Tarea Sesión 3

Objetivo

Esta tarea está compuesta por dos ejercicios. El primero consiste en ampliar el proyecto DOME comenzado en la sesión 3. El segundo consiste en resolver un problema de estructura equivalente al DOME pero dentro de otro dominio: la gestión de posts de una red social.

En ambos casos se trabajará el **concepto de herencia**, a través de la creación de una jerarquía de clases y el **polimorfismo**, a través del uso de una clase que maneja todos los objetos de la jerarquía mediante métodos y variables polimórficas.

1 Instrucciones

- Esta tarea deberá ser entregada **24 horas antes** de la siguiente clase de laboratorio.
- Antes de entregar, utiliza **la *checkList*** para revisar los proyectos. No es necesario entregarla.
- Añade también el **UML** realizado en cada proyecto. Incluye una imagen (jpg) con el mismo dentro del directorio raíz del proyecto.
- **Renombra los proyectos dentro de Eclipse** (usando `Refactor-> Rename`) con:
  `apellido_1_apellido2_nombre_lab03_task_dome`
  `apellido1_apellido2_nombre_lab03_task_post` en minúsculas y sin tildes.
- **Exporta los proyectos juntos** a un fichero comprimido en formato zip (botón derecho `enviar a/carpeta comprimida`). Súbelo al Campus Virtual.

Para cualquier ejercicio, asegúrate de que siempre cumplas con las siguientes reglas:

1. Plantea el diseño UML (o amplía el que ya tienes)
2. Antes de implementar un método o constructor, implementa las pruebas
3. **Añade comentarios javadoc** para todos los métodos públicos y constructores. Genera javadoc.
4. **Valida** argumentos (parámetros) en todos los métodos públicos y constructores, usando solo `ArgumentChecks`. A menos que se diga otra cosa, `null` será considerado argumento inválido, así como cadenas vacías o cadenas de blancos.
5. En la fase de **actualización** de un proyecto, comprueba que los métodos existentes siguen funcionando correctamente. Por ejemplo, cuando se añade `videogames`, el método `print` de `MediaLibrary` también imprime los datos de estos objetos en consola. **Revisa los test que ya existen** pues pueden necesitar incluir nuevos aspectos para tratar los cambios realizados.

2 Proyecto DOME

A partir del proyecto dome realizado en la sesión 3 `(lab03)`, realiza las siguientes ampliaciones

2.1 Pruebas

1. Añade o finaliza las pruebas de la clase `MediaLibrary`, para los métodos que se han implementado en clase, **salvo el método** `list`. Crea `getItems()` para devolver una **copia de la lista** de ítems.

2.2 Nuevos métodos públicos en Medialibrary

2. `Item searchItem(Item theItem)` Busca en la librería el ítem recibido como parámetro (el mismo ítem, no otro ítem con el mismo contenido). Devuelve el ítem encontrado en la librería, si lo encuentra, o bien null si no lo ha encontrado o el parámetro es null. **Comienza realizando las pruebas** unitarias (buscando `cds` y `dvds`), a continuación, implementa el código del método. Utiliza `getItems` y el método `contains` de ArrayList para realizar las pruebas.

3. `String getResponsables()`. Devuelve un `String` (posiblemente vacío si no hay datos) que contenga las personas responsables de todos los ítems existentes, **separadas por comas**. En el caso de los `CDs`, el responsable es el artista. En el caso de los `DVDs` el responsable es el director.

   Comienza realizando las pruebas de este método, a continuación, implementa el código delegando la operación en cada tipo de objeto, lo que permite aplicar el polimorfismo. Utiliza StringBuilder para crear la cadena de forma más eficiente.

2.3 Nuevo tipo de item VideoGame

Incluye en la aplicación la posibilidad de guardar también videojuegos, además de `CDs` y `DVDs`. Analiza cómo se añade esta información al diseño UML.

Un videojuego `(Videogame)` tiene las siguientes características:

- Un título
- Un autor (que será el responsable).
- No se guarda la duración. *(Ojo!, no tendrá la propiedad playingTime, por lo que este atributo deja de ser común a todos, habrá que sacarlo de nuevo a las subclases que lo tienen)*
- Propietario
- Comentario
- Número de jugadores
- Una plataforma que podrá ser `XBOX, PLAYSTATION o NINTENDO`. (Nota: deberás crear un tipo enumerado `Platform` con los tres valores).
- Un método `print(PrintStream out)` para imprimir sus características en consola.

