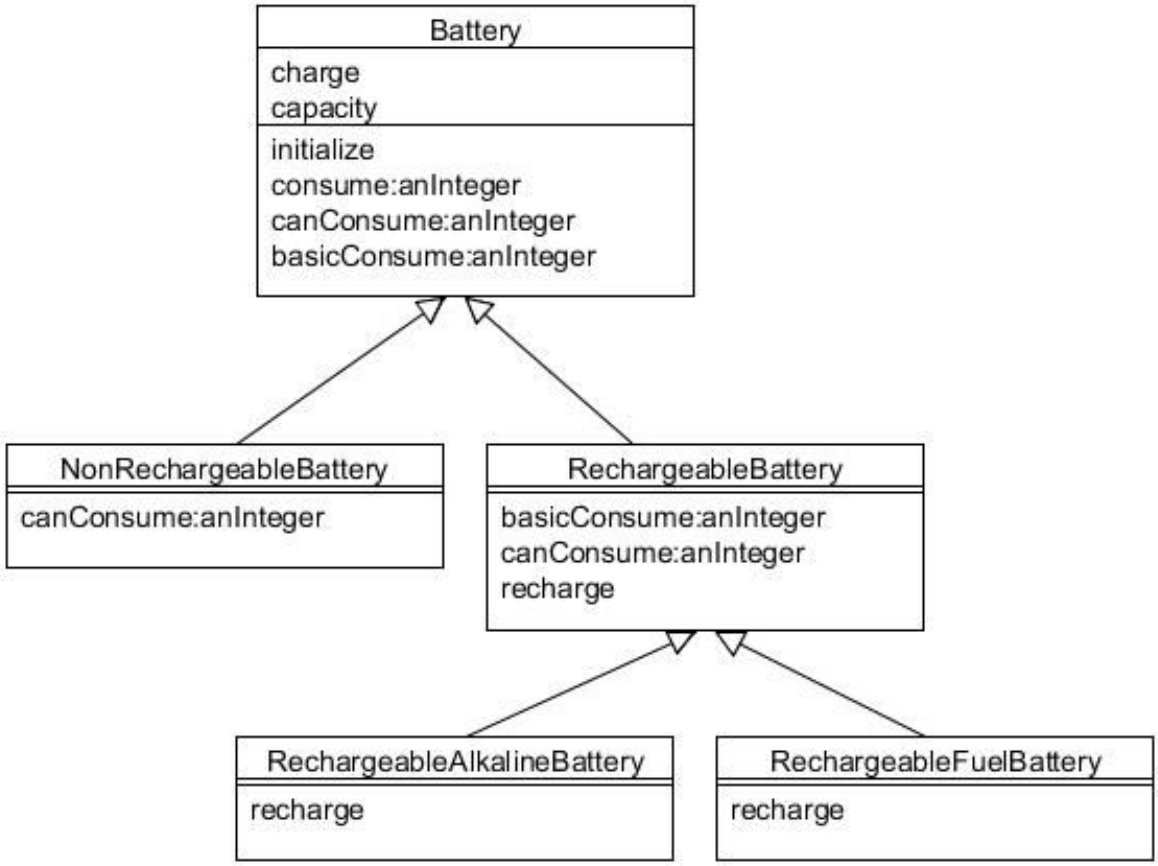
```
1.   | doc |
2.   doc:= Document titled: 'Objetos'.
3.   doc size.
4.   doc class.
5.   doc class allInstances.
6.   Document allInstances.
7.   doc addLine: 'bla'.
8.   doc class == Document
```

Conteste:

a. ¿*doc* es una clase o una instancia?
b. Al evaluar la línea 3, ¿de qué clase es el resultado de dicha evaluación?
c. El resultado de la evaluación de la línea 4, ¿es una clase o una instancia?
d. ¿Cuál es el resultado de evaluar la línea 8? ¿Qué conclusión puede sacar al respecto?

