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

Actualizado: 7 de abril 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

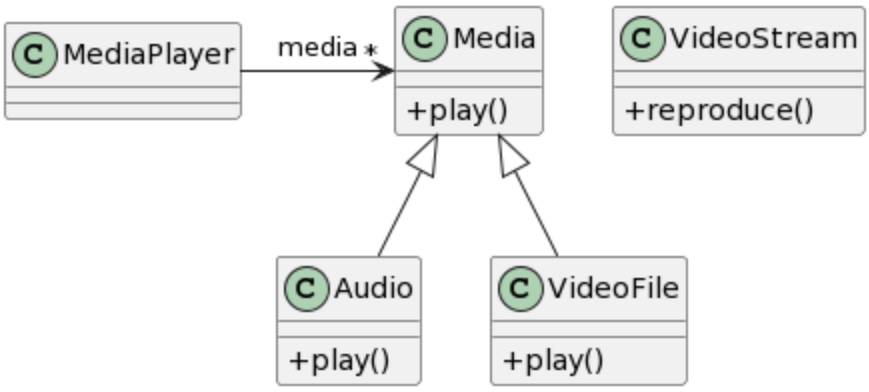
```
<dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
</dependency>
```

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:



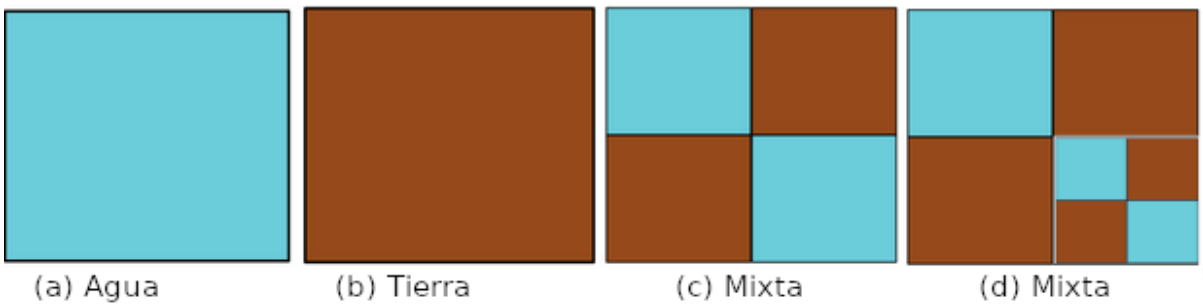
Tareas:

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.

Ejercicio 7: 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()

	/**
	* 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,
```

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

```
        * 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.

Ejercicio 8: Excursiones

Sea una aplicación que ofrece excursiones como por ejemplo “dos días en kayak bajando el Paraná”. Una excursión posee nombre, fecha de inicio, fecha de fin, punto de encuentro, costo, cupo mínimo y cupo máximo.

La aplicación ofrece las excursiones pero éstas sólo se realizan si alcanzan el cupo mínimo de inscriptos. Un usuario se inscribe a una excursión y si aún no se alcanzó el cupo mínimo, la inscripción se considera provisoria. Luego, cuando se alcanza el cupo mínimo, la inscripción se considera definitiva y podrá llevarse a cabo. Finalmente, cuando se alcanza el cupo máximo, la excursión solo registrará nuevos inscriptos en su lista de espera.

De los usuarios inscriptos, la aplicación registra su nombre, apellido e email.

Por otro lado, en todo momento la excursión ofrece información de la misma, la cual consiste en una serie de datos que varían en función de la situación.

- Si la excursión no alcanza el cupo mínimo, la información es la siguiente: nombre, costo, fechas, punto de encuentro, cantidad de usuarios faltantes para alcanzar el cupo mínimo.
- Si la excursión alcanzó el cupo mínimo pero aún no el máximo, la información es la siguiente: nombre, costo, fechas, punto de encuentro, los mails de los usuarios inscriptos y cantidad de usuarios faltantes para alcanzar el cupo máximo.
- Si la excursión alcanzó el cupo máximo, la información solamente incluye nombre, costo, fechas y punto de encuentro.

