



# Orientación a Objetos 2

## Cuadernillo Semestral de Actividades

### - Patrones de diseño -

**Actualizado: 21 de abril de 2024**

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.
- Un re-tweet no tiene texto adicional.



## 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: Piedra Papel o Tijera

Se quiere programar en objetos una versión del juego Piedra Papel o Tijera. En este juego dos jugadores eligen entre tres opciones: piedra, papel o tijera. La piedra aplasta la tijera, la tijera corta el papel, y el papel envuelve la piedra. Los jugadores eligen una opción y se determina un ganador según las reglas:

-

	Piedra	Papel	Tijera
Piedra	Empate	Papel	Piedra
Papel	Papel	Empate	Tijera
Tijera	Piedra	Tijera	Empate

## Tareas:

1. Diseñe e implemente una solución a este problema, de forma tal que dadas dos opciones, determine cuál fue la ganadora, o si hubo empate
2. Se desea extender al juego a una versión más equitativa que integre a lagarto y Spock, con las siguientes reglas:
  - a. Piedra aplasta tijera y aplasta lagarto.
  - b. Papel cubre piedra y desaprueba Spock.
  - c. Tijera corta papel y decapita lagarto.
  - d. Lagarto come papel y envenena Spock.
  - e. Spock rompe tijera y vaporiza piedra.

¿Qué cambios se necesitan agregar?

3. Agregue los cambios a la solución anterior.

## Ejercicio 3: 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:

<ul style="list-style-type: none"><li>• Nombre: Arya Stark</li><li>• e-mail:needle@stark.com</li><li>• legajo: 5234-5</li></ul>	<ul style="list-style-type: none"><li>• Nombre: Tyron Lannister</li><li>• e-mail:tyron@thelannisters.com</li><li>• legajo: 2345-2</li></ul>
---	---

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 3.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.



- 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 4: 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

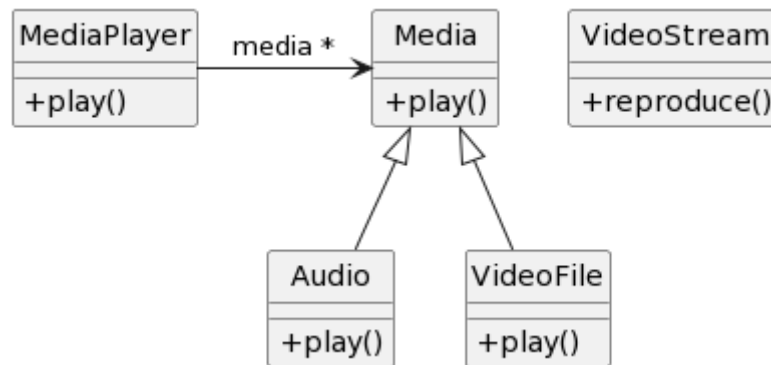
### Tareas:

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

## Ejercicio 5: 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:

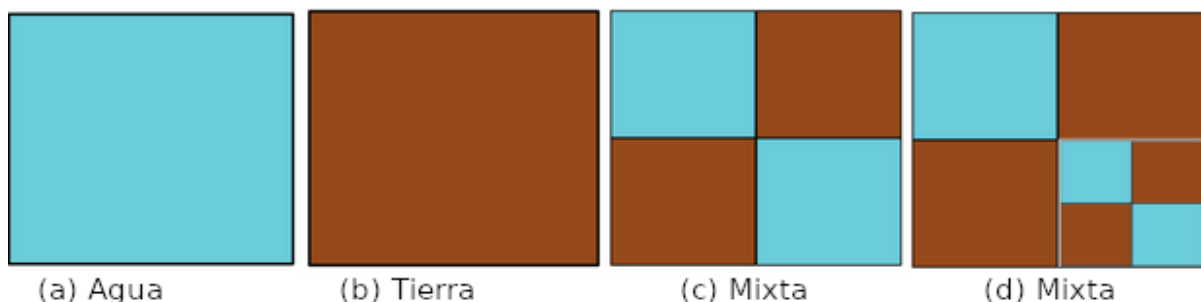
1. Modifique el diagrama de clases UML para considerar los cambios necesarios. Si utiliza patrones de diseño indique los roles en las clases utilizando estereotipos.
2. Implemente en Java

## Ejercicio 6: Topografías

Un uso común de imágenes satelitales es el estudio de las cuencas hídricas que incluye saber la proporción entre la parte seca y la parte bajo agua. En general las imágenes satelitales están divididas en celdas. Las celdas son imágenes digitales (con píxeles) de las cuales se quiere extraer su “topología”.