Ejercicio 3: Method Lookup

Se tienen definidas en Smalltalk las siguientes clases con el siguiente comportamiento




```
Object subclass: #Battery
instanceVariables: 'capacity charge'

Battery>> initialize
   capacity := 100.
   charge := capacity.

Battery>> consume: anInteger
   (self canConsume: anInteger)
     ifTrue:[self basicConsume:anInteger]
     ifFalse:[Transcript show: 'Batería descargada']

Battery>> canConsume: anInteger
   ^ self subclassResponsibility

Battery>> basicConsume: anInteger
   charge := charge - anInteger

------------------------
Battery subclass: #NonRechargeableBattery
```

```
NonRechargeableBattery>> canConsume: anInteger
 ^ charge >= anInteger

----------------------
Battery subclass: #RechargeableBattery
RechargeableBattery>> basicConsume: anInteger
     charge := charge - anInteger.
     charge <= (capacity / 10)
       ifTrue: [self recharge]

RechargeableBattery>> canConsume: anInteger
  ^ anInteger <= capacity

RechargeableBattery>> recharge
 ^ self subclassResponsibility

----------------------
RechargeableBattery subclass: #RechargeableAlkalineBattery

RechargeableAlkalineBattery>> recharge
   charge := capacity.
   Transcript show: 'Recharging alkaline battery...'
--------------------
RechargeableBattery subclass: #RechargeableFuelBattery
RechargeableFuelBattery >> recharge
   charge := capacity.
   Transcript show: 'Recharging fuel battery...'
```

Conteste:

¿Qué mensajes se imprimen en el Transcript[1] y con cuánta carga queda cada una de las baterías, si dentro de una ventana de Playground se evalúan las expresiones de los siguientes ejemplos?

Ejemplo 1:

```
nrb := NonRechargeableBattery new.
nrb consume: 70.
nrb consume: 35.
```

Ejemplo 2:

```
rab := RechargeableAlkalineBattery new.
rab consume: 70.
rab consume: 35.
```

[1] El Transcript puede visualizarse en Pharo accediendo a World / Tools / Transcript del menú contextual.

Ejercicio 4: Perfil y Karma

Sea una red social en la que se requiere representar el perfil de una persona de la siguiente manera: de cada persona interesa saber el nombre, la cantidad de posts que realizó en el último mes, la cantidad total de likes que obtuvo en el último mes (el total, no por post) y el "karma" que tiene esa persona.
El karma es un número que representa la relevancia del usuario en la red social y se incrementa con el correr de los meses, tal como se explica a continuación.

Mensualmente se computa nuevos puntos de karma a partir de la cantidad de posts y la cantidad de likes. La forma de calcular el nuevo valor de karma es la siguiente: se realiza la división de la cantidad de posts en el último mes por la cantidad de likes obtenidos también en el último mes.

Si el valor obtenido está entre 0 y 30 no obtiene nuevos puntos de karma.
Si el valor obtenido está entre 31 y 50 obtiene 2 puntos de karma (que se suman al karma que el usuario ya tiene).
Por 51 o más obtiene 3 puntos de karma.
Una vez realizado el cálculo de karma, los contadores de posts y likes mensuales se setean en 0 nuevamente.

Tarea:
1- Identifique objetos y responsabilidades.
2- Identifique las clases: nombres, variables, métodos que implementan y relaciones entre ellas .
3- Implemente en Smalltalk.
4- Instancie en un Playground y verifique que el cálculo de karma funciona correctamente

Considere que cuando se crea un usuario debe tener sus contadores y nombre correctamente inicializados. Por lo tanto, implemente el constructor correspondiente.

Ejercicio 5: Perfil y Karma Extendido

La red social del ejercicio anterior se propuso tener 2 tipos de usuarios: Silver y Gold. Los números de posts y likes se manejan de la misma que en el enunciado anterior. Lo que se modifica es el cálculo del karma.