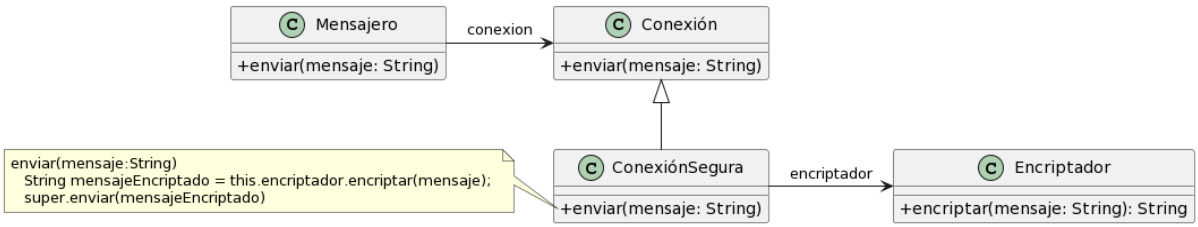
En una primera versión, al no contar con una interfaz de usuario y a los efectos de *debugging,* este comportamiento puede implementarlo en un método que retorne un String con la información solicitada.

Tareas:

1.- Realice un diseño UML. Si utiliza algún patrón indique cuál(es) y justifique su uso.
2.- Implemente lo necesario para instanciar una excursión y para instanciar un usuario.
3.- Implemente los siguientes mensajes de la clase Excursion:
(i) public void inscribir (Usuario unUsuario)
(ii) public String obtenerInformacion().
4.- Escriba un test para inscribir a un usuario en la excursión “Dos días en kayak bajando el Paraná”, con cupo mínimo de 1 persona y cupo máximo 2, con dos personas ya inscriptas. Implemente todos los mensajes que considere necesarios.

Ejercicio 9: Encriptador

En un sistema de mensajes instantáneos (como Hangouts) se envían mensajes de una máquina a otra a través de una red. Para asegurar que la información que pasa por la red no es espiada, el sistema utiliza una conexión segura. Este tipo de conexión encripta la información antes de enviarla y la desencripta al recibirla. La siguiente figura ilustra un posible diseño para este enunciado.



El encriptador utiliza el algoritmo RSA. Sin embargo, se desea agregar otros algoritmos (diferentes algoritmos ofrecen distintos niveles de seguridad, overhead en la transmisión, etc.).

Tareas:

1. Modifique el diseño para que el objeto Encriptador pueda encriptar mensajes usando los algoritmos Blowfish y RC4, además del ya soportado RSA.
2. Documente mediante un diagrama de clases UML indicando los roles de cada clase.

Ejercicio 10: Administrador de proyectos

Consideremos una empresa que brinda servicios y los gestiona a través de proyectos. Los proyectos tienen una fecha de inicio y de fin, un objetivo, un número de integrantes (quienes cobran un monto fijo por día) y un margen de ganancia. Durante el armado del proyecto, el mismo debe pasar por un proceso de aprobación que involucra las etapas: En construcción -> En evaluación -> Confirmada. Se desea implementar la siguiente funcionalidad:

Funcionalidad	**Etapa actual del proyecto**	**Resultado esperado**
Crear proyecto	-	Se crea el proyecto en etapa “En construcción” con nombre, fecha de inicio y fin, objetivo, margen de ganancia de 7%, un número de integrantes y el monto de pago por integrante por día.
Aprobar etapa	En construcción	El proyecto pasa a etapa “En evaluación“ siempre y cuando su precio no sea 0 (cero). De lo contrario genera un error.
	En evaluación	El proyecto pasa a etapa “Confirmada”
	En otra situación	No produce efecto alguno en el proyecto.
Costo del proyecto	En cualquier etapa	Retorna la suma de los costos de las personas involucradas. Considerar que las personas trabajan todos los días que dura el proyecto.
Precio del proyecto	En cualquier etapa	Retorna el valor obtenido luego de aplicar el margen de ganancia al costo del proyecto.
Modificar margen de ganancia	En etapas “En construcción” y “En evaluación”	Actualiza el margen de ganancia si se encuentra en los siguientes valores: Para “En construcción” -> valores entre 8% y 10% Para “En evaluación” -> valores entre 11% y 15% Para valores fuera de los rangos permitidos no produce efecto alguno en el proyecto.
	Otra situación	No produce efecto alguno en el proyecto.
Cancelar proyecto	En cualquier etapa	Agrega “(Cancelado) ” al objetivo del proyecto. Deja el proyecto cancelado.

