Programación con Objetos 1 – Práctica 3

Ejercicio 1: Más sobre colecciones

Analice y compare, evaluando en el Workspace qué devuelven y qué hacen los siguientes mensajes para cada una de las siguientes clases: `OrderedCollection`, `SortedCollection`, `Array`, `Dictionary`, `Bag` y `Set`.

`addAll:`	`at:`	`removeAll`
`asSet`	`anySatisfy:`	`detect:`
`includesAny:`	`collect:`	`reject:`
`detectMax:`	`,` (coma)	`detect:ifFound:ifNone:`

Recorra la jerarquía de clases en Smalltalk y compruebe el por qué de los diferentes resultados.

Ejercicio 2: Cuentas bancarias

Implemente en Smalltalk cuentas bancarias como las vistas en la teoría. Se necesita tener cajas de ahorro y cuentas corrientes. De ambas puede extraerse, depositarse dinero y pedir el saldo. La única diferencia entre ellas es que las cajas de ahorro no permiten extraer más de 500 pesos de una vez ni en descubierto (es decir, si el saldo queda bajo cero como resultado de la extracción), mientras que las cuentas corrientes no tienen un límite predefinido de monto para cada extracción y tienen un límite de descubierto permitido. Puede que distintas cuentas corrientes tengan distintos límites de descubierto, y ese dato se indica al momento de crear la cuenta.

Se pide realizar dos implementaciones:

a) Llevando cuenta únicamente del saldo actual.

b) Llevando cuenta de cada uno de los movimientos (depósitos y extracciones) que se fueron realizando.

Ejercicio 3: Carritos de supermercado

Un nuevo y moderno supermercado ha decidido equipar a sus carritos con un sistema de última tecnología que les permite a los compradores mantener un mejor control sobre sus compras. El carrito permite realizar a los usuarios distintas consultas acerca de los productos que contiene, entre ellas:

a) Saber cuántos artículos cuyo costo es mayor a 50 pesos hay en el carrito.

b) Obtener todos los artículos comprados, sin repeticiones.

c) Obtener el primer producto de la marca “Bagley”. Si no hay producto de esa marca, devolver `nil`.

d) Obtener el monto total de dinero gastado hasta el momento.

e) Saber la cantidad de Coca Colas que se compraron.

f) Dejar en el carrito sólo aquellos productos que no son de marca “Arcor”.

g) Obtener el producto más caro.

h) Preguntar si ya se compraron todos los productos de una lista de compras.

De los productos nos interesa saber entonces, el precio, la marca, y el tipo. Modele el `CarroDeSuper` que nos permita realizar todas las consultas mencionadas. No olvide que a un carro de compras se le deber poder agregar y quitar productos.

Ejercicio 4: Trabajando con fechas

1. Las instancias de `Date` representan fechas en Smalltalk. Por ejemplo, la expresión `Date today` retorna un objeto que representa la fecha de hoy. Se pide investigar, y probar:
   - cómo crear una fecha indicando día, mes y año.
   - cómo saber si una fecha está entre otras dos.
2. Implementar el `IntervaloDeTiempo` que posee una fecha de inicio y una fecha de fin, y se le puede preguntar:
   - la duración (expresada en cantidad de días).
   - si incluye a una fecha dada (es decir, cae dentro del intervalo, incluyendo los extremos).
   - si se intersecta en algún punto con otro intervalo de tiempo.

   No olvidar definir mensajes de clase para crear instancias correctamente inicializadas.
3. Agregar al `IntervaloDeTiempo` los siguientes mensajes:
   - `= otroIntervaloDeTiempo`, que define el criterio de igualdad entre intervalos, que es fechas de inicio y de fin iguales respectivamente.
   - `desplazado: unaCantidadDeDias}`, que retorna un nuevo `IntervaloDeTiempo` con sus fechas de inicio y fin desplazadas `unaCantidadDeDias` hacia el futuro.

Ejercicio 5: Ciber

En un ciber nos piden modelar un sistema que se ajuste a sus necesidades. En el mismo hay muchas computadoras, de cada una nos van a interesar: el espacio libre en disco (expresado en Gb), la marca del mother, y los nombres de los programas cargados. Este ciber puede recibir pedidos, en cada pedido se indica qué programas se van a usar y cuánto espacio en disco requieren.

Implementar el modelo del ciber de forma tal que se pueda:

a) Agregarle una compu a un ciber.

b) Preguntar cuántas computadoras hay en un ciber con más de un determinado espacio libre en disco (por ejemplo, ¿cuántas compus hay en el ciber con más de 10 Gb libres en disco?)

c) Pedir una compu que tenga instalado un determinado programa, `nil` si no hay ninguna.

d) Obtener todas las marcas de los mother de un ciber.

e) Obtener el espacio libre en disco total de las compus de un ciber.

f) Obtener todas las compus copadas de un ciber. Una compu es copada si tiene al menos 5 programas instalados y además el mother es Asus.

g) Obtener las compus de un ciber que pueden satisfacer un pedido, o sea, que estén instalados todos los programas que el pedido necesita, y que tenga espacio en disco suficiente para lo que requiere el pedido.

h) Manejar la lista de pedidos de un ciber. Hay que registrar los pedidos que llegan que quedan como pedidos pendientes. Los pedidos se atienden por orden de llegada. El ciber tiene que entender un mensaje `atenderPrimerPedido` que toma el primer pedido pendiente, obtiene una compu que lo puede atender, le asigna la compu al pedido (el pedido se tiene que acordar qué compu le fue asignada), y pasa el pedido de la lista de pendientes a la lista de despachados. Si no hay ninguna compu que satisfaga al pedido, éste pasa de la lista de pendientes a la lista de rechazados.

i) Conocer el conjunto de programas que no están instalados en ninguna compu y que sí están en al menos un pedido rechazado, una vez que ya hay pedidos rechazados.

