Orientación a Objetos 2
Cuadernillo Semestral de Actividades
- Patrones de diseño -

Actualizado: 07 de marzo de 2023

El presente cuadernillo estará en elaboración durante el semestre y tendrá un compilado con todos los ejercicios que se usarán durante la asignatura. Se irán agregando ejercicios al final del cuadernillo para poder poner en práctica los contenidos que se van viendo en la materia.

Cada semana les indicaremos cuáles son los ejercicios en los que deberían enfocarse para estar al día y algunos de ellos serán discutidos en la explicación de práctica.

Recomendación importante:
Los contenidos de la materia se incorporan y fijan mejor cuando uno intenta aplicarlos - no alcanza con ver un ejercicio resuelto por alguien más. Para sacar el máximo provecho de los ejercicios, es importante asistir a las consultas de práctica habiendo intentado resolverlos (tanto como sea posible). De esa manera las consultas estarán más enfocadas y el docente podrá dar un mejor feedback.

Ejercicio 1: Red social

Se quiere programar en objetos una versión simplificada de una red social parecida a Twitter. Este servicio debe permitir a los usuarios registrados postear y leer mensajes de hasta 280 caracteres. Ud. debe modelar e implementar parte del sistema donde nos interesa que quede claro lo siguiente:

- Cada usuario conoce todos los Tweets que hizo.
- Un tweet puede ser re-tweet de otro, y este tweet debe conocer a su tweet de origen.
- Twitter debe conocer a todos los usuarios del sistema.
- Los tweets de un usuario se deben eliminar cuando el usuario es eliminado. No existen tweets no referenciados por un usuario.
- Los usuarios se identifican por su screenName.
- No se pueden agregar dos usuarios con el mismo screenName.
- Los tweets deben tener un texto de 1 carácter como mínimo y 280 caracteres como máximo.

Tareas:

Su tarea es diseñar y programar en Java lo que sea necesario para ofrecer la funcionalidad antes descrita. Se espera que entregue los siguientes productos.

1. Diagrama de clases UML.
2. Implementación en Java de la funcionalidad requerida.
3. Implementar los tests (JUnit) que considere necesarios.

Nota: para crear el proyecto Java, lea el material llamado “Trabajando en OO2 con proyectos Maven”.

Ejercicio 2: Friday the 13th en Java

Nota: Para realizar este ejercicio, utilice el recurso que se encuentra en el sitio de la cátedra. Allí encontrará un proyecto Maven que contiene el código fuente de las clases Biblioteca, Socio y VoorheesExporter.

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

```
/**
 * Retorna la representación JSON de la colección de socios.
 */
public String exportarSocios() {
    return exporter.exportar(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 **`exportar(List<Socio>)`**.

De un socio se conoce 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

Ud. puede probar la funcionalidad ejecutando el siguiente código:

```
Biblioteca biblioteca = new Biblioteca();
```

```
biblioteca.agregarSocio(new Socio("Arya Stark", "needle@stark.com", "5234-5"));
biblioteca.agregarSocio(new Socio("Tyron Lannister", "tyron@thelannisters.com",
"2345-2"));
System.out.println(biblioteca.exportarSocios());
```

Al ejecutar, el mismo imprimirá el siguiente JSON:

```
[
    {
        "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.

Tareas:

1. Analice la implementación de la clase Biblioteca, Socio y VoorheesExporter que se provee con el material adicional de esta práctica (Archivo biblioteca.zip).
2. Documente la implementación mediante un diagrama de clases UML.
3. Programe los Test de Unidad para la implementación propuesta.

Ejercicio 2.b - Usando la librería JSON.simple

Su nuevo desafío consiste en utilizar la librería JSON.simple 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.**

La librería JSON.simple es liviana y muy utilizada para leer y escribir archivos JSON.
Entre las clases que contiene se encuentran:

- JSONObject : Usada para representar los datos que se desean exportar de un objeto. Esta clase provee el método **`put(Object, Object)`** para agregar los campos al mismo. Aunque el primer argumento sea de tipo Object, usted debe proveer el nombre del atributo como un string. El segundo argumento contendrá el valor del mismo. Por ejemplo, si point es una instancia de JSONObject, se podrá ejecutar point.put("x", 50);

- JSONArray: Usada para generar listas. Provee el método **`add(Object)`** para agregar los elementos a la lista, los cuales, para este caso, deben ser JSONObject.

Ambas clases implementan el mensaje **`toJSONString()`** el cual retorna un String con la representación JSON del objeto.

- JSONParser : Usada para recuperar desde un String con formato JSON los elementos que lo componen.

