UT04 Práctica Video en Streaming

Vamos a construir la estructura de objetos necesaria para implementar un pequeño framework que represente la estructura de datos para soportar un sistema de video en streaming. En concreto, el sistema estará preparado para reproducir tanto películas como series, las cuales estarán clasificadas en categorías para facilitar su búsqueda.

A continuación, se detalla los objetos que debemos implementar junto con su funcionalidad. Ten en cuenta que deberás implementar también la estructura de objetos necesaria para gestionar las excepciones que genere la aplicación.

Para el desarrollo de la práctica y su superación deberás utilizar la herramienta de gestión de versiones **GIT** e ir guardando una copia de la versión realizada en **GitHub** de forma obligatoria. La práctica podrá considerarse no válida si no se ha utilizado dichas herramientas.

1. Listado de objetos

El siguiente listado son objetos de entidad, es decir, objetos planos que tan solo tienes propiedades que almacenar, pero no tienen relación con otros objetos. Para cada objeto deberás implementar las propiedades getter y setter correspondientes y añadir los métodos que consideres útiles, en principio será obligatorio un toString.

1.1. Objeto Person

Objeto para identificar los datos de una persona. Sus propiedades y métodos serán:

Propiedad	Tipo	Obligatorio	Descripción
name	String	Si	Nombre de la persona.
lastname1	String	Si	Primer apellido de la persona.
lastname2	String	No	Segundo apellido de la persona.
born	Date	Si	Fecha de nacimiento.
picture	String	No	String con la ruta donde está ubicada la imagen.

Tabla 1 Descripción del objeto Person

1.2. Objeto Category

Con este objeto podemos crear la estructura de categorías. Sus propiedades y métodos serán:

Propiedad	Tipo	Obligatorio	Descripción
name	String	Si	Nombre de la categoría.
description	String	No	Descripción de la categoría.

Tabla 2 Descripción del objeto Category

1.3. Objeto Resource

Representa un recurso audiovisual.

Propiedad	Tipo	Obligatorio	Descripción
duration	Number	Si	Nº de minutos de la película.
link	String	Si	Ruta donde se ubica el recurso.

Tabla 3 Descripción del objeto Resource

1.4. Objeto Production

Este será un evento genérico para que tanto los siguientes objetos, Movie y Serie, hereden de él, y así podamos tener una estructura homogénea a la hora de buscar recursos en el sistema. Este objeto debe ser **implementado de forma abstracta** para que no se pueda instanciar objetos de la misma, ya que solo se utiliza para la herencia.

Propiedad	Tipo	Obligatorio	Descripción
title	String	Si	Título de la producción.
nationality	String	No	Define la nacionalidad de la producción.
publication	Date	Si	Fecha de publicación de la producción.
synopsis	String	No	Resumen del contenido de la producción.
image	String	No	String con la ruta donde está ubicada la imagen.

Tabla 4 Descripción del objeto Production

1.5. Objeto Movie

Representa un recurso película que podremos reproducir en el sistema. Hereda de Producción.

Propiedad	Tipo	Obligatorio	Descripción
resource	Resource	No	Recurso con el contenido de la película.
locations	[Coordinate]	No	Array con diferentes ubicaciones donde transcurre la película.

Tabla 5 Descripción del objeto Movie

1.6. Objeto Serie

Representa un recurso serie que podremos reproducir. Hereda de Production.

Propiedad	Tipo	Obligatorio	Descripción
resources	[Resource]	No	Array de Recursos con el contenido de los episodios.
locations	[Coordinate]	No	Array con diferentes ubicaciones donde transcurre la película.
seasons	Number	No	Número de temporadas.

Tabla 6 Descripción del objeto Serie

1.7. Objeto User

Representa un usuario del sistema.

Propiedad	Tipo	Obligatorio	Descripción
username	String	si	Nombre del usuario
email	String	Si	Correo electrónico del usuario.
password	String	Si	Contraseña del usuario.

Tabla 7 Descripción del usuario del sistema

1.8. Objeto Coordinate

Son coordenadas para localizar una ubicación.

Propiedad	Tipo	Obligatorio	Descripción
latitude	Number	si	Latitud de la ubicación.
longitude	Number	si	Longitud de la ubicación.

Tabla 8 Descripción de Coordinate.

2. Sistema de reproducción

En un objeto **VideoSystem** vamos a mantener el estado del propio sistema, donde vamos a relacionar todos los objetos anteriores.

La información que debe mantener es:

- Nombre del sistema.
- Los usuarios que tienen acceso al sistema.
- Listado de producciones que tenemos en el sistema.
- Las categorías de las producciones.
- Actores y actrices que tenemos registrados.
- Directores que tenemos en el sistema.