Ejercicio 6: Jugadores de fútbol

Los jugadores se fútbol se puede clasificar en:

- los que antes de correr protestan y corren sólo para quitar la pelota (a los que llamaremos defensores),
- los que sólo corren para patear al arco (a los que llamaremos delanteros)
- los que pueden tomar la pelota con sus manos (arqueros).

```
Object subclass: Jugador
>>jugar
    self correrLaPelota.

>>jugarCon: unJugador
    unJugador jugar.
    self jugar.

Jugador subclass: Defensor
>>jugar
   self protestar.
   self jugar.
   self quitarLaPelota.

Jugador subclass: Delantero
>>jugar
   super jugar.
   self patearAlArco.

Jugador subclass: Arquero
>>jugar
    self atajar.
```

Si asumimos que todos los métodos que no aparecen en la tabla están implementados en la misma clase donde son invocados y contamos con las siguientes instancias, responder las siguientes preguntas respecto de la ejecución de las expresiones:

1. ```
   gigliotti := Delantero new.
   insua := Defensor new.
   orion := Arquero new.

   gigliotti jugarCon: insua.
   orion jugar.
   ```
   ```

2.
```
teo := Delantero new.
mora := Delantero new.
funesMori := Defensor new.
barovero := Arquero new.

mora jugarCon: teo.
barovero jugarCon: funesMori.
```

- Indique si es posible, cuántas veces se manda el mensaje `jugar`. ¿A qué objetos?
- Indique si es posible, cuántas veces se manda el mensaje `correrLaPelota`. ¿A qué objetos?

Ejercicio 7: Extendiendo clases existentes

Se pide implementar y probar los siguientes mensajes:

1. `Boolean>>nor:` que recibe un bloque como colaborador externo y retorna el resultado del operador lógico NOR, que es la negación del OR.
2. `Boolean>>nand:` que recibe un bloque como colaborador externo y retorna el resultado del operador lógico NAND, que es la negación del AND.
3. `Integer>>,` (coma) que recibe otro entero como colaborador externo, y construye el número decimal correspondiente, que debe ser instancia de `Float`. Ejemplo, la expresión `42 , 23` debería retornar el objeto `42.23`.
4. `Date>>hasta:`, que recibe otra fecha como colaborador externo y retorna un `IntervaloDeTiempo` (del ejercicio anterior) con el objeto receptor y el colaborador externo como fechas de inicio y fin respectivamente.

Nota 1: se verá como hacer extensiones de clases durante la práctica.
Nota 2: la notación `Clase>>mensaje` se refiere al método cuyo nombre de mensaje es `mensaje` y está implementado en la clase `Clase`.

Ejercicio 8: Laboratorio de Biología

En una aplicación que modela el comportamiento de los animales que estudia un conocido laboratorio de biología, tenemos entre otras las siguientes clases, de cada una se indica superclase y algunos mensajes que nos interesan con sus respectivos métodos:

```
Object subclass: Animal
  >>energia
    ^25

Animal subclass: AnimalConPatas
  >>energia
    ^super energia + (self cantidadDePatas * self energiaPorPata)
  >>cantidadDePatas
    self subclassResponsibility
  >>energiaPorPata
    self subclassResponsibility

AnimalConPatas subclass: Cuadrupedo
  >>cantidadDePatas
    ^4
```

```
Cuadrupedo subclass: Perro
  >>energiaPorPata
    ^12

AnimalConPatas subclass: Hormiga
  >>cantidadDePatas
    ^6

Perro subclass: PerroFlaco
  >>energia
    ^20
  >>energiaPorPata
    ^2

Animal subclass: Tiburon
  >>cantidadDeDientes
    ^2500
```

Se pide:

1. Indicar para cada una de las clases descriptas si es abstracta o concreta, justificando.
2. Para cada una de estas expresiones:
   - `Perro new energia`
   - `PerroFlaco new energia`
   - `Hormiga new energia`
   - `Tiburon new energia`

   Indicar si la ejecución termina o no, y en caso de terminar qué devuelve. Justificar en cada caso.
3. Indicar qué método/s hay que agregar en qué clase/s para que el/los caso/s que no ande/n del punto anterior pase/n a andar.

Ejercicio 9: Set con filtro

Implemente y pruebe en Smalltalk la clase `SetConFiltro`. Un `SetConFiltro` es un `Set` que sólo agrega aquellos elementos que cumplen una condición. La condición se indica al momento de crear una instancia, y es particular a cada una. ¿Cuál es la superclase de `SetConFiltro`? ¿cómo modela la condición?