FACULTAD DE INFORMATICA | UNIVERSIDAD NACIONAL DE LA PLATA

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

Actualizado: 14 de marzo de 2025

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: Friday the 13th en Java

Nota: Para realizar este ejercicio, utilice el material adicional que se encuentra en el siguiente link. 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 1.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 2: 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.

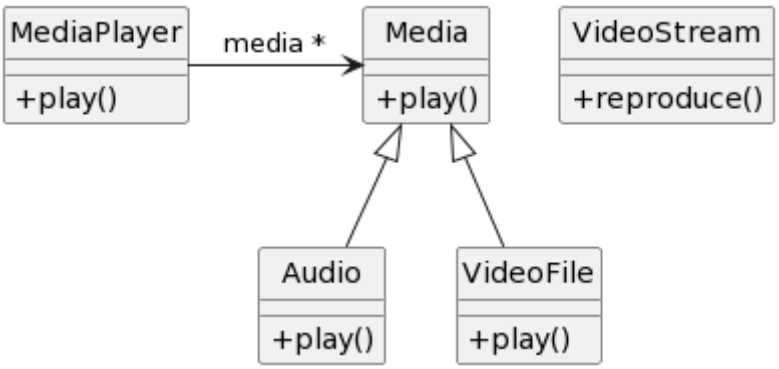
	Temporario	Pasante	Planta
básico	$ 20.000 + cantidad de horas que trabajó * $ 300.	$20.000	$ 50.000
adicional	$5.000 si está casado $2.000 por cada hijo	$2.000 por examen que rindió	$5.000 si está casado $2.000 por cada hijo $2.000 por cada año de antigüedad
descuento	13% del sueldo básico 5% del sueldo 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.

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: ToDoItem

Se desea definir un sistema de seguimiento de tareas similar a Jira[1].
En este sistema hay tareas en las cuales se puede definir el nombre y una serie de comentarios. Las tareas atraviesan diferentes etapas a lo largo de su ciclo de vida y ellas son: *pending*, *in-progress*, *paused* y *finished*. Cada tarea debe estar modelada mediante la clase ToDoItem con el siguiente protocolo:

```
public class ToDoItem {
	/**
	* Instancia un ToDoItem nuevo en estado pending con <name> como nombre.
	*/
	 public ToDoItem(String name)

	/**
	* Pasa el ToDoItem a in-progress, siempre y cuando su estado actual sea
	* pending. Si se encuentra en otro estado, no hace nada.
	*/
	 public void start()

	/**
	* Pasa el ToDoItem a paused si su estado es in-progress, o a in-progress si
su
	* estado es paused. Caso contrario (pending o finished) genera un error
	* informando la causa específica del mismo.
	*/
	 public void togglePause()
```

[1] https://es.atlassian.com/software/jira

```
        /**
        * Pasa el ToDoItem a finished, siempre y cuando su estado actual sea
        * in-progress o paused. Si se encuentra en otro estado, no hace nada.
        */
         public void finish()

        /**
        * Retorna el tiempo que transcurrió desde que se inició el ToDoItem (start)
        * hasta que se finalizó. En caso de que no esté finalizado, el tiempo que
        * haya transcurrido hasta el momento actual. Si el ToDoItem no se inició,
        * genera un error informando la causa específica del mismo.
        */
         public Duration workedTime()


        /**
        * Agrega un comentario al ToDoItem siempre y cuando no haya finalizado.
Caso
        * contrario no hace nada."
        */
         public void addComment(String comment)
}
```

Nota: para generar o levantar un error debe utilizar la expresión

```
throw new RuntimeException("Este es mi mensaje de error");
```

El mensaje de error específico que se espera en este ejercicio debe ser descriptivo del caso. Por ejemplo, para el método togglePause() , el mensaje de error debe indicar que el ToDoItem no se encuentra en in-progress o paused:

```
throw new RuntimeException("El objeto ToDoItem no se encuentra en pause o
in-progress");
```

Tareas:

1. Modele una solución orientada a objetos para el problema planteado utilizando un diagrama de clases UML. Si utilizó algún patrón de diseño indique cuáles son los participantes en su modelo de acuerdo a Gamma et al.
2. Implemente su solución en Java. Para comprobar cómo funciona recomendamos usar test cases.