Objetivo general del taller

El objetivo de este taller es aprender a implementar en Java un modelo de clases dado, reconocer los diferentes componentes que podrían aparecer en un modelo de clases y practicar la implementación de algoritmos básicos en Java.

Objetivos específicos del taller

Durante el desarrollo de este taller se buscará el desarrollo de las siguientes habilidades:

1. Implementar en Java un modelo de clases expresado como un diagrama de clases de UML, incluyendo atributos de los principales tipos básicos y métodos con funcionalidades básicas.
2. Entender el significado de la cardinalidad (*multiplicity*) en las asociaciones entre clases y ser capaz de implementar diferentes tipos de asociaciones usando relaciones directas, listas (`ArrayList`) y mapas (`Maps`)
3. Entender el significado de `null` en Java, saber verificar si una referencia tiene valor `null` y entender los problemas relacionados con invocar métodos sobre referencias con valor `null`.
4. Conocer el método `equals`.
5. Utilizar con habilidad el ambiente de programación del curso y ser capaz de buscar recursos externos que les permitan resolver problemas técnicos puntuales.

Instrucciones generales

1. Descargue de Bloque Neón el archivo `Taller2-Hamburguesas_esqueleto.zip` y descomprima los archivos que tiene por dentro.
2. Cree un nuevo proyecto en Eclipse para implementar el taller y copie los archivos del esqueleto a la carpeta del proyecto.
3. Lea con cuidado este documento y vaya realizando las actividades. En este documento encontrará información importante para completar el programa entendiendo lo que está haciendo.

No solo es posible usar recursos externos (Google, StackOverflow, libros, etc.) para resolver el taller, sino que es deseable que se consulten fuentes externas.

Descripción del caso: Hamburguesas

Para este taller vamos a trabajar sobre una aplicación para manejar el menú y los pedidos de una tienda de hamburguesas. Empezaremos dándole información general sobre el caso y luego le presentaremos un modelo de clases para implementar una solución.

La tienda de hamburguesas tiene un menú compuesto por unos productos básicos, combos e ingredientes adicionales. Los productos básicos son elementos que pueden hacer parte de un pedido (ej. papas fritas medianas) y cada uno tiene un precio. Los combos son agrupaciones predefinidas de productos básicos que tienen un

descuento (ej. el combo especial incluye una hamburguesa sencilla y unas papas medianas y tiene un descuento del 7% sobre el precio de los productos por separado). Los ingredientes son elementos que se pueden agregar a cualquier producto o quitar de cualquier producto (ej. agregarle queso a una hamburguesa o quitarle el tomate). Los ingredientes adicionales tienen un precio adicional, pero quitar ingredientes no reduce el precio de un producto. Un combo no se puede ajustar agregándole o quitándole ingredientes.

Los clientes de la tienda pueden hacer pedidos y la tienda puede recibir un pedido a la vez (el pedido en curso). Un pedido debe tener el nombre del cliente, la dirección del cliente y un conjunto de elementos que hacen parte del pedido. Esos elementos pueden ser productos básicos del menú, pueden ser combos, o pueden ser productos modificados, es decir productos base a los cuales se les quitó o se les agregó uno o varios ingredientes.

Nota: no esperamos que la aplicación que usted construya garantice que las modificaciones a los ingredientes tengan sentido (ej. “agua sin gas con adición de suero costeño y sin cebolla” sería un elemento válido en un pedido).

Las facturas de los pedidos deben discriminar el valor de cada elemento, el valor neto total, el IVA (19%) y el valor total (neto + IVA).

Requerimientos iniciales de la aplicación

A continuación, presentaremos los principales requerimientos y restricciones de la aplicación que usted tiene que construir.

Funcionalidades de la aplicación

La aplicación debe ofrecer las siguientes opciones:

1. Mostrar el menú
2. Iniciar un nuevo pedido
3. Agregar un elemento a un pedido
4. Cerrar un pedido y guardar la factura
5. Consultar la información de un pedido dado su id

Restricciones

La interacción con el usuario debe hacerse a través de una interfaz de consola, pero debe ser tan amigable como sea posible (ej. ¡no lo ponga a teclear el nombre completo de un ingrediente!)