Los métodos a implementar para este objeto son:

Tabla 9 Relación de métodos del objeto VideoSystem

Método	Funcionalidad	Argumentos	Retorno	Excepciones
Getter/Setter name	Devuelve el nombre del sistema	String	String	El nombre no puede ser vacío
Getter categories	Devuelve un iterador que permite recorrer las categorías del sistema	-	Iterador de categorías	
addCategory	Añade una nueva categoría	Objeto Category	Number con el nº de elementos	- La categoría no puede ser null o no es un objeto Category. - La categoría ya existe
removeCategory	Elimina una categoría. Al eliminar la categoría, sus productos pasan a la de por defecto.	Objeto Category	Number con el nº de elementos	- La categoría no está registrada
Getter users	Devuelve un iterador que permite recorrer los usuarios del sistema	-	Iterador de users	
addUser	Añade un nuevo usuario	Objeto User	Number con el nº de elementos	- El usuario no puede ser null o no es un objeto User. - El username ya existe. - El email ya existe
removeUser	Elimina un usuario del sistema	Objeto User	Number con el nº de elementos	- El usuario no puede ser null o no es un objeto User. - El usuario no existe en el sistema.
Getter productions	Devuelve un iterador que permite recorrer las producciones del sistema	-	Iterador de producciones	
addProduction	Añade una nueva producción	Objeto Production	Number con el nº de elementos.	- La producción no puede ser null o no es un objeto Production. - La producción ya existe.
removeProduction	Elimina una producción del sistema.	Objeto Production	Number con el nº de elementos	- La producción no puede ser null o no es un objeto Production. - La producción no está registrada
Getter actors	Devuelve un iterador que permite recorrer los actores registrados en el sistema	-	Iterador de actores	

addActor	Añade un nuevo actor	Objeto Person	Number con el nº de elementos	- El actor no puede ser null o no es un objeto Person. - El actor ya existe.
removeActor	Elimina un actor del sistema	Objeto Person	Number con el nº de elementos	- El actor no puede ser null o no es un objeto Person. - El actor no existe en el sistema.
Getter directors	Devuelve un iterador que permite recorrer los directores registrados en el sistema	-	Iterador de directores	
addDirector	Añade un nuevo director.	Objeto Person	Number con el nº de elementos	- El director no puede ser null o no es un objeto Person. - El director ya existe.
removeDirector	Elimina un director del sistema	Objeto Person	Number con el nº de elementos	- El director no puede ser null o no es un objeto Person. - El director no existe en el sistema.
assignCategory	Asigna uno más producciones a una categoría. Si el objeto Category o Production no existen se añaden al sistema.	-Category -Production	Number con el nº total de producciones asignadas a la categoría.	- Category es null - Production es null.
deassignCategory	Desasigna una o más producciones de una categoría.	-Category -Production	Number con el nº total de producciones asignadas a la categoría.	- Category es null - Production es null.
assignDirector	Asigna uno más producciones a un director. Si el director o el objeto Production no existen se añaden al sistema.	-Person -Production	Number con el nº total de producciones asignadas al director.	- Person es null - Production es null.
deassignDirector	Desasigna una o más producciones de un director.	-Person -Production	Number con el nº total de producciones asignadas al director.	- Person es null - Production es null.
assignActor	Asigna uno más producciones a un actor. Si el actor o el objeto Production no existen se añaden al sistema.	-Person -Production	Number con el nº total de producciones asignadas al director.	- Person es null - Production es null.

deassignActor	Desasigna una o más producciones de un actor.	-Person -Production	Number con el nº total de producciones asignadas al director.	- Person es null - Production es null.
getCast	Obtiene un iterador con la relación de los actores del reparto una producción y sus personajes.	- Production	iterador	- Production es null
getProductionsDirector	Obtiene un iterador con las producciones de un director.	- Person	iterador	- Person es null
getProductionsActor	Obtiene un iterador con las producciones de un actor y su papel en la producción.	- Person	iterador	- Person es null
getProductionsCategory	Obtiene un iterador con las producciones de una categoría determinada.	- Category	iterador	- Category es null

3. Relación entre objetos

Las bases de datos **NOSQL** son aquellas que utilizan almacenes de objetos para contener la información, en lugar de utilizar relaciones entre tablas como ocurre con las relacionales. Un ejemplo es **MongoDB** la cual almacena documento en un formato parecido a **JSON**. Como veremos en próximas unidades, un documento **JSON** es un *objeto literal* traducido a *string* para que pueda ser portable. Estos documentos son fácilmente interpretables por un humano, por lo que no necesitan ser procesados. Por último, pueden ser transferibles a través de red para comunicar componentes de una aplicación, o de forma más habitual, entre un cliente y servidor.

Un ejemplo de documento JSON podría ser utilizado para representar un usuario. El siguiente ejemplo vemos cómo podemos utilizar esta representación para un usuario con una propiedad *_id*, *name*, *contact*, y *dob* con la fecha de nacimiento.

```
{
    "_id": "52ffc33cd85242f436000001",
    "name": "Tom Hanks",
    "contact": "987654321",
    "dob": "01-01-1991"
 }
```

Otro ejemplo podría ser utilizado para representar una dirección.

```
{
    "_id": "52ffc4a5d85242602e000000",
    "building": "22 A, Indiana Apt",
    "pincode": 123456,
    "city": "Los Angeles",
    "state": "California"
}
```

Como vemos, ambos siguen una estructura similar a la de un objeto literal. Ambos, JSON y los objetos literales son formatos intercambiables.