	Si ya está Cancelado.	No produce efecto alguno en el proyecto.

Tareas:

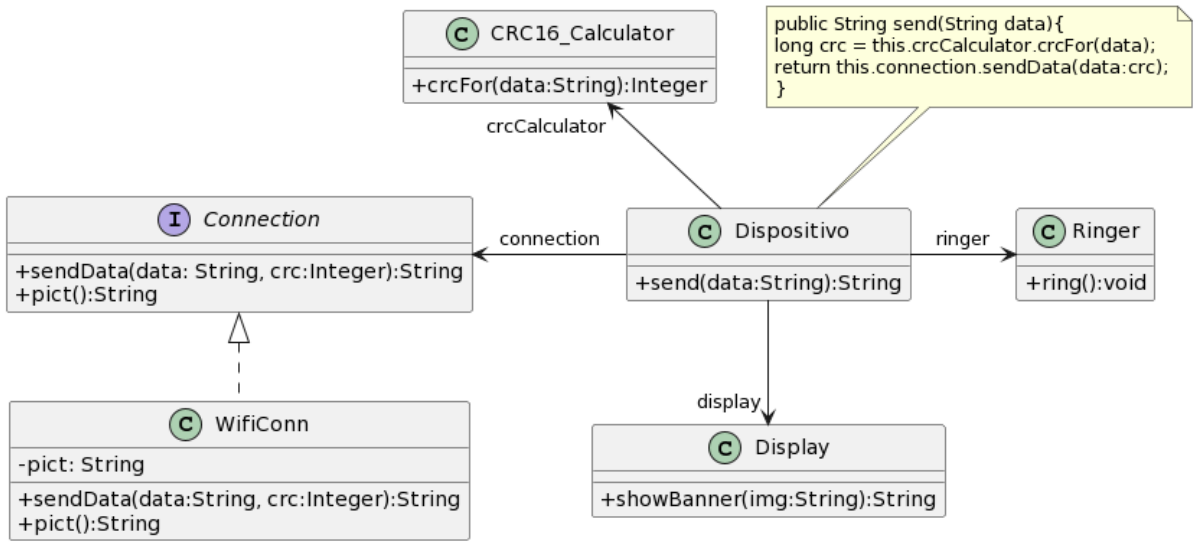
1- Modele una solución y provea el diagrama de clases UML para el problema planteado. Si utiliza algún patrón, indique cuál.
2- Implemente en Java.
3- Implemente un test para aprobar un proyecto con las siguientes características: (i) se encuentra en evaluación, (ii) se llama “Vacaciones de invierno”, (iii) tiene como objetivo “salir con amigos”, y (iv) lo integran 3 personas.

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

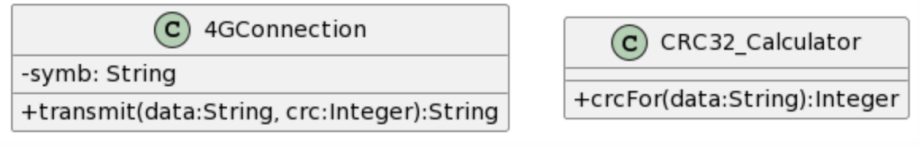
```
throw new RuntimeException("Este es mi mensaje de error");
```

Ejercicio 11 - Dispositivo móvil y conexiones

Sea el software de un dispositivo móvil que utiliza una conexión WiFi para transmitir datos. La figura muestra parte de su diseño:



Nuevas clases a utilizar:



El dispositivo utiliza, para asegurar la integridad de los datos emitidos, el mecanismo de cálculo de redundancia cíclica que le provee la clase CRC16_Calculator que recibe el mensaje `crcFor(data: String)` con los datos a enviar y devuelve un valor numérico. Luego el dispositivo envía a la conexión el mensaje `sendData` con ambos parámetros (los datos y el valor numérico calculado).