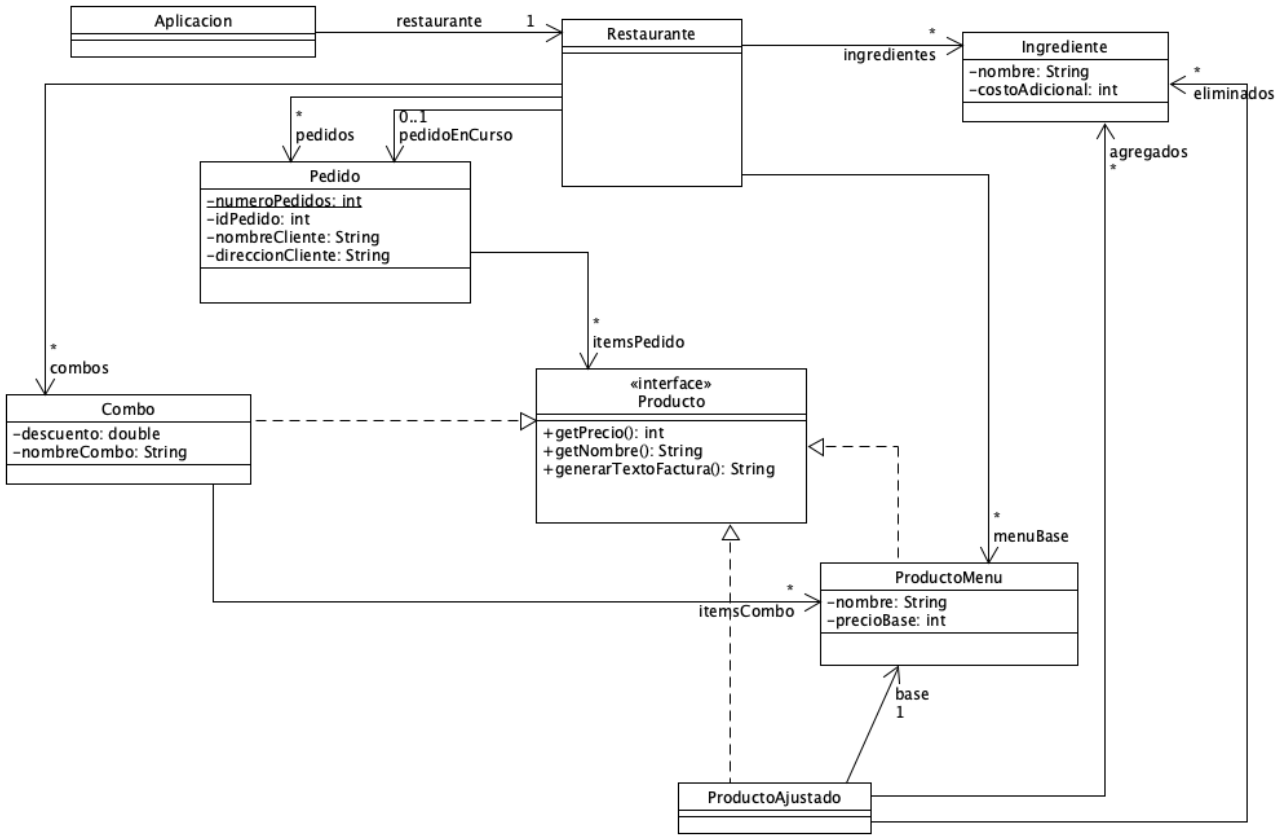
Cuando se abra, la aplicación debe cargar la información de los ingredientes, los productos base, y los combos de archivos de texto como los que se encuentran en el esqueleto del taller. No le debe preguntar al usuario por el nombre de estos archivos.

Cuando se cierre una factura, se debe guardar automáticamente en un archivo basado en el id de la factura, sin preguntarle al usuario por el nombre.

Las clases utilizadas para implementar la interacción con el usuario y las clases para implementar el estado de la aplicación y su comportamiento deben estar separadas en dos paquetes.

Clases e interfaces

Las clases e interfaces utilizadas para implementar su aplicación deben estar basadas en el siguiente diagrama de clases. Dentro del esqueleto hay una imagen mucho más detallada que indica los métodos que usted debe implementar en cada clase.



- La clase `Aplicación` es la única clase del diagrama que es parte del paquete de la consola. Usted puede agregar más clases a este paquete si es necesario, pero tenga cuidado de que sean clases dedicadas exclusivamente a mediar la interacción con los usuarios (no pueden implementar "la lógica" de la aplicación).
- La clase `Restaurante` es el corazón de la lógica de la aplicación y sirve principalmente para agrupar el resto de las cosas que maneja la aplicación.
- La interfaz `Producto` abstrae el comportamiento de todos los elementos que pueden hacer parte de un pedido.
- La clase `Ingrediente` representa a un ingrediente que se puede agregar o eliminar de algún elemento en un pedido.
- La clase `ProductoMenu` representa un producto que aparece en el menú y que puede ser agregado en un pedido.
- La clase `ProductoAjustado` representa un producto que se ajustó cuando se agregó a un pedido: un `ProductoAjustado` tiene un producto base y una serie de ingredientes que se agregan o se eliminan a este producto.
- La clase `Combo` agrupa un conjunto de productos que tienen un descuento al comprarse en combo.

