Orientación a Objetos 2 – Práctica 4

Rev 2020: Antonelli, Zambrano, Firmenich, Torres
Rev 2019: Diego Torres - Leandro Antonelli
Rev 2018: Diego Torres - Leandro Antonelli
Rev 2017: Leandro Antonelli
Rev 2016: Julián Grigera - Emiliano Perez - Mariela Zurbano

Ejercicio 1: Friday the 13th en Smalltalk

La clase Biblioteca implementa la funcionalidad de exportar el listado de sus socios en un formato JSON. Para ello define el método #exportarSocios de la siguiente forma:

```
Biblioteca>>exportarSocios
"Retorna la representación JSON de la colección de socios."
      ^ self exporter export: (self socios).
```

La Biblioteca delega la responsabilidad de exportar en una instancia de la clase VoorheesExporter que dada una colección de socios retorna un texto con la representación de la misma en formato JSON, esto lo hace mediante el mensaje de instancia **`#export:`**.

De un socio es posible conocer el nombre, el email y el número de legajo. Por ejemplo, para una biblioteca que posee una colección con los siguientes socios:

• Nombre: Arya Stark • e-mail:needle@stark.com • legajo: 5234/5	• Nombre: Tyron Lannister • e-mail:tyron@thelannisters.com • legajo: 2345/2

Haciendo “Print it” de las siguientes expresiones en un Playground:

```
| miBiblioteca arya tyron |
miBiblioteca:= Biblioteca new: VoorheesExporter new.
arya:= Socio nombre:'Arya Stark' email:'needle@stark.com' legajo: '5234/5'.
tyron:= Socio nombre:'Tyron Lannister' email:'tyron@thelannisters.com' legajo:'2345/2'.
miBiblioteca agregarSocio: arya.
miBiblioteca agregarSocio: tyron.
miBiblioteca exportarSocios.
```

El JSON que se generará es :

```
[
      {
            "nombre" : "Arya Stark",
            "email" : "needle@stark.com",
            "legajo" : "5234/5"
```

```
	},
	{
		"nombre" : "Tyron Lannister",
		"email" : "tyron@thelannisters.com",
		"legajo" : "2345/2"
	}
]
```

Note los corchetes de apertura y cierre de la colección, las llaves de apertura y cierre para cada socio y la coma separando a los socios.

Usando el paquete NeoJSON

El paquete NeoJSON para Pharo incluye a la clase NeoJSONWriter la cual permite generar la representación JSON de diferentes objetos. Mediante el mensaje de clase #toStringPretty: es posible enviarle un diccionario o una colección con diccionarios para obtener su representación en formato JSON.

Su nuevo desafío consiste en utilizar la clase NeoJSONWriter para imprimir en formato JSON a los socios de la Biblioteca en lugar de utilizar la clase VoorheesExporter. Pero con la siguiente condición: **nada de esto debe generar un cambio en el código de la clase Biblioteca.**

Tareas

1. Analice la implementación de la clase Biblioteca y VoorheesExporter que se provee con el material adicional de esta práctica (Archivo Biblioteca.st).
2. Programe Test de Unidad para la implementación propuesta.
3. Instale en el ambiente Pharo el paquete NeoJSON. Haga “Do it” de la siguiente expresiones en un Playground:

```
Metacello new
  repository: 'github://svenvc/NeoJSON/repository';
  baseline: 'NeoJSON';
  load.
```

4. Inspeccione las siguientes expresiones en un playground y analice el resultado.
   col:= OrderedCollection with: (Dictionary new at: #x put: 1; at: #y put: 2; yourself) with: (Dictionary new at: #x put: 3; at: #y put: 4; yourself).
   NeoJSONWriter toStringPretty: col.
5. Ahora debe utilizar la clase NeoJSONWriter para imprimir en formato JSON a los socios de la Biblioteca en lugar de la clase VoorheesExporter sin que esto genere un cambio en el código de la clase Biblioteca.

    a. Modele una solución a esta alternativa utilizando un diagrama de clases UML. Si utiliza patrones de diseño indique los roles en las clases utilizando estereotipos.
    b. Implemente en Smalltalk la totalidad de la solución siguiendo TDD.
6. Estudie la API de STON (busque la clase STON en el Pharo), el cual es un formato alternativo a Json para serializar objetos. Considere la posibilidad de extender su implementación para exportar en STON

Ejercicio 2: Template Method

1. Busque un ejemplo de aplicación del pattern en:
    a. Clase #Magnitude, categoría #comparing
    b. Clase #Collection, categoría #accessing
2. Busque dos ejemplos más de uso del patrón en Smalltalk.

Ejercicio 3: Topografía

Un objeto Topografía representa la distribución de agua y tierra de una región cuadrada del planeta, la cual está formada por porciones de “agua” y de “tierra”. La siguiente figura muestra:

(a) el aspecto de una topografía formada únicamente por agua.
(b) otra formada sólamente por tierra.
(c) una topografía mixta.

(a) Agua (b) Tierra (c) Topografía Mixta

Una topografía mixta está formada por partes de agua y partes de tierra (4 partes en total). Éstas a su vez podrían descomponerse en 4 más y así consecutivamente.