Un objeto Topografía representa la distribución de agua y tierra de una celda satelital, 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) y (d) topografías mixtas.



Una topografía mixta está formada por partes de agua y partes de tierra (4 partes en total). Estas 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 6b: 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 7: FileSystem

Un File System es un componente que forma parte del sistema operativo. Este está estructurado jerárquicamente en forma de árbol, comenzando con un directorio raíz.

Los elementos del file system pueden ser directorios o archivos. Los archivos contienen datos y los directorios contienen archivos u otros directorios. De cada archivo se conoce el nombre, fecha de creación y tamaño en bytes. De cada directorio se conoce el nombre, fecha de creación y contenido (el tamaño es siempre la cantidad inicial de 32kb más la suma del tamaño de su contenido). Modele el file system y provea la siguiente funcionalidad:

```
public class FileSystem {  
    /**  
     * Retorna el espacio total ocupado, incluyendo todo su contenido.  
     */  
    public int tamanoTotalOcupado()  
  
    /**  
     * Retorna el archivo con mayor cantidad de bytes en cualquier nivel del  
     * filesystem  
     */  
    public Archivo archivoMasGrande()  
}
```



```
/**
 * Retorna el archivo con fecha de creación más reciente en cualquier nivel
 * del filesystem
 */
public Archivo archivoMasNuevo()

/**
 * Retorna el primer elemento con el nombre solicitado contenido en
 * cualquier
 * nivel del filesystem
 */
public ?? buscar(String nombre)

/**
 * Retorna la lista con los elementos que coinciden con el nombre solicitado
 * contenido en cualquier nivel del filesystem
 */
public List<??> buscarTodos(String nombre)

/**
 * Retorna un String con los nombres de los elementos contenidos en todos
los
 * niveles del filesystem. De cada elemento debe retornar el path completo
 * (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
    /Directorio A/Directorio A.1/Directorio A.1.2
    /Directorio A/Directorio A.2
    /Directorio B
 */
public String listadoDeContenido()

}
```

### 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.
3. Implemente completamente en Java.





## Ejercicio 8: ToDoItem

Se desea definir un sistema de seguimiento de tareas similar a Jira<sup>1</sup>.

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,
     * genera un error informando la causa específica del mismo.
     */
    public Duration workedTime()

    /**
    Caso
     * Agrega un comentario al ToDoItem siempre y cuando no haya finalizado.
     * contrario no hace nada."
     */
}
```

---

<sup>1</sup> <https://es.atlassian.com/software/jira>



```
    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 9: 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.

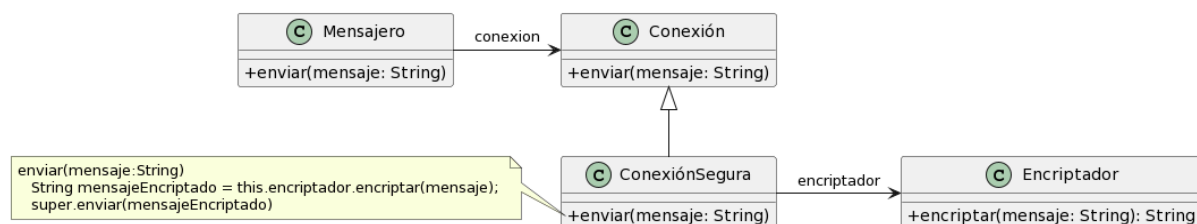
En una primera versión, al no contar con una interfaz de usuario y a los efectos de *debugging*, este comportamiento puede implementarlo en un método que retorne un `String` con la información solicitada.

