Ejercicios de Introducción al lenguaje Java

Índice

1. Uso de interfaces (1 punto)

Tenemos una aplicación para gestionar nuestra colección de DVDs, en la que podemos añadir películas a la colección, eliminarlas, o consultar la lista de todas las películas que poseemos. En esta aplicación tenemos una clase `FilePeliculaDAO` que nos ofrece los siguientes métodos:

```
public void addPelicula(PeliculaTO p);
public void delPelicula(int idPelicula);
public List<PeliculaTO> getAllPeliculas();
```

Esta clase nos permitirá acceder a los datos de nuestra colección almacenados en un fichero en disco. Siempre que en algún lugar de la aplicación se haga un acceso a los datos se utilizará esta clase. Por ejemplo, si introducimos los datos de una nueva película en un formulario y pulsamos el botón para añadir la película, se invocaría un código como el siguiente:

```
FilePeliculaDAO fpdao = GestorDAO.getPeliculaDAO();
fpdao.addPelicula(pelicula);
```

La clase auxiliar `GestorDAO` tiene un método estático que nos permitirá obtener una instancia de los objetos de acceso a datos desde cualquier lugar de nuestro código. En el caso anterior este método sería algo así como:

```
public static FilePeliculaDAO getPeliculaDAO() {
  return new FilePeliculaDAO();
}
```

Como la aplicación crece de tamaño decidimos pasar a almacenar los datos en una BD en lugar de hacerlo en un fichero. Para ello creamos una nueva clase `JDBCPeliculaDAO`, que deberá ofrecer las mismas operaciones que la clase anterior. ¿Qué cambios tendremos que hacer en nuestro código para que nuestra aplicación pase a almacenar los datos en una BD? (Imaginemos que estamos accediendo a `FilePeliculaDAO` desde 20 puntos distintos de nuestra aplicación).

En una segunda versión de nuestra aplicación tenemos definida una interfaz `IPeliculaDAO` con los mismos métodos que comentados anteriormente. Tendremos también la clase `FilePeliculaDAO` que en este caso implementa la interfaz `IPeliculaDAO`. En este caso el acceso a los datos desde el código de nuestra aplicación se hace de la siguiente forma:

```
IPeliculaDAO pdao = GestorDAO.getPeliculaDAO();
pdao.addPelicula(pelicula);
```

El `GestorDAO` ahora será como se muestra a continuación:

```
public static IPeliculaDAO getPeliculaDAO() {
  return new FilePeliculaDAO();
}
```

¿Qué cambios tendremos que realizar en este segundo caso para pasar a utilizar una BD? Por lo tanto, ¿qué versión consideras más adecuada?

2. Refactorización (1 punto)

En las plantillas de la sesión podemos encontrar un proyecto `lja-filmoteca` en el que tenemos implementada la primera versión de la aplicación anterior. Realiza una refactorización del código para facilitar los cambios en el acceso a datos (deberá quedar como la segunda versión comentada en el ejercicio anterior).

Ayuda

Resultará útil el menú *Refactor* de Eclipse. En él podemos encontrar opciones que nos permitan hacer los cambios necesarios de forma automática.

Una vez hechos los cambios añadir el nuevo DAO `JDBCPeliculaDAO` y probar a cambiar de un DAO a otro (si se ha hecho bien sólo hará falta modificar una línea de código). No hará falta implementar las operaciones de los DAO. Bastará con imprimir mensajes en la consola que nos digan lo que estaría haciendo cada operación.

3. Documentación (0.5 puntos)

El proyecto anterior tiene una serie de anotaciones en los comentarios que nos permiten generar documentación Javadoc de forma automática desde Eclipse. Observa las anotaciones puestas en los comentarios, las marcas `@param`, `@return`, `@deprecated`...

Genera la documentación de este proyecto (menú *Project* > *Generate Javadoc*...) en una subcarpeta `doc` dentro del proyecto actual. Eclipse nos preguntará si queremos establecer este directorio como el directorio de documentación de nuestro proyecto, a lo que responderemos afirmativamente.

Añade comentarios Javadoc a las nuevas clases creadas en el ejercicio anterior y genera nuevamente la documentación. Prueba ahora a escribir código que utilice alguna de las clases de nuestro proyecto, usando la opción de autocompletar de Eclipse. Veremos que junto a cada miembro de la clase nos aparecerá su documentación.

4. Centro cultural (1 punto)

Un centro cultural se dedica al préstamo de dos tipos de materiales de préstamo: discos y libros. Para los dos se guarda información general, como su código identificativo, el título y el autor. En el caso de los libros, almacenamos también su número de páginas y un capítulo de muestra, y para los discos el nombre de la discográfica.

También podemos encontrar una serie de documentos con las normas e información sobre el centro de los que guardamos su título, fecha de publicación y texto. Tanto estos documentos como los libros se podrán imprimir (en el caso de los libros se imprimirá el título, los autores, y el capítulo de muestra, mientras que de los documentos se imprimirá su título, su fecha de publicación y su texto).

Escribe la estructura de clases que consideres adecuada para representar esta información en un nuevo proyecto `lja-centrocultural`. Utiliza las facilidades que ofrece Eclipse para generar de forma automática los constructores y *getters* y *setters* necesarios.

5. Copia de propiedades con BeanUtils (0.5 puntos)

En las plantillas de la sesión contamos con un proyecto llamado `lja-copyproperties` en el que hay una clase principal que crea dos objetos: uno de clase `PeliculaTO` y otro de clase `Mpeg4fileTO`. Se trata de dos transfer objects muy similares que comparten varios campos.

Copia todos los campos que puedas desde `PeliculaTO` hacia `Mpeg4fileTO`, sobreescribiendo los campos que se pueda y dejando como están los campos que no coincidan.

Ahora utiliza el método `BeanUtils.copyProperties(dest,orig)` para hacer lo mismo en una sola línea. Tendrás que incluir las bibliotecas correspondientes, que se encuentran en la carpeta `lib`.

Observa el resultado para ver si se han copiado bien todos los campos que esperabas. ¿Encuentras algún fallo en la copia de la fecha de estreno? Corrígelo y comprueba que se copia correctamente.