En el caso del usuario Silver el karma se calcula como: posts * likes / Float pi

En el caso del usuario Gold el karma se calcula como: posts * likes / Float halfPi

Considerando la implementación del ejercicio 4 determine:

1. ¿Qué modificaciones o extensiones debe hacer a la resolución del ejercicio 4?

2. ¿Qué comportamiento tienen en común los usuarios Gold y Silver?
3. La superclase de ambos usuarios, ¿puede instanciarse?
4. En el caso en que no pueda instanciarse, ¿cuál es la manera en Smalltalk de definir una clase de ese tipo?

Ejercicio 6: Carrito de compras

En un sitio de compras por internet un cliente puede comprar muchos productos que pone en su carrito. Al finalizar la compra se debe calcular el total a abonar.
De cada producto se conoce su nombre y su precio. El carrito debe permitir ir sumando de a uno los productos, hasta que el cliente decide hacer el checkout. No es necesario almacenar los productos, sino sumar el valor total de la compra y la cantidad de productos adquiridos. Si la cantidad de productos supera los 10, debe aplicar un descuento del 10%. Si el monto total a abonar supera los $ 10.000 se le aplica un 5% de descuento.

Tareas
1. Identifique los objetos y responsabilidades.
2. ¿Qué comportamiento debe proveer el carrito?
3. Implemente en Smalltalk (recuerde implementar los constructores)
4. Piense 3 casos de prueba que le permitan verificar que su implementación funciona de la manera esperada.
5. Implemente esos casos de prueba y ejecútelos en un Playground.

Ejercicio 7:Robot

Analice la jerarquía de clases correspondientes al robot que usó en las 2 primeras prácticas (`WalkingBrushRobot` [2],[3]).

1. Analice la implementación de #position en toda la jerarquía. ¿Es necesaria la redefinición que se hizo en `WalkingBrushRobot` ? Justifique.
2. ¿Cuáles son los constructores que se definieron? ¿Para qué sirve cada uno?
3. Analice las clases `Battery` y `EndlessBattery`. ¿Cuál es el protocolo en común? ¿Qué puede concluir al respecto? ¿Es necesario que objetos polimórficos pertenezcan a la misma jerarquía?
4. Liste las clases de las cuales hereda `WalkingBrushRobot` y para cada caso documente las variables de instancia
5. Analize las referencias a la variable de instancia `#body` de la clase `BGSRobot`,
    a. cómo se inicializa esta variable?
    b. que retorna el mensaje #`bodyClass` de la clase `BGSRobot`?
6. Modifique la clase `WalkingBrushRobot` para que se inicialice con otros tipos de "`body`"

[2] Puede buscar una clase usando la combinación de teclas Cmd o Ctrl + F,C
[3] Para ver la jerarquía haga click en el botón Hierarchy.

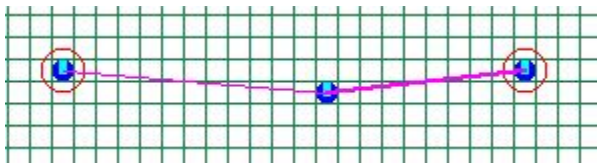
Ejercicio 8: Robot

En este ejercicio utilizará una nueva configuración de BotArena dos nuevos tipos de robot provistos por la cátedra. Una de las características que utilizará en este ejercicio es la capacidad de comunicarse con otros robots que se encuentran en la “arena” a cierta distancia. Esto ocurre dado que los robots tienen un módulo “wireless” que se activa cuando dos robots están a la distancia apropiada.

Para utilizar esta configuracion recuerde actualizar BotArena. El nuevo BotArena se abre evaluando la siguiente expresión:

BGSArenaWindow wirelessViewOn: (OnTheFlyConfigurableSimulation wirelessBrush).

En la arena siempre hay un robot que se llama “c3po”. El robot “c3po” es instancia de una clase diferente al que se puede agregar en la arena usando el botón “Add Robot” o haciendo click en una celda vacía de la arena. Los robots que se agregan (por botón o por click) se pueden identificar por el círculo rojo que rodea su cuerpo.



