PEC 3

Asociaciones

Universitat Oberta de Catalunya

Información relevante:

- Fecha límite de entrega: 28 de noviembre de 2022.
- Peso en la nota de EC: 30%.

Contenido

Información docente

Esta PEC está vinculada con el módulo teórico "Asociaciones (relaciones entre objetos)" de los apuntes de la asignatura. Léelo antes de empezar la PEC.

Prerrequisitos

Para hacer esta PEC necesitas:

- Tener adquiridos los conocimientos de las PEC 1 y PEC 2. Para ello te recomendamos que mires las soluciones que se publicaron en el aula y las compares con las tuyas.

Objetivos

Con esta PEC el equipo docente de la asignatura busca que:

- Entiendas las diferencias entre asociación binaria, de agregación y de composición.
- Codifiques asociaciones en Java.
- Entiendas qué es una enumeración y el potencial que tienen en Java.
- Utilices una librería proporcionada por terceros.

Resultados de aprendizaje

Con esta PEC debes demostrar que eres capaz de:

- Codificar una clase a partir de unos requisitos y su representación como diagrama de clases.
- Entender la diferencia entre los diferentes modificadores de acceso.
- Saber cómo codificar una asociación de composición en Java (y en cualquier lenguaje).
- Saber cómo codificar una asociación binaria/de agregación bidireccional de manera que la información del programa sea siempre consistente/coherente.
- Codificar un `enum` en Java añadiéndole funcionalidades.
- Saber cómo tratar las excepciones de un programa Java.
- Usar con cierta soltura un entorno de desarrollo integrado (IDE) como IntelliJ.
- Utilizar test unitarios para determinar que un programa es correcto.
- Entender qué es un test de integración.

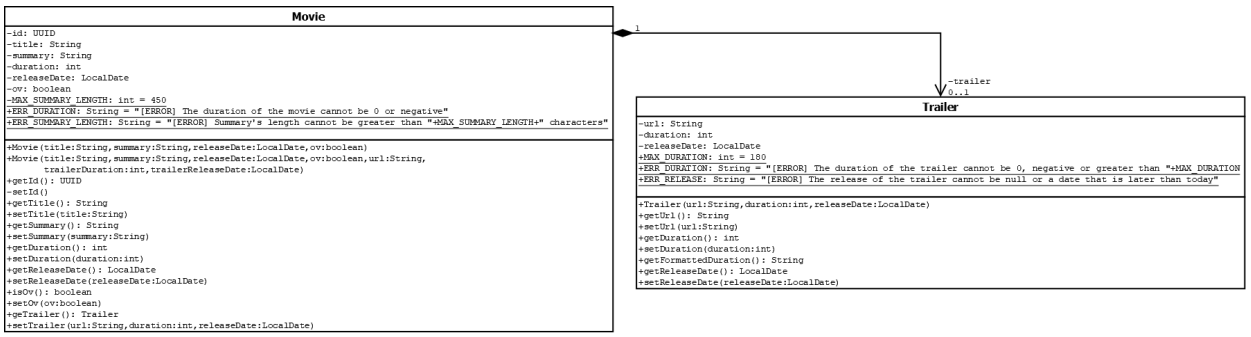
Enunciado

Esta PEC contiene 4 ejercicios evaluables. Debes entregar tu solución de los 4 ejercicios evaluables (ver el último apartado).

Debido a que las actividades están encadenadas (i.e. para hacer una se debe haber comprendido la anterior), **es altamente recomendable hacer los ejercicios en el orden en que aparecen en este enunciado.**

Ejercicio 1 (1.5 puntos)

Abre el proyecto `PAC3Ex1` con IntelliJ. En el *package* `edu.uoc.pac3` del directorio `/src/test/java` está el fichero con los test unitarios que te proporcionamos. En el *package* `edu.uoc.pac3` del directorio `/src/main/java` debes codificar la clases `Movie` y `Trailer`, cuya representación en diagrama de clases UML es:



Como puedes ver en el diagrama UML (te lo adjuntamos con el enunciado para que lo veas mejor), vamos a modelar el comportamiento de una película (`Movie`) y su relación con su trailer (`Trailer`). La clase `Trailer` abstrae la información relacionada con los datos del trailer que presenta una película. **La relación entre las clases `Movie` y `Trailer` es una asociación unidireccional de composición**, con la clase `Movie` como clase compuesta y la clase `Trailer` como componente, de manera que si desaparece el objeto `Movie`, desaparezca el objeto `Trailer` que estaba vinculado.