## 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 10: Encriptador

En un sistema de mensajes instantáneos (como WhatsApp) 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 11: 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	Etapla 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:

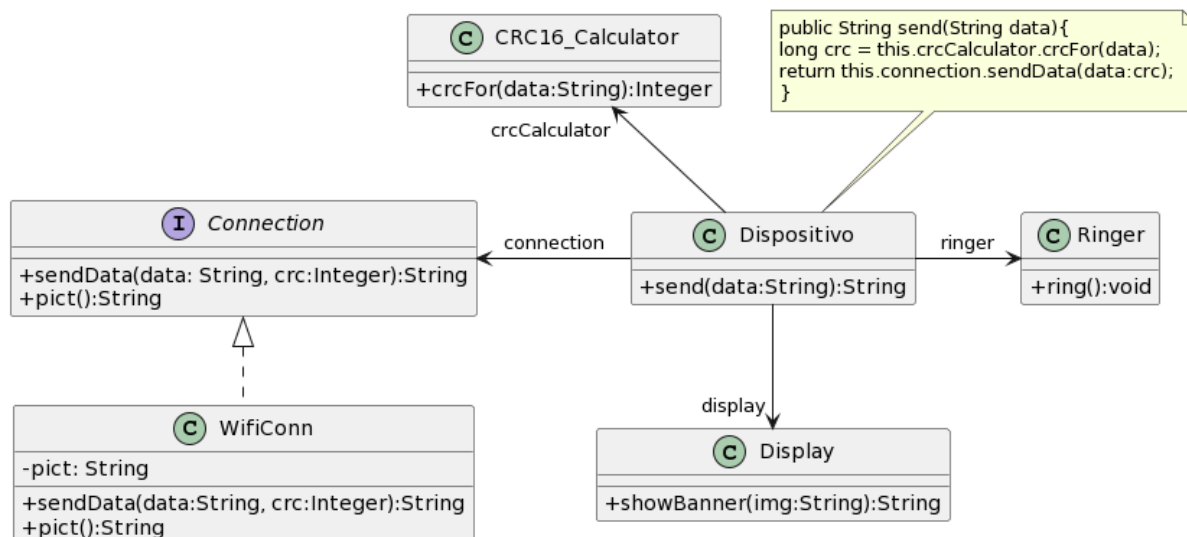
- 1- Modele una solución y provea el diagrama de clases UML para el problema planteado. Si utiliza algún patrón, indique cuál.
- 2- Implemente en Java.
- 3- Implemente un test para aprobar un proyecto con las siguientes características: (i) se encuentra en evaluación, (ii) se llama "Vacaciones de invierno",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

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

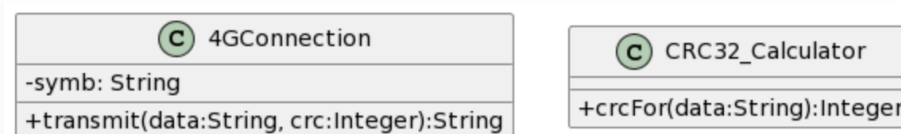
```
throw new RuntimeException("Este es mi mensaje de error");
```

## Ejercicio 12 - 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 13 - 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 similares a alguna película que reprodujo, ordenadas de más a menos reciente.
- (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

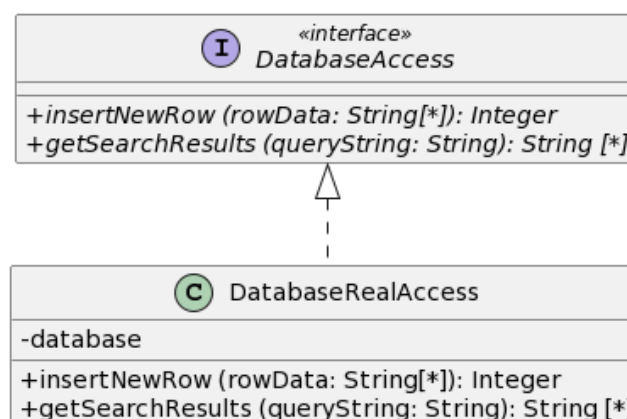
**Nota:** si existen más de 3 películas con el mismo criterio, retorna 3 de ellas sin importar cuales. Por ejemplo, si las 6 películas son del 2018, el criterio (i) retorna 3 cualquiera.