Como puede verse en los ejemplos, hay una relación muy estrecha entre la proporción de agua o tierra de una topografía mixta y la proporción de agua o tierra de sus componentes (compuestas o no). Por ejemplo:

La proporción de agua de una topografía sólo agua es 1. La proporción de agua de una topografía sólo tierra es 0. La proporción de agua de una topografía compuesta está dada por la suma de la proporción de agua de sus componentes dividida por 4. En el ejemplo, la proporción de agua es: (1 + 0 + 0 + 1) / 4 = 1/2. La proporción siempre es un valor entre 0 y 1.

1. Diseñe e implemente las clases necesarias para que sea posible:
    a. crear Topografías,
    b. calcular su proporción de agua y tierra,
    c. comparar igualdad entre topografías (dada por igual proporción de agua y tierra e igual distribución).
2. Instancie la topografía compuesta del ejemplo, y diseñe e implemente test cases para probar la funcionalidad implementada.

Tenga presente que sólo se descomponen topografías para conseguir combinaciones. No es correcto construir una topografía compuesta por cuatro topografías del mismo tipo. No debe implementar funcionalidad para hacer tal verificación.

Ejercicio 4: Modele el comportamiento de un FileSystem

Un file system contiene un conjunto de directorios y archivos organizados jerárquicamente mediante una relación de inclusión. De cada archivo se conoce el nombre, fecha de creación y tamaño en bytes. De un directorio se conoce el nombre, fecha de creación y contenido (el tamaño es siempre 32kb). Modele el file system y provea la siguiente funcionalidad:

```
Archivo>>llamado: unString creadoEl: unaFecha kBytes: unNumero
“Método de clase. Crea un nuevo archivo con nombre unString, de unNumero kBytes y en la fecha unaFecha.”

Directorio>>llamado: unString creadoEl: unaFecha
“Método de clase. Crea un nuevo Directorio con nombre unString y en la fecha unaFecha.”

Directorio>>tamanoTotalOcupado
"Retorna el espacio total ocupado en KB, incluyendo su contenido."

Directorio>>listadoDeContenido
 "Retorna un string con el listado del contenido del directorio imprimiendo el path completo de cada elemento (similar al comando pwd de linux) siguiendo el modelo presentado a continuación:
/Directorio A
/Directorio A/Directorio A.1
/Directorio A/Directorio A.1/Directorio A.1.1 (3 archivos)
/Directorio A/Directorio A.1/Directorio A.1.2 (2 archivos)
/Directorio A/Directorio A.2
/Directorio B

Directorio>>archivoMasGrande
 "Retorna el archivo con mayor cantidad de bytes en cualquier nivel del filesystem contenido por directorio receptor."
```

```
Directorio>>archivoMasNuevo
"retorna el archivo con fecha de creacion más reciente en cualquier nivel del
filesystem contenido por directorio receptor."
```

Tareas:

1. Diseñe y represente un modelo UML de clases de su aplicación, identifique el patrón de diseño empleado (utilice estereotipos UML para indicar los roles de cada una de las clases en ese patrón).
2. Diseñe, implemente y ejecute test cases para verificar el funcionamiento de su aplicación. En el archivo DirectorioTest.st del material adicional se provee la clase DirectorioTest que contiene el test para el método listadoDeContenido y la definición del método setUp. Utilice el código provisto como guía de su solución y extienda lo que sea necesario.
3. Implemente completamente en Smalltalk.

Ejercicio 5: Cálculo de sueldos

Sea una empresa que paga sueldos a sus empleados, los cuales están organizados en tres tipos: Temporarios, Pasantes y Planta. El sueldo se compone de 3 elementos: sueldo básico, adicionales y descuentos.

	Temporario	Pasante	Planta
básico	$4000 + cantidad de horas que trabajo * $10.	$2000	$3000
adicional	$500 si está casado $100 por cada hijo	$100 por examen que rindió	$500 si está casado $100 por cada hijo $100 por cada año de antigüedad
descuento	13% del sueldo básico 5% del sueldo adicional	13% del sueldo básico 5% del sueldo adicional	13% del sueldo básico 5% del sueldo adicional

Tareas:

1. Diseñe la jerarquía de Empleados de forma tal que cualquier empleado puede responder al mensaje #sueldo.
2. Desarrolle los test cases necesarios para probar todos los casos posibles.
3. Implemente en Pharo.

Ejercicio 6: Sunit

Lea http://sdmeta.gforge.inria.fr/Programmez/OnTheWeb/Eng-Art8-SUnit-V1.pdf y verifique como funciona SUnit de acuerdo a la sección 4 del pdf.
Luego responda:

1. ¿A qué patrón corresponde la implementación TestCase>>runCase?
2. ¿Dónde se implementan las otras partes del patrón?
3. ¿Quién invoca TestCase>>runCase?
4. Para el patrón de diseño detectado en el inciso 1, indique quiénes cumplen los roles del patrón indicados por Gamma et al.
5. Extienda la implementación para que al terminar cada test case se imprima en el transcript el resultado del test. (debe indicar el nombre del método que se testeó y si pasó el test, o fue error o failure).
6. Identifique su extensión, de acuerdo a los roles del patrón.