

Orientación a Objetos 2 – Práctica 3

Rev 2019: Diego Torres - Leandro Antonelli

Rev 2018: Diego Torres - Leandro Antonelli

Rev 2017: Leandro Antonelli

Rev 2016: Julián Grigera - Emiliano Perez - Mariela Zurbano

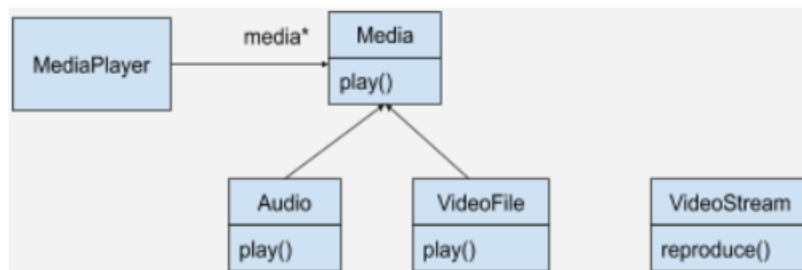
Ejercicio 1

Lea el capítulo 4 del libro Design Patterns de Gamma et al. y responda a las siguientes preguntas:

1. ¿Qué es un patrón estructural?
2. El capítulo menciona dos formas de implementar el patrón Adapter: como patrón estructural de clase y como patrón estructural de objeto. ¿Cuáles son esas dos formas? ¿Cuál de ellas no es implementable en Smalltalk y por qué?

Ejercicio 2

Usted ha implementado un exitoso Media player, el cual reproduce archivos de audio y video en formatos que usted ha diseñado. Cada Media se puede reproducir con el mensaje `play()`. Para continuar con el éxito, usted desea incorporar la posibilidad de reproducir Video Stream. Para ello, dispone de la clase `VideoStream` que pertenece a una librería de terceros y usted no puede ni debe modificarla. El desafío que se le presenta es hacer que la clase `MediaPlayer` pueda interactuar con la clase `VideoStream`. La situación se resume en la siguiente figura:



Tareas:

1. Indique que patrón resuelve el problema planteado.
2. Extienda el modelo con la incorporación del patrón.
3. Implemente en Smalltalk.

Ejercicio 3: 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:

<ul style="list-style-type: none">• Nombre: Arya Stark• e-mail:needle@stark.com• legajo: 5234/5	<ul style="list-style-type: none">• 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. Instale en el ambiente Pharo el paquete NeoJSON. Haga "Do it" de la siguiente expresiones en un Playground:

Metacello new

```
smalltalkhubUser: 'Pharo' project: 'MetaRepoForPharo60';  
configuration: 'NeoJSON';  
version: #stable;  
load.
```

3. 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.
4. 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.

Ejercicio 4: Template Method

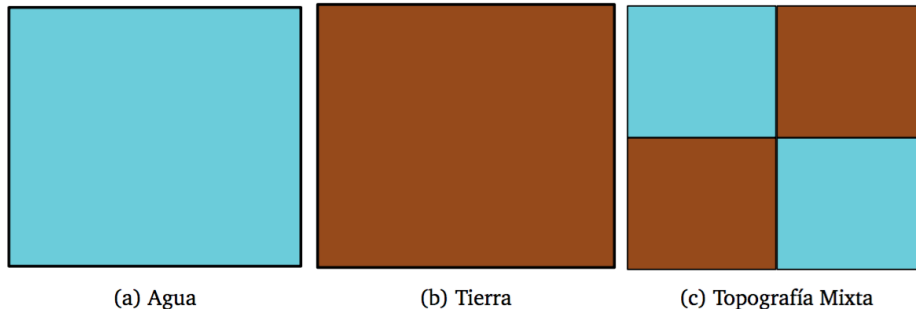
1. Lea el archivo templateMethod.pdf del material adicional y responda:
 - ¿Dónde se define el esqueleto del algoritmo?
 - ¿Se puede redefinir el esqueleto?
 - ¿Qué es lo que se puede redefinir?
 - ¿Qué es un hook method?
2. Busque un ejemplo de aplicación del pattern en:
Clase #Magnitude categoría #comparing
Clase #Collection categoría #accessing