### 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 similitud (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 14: 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 de 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 15 - 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 16 - 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 getTemperatura()
```

```
//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> getTemperaturas()

//retorna un reporte de todos los datos: temperatura, presión, y
radiación solar.
public String displayData() {
    return "Temperatura F: " + this.getTemperatura() +
        "Presión atmosf: " + this.getPresion() +
        "Radiación solar: " + this.getRadiacionSolar();
}
```

Esta clase se encuentra implementada por terceros y no se puede modificar. Pero sabemos que implementa la interfaz WeatherData que define los mismos mensajes.

Si bien el código de la clase HomeWeatherStation no se puede modificar, se requiere poder integrar diferentes configuraciones que combinen algunas de las siguientes funcionalidades:

- La temperatura en grados Celsius ( $^{\circ}\text{C} = (^{\circ}\text{F} - 32) \div 1.8$ ).
- El promedio de las temperaturas históricas.
- Las temperaturas mínima y máxima histórica.

Esto implica que la aplicación debe ser capaz de adaptarse a diferentes necesidades de visualización. Por ejemplo:

Ej 1: "Temperatura F: 86; Presión atmosf.: 1008; Radiación solar: 200;"  
Ej 2: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200;"  
Ej 3: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200; Promedio: 30;"  
Ej 4: "Temperatura F: 86; Presión atmosf: 1008; Radiación solar: 200; Promedio: 86;"  
Ej 5: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200; Promedio: 30; Mínimo: 27 Máximo: 32;"  
Ej 6: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200; Mínimo: 27 Máximo: 32;"  
Ej 7: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200; Mínimo: 27 Máximo: 32; Promedio: 30;"

En cada uno de los ejemplos, la aplicación puede mostrar diferentes configuraciones de los datos, según lo que el usuario haya seleccionado previamente. Por ejemplo, la inclusión del promedio de temperatura (ya sea en grados Celsius o Fahrenheit) dependerá de la configuración de temperatura previamente establecida por el usuario.

Usted debe proveer la implementación del mensaje public String displayData() que devuelva los datos según lo configurado (dado que la app aun no cuenta con interface de usuario).

### Tareas:

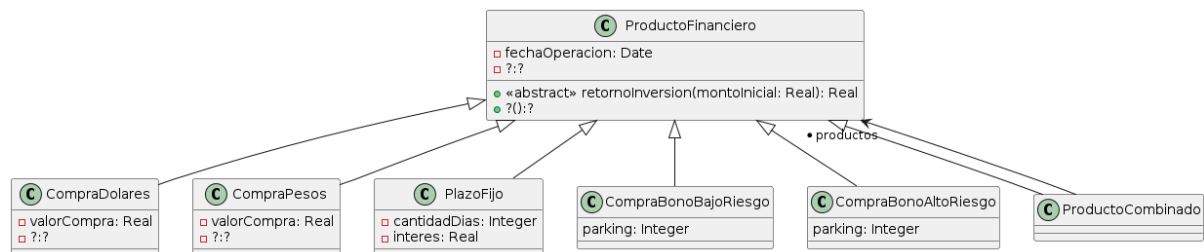
- 1- Modele una solución para el problema planteado. Si utiliza algún patrón, indique cuál
- 2- Implemente en Java
- 3- Implemente un test para validar la configuración del ejemplo 5, asumiendo que en el momento de la ejecución del mismo, los sensores arrojan los valores del ejemplo.

## Ejercicio 17 - Productos Financieros

El banco ofrece los siguientes productos financieros a sus clientes:

- Compra de Dólares
- Compra de Pesos
- Plazo Fijo (requiere indicar plazo e interés diario)
- Compra de bonos de bajo riesgo (tienen una variación máxima del 10% y se debe indicar un plazo mínimo o parking)
- Compra de bonos de alto riesgo (tienen una variación máxima del 70% y se debe indicar un plazo mínimo o parking)

A menudo, el banco también innova creando nuevos productos financieros que son combinaciones de estos productos existentes.



