Laboratorio S12
Jerarquía, herencia, programación incremental, polimorfismo, ...

Objetivos

- Trabajar los conceptos de herencia, programación incremental y polimorfismo en Java
- *Realizar diseño con StarUML (opcional)*
- *Realizar verificaciones con JUnit (opcional)*

Herramientas a utilizar

- Eclipse, Javadoc y opcionalmente *StarUML y JUnit*

Entregable

Hay que entregar un fichero que contenga el proyecto y la exportación del proyecto de Eclipse (código + documentación en carpeta doc) que contenga todo lo que se solicita en este laboratorio. Nomenclatura para los diferentes elementos.

- Nombre del proyecto Java: *apellido1_apellido2_nombre_S12*
- Nombre del fichero a entregar: *apellido1_apellido2_nombre*.zip
- La entrega es individual y se entregará a través de eGela
- **El último día para realizar la entrega es el 23 de abril antes de las 23:55**

Contexto del laboratorio

Queremos realizar una aplicación que gestione los músicos y actuaciones de una orquesta. El objetivo inicial del desarrollo es contratar a músicos, actuar interpretando cada componente su fragmento de una pieza, y conocer el coste total de las contrataciones.

Para ello, crearemos la clase **`Orchestra`** que definirán los atributos y operaciones adecuados para alcanzar los objetivos marcados tal y como se describirá más adelante,

Por otro lado, los componentes de la orquesta serán músicos. Aunque los músicos compartan muchas características, también tienen muchas diferencias. La clase **`Musician`** reunirá las características que tienen todos los músicos en común y a partir de ella se derivan otras clases tales como **`Director`** o **`Instrumentalist`**. Aprovecharemos la jerarquía para implementar de manera incremental las constructoras, el salario de los músicos y el método `toString` (este método es opcional).

Antes de empezar a realizar las tareas lee detenidamente todo el enunciado del laboratorio para que te ayude en todo el proceso de diseño e implementación.

Tareas a realizar en el laboratorio

Tarea 1. Implementa y documenta la aplicación, según se indica en el apartado de especificaciones.

a. La clase **`Musician`** y las clases que se deriven de la misma se definirán en el paquete **`packmusicians`**.

b. La clase **`Orchestra`** se debe definir dentro del paquete **`packorchestra`**

c. Una vez realizadas las tareas anteriores, crea una clase **`DemoOrchestra`** en el paquete **`packdemo`**. En el método main de esta clase se debe realizar lo siguiente:

   i. Crear al menos 7 músicos incluyendo: director o directora, pianista, dos sopranos (una de talla internacional y otra no), tenor, dos trompetistas. Mira la ejecución más abajo.

   ii. Crea la orquesta y contrata a los y las músicas que has creado. A dos hay que intentar contratarlos dos veces.

   iii. Haz que actúe la orquesta.

   iv. Muestra por pantalla el presupuesto de la orquesta

Ejemplo de ejecución (el director de esta orquesta tiene 5 años de antigüedad):

```
Robert Segovia pianist, hired for 1500.0 euros.
Chris Black tenor, hired for 1000.0 euros.
Edurne Berasaluze soprano, hired for 1000.0 euros.
Ines Barrutieta trumpeter, hired for 1100.0 euros.
Robert Segovia pianist is alredy hired.
Chris Black tenor is alredy hired.
Rafael Zurbano director, hired for 1500.0 euros.
Jone Kaperotxipi trumpeter, hired for 1100.0 euros.
Caroline Linecarol soprano, hired for 1330.0 euros.

This is the Jazzband Orkestra performance:
  [Rafael Zurbano]: Tok tok tok: (silence)
  [Robert Segovia, Piano]: Cliing-cliing-clang-cliing-claang-cli
  [Chris Black]: Lorolo-lorolo-looooo
  [Edurne Berasaluze]: Liliri-liliri-liiiiiiiii
  [Ines Barrutieta, Trumpet]: Tuturu-tuturu-tururu
  [Jone Kaperotxipi, Trumpet]: Tuturu-tuturu-tururu
  [Caroline Linecarol]: Liliri-liliri-liiiiiiiii

Spent onJazzband Orkestra participants' salaries: 8530.0 euro.
```

Tareas opcionales:

Tarea 2. Crea el UML de la jerarquía de clases de **`Musician`**.

Tarea 3. Verifica con JUnit los siguientes métodos:

a. Los métodos **`perform`**, **`hire`** y **`getSalary (`**Clases **`Pianist, Director y Soprano`**, si son necesarias varias)**.**

b. Clase **`Orchestra`**: método **`getFee`**

Especificaciones de las clases de la aplicación

A continuación, se describen todas las clases de las que constará la aplicación:

C1. **`Musician`** representa a los músicos en general. Los músicos se caracterizan por su nombre (`name`), salario (salary, double. 800 euros de salario por defecto) y si están o no contratados en alguna orquesta (`hired`). Además, debe proporcionar los siguientes métodos:

a. Una constructora que reciba por parámetro el nombre. Se establece que no estará contratado.

b. Getters y setters para los atributos nombre y salario.

c. **`isHire:`** Indicará (boolean) si el músico está o no contratado.

d. **`hire`**: método que simula la contratación de un músico. Si el músico no está contratado, el atributo contratado del músico pasará a ser true. Si el músico ya está contratado entonces se elevará la excepción **`AlreadyHiredMusician`**. Esta excepción se declarará en el paquete **`packexceptions`** y tendrá una constructora con un parámetro mensaje. Este método eleva la excepción indicando como mensaje el nombre del músico, el tipo de músico[1] y que ya está contratado. Por ejemplo, el String del mensaje podría ser cualquiera de los siguientes:

```
Robert Segovia pianist is alredy hired.
Chris Black tenor is alredy hired.
```

e. **`perform`**: método que devuelve un `String` que representa la actuación del músico. El método **`perform`** hará que el músico actúe.

f. Se debe sobrescribir el método **`equals`**.

g. (**opcional**) Se debe sobrescribir el método **`toString`** que mostrará la clase[2] a la que pertenece el músico actual junto con la información de todos los atributos de la clase. Este método se debe sobrescribir de manera incremental en todas las clases necesarias de la jerarquía. Ver la ejecución al final en el ejemplo del método **`toString`** de **`Orchestra`**.