Se desea hacer dos cambios en el software. En primer lugar, se quiere que el dispositivo tenga capacidad de ser configurado para utilizar conexiones 4G. Para este cambio se debe utilizar la clase 4GConnection.
Además se desea poder configurar el dispositivo para que utilice en distintos momentos un cálculo de CRC de 16 o de 32 bits. Es decir que en algún momento el dispositivo seguirá utilizando CRC16_Calculator y en otros podrá ser configurado para utilizar la clase CRC32_Calculator. Se desea permitir que en el futuro se puedan utilizar otros algoritmos de CRC.

Cuando se cambia de conexión, el dispositivo muestra en pantalla el símbolo correspondiente (que se obtiene con el getter `pict()` para el caso de WiFiConn y `symb()` de 4GConnection) y se utiliza el objeto Ringer para emitir un `ring()`.

Tanto las clases existentes como las nuevas a utilizar pueden ser ubicadas en las jerarquías que corresponda (modificar la clase de la que extienden o la interfaz que implementan) y se les pueden agregar mensajes, pero no se pueden modificar los mensajes que ya existen porque otras partes del sistema podrían dejar de funcionar.

Dado que esto es una simulación, y no dispone de hardware ni emulador para esto, la signatura de los mensajes se ha simplificado para que se retorne un String descriptivo de los eventos que suceden en el dispositivo y permitir de esta forma simplificar la escritura de los tests.

Modele los cambios necesarios para poder agregar al protocolo de la clase Dispositivo los mensajes para

- cambiar la conexión, ya sea la 4GConnection o la WifiConn. En este método se espera que se pase a utilizar la conexión recibida, muestre en el display su símbolo y genere el sonido.
- poder configurar el calculador de CRC, que puede ser el CRC16_Calculator, el CRC32_Calculator, o pueden ser nuevos a futuro.

Tareas:

1. Realice un diagrama UML de clases para su solución al problema planteado. Indique claramente el o los patrones de diseño que utiliza en el modelo y el rol que cada clase cumple en cada uno.
2. Implemente en Java todo lo necesario para asegurar el envío de datos por cualquiera de las conexiones y el cálculo adecuado del índice de redundancia cíclica.
3. Implemente test cases para los siguientes métodos de la clase Dispositivo:

a. send
b. conectarCon
c. configurarCRC

En cuanto a CRC16_Calculator, puede utilizar la siguiente implementación:

```
public long crcFor(String datos) {
	int crc = 0xFFFF;
	for (int j = 0; j < datos.getBytes().length; j++) {
		crc = ((crc >>> 8) | (crc << 8)) & 0xffff;
		crc ^= (datos.getBytes()[j] & 0xff);
		crc ^= ((crc & 0xff) >> 4);
		crc ^= (crc << 12) & 0xffff;
		crc ^= ((crc & 0xFF) << 5) & 0xffff;
	}
	crc &= 0xffff;
	return crc;
}
```

Nota: para implementar CRC32_Calculator utilice la clase java.util.zip.CRC32 de la siguiente manera:

```
CRC32 crc = new CRC32();
String datos = "un mensaje";
crc.update(datos.getBytes());
long result = crc.getValue();
```

Ejercicio 12 - Decodificador de películas

Sea una empresa de cable *on demand* que entrega decodificadores a sus clientes para que miren las películas que ofrece. El decodificador muestra la grilla de películas y también sugiere películas.

Usted debe implementar la aplicación para que el decodificador sugiera películas. El decodificador conoce la grilla de películas (lista completa que ofrece la empresa), como así también las películas que reproduce. De cada película se conoce título, año de estreno, películas similares y puntaje. La similaridad establece una relación recíproca entre dos películas, por lo que si A es similar a B entonces también B es similar a A.

Cada decodificador puede ser configurado para que sugiera 3 películas (que no haya reproducido) por alguno de los siguientes criterios:

(i) novedad: las películas más recientes.