No es necesario que implementes las pruebas para la clase VideoGame.

2.4 Revisa clases previas y tests

1. Modifica las clases de prueba de los métodos de `MediaLibrary`, para contemplar objetos `Videogame`.
2. Añade las pruebas del método `getNumberOfItemsOwned()` si no lo has realizado. Deberá contemplar también objetos `Videogame`.
3. **Añade código al método run** de la clase `MediaPlayer`. Aquí se deberán crear al menos 1 objeto de cada clase (`CD`, `DVD`, `Videogame`) y un objeto de la clase `MediaLibrary` donde se añadan los ítems. Muestra el resultado de la ejecución de los métodos `getNumberOfItemsOwned, getResponsables` y `list`. Al ejecutar `list`, pasa como objeto `out` la pantalla `(System.out)`.
4. **Comentarios.** Revisa que las clases, los constructores y los métodos tengan comentarios *javadoc*. Genera la documentación para el proyecto `(project/generate javadoc)`.
5. **Validación de argumentos (o parámetros)**. Revisa constructores y métodos públicos para asegurar que tengan todos validación de parámetros, lanzando una excepción

`IllegalArgumentException` caso erróneo. En el caso de cadenas, comprueba que la cadena existe y también que no esté formada únicamente por espacios en blanco o es cadena vacía. Para campos numéricos comprueba que es mayor que 0. Utiliza el proyecto `util` para validar. Realiza las pruebas de robustez para comprobar que se validan los argumentos en constructores y métodos.

3 Proyecto Red Social

Se pretende desarrollar una aplicación para gestionar una red social que permitirá a los usuarios publicar información (posts) de dos tipos: mensajes y fotos. La red social tendrá en su última versión múltiples operaciones, pero, para esta primera versión se centra exclusivamente en **enviar posts de dos tipos**, **recuperar todos los posts de un usuario** y **obtener todos los posts de la red**.

3.1 Modelo de datos para almacenar posts

3.1.1 Diagrama UML

Antes de realizar la implementación realiza el diseño UML de las clases que modelan los posts. Ten en cuenta el concepto de herencia para establecer el diseño. Incluye atributos, constructor y métodos públicos en las clases. Establece las relaciones entre las mismas.

Puedes usar la herramienta Visual Paradigm (Community edition 16.3) para realizar el diseño, si lo realizas en papel haz una foto.

3.1.2 Definición

Los usuarios tienen la posibilidad de publicar dos tipos de posts:

- **TextMessage**. Un mensaje de texto consiste en una cadena de caracteres en la que se almacena el mensaje.
- **Image**. Publicar una foto implica guardar el **nombre del fichero (filename)** donde se almacena la foto y un **pie de foto (caption)**.

Además, todos los posts, llevan asociado el identificador del **usuario (user)** que lo ha creado. También guardarán información de los **likes (like)** que han recibido y cada post guarda una lista con los comentarios asociados al mismo. Un **comentario (comment)** será almacenado como una cadena.

3.1.3 Constructor

Cuando se crea un post, se reciben los datos como parámetros, salvo los *likes* que se inicializan a 0 y la lista de comentarios que se crea vacía. Así la creación de un mensaje recibe el usuario y el mensaje, y la creación de una foto recibe el usuario, el nombre del fichero y el pie de foto. Al crear cualquiera de ellos se lanza una excepción `IllegalArgumentException` si el nombre de usuario, del fichero o el mensaje es `null` o cadena vacía.

3.1.4 Métodos públicos

- Deberá existir un método público `String toString()` que devuelva de manera adecuada para el usuario una cadena describiéndose a sí mismo con el siguiente formato:
  - TEXT MESSAGE Posted by: nombre de usuario, Content: el mensaje
  - PHOTO Posted by: nombre de usuario, File: el nombre de fichero, Caption: el pie de foto
- Además, se dejarán públicos los métodos para modificación de likes y comentarios.