Para la codificación de estas clases debes tener en cuenta las siguientes especificaciones/consideraciones:

Clase `Trailer`

- Los atributos de tipo “constante” deben ser inicializados en la propia declaración, no en el constructor.
- **Siempre que exista conflicto de nombres**, sobre todo en los métodos *setter* (i.e. `setXXX`), **debes usar la palabra reservada `this`**.

- Si la duración (que en este caso se expresa en segundos) que se quiere asignar al atributo `duration` es negativa, cero o superior al valor de `MAX_DURATION`, entonces no se debe asignar dicho valor y en su lugar se debe lanzar una excepción de tipo `Exception` con el mensaje de `ERR_DURATION`.

- Si el valor que se desea asignar al atributo `releaseDate` es `null` o posterior a la fecha actual, entonces no se debe asignar dicho valor y se debe lanzar una excepción de tipo `Exception` con el mensaje de `ERR_RELEASE`.

Pista: investiga en Internet acerca de la clase `LocalDate` para determinar si una fecha es anterior o posterior a otra, así como para obtener la fecha actual.

- El método `getFormattedDuration` debe devolver un `String` con el formato siguiente: `mm:ss`, es decir, 2 dígitos para los minutos y 2 para los segundos. En este caso, sabemos que el valor de `duration` nunca será menor a 1, puesto que `setDuration` controla que esto no ocurra. De todas maneras, en este caso vamos a usar el `assert` de Java para aseverar que en este punto del código dicha condición (i.e. `duration` es mayor a cero) debe ser cierta. Aquí tienes unos enlaces que hablan de los `asserts` de Java: https://www.baeldung.com/java-assert y https://www.delftstack.com/es/howto/java/java-assert/.

Pista: investiga en Internet acerca de la clase `Duration` para obtener los segundos y minutos a partir de una cantidad expresada en segundos. Asimismo, investiga acerca del método `format` de la clase `String` para poder cumplir con el patrón que se pide, i.e. `mm:ss`.

Clase `Movie`

- Los atributos de tipo “constante” deben ser inicializados en la propia declaración, no en el constructor.

- La gestión del `id` de dicha clase se basará en el formato **UUID** (*Universally Unique Identifier*), que son números de 128 bits y son únicos en todos los sistemas locales en los que se crean, y también en los UUID creados entre otros sistemas.

  El UUID se compone de dígitos hexadecimales (4 caracteres cada uno) junto con 4 símbolos "-" que hacen que su longitud sea igual a 36 caracteres (p.ej. **`123e4567-e89b-12d3-a456-556642440000`**).

  Para trabajar con estos identificadores vamos a utilizar la clase `UUID` de Java del *package* `java.util`. Para la gestión de nuestros id’s, utilizaremos el **método estático** de la clase `UUID` que genera identificadores aleatorios. Esta es la manera más conveniente de crear un UUID, en lugar de ser nosotros mismos los que construyamos el patrón de bits “manualmente”.

  Para obtener más información sobre este tema, aquí tienes una buena guía de UUID: https://www.baeldung.com/java-uuid.

- El método `getId` debe devolver el objeto `UUID` de la clase `Movie`.
- El método `setId` debe crear un `UUID` aleatorio utilizando la clase `UUID` de Java.
- Al crear un objeto de tipo `Movie` se debe crear el objeto `Trailer` que le corresponda (puede ser `null`).
- En el caso del primer constructor, el objeto `Movie` creado tendrá un `Trailer` con valor `null`.
- Si el resumen (`summary`) que se quiere asignar es mayor al valor de `MAX_SUMMARY_LENGTH`, entonces se debe lanzar una excepción de tipo `Exception` con el mensaje que contiene el atributo `ERR_SUMMARY_LENGTH`. En caso contrario, se debe asignar el valor del parámetro al atributo `summary`.
- Si la duración (que se expresa en minutos) que se quiere asignar a la película es cero o inferior, entonces se debe lanzar una excepción de tipo `Exception` con el mensaje que contiene el atributo `ERR_DURATION`. En caso contrario, se debe asignar el valor del parámetro al atributo `duration`.