(ii) similaridad: las películas más nuevas son similares a alguna película que reprodujo.
(iii) puntaje: las películas de mayor puntaje, para igual puntaje considera las más recientes.

Tenga en cuenta que la configuración del criterio de sugerencia del decodificador no es fija, sino que el usuario la debe poder cambiar en cualquier momento. El sistema debe soportar agregar nuevos tipos de sugerencias aparte de las tres mencionadas.

Sea un decodificador que reprodujo Thor y Rocky, y posee la siguiente lista de películas:

Thor, 7.9, 2007 (Similar a Capitan America, Iron Man)
Capitan America, 7.8, 2016 (Similar a Thor, Iron Man)
Iron man, 7.9, 2010 (Similar a Thor, Capitan America)
Dunkirk, 7.9, 2017
Rocky, 8.1, 1976 (Similar a Rambo)
Rambo, 7.8, 1979 (Similar a Rocky)

Las películas que debería sugerir son:

(i) Dunkirk, Capitan America, Iron man
(ii) Capitán América, Iron man, Rambo
(iii) Dunkirk, Iron man, Capitan America

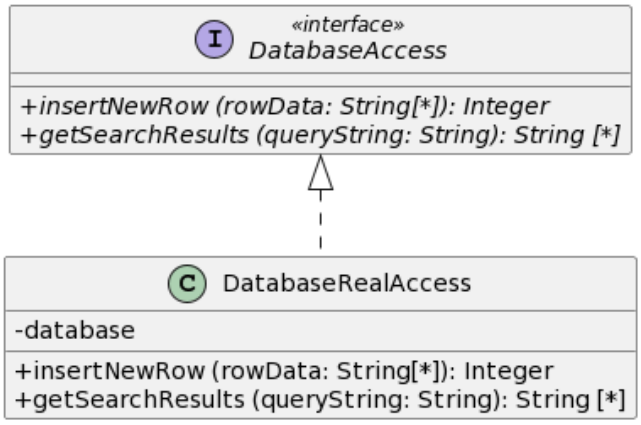
Nota: si existen más de 3 películas con el mismo criterio, retorna 3 de ellas sin importar cuales. Por ejemplo, si las 6 películas son del 2018, el criterio (i) retorna 3 cualquiera.

Tareas:

1. Realice el diseño de una correcta solución orientada a objetos con un diagrama UML de clases.
2. Si utiliza patrones de diseño indique cuáles y también indique los participantes de esos patrones en su solución según el libro de Gamma et al.
3. Escriba un test case que incluya estos pasos, con los ejemplos mencionados anteriormente:
    - configure al decodificador para que sugiera por similaridad (ii)
    - solicite al mismo decodificador las sugerencias
    - configure al mismo decodificador para que sugiera por puntaje (iii)
    - solicite al mismo decodificador las sugerencias
4. Programe su solución en Java. Debe implementarse respetando todas las buenas prácticas de diseño y programación de POO.

Ejercicio 13: Acceso a la base de datos

Queremos acceder a una base de datos que contiene información sobre cómics. Este acceso está dado por el comportamiento de la clase DatabaseRealAccess con el siguiente protocolo y modelado como muestra la siguiente figura.




```
public interface DatabaseAccess {

   /**
    * Realiza la inserción de nueva información en la base de datos y
    * retorna el id que recibe la nueva inserción
    *
    * @param rowData
    * @return
    */
   public int insertNewRow(List<String> rowData);
}

   /**
    * Retorna una colección de acuerdo al texto que posee
"queryString"
    *
    * @param queryString
    * @return
    */
   public Collection<String> getSearchResults(String queryString);

}
```

En este caso, ustedes recibirán una implementación prototípica de la clase **DatabaseRealAccess** (ver material extra) que simula el uso de una base datos de la siguiente forma (mire el código y los tests para entender cómo está implementada).

```
// Instancia una base de datos que posee dos filas
   database = new DatabaseRealAccess();

   // Retorna el siguiente arreglo: ['Spiderman' 'Marvel'].
   database.getSearchResults("select * from comics where id=1");

   // Retorna 3, que es el id que se le asigna
   database.insertNewRow(Arrays.asList("Patoruzú", "La flor"));

   // Retorna el siguiente arreglo: ['Patoruzú', 'La flor'], ya que
lo insertó antes
   database.getSearchResults("select * from comics where id=3");
```

Tareas:

En esta oportunidad, usted debe proveer una solución utilizando un patrón que le permita brindar protección al acceso a la base de datos de forma que lo puedan realizar solamente usuarios que se hayan autenticado previamente. 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. Implementación de los tests (JUnit) que considere necesarios.