En el archivo que contiene la imagen detallada de este modelo las clases tienen todos sus métodos especificados, incluyendo métodos públicos y privados. Los métodos privados aparecen ahí para sugerirles una descomposición de la lógica: asegúrese de implementar todos los métodos, incluyendo los privados, y que esos métodos privados sean invocados desde alguno de los otros métodos.

Parte 1: Implementar el modelo de clases de la lógica

La primera tarea para realizar es implementar completamente el modelo de clases con la lógica de la aplicación: debe incluir todas las clases e interfaces mostradas en el diagrama de clases, y debe necesariamente implementar los métodos definidos en los modelos.

Usted puede hacer modificaciones al modelo y a los métodos siempre y cuando los declare: su entrega debe incluir un documento donde explique cada una de las desviaciones de su implementación con respecto al modelo y justificando la modificación.

Parte 2: Implementar una consola para la aplicación

El modelo UML propone una clase `Aplicación` con unos pocos métodos básicos. Usted debe decidir qué otras clases y métodos quiere incluir en la consola de la aplicación.

1. Construya un mapa de las opciones que va a ofrecer su aplicación: puede utilizar el mecanismo que quiera para explicar cuáles van a ser los menús, cómo van a estar organizados y conectados entre ellos, cómo va a seleccionar las opciones el usuario, etc.
2. Construya un diagrama UML donde aparezcan todas las clases y métodos que usted va a implementar.
3. Implemente la consola para la aplicación
4. Ejecute su aplicación y compruebe que funciona correctamente

Parte 3: Modificaciones a la aplicación

En esta parte usted debe incorporar las siguientes modificaciones a su aplicación (asegúrese de que su entrega incluya tanto la versión original como la versión modificada):

1. Debe ser capaz de manejar las bebidas por separado del resto de productos del menú, y la lista de bebidas disponibles debe cargarse desde un archivo diferente al resto del menú.
2. La factura de un pedido debe incluir la cantidad de calorías del pedido completo, las cuales deben estar basadas en las calorías de cada elemento que lo conformen.
3. Cuando se cierre un pedido se debe verificar si alguien había hecho un pedido idéntico. Implemente esta funcionalidad sobreescribiendo el método `equals` de pedido y de las otras clases que sean necesarias. Piense con mucho cuidado cómo va a saber de qué tipo específico es un elemento particular dentro de un pedido.

Teoría: Comparar valores en Java

En Java, hay dos formas de comparar valores. Cuando se trata de **tipos simple**s (`int`, `float`, `boolean`, `long`, etc.) se debe usar el operador `==`. Es decir que si tenemos una variable llamada `valor` y queremos saber si tiene el valor 5 podemos usar la siguiente expresión:

```
valor == 5
```

Por otro lado, para comparar objetos NO SE DEBE USAR `==`. Si usamos el operador `==` cuando comparemos dos objetos el resultado será similar al que habríamos obtenido con el operador `is` en Python. Es decir, cuando en Java comparamos dos objetos usando == no estamos preguntando si son iguales, estamos

preguntando si son el mismo. Para comparar si dos objetos son iguales, debe usarse el método `equals`. Veamos ahora unos ejemplos de esto usando la clase String[1]:

```
String cadena1 = “Hola”;
String cadena2 = “Mundo”;
String cadena3 = cadena1 + cadena2;
String cadena4 = “HolaMundo”;
boolean b1 = (cadena1 == cadena2);                // false
boolean b2 = (cadena1 == cadena1);                // true
boolean b3 = (cadena3 == cadena1+cadena2);        // false
boolean b4 = (cadena3 == cadena4);                // false
boolean b5 = (cadena4.equals(cadena1+cadena2));   // true
boolean b6 = (cadena3.equals(cadena4));           // true
```

Finalmente, tenga en cuenta que para que tenga sentido usar el método `equals` dentro de una clase que usted haya definido, debería haber *sobrecargado* el método.

Entrega

1. Cree un repositorio privado donde quedarán sus talleres y deje ahí todos los archivos relacionados con sus entregas.

2. Entregue a través de Bloque Neón el URL para el repositorio, en la actividad designada como “**Taller 2**”.

[1] La clase `String` es la más fácil de usar para dar ejemplos, pero también tiene algunas particularidades ligadas a optimizaciones que hace la JVM para que manejar cadenas sea mucho más rápido. En particular, si en su código aparecen explícitamente dos cadenas idénticas, la optimización las convertirá en una sola y su código podría comportarse de formas inesperadas. No es necesario que conozca ahora todos los detalles: sólo asegúrese de siempre usar `equals` para comparar objetos y no debería tener sorpresas.