Importante: todas las clases deben quedar correctamente documentadas con comentarios Javadoc. Sin embargo, no es necesario generar la documentación.

Requisito mínimo para evaluar este ejercicio: el programa debe pasar los test `TrailerTest` y `MovieTest`, sin incluir `testGetFormattedDuration`.

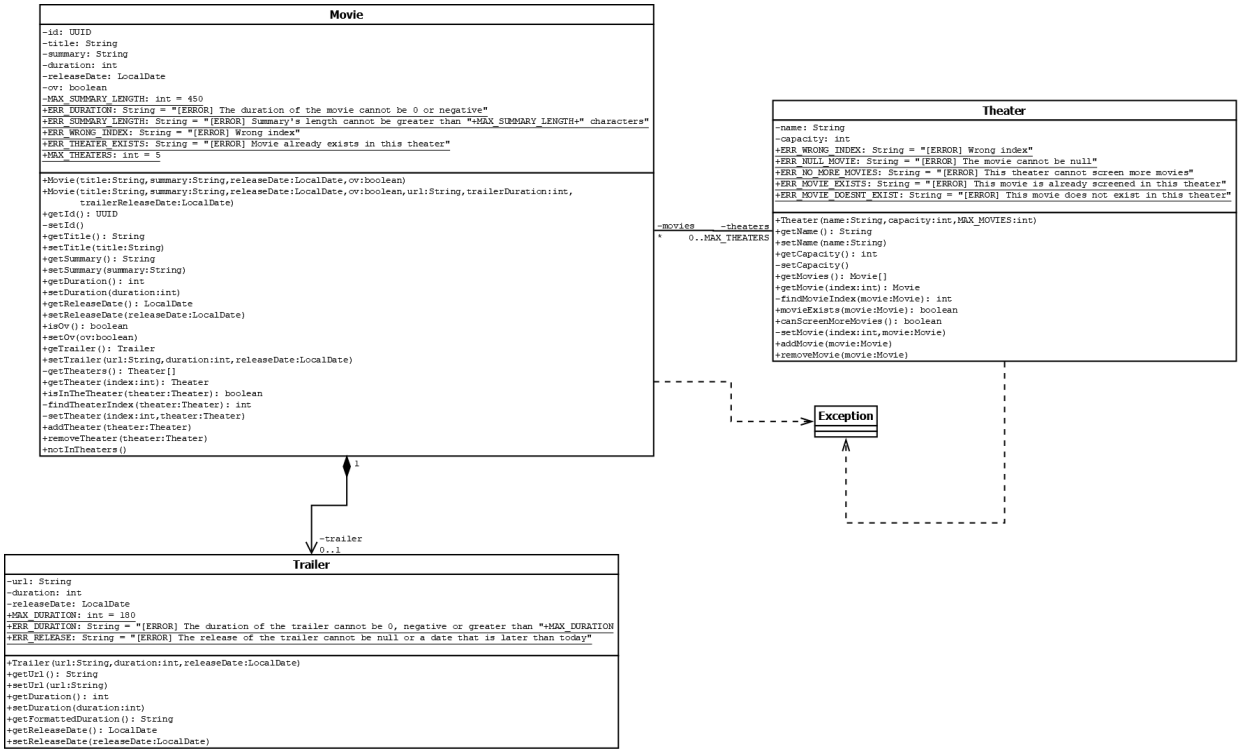
Nota: El estudiante puede recibir una penalización de hasta **0.5 pts.** de la nota obtenida en este ejercicio en función de la calidad del código proporcionado.

(1.5 puntos: 0.5 pts. TrailerTest; 0.75 pts. MovieTest sin testGetFormattedDuration; 0.25 pts. testGetFormattedDuration)

Ejercicio 2 (6 puntos)

Abre el proyecto `PAC3Ex2` desde IntelliJ. En el *package* `edu.uoc.pac3` del directorio `/src/test/java` está el fichero con los test unitarios que te proporcionamos. Copia el *package* `edu.uoc.pac3` del directorio `/src/main/java` del Ejercicio 1 de esta PEC3 una vez lo hayas completado.