3. Busque dos ejemplos más de uso del patrón en Smalltalk.
4. Para cada uno de los cuatro ejemplos de los dos ejercicios anteriores, realice un diagrama de clase indicando:
 - La clase abstracta.
 - El template method.
 - Al menos 2 clases concretas.
 - Los hook methods.

Ejercicio 5: 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 solamente por tierra
- (c) una 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 6: 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 siguiendo el modelo presentado a continuación:
- Directorio A
-- Directorio A.1
--- Directorio A.1.1 (3 archivos)
--- Directorio A.1.2 (2 archivos)
-- 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."
```

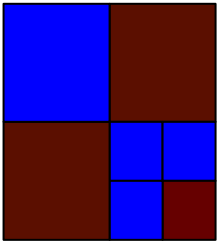
Tareas:

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 7: Visualización de topografías

A partir de las topografías vistas en el previo, se desea implementar funcionalidad para visualizarlas de dos formas distintas. Por un lado indentadas y por otro lado en formato JSON. A continuación se muestra un ejemplo de una topografía Mixta y cómo deben visualizarse.

	<pre>Mixta (Agua Tierra Tierra Mixta (Agua Agua Agua Tierra))</pre>	<pre>{ "Mixta": ["Agua", "Tierra", "Tierra", { "Mixta": ["Agua", "Agua", "Agua", "Tierra"] }] }</pre>
--	---	---

Para producir la representación indentada, se dispone del siguiente código:

```
Topografia>>printOn: aStream
  self printOn: aStream indentation: 0
```

```
Agua>>printOn: aStream indentation: spaces
  spaces timesRepeat: [ aStream nextPut: Character space ].
  aStream nextPutAll: 'Agua'.
  aStream nextPut: Character cr.
```

```
Tierra>>printOn: aStream indentation: spaces
  spaces timesRepeat: [ aStream nextPut: Character space ].
  aStream nextPutAll: 'Tierra'.
  aStream nextPut: Character cr.
```

```
Mixta>>printOn: aStream indentation: spaces
  spaces timesRepeat: [ aStream nextPut: Character space ].
  aStream nextPutAll: 'Mixta ('.
  aStream nextPut: Character cr.
  self partes do: [ :each | each printOn: aStream indentation: spaces + 4 ].
  1 to: spaces do: [ :i | aStream nextPut: Character space ].
```

```
aStream nextPutAll: ')).  
aStream nextPut: Character cr
```

Tareas:

1. Incluya la funcionalidad necesaria para mostrar el JSON de las topografías.
2. Construya un template method con la funcionalidad común de la visualización indentada y el JSON. Reescriba los métodos que visualizan en forma indentada y JSON.

Ejercicio 8: Cálculo de sueldos

Sea una empresa que paga mensualmente sueldos y adicionales a cada uno de sus empleados, quienes están organizados en tres tipos: Temporarios, Pasantes y Planta. El sueldo de un empleado Temporal está determinado por el pago de \$5 por hora que trabajó más el sueldo básico que es de \$1000; además se le paga \$100 si posee hijos y/o está casado. A los Pasantes se les paga \$40 las horas trabajadas. Por último a los empleados de Planta se les paga un sueldo básico de \$ 3000 y un plus por cada hijo que posea de \$ 150 cada hijo. Por otro lado, de cada sueldo se deben descontar en conceptos de aportes y obra social un 13 % del sueldo.

Tareas:

1. Diseñe la jerarquía de Empleados de forma tal que cualquier empleado puede responder al mensaje #sueldo. Analice y decida el usar un template method para este método. Justifique su elección.
2. Codifique.