Tareas:

1. Instale la librería JSON.simple agregando la siguiente dependencia al archivo pom.xml de Maven

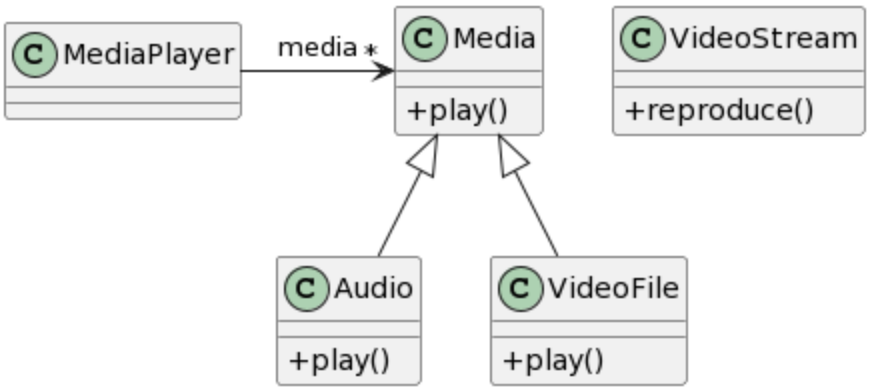
```
<dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
</dependency>
```

2. Utilice esta librería para imprimir, en formato JSON, los socios de la Biblioteca en lugar de utilizar 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 Java la solución incluyendo los tests que crea necesarios.

3. Investigue sobre la librería Jackson, la cual también permite utilizar el formato JSON para serializar objetos Java. Extienda la implementación para soportar también esta librería.

Ejercicio 3: Media Player

Usted ha implementado una clase Media player, para reproducir archivos de audio y video en formatos que usted ha diseñado. Cada Media se puede reproducir con el mensaje play(). Para continuar con el desarrollo, 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 el siguiente diagrama UML:



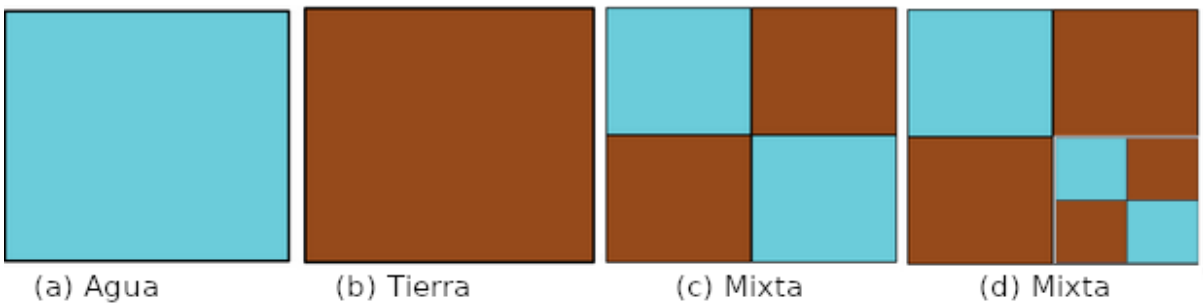
Tareas:

1. Modifique el diagrama de clases UML para considerar los cambios necesarios. Si utiliza patrones de diseño indique los roles en las clases utilizando estereotipos.
2. Implemente en Java

Ejercicio 4: Topografías

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) y (d) topografías mixtas.



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 siguiendo.

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.

Tareas:

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. Dos topografías son iguales si tienen exactamente la misma composición. Es decir, son iguales las proporciones de agua y tierra, y además, para aquellas que son mixtas, la disposición de sus partes es igual.
    Pista: notar que la definición de igualdad para topografías mixtas corresponde exactamente a la misma que implementan las listas en Java. https://docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html#equals-java.lang.Object-

2. Diseñe e implemente test cases para probar la funcionalidad implementada. Incluya en el set up de los tests, la topografía compuesta del ejemplo.

Ejercicio 4b: Más Topografías

Extienda el ejercicio anterior para soportar (además de Agua y Tierra) el terreno Pantano. Un pantano tiene una proporción de agua de 0.7 y una proporción de tierra de 0.3. No olvide hacer las modificaciones necesarias para responder adecuadamente la comparación por igualdad.

Ejercicio 5: 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:

```
public class Archivo {
    /**
       * Crea un nuevo archivo con nombre <nombre>, de <tamano> tamano
       * y en la fecha <fecha>.
    */
    public Archivo (String nombre, LocalDate fecha, int tamano)

}

public class Directorio {
       /**
       * Crea un nuevo Directorio con nombre <nombre> y en la fecha <fecha>.
       */
```

```
    public Directorio(String nombre, LocalDate fecha)

    /**
    * Retorna el espacio total ocupado, incluyendo su contenido.
    */
    public int tamanoTotalOcupado()

    /**
    * Retorna el archivo con mayor cantidad de bytes en cualquier nivel del
    * filesystem contenido por directorio receptor
    */
    public Archivo archivoMasGrande()
    /**
    * Retorna el archivo con fecha de creación más reciente en cualquier nivel
    * del filesystem contenido por directorio receptor.
    */
    public Archivo archivoMasNuevo()

}
```

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.java del material adicional se provee la clase DirectorioTest que contiene tests para los métodos arriba descritos 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 Java.

Ejercicio 6: 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	$ 20.000 + cantidad de horas que trabajo * $ 300.	$20.000	$ 50.000

adicional	$5.000 si está casado $2.000 por cada hijo	$2.000 por examen que rindió	$5.000 si está casado $2.000 por cada hijo $2.000 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 Java.