En este ejercicio ampliamos el programa anterior añadiendo una nueva clase, concretamente la clase `Theater`, la cual modela/representa las salas donde podemos visionar las diferentes películas. El nuevo diagrama de clases UML es (te lo adjuntamos con el enunciado para que lo veas mejor):



Ten presente las siguientes especificaciones para las clases `Movie` y `Theater`:

Clase `Movie`

- Una película inicialmente no es asignada a ninguna sala (i.e. valor `null`).
- El método `getTheaters` devuelve un array de objetos `Theater` o, en caso de que la película no esté en ninguna sala, el array con todos sus posiciones a `null`.
- El método `getTheater` devuelve el objeto de tipo `Theater` que hay en la posición `index`. En caso de que `index` sea un valor negativo o igual o superior a `MAX_THEATERS`, entonces lanzará una `Exception` con el mensaje que hay en `ERR_WRONG_INDEX`.

- El método `isInTheTheater` dice si la sala pasada como parámetro proyecta la película (`true`) o no (`false`). Si el valor pasado como argumento es `null`, entonces devuelve `false`.
- El método `findTheaterIndex` devuelve el índice/posición donde se encuentra la sala pasada como parámetro. Este método puede recibir el valor `null`. Si el objeto pasado como parámetro no existe, entonces este método devuelve `-1`.
- El método `addTheater` añade un sala a la película. Ten en cuenta:
    - Si la película ya está en la sala, entonces debe lanzar una `Exception` con el mensaje contenido en `ERR_THEATER_EXISTS`.
    - La película será guardada en la primera posición libre que haya en el array `theaters`. Una posición está libre si su valor es `null`.
    - Si el valor pasado como argumento a `addTheater` es `null`, entonces este método no debe hacer nada.
    - Este método debe invocar el método privado `setTheater`, encargado de asignar el parámetro `theater` en la posición `index` del array `theaters`. En caso de que `index` sea un valor negativo o igual o superior a `MAX_THEATERS`, entonces lanzará una `Exception` con el mensaje que hay en `ERR_WRONG_INDEX`.
    - Si se le asigna una sala concreta a la película, entonces la sala debe ser actualizada para que contenga la película. De esta manera, la información del programa es consistente y coherente. Este método debe propagar las `Exception` que lancen los métodos que se utilicen dentro.
- El método `removeTheater` elimina de la película la sala pasada como argumento. Al quitar la sala de la película, la información que la sala guarda acerca de la película debe ser actualizada. En el momento de dejar de proyectar una película en una sala, dicha película no debe constar tampoco como película que se proyecta en la sala. De esta manera, la información es consistente.
- El método `notInTheaters` elimina la película de todas las salas en las que esté. Es decir, hace que la película no se proyecte en ninguna sala.

Clase `Theater`

- El constructor debe inicializar el *array* de películas (atributo `movies`) con un tamaño igual al valor del parámetro `MAX_MOVIES`.
- `setCapacity`: este método asigna el valor pasado como parámetro al atributo `capacity` (aforo).