MongoDB utiliza los dos modelos de relación de objetos citados. Veamos unos ejemplos.

3.1. Modelo embebido o de relación embebida

En este modelo, los objetos subordinados están embebidos dentro del objeto principal. Un ejemplo es el siguiente, donde tenemos un usuario y un array con todas sus posibles direcciones.

```
{
    "_id": "52ffc33cd85242f436000001",
    "contact": "987654321",
    "dob": "01-01-1991",
    "name": "Tom Benzamin",
    "address": [
       {
          "building": "22 A, Indiana Apt",
          "pincode": 123456,
          "city": "Los Angeles",
          "state": "California"
       },
```

```
        {
            "building": "170 A, Acropolis Apt",
            "pincode": 456789,
            "city": "Chicago",
            "state": "Illinois"
        }
    ]
}
```

Esto nos crea la **ventaja** de que si queremos acceder al contenido se encuentra disponible inmediatamente. No tenemos que hacer ningún paso extra para recuperarlo. El **inconveniente** viene en el mantenimiento de los datos y su repetición. Dos usuarios podrían tener la misma dirección, si queremos modificar un dato de la dirección tendríamos que ir objeto tras objeto comprobando si es la dirección que estamos buscando para actualizar el dato.

3.2. Modelo relación por referencia

En este caso los objetos subordinados residen en otro documento o estructura, y son referenciados mediante una propiedad con forma de identificador. En este ejemplo, las direcciones son referenciadas por un identificador.

```
{
    "_id":"52ffc33cd85242f436000001",
    "contact": "987654321",
    "dob": "01-01-1991",
    "name": "Tom Benzamin",
    "address_ids": [
        "52ffc4a5d85242602e000000",
        "52ffc4a5d85242602e000001"
    ]
}
```

La **ventaja** de este modelo es el mantenimiento, ya que los datos están centralizados, pero el **inconveniente** es que tendremos que realizar varios pasos para encontrar la información, el usuario, y una vez encontrado, habría que buscar todas las direcciones que le pueden pertenecer a través de los *id* aportados por el objeto del usuario.

Nunca hay soluciones perfectas, todo depende de las necesidades de nuestra aplicación, y qué beneficia más a nuestra lógica de negocio.

4. Consejos de implementación

Para la implementación de la práctica necesitaremos diseñar una estructura que nos permita mantener todos los objetos que vayamos creando, tiendas, categorías y productos.

Como propuesta de implementación, aunque no tiene que ser la única, y otros tipos también podrían considerarse válidos, tenemos una solución mixta con los dos modelos.

Solución 1

- Crear un array con las tiendas. El cuál almacenará cada uno de los objetos Store.

```
let _stores = [];
```

- Crear un array con las categorías.

```
let _categories = [];
```

- Cada elemento en la categoría puede ser un objeto literal con dos propiedades, una con la categoría, y la otra un array que contenga los productos asignados a esa categoría.

```
{
    category: category,
    products:[]
}
```

- Los objetos Product en el array de productos pueden ser un objeto literal con el objeto Product y una propiedad que *referencie* la tienda en la que está ubicado.

```
{
    product: product,
    store: _stores[storePosition].CIF
}
```

Solución 2

La solución dos es hacer el objeto Store el principal:

- Partimos de los dos arrays.

```
let _stores = [];
let _categories = [];
```

- Las categorías son objetos independientes, sin embargo, las tiendas deben ser objetos literales para contener el array de productos.

```
{
    store: store,
    products:[]
}
```

- Por cada producto añadido debemos crear un objeto literal indicando el objeto Product y referenciando el título de la categoría asignada.

```
{
    product: product,
    category: _categories [categoryPosition].title
}
```

Esta aproximación nos podría dar la ventaja de transformar la estructura para que un producto pudiese pertenecer a varias categorías utilizando un array, pudiendo guardar en el array más de una referencia de categoría a través de su título.

```
{
    product: product,
    categories: []
}
```

Nota

Implementa una función de testeo de toda la funcionalidad implementada en la aplicación a través de la consola. **Esta funcionalidad es imprescindible para corregir la práctica. Si la función no está implementada la nota final será de 0.**

Podrás realizar cualquier cambio en la funcionalidad propuesta siempre y cuando esté justificada y mejore la funcionalidad propuesta.

Si consideras el diseño de los objetos, excepciones o argumentos de entrada y salida de los diferentes métodos también lo pueden realizar, siempre y cuando estén justificados.

Calificación

Criterio de evaluación	Puntos
Implementación de la aplicación. Verificación de funcionamiento.	3 puntos
Gestión de excepciones	1 punto
Eficiencia / Seguridad. Verifica el uso de funciones estándar y verifica los ámbitos de acceso a los objetos de tu aplicación.	1 punto
Estructura OO. Verificación del código siga el paradigma de orientación a objetos.	1 punto
Comentarios. Deberás comentar el código que has implementado.	1 punto
Uso de patrones de diseño y características avanzadas de objetos - Transformación de los arrays por iteradores. (1,5 puntos) - Uso del patrón Singleton (1 puntos) - Empaquetado en módulos (0,5 puntos)	3 puntos

Tabla 10 Calificación