Cuando se invoca el mensaje `retornoinversion()`, se indica un monto inicial, para que calcule y retorne el capital que se obtiene al invertir ese monto inicial.

Algunos de los productos que ofrece, son combinaciones de otros, como por ejemplo:

<b>Producto 1:</b> 1) Compra de dólares 2) Plazo fijo en dólares (indicar plazo e interés) 3) Compra de pesos	<b>Producto 2:</b> 1) Compra de bonos de bajo riesgo (indicar parking) 2) Compra de dólares 3) Plazo fijo en dólares (indicar plazo e interés) 4) Compra de pesos
<b>Producto 3:</b> 1) Compra de bonos de alto riesgo (indicar parking) 2) Compra de bonos de bajo riesgo (indicar parking) 3) Plazo fijo (indicar plazo e interés)	<b>Producto 4:</b> 1) Compra de bonos de alto riesgo (indicar parking) 2) Compra de bonos de alto riesgo (indicar parking)



	3) Compra de bonos de alto riesgo (indicar parking)
--	---

En el banco, cada cliente posee un tipo de membresía que determina sus beneficios. Contamos con dos niveles de membresía: Silver y Gold. Los clientes con membresía Silver tienen acceso exclusivo a los productos 1 y 2. Además, al realizar plazos fijos, disfrutan de una tasa del 5% y un plazo mínimo de 35 días. Asimismo, al comprar bonos, el periodo mínimo de parking es de 72 horas. Por otro lado, los clientes con membresía Gold reciben una tasa de plazo fijo del 6%, con un plazo mínimo de 30 días, y un periodo de parking de 24 horas.

Se sabe que el banco se esfuerza constantemente por lanzar nuevos productos para satisfacer aún más las necesidades de inversión de sus clientes.

**Tareas:**

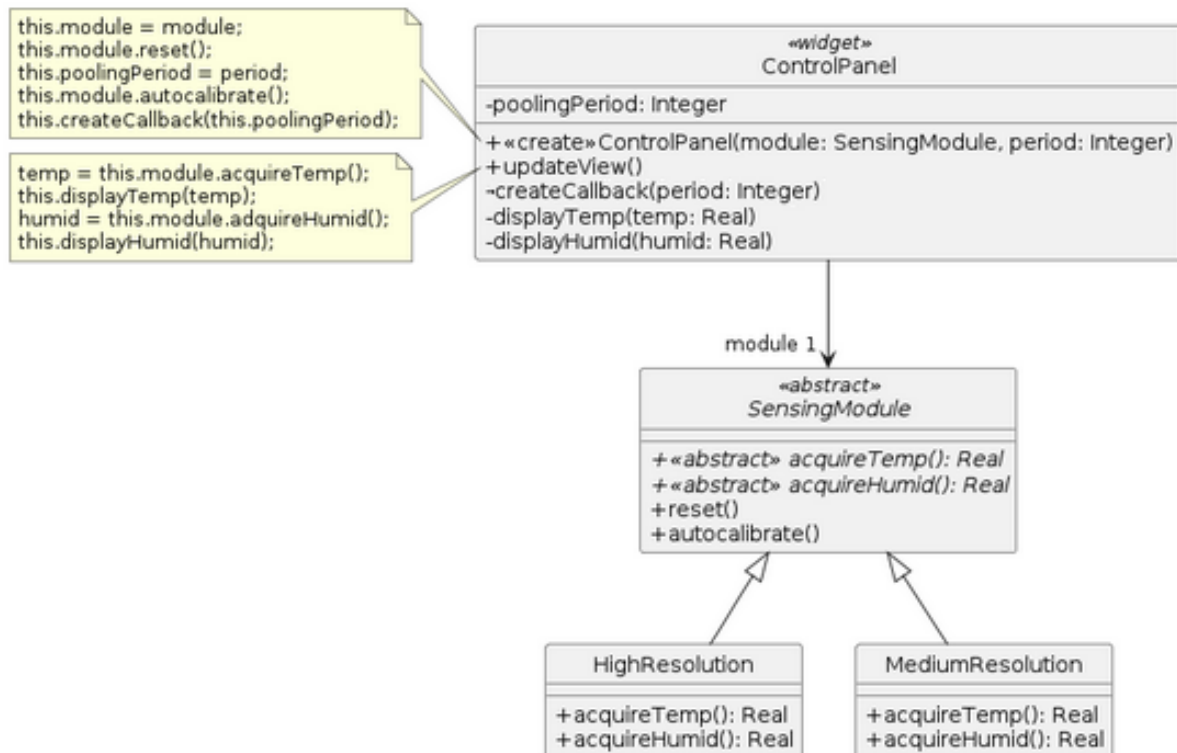
- a) Proponga una solución para que los desarrolladores no tengan que instanciar el Composite de Productos financieros explícitamente cada vez que sea necesario crear uno nuevo.
  - i) ¿Qué ventaja y que desventaja tiene su solución?

**Nota:** Para simular el comportamiento de los bonos se puede generar un número aleatorio dentro del rango de variación máxima, reflejando la típica volatilidad del mercado financiero

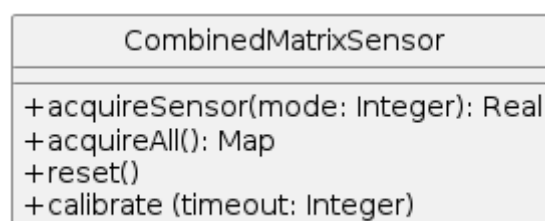
## Ejercicio 18 - Sistema de monitoreo

Una empresa ha desarrollado un sistema de monitoreo de viveros que incluye: temperatura y humedad. Para realizar esta tarea se utilizan módulos que incluyen sensores para ambos temperatura y humedad. El sistema fue desarrollado utilizando una familia de módulos que tienen diferentes prestaciones y que tienen un protocolo en común.

Instancias de ControlPanel cumplen el rol de cliente de las instancias de alguna de las clases de la Jerarquía SensingModule (variable module). Por la manera en la que una instancia de ControlPanel es inicializada con un "pollingPeriod" (ver pseudocódigo del constructor de ControlPanel), la instancia de ControlPanel recibirá periódicamente el mensaje updateView(), lo cual hace que se envíen los mensajes apropiados al "module" asociado.



La empresa planea agregar un nuevo módulo que corresponda a uno de baja resolución (Low Resolution). Para implementar esto la empresa se propone adquirir un nuevo tipo de sensor cuya librería tiene el protocolo que muestra el diagrama de la clase CombinedMatrixSensor.



Las operaciones que implementa la CombinedMatrixSensor son:

- `acquireSensor(mode: Integer)`. El parámetro "mode" que indica que tipo de medición se realizará. El valor 1 es para temperatura y el valor 2 es para humedad.