- `getMovies`: este método devuelve el *array* de películas (`movies`).
- `getMovie`: devuelve la película que está en la posición `index`. Si el índice es negativo o igual o superior al valor de `MAX_MOVIES`, entonces debe lanzar la `Exception` `ERR_WRONG_INDEX`.
- `findMovieIndex`: este método privado devuelve el índice en el que está el objeto de tipo `Movie` pasado como parámetro. Si el valor recibido es `null`, también lo busca, devolviendo la primera posición `null` que haya en `movies`. Si el objeto no existe en `movies`, entonces devuelve el valor `-1`. Este método propaga las excepciones de los métodos a los que invoca.
- `movieExists`: devuelve `true` si el la película pasada como parámetro está en la sala. Si la película no está o el valor pasado como parámetro es `null`, entonces este método debe devolver `false`. Este método propaga las excepciones de los métodos a los que invoca.
- `canScreenMoreMovies`: devuelve `true` si la sala puede en ese momento proyectar más películas. Es decir, si el atributo `movies` contiene alguna casilla a `null`. **Este método NO debe lanzar ninguna excepción**.
- `setMovie`: asigna la película recibida como parámetro en la posición `index` indicada, siempre y cuando, el valor de `index` sea positivo y menor a `MAX_MOVIES`. En caso de que el parámetro `index` no satisfaga el requisito anterior, este método debe lanzar una `Exception` con el mensaje `ERR_WRONG_INDEX`.
- `addMovie`: guarda la película pasada como argumento en la primera posición libre del array `movies`. Ten en cuenta que:
    - Si el objeto pasado como parámetro es `null`, entonces debe lanzar una `Exception` con el mensaje de `ERR_NULL_MOVIE`.
    - Si la sala no puede proyectar más películas, entonces debe lanzar una `Exception` con el mensaje de `ERR_NO_MORE_MOVIES`.
    - Si la película ya se proyecta en la sala, entonces debe lanzar una `Exception` con el mensaje de `ERR_MOVIE_EXISTS`.

  Si es posible asignar la película, entonces ésta debe ser añadida. Al mismo tiempo se debe actualizar la información que la película tiene sobre la sala en la que se encuentra para que la información almacenada en el programa sea consistente.
- `removeMovie`: elimina de la sala la película pasada como argumento. Al quitar la película de la sala, la información que la película guarda acerca de la sala en el que está debe ser actualizada.

Si la película que se desea eliminar de la sala, no se proyecta en dicha sala, entonces este método debe lanzar una `Exception` con el mensaje de `ERR_MOVIE_DOESNT_EXIST`.

MUY IMPORTANTE: debes controlar que toda la información que se almacena en el programa sea coherente/consistente. Así pues, cuando se añade una película a una sala, la información que la película tiene de las salas en las que se proyecta, también debe ser actualizada. De igual modo, si añadimos o borramos alguna de las salas de una película a través de sus métodos `addTheater` o `removeTheater`, entonces la información de las salas también debe ser actualizada.

Requisito mínimo para evaluar este ejercicio: todos los test proporcionados en las clases `MovieTest` y `TheaterTest` deben ser superados con éxito de manera independiente.

Integration Test: las dos clases de test anteriores no ejecutan los métodos implicados en la inserción y eliminación de salas y películas. Éstos son comprobados por `IntegrationTest`. No es necesario pasar todos los tests de `IntegrationTest` para obtener puntuación, ya que ésta se calculará de la siguiente manera: +0.45 pts por cada test pasado satisfactoriamente.

Nota: El estudiante puede recibir una penalización de hasta **1 punto** de la nota obtenida en este ejercicio en función de la calidad del código proporcionado.

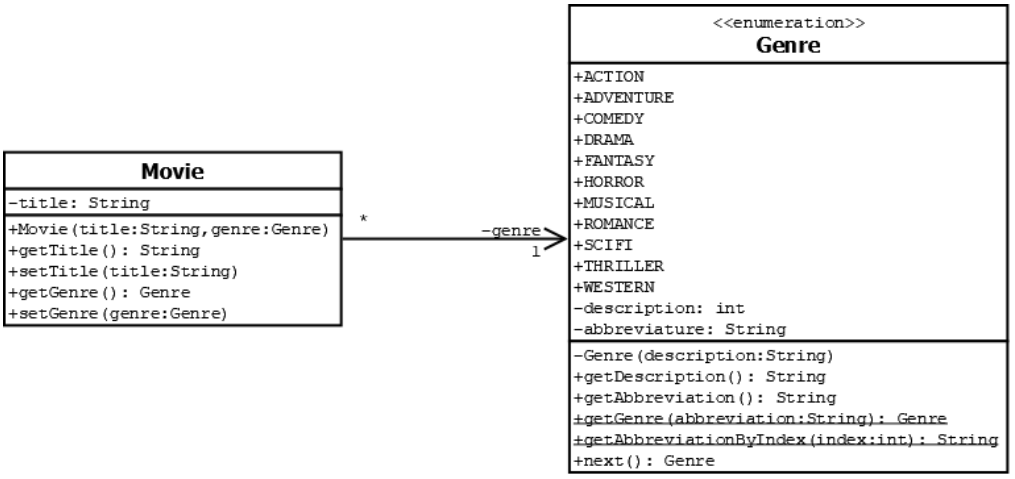
(6 puntos: 0.5 pts. test `MovieTest`***; 1 pt. test*** `TheaterTest`***; 4.5 pts.*** `IntegrationTest`***)***

Ejercicio 3 (1.5 puntos)

Antes de empezar debes:

> Leer el apartado 3.13 de la Guía de Java que habla sobre las enumeraciones.
>
> Te recomendamos ver el vídeo sobre enumeraciones que encontrarás en UOCoders.

