Proyecto: Pub

Objetivos

El objetivo principal de este proyecto es poner en práctica de forma autónoma los conceptos trabajados durante el curso. Este proyecto plantea realizar una aplicación para gestionar un pub. Las entidades a gestionar serán las bebidas que ofrecen al público, los alimentos para picar que tienen y el maridaje entre ambos, implementando las clases y funcionalidades que se describen en cada parte del proyecto.

Entregables

El proyecto tendrá una única entrega, siendo la fecha límite de entrega el **domingo 22 de mayo a las 23:55**. La entrega debe ser **individual** y **a través de eGela** y consistirá en un fichero comprimido cuyo nombre debe ser **`apellido1Apellido2_Nombre_Proyecto.zip`**. Este fichero comprimido debe contener:

- El diseño completo de la aplicación utilizando StarUML:
  - Imagen UML en formato jpg/jpeg (apellido1Apellido2_Nombre.jpg).
  - Archivo **.uml** (apellidoApellido2_Nombre.uml) o cualquier otra extensión correspondiente a StartUML.
- Proyecto exportado desde Eclipse, compatible con JDK1.8. y cuyo nombre tiene que ser **apellido1Apellido2_Nombre.zip.** Este proyecto contendrá la implementación de todas las clases definidas en el proyecto.
- Documentación, debe seguir el formato indicado en la plantilla que encontrareis eGela.

¡Aviso! Para que la práctica se pueda corregir hay que respetar todos los requisitos previamente indicados (nombres de ficheros, proyectos, clases, etc.). Además, el proyecto no podrá tener errores de compilación y su ejecución no puede finalizar en ningún **estado de excepción**. En caso de infringir cualquiera de estas normas, **la calificación del proyecto será 0**.

Primera parte

1. La clase Food establece la representación de cada alimento disponible en el pub. Esta clase debe definirse en un paquete llamado `packfeed` y debe incluir los componentes que se describen a continuación:

1.1. El nombre (es una constante), la *descripción* y las *calorías* del alimento.

1.2. Dos `constructores`: uno tendrá como parámetros el *nombre* y las *calorías* del alimento y otro sólo el *nombre* (el valor por defecto de las calorías es 50).

1.3. Los métodos `get` y `set` **que sean necesarios**.

1.4. Redefinición del método `equals`. Dos alimentos (instancias de Food) serán iguales cuando los nombres de ambas sean iguales.

1.5. Redefinición del método `toString`, de forma que, para cada alimento (intancia de `Food`), devuelva un String con el nombre y las calorías del alimento.

2. Comprueba el correcto funcionamiento de la clase `Food` incluyendo en el paquete `packcheck` la clase `FoodCheck` que se te proporciona junto con el enunciado del proyecto. Ejecuta el proyecto y comprueba que los resultados que da son correctos.

3. La clase[1] Drink debe incluirse también en el paquete `packFeed` y representa una marca de bebida disponible en el pub. Sus características generales son el *nombre* de la marca (String), el atributo calorías por cada 100ml (int), el atributo glutenFree dirá si está libre de gluten o no (booleano) y el atributo *pairing*, que indica con qué alimentos resulta más adecuado combinarla (un Array de hasta 10 elementos de tipo tipo `Food`, que se irán añadiendo al array de uno en uno). Por otra parte, debe contener las siguientes operaciones:

3.1. Un **método constructor** que construirá un objeto de tipo Drink a partir de su nombre.

3.2. Los **métodos de consulta y actualización** (`get` y `set` que sean necesarios.

3.3. Redefinición del método `equals`. Dos marcas de bebida serán iguales cuando el nombre de ambas sea igual.

3.4. Redefinición del método **`toString`**, que construye y devuelve un String con todas las características de una bebida.

3.5. Método **addFood**: dado un elemento de tipo `Food`, lo añadirá al final del array *pairing*, si previamente no estaba ya incluido en dicho array.

3.6. Método **removeLastFood**: elimina y devuelve el último elemento del array *pairing*.

---

[1]Se debe documentar correctamente utilizando JavaDoc. Cuando esté terminado generar la documentación JavaDoc dentro de la carpeta doc.

3.7. Método **removeFirstFood**: elimina y devuelve el primer elemento del array *pairing*.

3.8. Método **howManyPairing**: devuelve el número de alimentos con los que se combina adecuadamente la bebida desde la que se invoca este método.

3.9. Método **pairs**: dado un alimento, devuelve si la bebida desde la que se invoca este método se combina o no con el alimento dado.

3.10. Método **pairingList**: Devuelve en un String los alimentos (Food) con los que combina bien la bebida desde la que se invoca este método.

4. Comprueba el correcto funcionamiento de la clase `Drink`. Para ello, coloca en el paquete packcheck, la clase `DrinkCheck` que se te proporciona junto con el enunciado del Proyecto. Ejecuta y comprueba que da los resultados esperados.

5. Para representar la cervecería se debe definir el paquete `packpub` con la clase `Pub`. Esta clase constará de dos listas del tipo ArrayList, la primera para guardar las bebidas y la segunda para almacenar los alimentos de la cervecería. Además, esta clase debe contener las siguientes operaciones:

5.1. Un **método constructor**, que construye las dos listas vacías.

5.2. Método **addDrink**: dada una bebida, la añade al final de la lista de bebidas si previamente no estaba ya incluida en dicha lista.

5.3. Método **removeDrink**: dada una bebida, la elimina de la lista de bebidas de la cervecería.

5.4. Método **removeDrinksByCalories**: dado un valor de calorías se eliminarán todas las bebidas que tengan esas calorías o más y sus nombres se devolverán en un ArrayList.

5.5. Método **mostCaloricDrink**: Devuelve la bebida con más calorías.

5.6. Método **showDrink**: muestra en pantalla la información de todas las bebidas de la cervecería

5.7. Método **obtainDrink**: dado un nombre de bebida, devuelve la bebida que tiene ese nombre. Si no hay ninguna con el nombre dado, devuelve **`null`**.

6. Comprueba el correcto funcionamiento de la clase `Pub`. Para ello, coloca en el paquete `packcheck` la clase `PubCheck`, que debes crear para tal proposito. Ejecuta y comprueba que da los resultados esperados.