- `acquireAll()`, no tiene parámetros y retorna un HashMap con valores para temperatura (clave "temp") y humedad (clave "hum").
- `reset()`, no tiene parámetros y resetea todos los sensores
- `calibrate(timeout: Integer)`, el parámetro "timeout" indica cuál es el tiempo máximo que el sensor puede demorar en ejecutar una lectura.

Tareas:

1. ¿Cuál es el principal problema que identifica en el enunciado?
2. ¿Qué patrón aplicaría para este problema y cuáles son las consecuencias de aplicarlo?
3. Presente el diseño resultante y especifique el código/pseudocódigo resultante



## Ejercicio 19 - Personajes de juegos de rol

Una empresa de videojuegos ofrece personajes para sus juegos de rol. Cada personaje tiene un nombre y viene equipado con armaduras, armas, y habilidades únicas que les permiten desempeñarse mejor.

Las armaduras pueden ser de cuero, de hierro y de acero. Las armas pueden ser espadas, arcos y bastón de mago. Las habilidades pueden ser de combate cuerpo a cuerpo, de combate a distancia, de curación y de magia.

Actualmente ofrece 3 personajes: guerreros, arqueros y magos.

Los magos son expertos en el uso de la magia. Están equipados con una armadura de cuero para permitir la máxima movilidad. Su arma es un bastón mágico, y su habilidad es la magia y el combate a distancia. Los guerreros son los expertos en combate cuerpo a cuerpo, por lo tanto requieren una armadura de acero y una espada. Finalmente, los arqueros son especialistas tiros de flechas. Como deben moverse rápidamente, tienen una armadura de cuero y están equipados con arcos.

En el juego, los personajes tienen la posibilidad de enfrentarse entre sí. Durante un enfrentamiento, el resultado dependerá del arma que utilice el atacante y de la armadura que lleve el defensor. A continuación, se muestra una tabla con el daño que cada arma causa. Ese valor afecta el puntaje del jugador atacado.

	ArmaduraDeCuero	ArmaduraDeHierro	ArmaduraDeAcero
Espada	8	5	3
Arco	5	3	2
Baston	2	1	1

Al inicio del juego, cada personaje comienza con 100 puntos de vida, los cuales se reducirán a medida que se enfrenten a otros jugadores. Un personaje podrá participar de un combate siempre que tenga vida.

### Tareas:

1- Diseñe una solución orientada a objetos que permita la creación de personajes y el enfrentamiento entre ellos. Tenga en cuenta que en un futuro, la empresa planea ofrecer nuevos personajes. Provea el diagrama de clases UML y si utiliza algún patrón de diseño, indique cuál.

2- Implemente en Java el comportamiento descrito.



## Ejercicio 19b - Mas Personajes

Extienda el ejercicio anterior para permitir que la empresa pueda ofrecer un nuevo personaje llamado “Thoor”, que requieren armadura de hierro y un martillo, con habilidades de lanzar rayos y combate a distancia.

Nota: puede completar la tabla con los valores que desee.

## Ejercicio 20 - Filtros de Imágenes

En este [proyecto](#) encontrará la librería ImageFilter que implementa algunos filtros básicos sobre imágenes PNG tales como: Rainbow, Repeater y RGBShifter entre otros.

La librería incluye una herramienta (PNGFilterLauncher) que permite aplicar secuencia de filtros sobre una archivo de entrada y generar un archivo (.png) de salida.



PNG Original



Rainbow



Repeater



RGBShifter

Para probar el código, se deben configurar los argumentos de ejecución del proyecto. Se debe especificar la ubicación de la imagen de entrada y de salida, así como indicar qué filtros se van a aplicar.