A continuación, **abre el proyecto `PAC3Ex3` desde IntelliJ**. En el *package* `edu.uoc.pac3` del directorio `/src/test/java` está el fichero con los test unitarios que te proporcionamos. En el directorio `/src/main/java` te proporcionamos la implementación de una versión simplificada de la clase `Movie` que hemos ido utilizando hasta ahora. Dicha versión –que usarás para este ejercicio– está especificada en el diagrama de clases UML que puedes ver a continuación.



En este ejercicio SÓLO tienes que codificar el `enum Genre` siguiendo las especificaciones del anterior diagrama de clases UML. Fíjate cómo se definen los `enum` en UML. Existe otra notación que consiste en poner el atributo de tipo `enum` dentro de la clase que lo utiliza indicando su tipo, igual que si fuera un atributo de tipo, por ejemplo, `int`. En este caso no habría flecha hacia el `enum`.

Enum `Genre`

El `enum Genre` modela el género principal de una película. Para ello, debes tener en cuenta:

- El `enum Genre` tiene un atributo `description` que guarda una descripción de qué significa el género y un atributo `abbreviation` que almacena la abreviación del nombre del género. La relación "género-descripción-abreviación" es la siguiente:

```
ACTION | "Movies with shots, explosions, etc." | A
ADVENTURE | "Thousands of things happen" | ADV
COMEDY | "Lot of laughs" | C
```

```
DRAMA   | "Plenty of sadness" | D
FANTASY | "Imaginary friends" | F
HORROR  | "Screams" | H
MUSICAL | "La, la, la" | M
ROMANCE | "Love, love and much love" | R
SCIFI | "Aliens, time travels and so on" | SF
THRILLER | "Crimes, clues, policemen..." | T
WESTERN | "Bang!!! Movies wanted" | W
```

Nota: Codifica los valores del enum en el orden en el que aparecen en el listado anterior.

- `Genre`: es el constructor, el cual debe guardar los valores de los parámetros `description` y `abbreviation`.
- `getDescription` y `getAbbreviation`: cuando son invocados usando un valor del `enum`, devuelven su valor de `description` y `abbreviation`, respectivamente.
- `getGenre`: este método recibe un `String` que representa un género en su formato abreviación, y devuelve cuál de los valores de la enumeración le corresponde a dica abreviación. Es decir, devuelve el `Genre` correspondiente, o `null` si el valor de la abreviación pasada como argumento no corresponde con ningún valor del `enum`.

  (0.25 puntos: 0.25 puntos test proporcionados)
- `getAbbreviationByIndex`: este método devuelve un `String` con el valor de `abbreviation` del género que se encuentra en el índice recibido como parámetro. En caso de que el índice que pasamos como argumento sea mayor que el número de valores del `enum`, debe devolver el valor `abbreviation` del último valor del `enum`. En caso de que el índice que se pasa como argumento sea menor que `0`, debe devolver el valor `abbreviation` del primer valor del `enum` (i.e. el del índice `0`).

  (0.5 puntos: 0.5 puntos test proporcionado)
- `next`: este método devuelve el valor del enum `Genre` posterior al valor con el que se invoca este método. Así, si hacemos `Genre.COMEDY.next()`, el resultado debe ser `DRAMA`. Si el valor con el que invocamos `next` es el último de la enumeración, entonces debe devolver el primer valor, i.e. `Genre.WESTERN.next()` → `ACTION`.