Ejercicio 14: File Manager

En un **File Manager** se muestran los archivos. De los archivos se conoce:

- Nombre
- Extensión
- Tamaño
- Fecha de creación
- Fecha de modificación
- Permisos

Implemente la clase **FileOO2**, con las correspondientes variables de instancia y *accessors*.

En el File Manager el usuario debe poder elegir cómo se muestra un archivo (instancia de la clase FileOO2), es decir, cuáles de los aspectos mencionados anteriormente se muestran, y en qué

orden. Esto quiere decir que un usuario podría querer ver los archivos de muchas maneras. Algunas de ellas son:

- `nombre - extensión`
- `nombre - extensión - fecha de creación`
- `permisos - nombre - extensión - tamaño`

Para esto, el objeto o los objetos que representen a los archivos en el FileManager debe(n) entender el mensaje `prettyPrint()`.

Es decir, un objeto cliente (digamos el FileManager) le enviará al objeto que Ud. determine, el mensaje `prettyPrint()`. **De acuerdo a cómo el usuario lo haya configurado se deberá retornar un String con los aspectos seleccionados por el usuario en el orden especificado por éste**. Considere que un mismo archivo podría verse de formas diferentes desde distintos puntos del sistema, y que el usuario podría cambiar la configuración del sistema (qué y en qué orden quiere ver) en runtime.

Tareas:

1. Discuta los requerimientos y diseñe una solución. Si aplica un patrón de diseño, indique cuál es y justifique su aplicabilidad.
2. Implemente en Java.
3. Instancie un objeto para cada uno de los ejemplos citados anteriormente y verifique escribiendo tests de unidad.

Ejercicio 15 - Estación meteorológica

Sea una estación meteorológica hogareña que permite conocer información de varios aspectos del clima. Esta estación está implementada con la clase HomeWeatherStation que interactúa con varios sensores para conocer fenómenos físicos. La misma implementa los siguientes métodos:

```
//retorna la temperatura en grados Fahrenheit
public double getTemperaturaFahrenheit()

//retorna la presión atmosférica en hPa
public double getPresion()

//retorna la radiación solar
public double getRadiacionSolar()
```

```
//retorna una lista con todas las temperaturas sensadas hasta el
momento, en grados Fahrenheit
public List<Double> getTemperaturasFahrenheit()
```

Esta clase se encuentra implementada por terceros y **no se puede modificar.**

Nos piden construir una aplicación que además de lo anteriormente descripto pueda obtener:

- La temperatura en grados Celsius (ºC = (ºF-32) ÷ 1.8).
- El promedio de las temperaturas históricas en grados Fahrenheit.

Además, la aplicación debe permitir al usuario configurar qué datos mostrar y en qué orden. Esto significa que podría querer ver la información de muchas maneras, por ejemplo:

- Ejemplo 1: “Presión atmosférica: 1008”
- Ejemplo 2: “Presión atmosférica: 1008 Radiación solar: 500”
- Ejemplo 3: “Radiación solar: 500 Temperatura C: 28 Promedio de temperaturas C: 25”

Para ello, usted debe proveer en algún punto de su solución, la implementación del mensaje public String displayData() que devuelva los datos elegidos en el orden configurado (dado que la app aun no cuenta con interface de usuario).

Haga uso de la clase HomeWeatherStation sin modificarla.

Tareas:
1- Modele una solución para el problema planteado. Si utiliza algún patrón, indique cuál
2- Implemente en Java
3- Implemente un test para validar la configuración del ejemplo 2, asumiendo que en el momento de la ejecución del mismo, los sensores arrojan los valores del ejemplo.