```
img\input.png img\output.png --rainbow --artifacter
```

### Tareas:

1. Descargue el proyecto y pruebe los diferentes Filtros
2. Analice el código y documente el proyecto con un Diagrama de Clases
3. Evalúe cuál de los siguientes patrones mejor describe el diseño de los Filtros: TemplateMethod, Strategy, Decorator. Para realizar la evaluación se sugiere contestar las siguientes preguntas aplicadas a cada uno de los patrones:
  - a. ¿El objetivo del patrón se distingue en el diseño? Elabore en un párrafo.
  - b. ¿La estructura del proyecto coincide con la estructura y los participantes del patrón? Elabore en un párrafo.
  - c. En el caso que el patrón coincida, puede distinguir un “smell” o algo que se aleja del patrón presentado en el libro?
4. Se requiere crear el filtro Monochrome. El pseudo código para crear una imagen monocrome es el siguiente:

Parámetro de entrada: BufferedImage image

```
1. Por cada pixel en la imagen {  
    int pixel = image.getRGB(x, y);  
    int alpha = (pixel >> 24) & 0xff;  
    int red = (pixel >> 16) & 0xff;  
    int green = (pixel >> 8) & 0xff;  
    int blue = pixel & 0xff;  
  
    int avg = (red + green + blue) / 3;  
    pixel = (alpha << 24) | (avg << 16) | (avg << 8) | avg;  
    image.setRGB(x, y, pixel);  
}
```

## Ejercicio 20b - Secuencia de Filtros

Basado en el proyecto ImageFilter presentado en el ejercicio 17, se desarrolló una nueva versión que permite componer secuencia de filtros. Por ejemplo, es posible crear una secuencia con los siguientes filtros: rgb-shifter > repeater > monochrome. Las secuencias se implementan utilizando las clases en el paquete “Pipes”

### Tareas:

1. Descargue [este proyecto](#) y pruebe los diferentes Filtros
2. Analice el código y documente el proyecto con un Diagrama de Clases
3. Evalúe cuál de los siguientes patrones mejor describe el diseño de los Pipes: TemplateMethod, Strategy, Decorator. Para realizar la evaluación se sugiere contestar las siguientes preguntas aplicadas a cada uno de los patrones:
  - a. ¿El objetivo del patrón se distingue en el diseño? Elabore en un párrafo.
  - b. ¿La estructura del proyecto coincide con la estructura y los participantes del patrón? Elabore en un párrafo.



- c. En el caso que el patrón coincida, puede distinguir un “smell” o algo que se aleja del patrón presentado en el libro?

## Ejercicio 20c - Instanciando Secuencia de Filtros

Se desea crear una nueva versión del proyecto ImageFilter con secuencias para implementar reglas de combinación de filtros. Se ha detectado que la incorporación del filtro Monochrome implica combinaciones que no tienen sentido o que no genera resultados aceptables. Para entender estos casos pruebe las siguientes secuencias:

- rainbow>monochrome vs monochrome>rainbow.
- monochrome>artifacter vs artifacter>monochrome
- rgb-shifter>monochrome vs monochrome>rgb-shifter

Se desea implementar dos tipos de filtros, aquellos que trabajan sobre el rango multicromáticos (Multichrome) y los monocromáticos (Monochrome). Considerando las siguientes reglas:

- Secuencias multicromáticas no incluyen el filtro Monochrome
- Secuencias monocromáticas no incluyen el filtro Rainbow
- Secuencias monocromáticas aplican el filtro Monochrome como último paso de la secuencia

### Tareas:

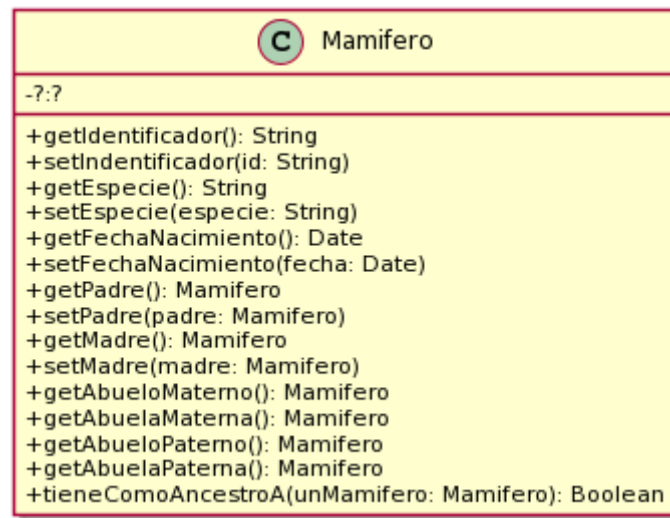
1. Se debe diseñar una solución que oculte las reglas antes descritas de tal manera que el código que use la librería simplemente delegue la creación de secuencias (consistentes) a los objetos resultantes de su diseño. Considere que en el futuro nuevos filtros o nuevas reglas pueden ser necesarias.
2. Implemente su diseño de tal manera que pueda ser utilizado con la herramienta ejecutable del proyecto (PNGFilterLauncher)
3. ¿Identifica algún patrón de diseño que sea esencial en su diseño?
  - a. ¿Cuál es el objetivo del patrón?
  - b. ¿Cuál es la estructura y cuáles son los participantes?
  - c. Analice la implementación y describa si existen detalles de implementación sean code smells o que no corresponden con el patrón

## Ejercicio 21 - Genealogía salvaje

Retomamos el ejercicio de [genealogía salvaje de Objetos 1](#), ahora utilizando los conceptos vistos en Objetos 2.

Se trata de una reserva de vida salvaje (como la estación de cría ECAS, en el camino Centenario), donde los cuidadores están interesados en llevar un registro detallado de los animales que cuidan y sus familias. Para ello nos han pedido ayuda.

El siguiente es un diagrama de clases inicial (incompleto) y nos da una idea de los mensajes que un mamifero entiende.



### Tareas:

1. Complete el diagrama de clases para reflejar los atributos y relaciones requeridos.
2. Implemente completamente en Java. Si utiliza algún patrón de diseño, indique cuál.

## Ejercicio 22 - SubteWay

Sugerencia: no resuelva este ejercicio en ayunas.

Una cadena de comidas rápidas especializada en sándwiches necesita resolver el cálculo de precios de éstos. El cálculo es simple: el precio de un sándwich equivale a la suma del precio de cada uno de sus componentes; el problema es la dificultad para representar y crear cada uno de los sándwiches distintos.

Existen cuatro sandwiches distintos, pero podrían aparecer nuevos en el futuro.

Clásico: consta de pan brioche (100 pesos), aderezo a base de mayonesa (20 pesos), principal de carne de ternera(300 pesos) y adicional de tomate (80 pesos).

Vegetariano: consta de pan con semillas (120 pesos), sin aderezo, principal de provoleta grillada (200 pesos) y adicional de berenjenas al escabeche (100 pesos).

Vegano: consta de pan integral (100 pesos), aderezo de salsa criolla (20 pesos), principal de milanesa de girgolas (500 pesos), sin adicional.



Sin TACC: consta de pan de chipá (150 pesos), aderezo de salsa tártara (18 pesos), principal de carne de pollo (250 pesos) y adicional de verduras grilladas (200 pesos).

**Tareas:**

Proponga un diseño e implementación para la creación y cálculo de precios de estas alternativas de sándwiches.

Si en su solución aplicó algún(os) patrón(es) de diseño, indique cuál(es) y sus roles.