  (0.75 puntos: 0.75 puntos test proporcionados)

Requisito mínimo para evaluar este ejercicio: los test `testGetDescription`, `testGetAbbreviation`, `testGetGenre` y `testgetAbbreviationByIndex` deben ser superados. Los métodos `testGetDescription` y `testGetAbbreviation` no tienen puntuación.

Nota: El estudiante puede recibir una penalización de hasta **0.5 pts.** de la nota obtenida en este ejercicio en función de la calidad del código proporcionado.

Ejercicio 4 (1 punto)

Abre el proyecto `PAC3Ex4` desde IntelliJ. En el *package* `edu.uoc.pac3` del directorio `/src/test/java` está el fichero con los test unitarios que te proporcionamos. En el directorio `/src/main/java` te proporcionamos la implementación de una versión reducida de la clase `Movie` y una versión reducida de la clase `Actor` (ambas versiones serán las que usarás para realizar este ejercicio). En este ejercicio se pide la implementación de dos métodos en la clase `Movie` que devuelvan el objeto Java en formato JSON. Antes de comenzar, vamos a hacer una pequeña introducción a JSON.

JSON (acrónimo de *JavaScript Object Notation*) es un formato de texto sencillo para el intercambio de datos, es decir, la principal función es crear un estándar para almacenar e intercambiar información de texto.

JSON posee algunas ventajas sobre XML como formato de intercambio de datos. Es fácil de aprender, de ser interpretado/parseado y generado. Además, es ligero al transportarse por la red y ya está soportado por la mayoría de los lenguajes de programación actuales. Para saber más sobre las especificaciones del formato JSON (cómo se representa cada elemento JSON: *object, array, string*, etc.), consulta https://www.json.org/json-es.html

Vamos a ver unos ejemplos concretos con la clase `Movie`:

1. Supongamos que tenemos el siguiente objeto `Movie`, sin tener en cuenta el array de `casting` que contiene objetos de tipo `Actor`.

```
Movie movie = new Movie("The Truman Show", "Bla, bla, bla");
```

La salida en JSON debería ser:

```
{
  "title": "The Truman Show",
  "summary": "Bla, bla, bla"
}
```

2. Supongamos que tenemos el siguiente objeto `Movie`, ahora teniendo en cuenta el array de `casting`.

```
Movie movie = new Movie("The Truman Show", "Bla, bla, bla", new
Actor[]{new Actor("Jim", "Carrey"), new Owner("Laura", "Linney"),
new Owner("Ed", "Harris")});
```

La salida en JSON debería ser:

```
{
  "title": "The Truman Show",
  "summary": "Bla, bla, bla",
  "casting": [
    {
      "name": "Jim",
      "surname": "Carrey"
    },
    {
      "name": "Laura",
      "surname": "Linney"
    },
   {
      "name": "Ed",
      "surname": "Harris"
    }
  ]
}
```

A este último JSON que hemos visto se le conoce como *"pretty print"*, es decir, "impresión bonita" o, más coloquialmente, *"formateada"*. Este mismo JSON sin formatear (en una única línea), sería:

```
{"title": "The Truman Show","summary": "Bla, bla, bla", "casting": [{"name": "Jim",
"surname": "Carrey"},{"name": "Laura","surname": "Linney"},{"name": "Ed","surname":
"Harris"  }]
```

Pero para trabajar con este formato de salida, no lo vamos a hacer como hemos hecho hasta ahora mediante `print/println` y concatenación de `String` hasta formar la salida que queremos, sino que vamos a utilizar una **librería de terceros,** y para ello vamos a utilizar el gestor de dependencias de Gradle.