Cuando los robots tienen la capacidad de comunicarse, se dibuja una línea. En la captura de pantalla de BotArena se puede ver al objeto “c3po” (no tiene circulo rojo) conectado con otros dos robots. Los robots no están conectados entre si dado que no hay una linea entre ellos. El objeto “c3po” entiende mensajes como: #checkNeighbours y #helpNeighbours los cuales se comunican con otros robots enviandoles mensajes.

a) Encuentre las clases de robots involucradas en esta configuración de BotArena
b) Liste los mensajes que el objeto “c3po” no entiende y que si eran entendidos por los robots de la práctica 1.
c) Encuentre los objetos que colaboran con “c3po” y los mensajes que deben implementar
d) Implemente los mensajes necesarios en las clases adecuadas para que “c3po” pueda recibir mensajes de la categoría “protocol”
e) Verifique que el objeto “c3po” puede recibir los mensajes de la categoría “protocol” de la manera esperada y sin generar errores.
f) Compruebe que su solución funciona para configuraciones del robot con diferentes tipos de batería.

Ejercicio 9

Instancias de la clase WalkingBrushWirelessRobot entienden el mensaje #peerBots. Este mensaje retorna otros bots que están dentro del alcance del módulo wireless.

Implemente los siguientes mensajes:

a) #availableCharge: retorna la sumatoria de las cargas de las baterías del receptor y de los bots que están a su alcance
b) #averageCharge: retorna el promedio de las cargas de las baterías del receptor y de los bots que están a su alcance
c) #balanceCharge: balancea la carga de las baterias del receptor y los bots que estan a su alcance para que todos tengan la misma carga

Ejercicio 10

Instancias de la clase WalkingBrushWirelessRobot deberán entender mensajes que le permitan interactuar con bots cercanos y otros conectados indirectamente.

Implemente los siguientes mensajes:

a) #relatedBots: retorna una colección de bots 1) conectados al receptor (peer) y 2) bots conectados a cada peer.
b) #availableDeepCharge: retorna la sumatoria de las cargas de las baterías del receptor, de los bots “relatedBots”.
c) #averageDeepCharge: retorna el promedio de las cargas de las baterías del receptor y de los relatedBots
d) #balanceDeepCharge: balancea la carga de la batería del receptor y las baterias de los relatedBots

Ejercicio 11

El sistema de monitoreo de un motor recibe y guarda mediciones de temperatura. Todas las mediciones son objetos que tienen un timestamp (un número entero) que es un valor de un único y creciente. Las mediciones de temperatura pueden ser valores absolutos de temperatura o pueden ser valores diferenciales en referencia a un valor absoluto anterior. Por esto es que se puede asumir que la primera medición de temperatura es siempre una absoluta.

El orden de las mediciones se considera por el timestamp/clock, es decir, la última medición tendrá el valor de clock más alto. Todas las mediciones retornan una medición absoluta que sirve de referencia, las mediciones diferenciales retornan la última medición absoluta mientras que una medición absoluta retorna a ella misma.

La siguiente tabla muestra la relación entre la temperatura real del motor y las mediciones que maneja el sistema de monitoreo

REAL	80	80.3	80.6	90	89.9	90.1	89.8	90
MEDICION	80	+0.3	+0.6	90	-0.1	+0.1	-0.2	+0.0

El sistema de monitoreo debe implementar los siguientes métodos:

#addAbsoluteTemperature: aValue clock: aNumber “agrega una medicion absoluta”	#currentTemperature “retorna la temperatura actual”
#addDiferentialTemperature: aValue clock: aNumber “agrega una medicion diferencial”	#averageTemperature “retorna el promedio de temperaturas”
#reset “borra las mediciones y toma la temperatura actual como la primera medicion absoluta”	#allTemperatureValues “retorna una colección con los valores de temperatura”

1. Describa las clases involucradas y como se relacionan
2. Implemente en Smalltalk las clases y los métodos de su solución
3. Presente código de un “playground” con las últimas 5 mediciones del ejemplo