C2. **`Director`**: Para los músicos que dirigen orquestas queremos guardar además información de los años que lleva dirigiendo. El salario de los directores será de 400 euros superior al salario habitual. Además, se aplica un incremento a su salario de un 5% por cada año de antigüedad. Su actuación consistirá en: "Tok tok tok: (silence)". Deberá incluir el getter y setter de su antigüedad.

C3. **`Singer`** y subclases: **`Singer`** representará a los músicos que son cantantes. De manera general, los cantantes cobrarán 200 euros más al salario habitual. Sin embargo, se distinguen entre **soprano** y **tenor**. Las Sopranos cobrarán un 33% más si son de talla internacional[3]. Los tenores cantan "Lorolo-lorolo-looooo" mientras que todas las sopranos cantan "Liliri-liliri-liiiiiiiii".

C4. La clase **`Instrumentalist`**: Todos los músicos que tocan algún instrumento en la orquesta quedan representados por este tipo especial de músico. Para esta clase se debe añadir información sobre el instrumento que toca (`instrument`). Se debe añadir un getter para saber cuál es el instrumento que toca. Por ejemplo, un violinista tocará el violín y su sueldo es el habitual de los músicos.

C5. **`Pianist`**: Esta clase representa a los pianistas. Su sueldo es de 700 euros más que el salario habitual y al interpretar produce el sonido "Cliing-cliing-clang-cliing-claang-cliing".

[1] `this.getClass().getSimpleName().toLowerCase()`
[2] `this.getClass().getSimpleName()`
[3] Será necesario añadir un atributo en Soprano que nos permita saber si es o no de talla internacional. No es necesario crear una clase para los sopranos internacionales y otra para los que no lo son. Basta con la información del atributo mencionado.

C6. **`Trumpeter`**: Esta clase representa a los trompetistas. Su sueldo es de 300 euros más que el salario habitual y al interpretar produce el sonido "Tuturu-tuturu-tururu".

C7. **`Orchestra`**: Esta clase representa la orquesta y se caracteriza por un nombre y sus músicos (un ArrayList). Esta clase debe además proporcionar la siguiente funcionalidad:

a. **`getName`**: Devuelve un String con el nombre de la orquesta

b. **`act`**: Escribe un mensaje indicando el nombre de la orquesta que actúa y a continuación, hace que suene la orquesta (escribiendo por pantalla cada actuación). Primero, debe actuar el Director y posteriormente cada componente de la orquesta. En todos los casos se debe indicar entre [ ], el nombre del músico y si es instrumentista además, deberé indicar cuál el instrumento que toca. Deberá completar la información con la música que genera. Para ello se debe utilizar el método polimórfico **`perform`** de la clase **`Musician`**. Ejemplo de ejecución:

```
This is the Jazzband Orkestra performance:
  [Rafael Zurbano]: Tok tok tok: (silence)
  [Robert Segovia, Piano]: Cliing-cliing-clang-cliing-claang-cli
  [Chris Black]: Lorolo-lorolo-looooo
  [Edurne Berasaluze]: Liliri-liliri-liiiiiiiii
  [Ines Barrutieta, Trumpet]: Tuturu-tuturu-tururu
  [Jone Kaperotxipi, Trumpet]: Tuturu-tuturu-tururu
  [Caroline Linecarol]: Liliri-liliri-liiiiiiiii
```

c. **`hire`**: Dado un músico, se realiza el proceso de su contratación en la orquesta. Si el músico no está ya contratado, se contratará y se añadirá a la lista de músicos de la orquesta; además, debe devolver un String indicando cuánto cobrará el músico. Si el músico ya estaba contratado con anterioridad, se tratará la excepción **`AlreadyHiredMusician`** mostrando un mensaje adecuado en pantalla. Un ejemplo de funcionamiento correcto sería la obtención del String: "`Robert Segovia pianist, hired for 1500.0 euros.`".

d. **`getFee`**: Calcula la suma de los sueldos de todos los componentes de la orquesta.

e. (**opcional**) **`toString`**: Obtiene un `String` con la información de todos los componentes de la orquesta (sobreescribe). Si implementas los métodos **`toString`** incluye en el main de la demo que se imprima el valor del `String` devuelto. Por ejemplo, escribir el `String` podría visualizarse como sigue:

```
***********  ORCHESTRA  ***************
There are the Jazzband OrkestraOrchestra participants:
[Directorname=Rafael Zurbano, salary=800.0, hired=true, antiquity=5
, Pianist name=Robert Segovia, salary=800.0, hired=true, instrument=Piano
, Tenor name=Chris Black, salary=800.0, hired=true
, Soprano name=Edurne Berasaluze, salary=800.0, hired=true international=false
, Trumpeter name=Ines Barrutieta, salary=800.0, hired=true, instrument=Trumpet
, Trumpeter name=Jone Kaperotxipi, salary=800.0, hired=true, instrument=Trumpet
, Soprano name=Caroline Linecarol, salary=800.0, hired=true international=true
]
```