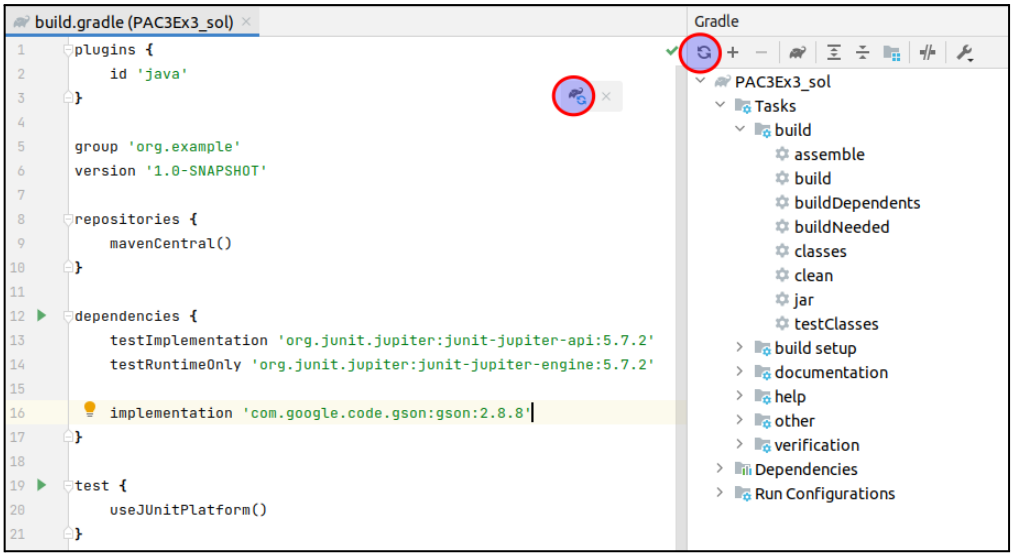
Existen varias librerías Java para trabajar con JSON. Las 5 más populares son: ***Jackson***, ***GSON***, ***json-simple***, ***Flexjson*** y ***JSON-lib***. En nuestro caso, y por simplicidad de uso para este ejercicio en concreto, vamos a utilizar GSON.

Para añadir la librería GSON a nuestro gestor de dependencias, debemos ir al fichero `build.gradle` de la raíz de nuestro proyecto y añadir la siguiente dependencia (ver la imagen que hay más adelante):

```
implementation 'com.google.code.gson:gson:2.9.1'
```

Cada vez que cambiamos la estructura del proyecto Gradle, como en este caso que hemos añadido una dependencia, debemos hacer un *reload* para que los cambios tengan efecto, y Gradle nos descargue del repositorio la librería (fichero `*.jar`) que hemos añadido.

En la siguiente imagen vemos cómo queda el fichero `build.gradle`, y a qué botones tienes que clicar (con clicar a uno de ellos es suficiente) para hacer el *reload* del proyecto.



Tras esta rápida introducción a JSON y al gestor de dependencias de Gradle, debes:

Clase `Car`

- Añadir el método `getJSON` a la clase `Movie` que os proporcionamos para que devuelva un `String` con el objeto Java en formato JSON (en una sola línea). A esto se le conoce como **serialización**.
- Añadir un nuevo método `getPrettyJSON` a la clase `Movie` que devuelva un `String` con el objeto Java en formato JSON, pero esta vez formateado en "impresión bonita".
- Añadir un nuevo método estático `getObject` en la clase `Movie` que devuelva un objeto tipo `Movie` a partir del String en formato JSON recibido como argumento. A esto se le conoce como **deserialización**.

Pista: debes consultar por Internet cómo utilizar la librería JSON escogida (GSON en nuestro caso) para convertir el objeto Java en un `String` JSON.

Requisito mínimo para evaluar este ejercicio: el programa debe pasar todos los test proporcionados en la clase `MovieTest`.

(1 punto: 1 punto test `MovieTest`)

Formato y fecha de entrega

Tienes que entregar un fichero *.zip, cuyo nombre tiene que seguir este patrón: `loginUOC_PEC3.zip`. Por ejemplo: `dgarciaso_PEC3.zip`. Este fichero comprimido tiene que incluir los siguientes elementos:

- El proyecto de IntelliJ `PAC3Ex1` completado siguiendo las peticiones y especificaciones del Ejercicio 1.
- El proyecto de IntelliJ `PAC3Ex2` completado siguiendo las peticiones y especificaciones del Ejercicio 2.
- El proyecto de IntelliJ `PAC3Ex3` completado siguiendo las peticiones y especificaciones del Ejercicio 3.
- El proyecto de IntelliJ `PAC3Ex4` completado siguiendo las peticiones y especificaciones del Ejercicio 4.

El último día para entregar esta PEC es el **28 de noviembre de 2022** antes de las 23:59. Cualquier PEC entregada más tarde será considerada como no presentada.