3.1.5 Implementación del modelo de datos

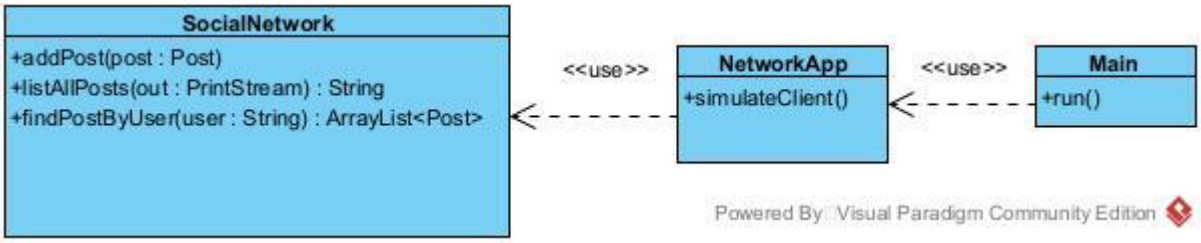
Una vez realizado el diseño UML ya puedes hacer la implementación. En este caso no es necesario que realices las pruebas de las clases del modelo para reducir la carga de trabajo, aunque en un diseño ideal si deberían ser realizadas.

3.2 Clase servicio SocialNetwork

Esta clase debe almacenar y gestionar todos los posts de la red. Guardará por tanto los posts en una colección.

3.2.1 Diagrama UML

Completa el siguiente diagrama de clases UML. Diseña las clases que consideres para el modelo de datos. Organiza la información común y establece una jerarquía de herencia. Al final del ejercicio exporta este diagrama en formato imagen (.jpg) y cópialo en la carpeta raíz del proyecto.



3.2.2 Pruebas

Escribe primero los test de funcionalidad y robustez para los métodos públicos de la clase `SocialNetwork` (salvo para el método listAllPosts).

Estas son algunas sugerencias para plantear los test:

- Utiliza el método `getPosts()` que devuelve todos los posts de la red social. Este método no estará en la interfaz pública de la clase `SocialNetwork`, pero resultará útil para la implementación de los tests.

- El método `void addPost(Post post)`. Añade un nuevo post de un usuario a la red. Cuando escribas los tests para `addPost`, comprueba no solo que la red contiene el post añadido, sino **también que los demás posts siguen en la lista**. Puede ayudarte el uso de `getPost()` y el método `contains()` de la clase `java.util.ArrayList` que retorna true si la lista contiene cierto elemento.

- Finalmente, para los tests de `ArrayList<Post> findPostsByUser(String user)`, puedes crear tu propio `ArrayList` con los posts esperados y usar métodos `contains` y `size` para comprobar si el esperado concuerda con el valor devuelto por `findPostsByUser`.

3.2.3 Interfaz pública de la clase SocialNetwork

Se podrán realizar las siguientes operaciones:

1. `void addPost(Post post)`. Añade el post a la red. Lanza una excepción `IllegalArgumentException` si al añadir un post se pasa `null`.

2. `void listAllPosts(PrintStream out)`. Escribe en el objeto `out` la información de todos los posts de la red social. La información de un post es la cadena que devuelve el método toString() del post. Cada cadena debe estar separada de la siguiente por el carácter de nueva línea ("\n"). Si no hay posts, devuelve la cadena vacía.

   Ejemplo de formato:

   ```
   PHOTO posted by: maria1999, File: gatos.jpg, Caption: Gatos divertidos
   TEXT MESSAGE Posted by: Descartes Content: Pienso, luego existo.
   ```

3. `ArrayList<Post> findPostsByUser(String userId)`. Devuelve la lista de todos los posts correspondientes al usuario recibido o lista vacía si no se pueden encontrar posts.

Revisa el diagrama UML y relaciona esta clase con las clases del modelo si no lo has hecho aún.

3.3 Otras clases

Crea una clase `NetworkApp` con un único método `public void simulateClient()`. Primero crea una instancia de SocialNetwork. Añade varias fotos y mensajes de dos usuarios diferentes, imprime todos los posts y finalmente busca e imprime todos posts asociados a un usuario. Piensa en qué paquete se crearía esta clase.

Finalmente, añade también una clase `Main` para ejecutar la clase `NetworkApp`.