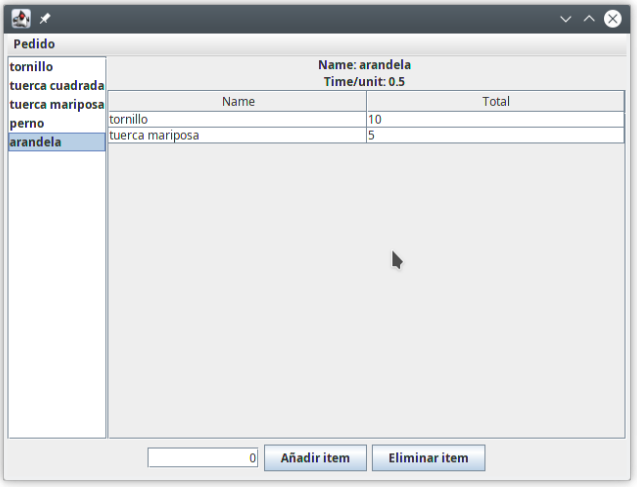
Programación III - Facultad de Ingeniería - Univ. de Deusto - 10/01/2020

Nombre _____ **DNI** _____ **Aula** _____

- En la web de la asignatura (sección "Examen Ordinaria") existe un archivo comprimido con los ficheros necesarios para llevar a cabo el desarrollo del examen.
- Para realizar la entrega del examen en ALUD comprime (`.zip`) todos los directorios (`src, test, ...`) que contengan código que hayas realizado. Asegúrate de incluir todos los ficheros que hayas modificado.

Se pide realizar las tareas indicadas a continuación que completan la funcionalidad de una aplicación para la gestión de pedidos de fabricación en una empresa. La captura muestra la aplicación terminada, sin embargo, en el código proporcionado faltan elementos de la interfaz visual (lista izq.) que se completarán como parte del examen.



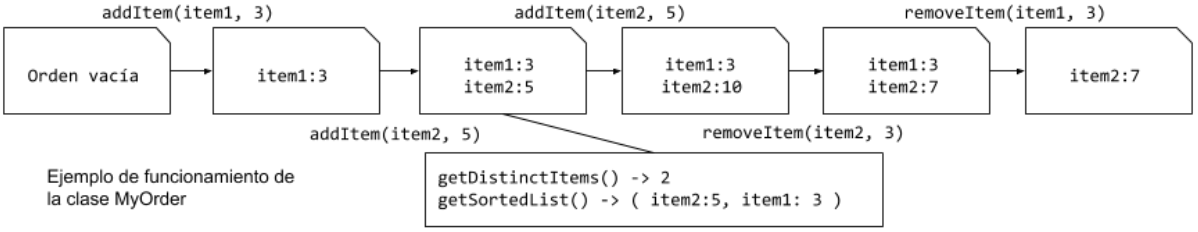
Tareas a realizar

1. **Base de datos (2 puntos).** Se proporciona una base de datos SQLite llamada *database.db*. Esta base de datos contiene una tabla `Items` con la siguiente estructura de columnas: `id (entero), name (texto), timePerUnit (float)`. Se pide implementar completamente una clase `DBManager` que además de métodos para conectar y desconectar con la base de datos **(1 punto)**, proporcione un método `loadItems()` **(1 punto)** que obtiene todas las filas de la tabla `Items` y retorna una lista de objetos `orders.Item`.

2. **Collections (2 puntos).** Se pide implementar la funcionalidad determinada por la interfaz `orders.Order` en una nueva clase `orders.MyOrder`. Esta nueva clase representa una orden de fabricación que contiene diferentes artículos (clase `orders.Item`) a fabricar y asociada a cada uno la cantidad que debe ser fabricada del mismo. Se indica a continuación la funcionalidad que se espera de cada método de esta nueva clase:
   a. `Para` implementar la funcionalidad se considerará que dos instancias de la clase `Item` son iguales si tienen el mismo valor para su propiedad `'id'`, independientemente del valor de sus otras propiedades. El código proporcionado deberá ser modificado para cumplir este criterio de igualdad. **(0,4 puntos)**
   b. `addItem(Item, int):` este método recibe un artículo (`Item`) y la cantidad de unidades a fabricar del mismo y guarda dicha información en la orden de fabricación. Si este método se utiliza para añadir un artículo ya existente en la orden de fabricación entonces se debe incrementar la cantidad de dicho artículo a fabricar (no debe haber artículos duplicados en una orden). **(0,4 puntos)**
   c. `removeItem(Item, int):` este método sustrae de la orden de fabricación, para el artículo (`Item`) indicado, la cantidad de unidades pasada como parámetro. Si el artículo indicado no existía en la orden no

tiene efecto. Si el número de unidades del artículo llega a cero dicho artículo es eliminado completamente de la orden de fabricación. **(0,4 puntos)**

d. `getTotalTime()`: calcula y retorna el tiempo total de fabricación de la orden según el número de artículos de cada tipo y a su tiempo de fabricación (propiedad `timePerUnit` en `Item`). **(0,2 puntos)**
e. `getDistinctItems()`: obtiene el número de artículos que hay en la orden de fabricación. **(0,2 puntos)**
f. `getSortedList()`: devuelve una lista de objetos Entry<Item, Integer> ordenada de mayor a menor de acuerdo al número de unidades que deben ser fabricadas de cada artículo. **(0,4 puntos)**



Tras implementar la clase se debe quitar el comentario existente contenido en el método `createNewOrder()` en `gui.MainWindow` para hacer uso de esta nueva clase en la aplicación.

3. JUnit (2 puntos). Se pide realizar un test unitario que pruebe la funcionalidad de la clase `orders.MyOrder` implementada anteriormente. El alumno es libre de definir los datos de prueba necesarios para poder comprobar los distintos métodos de la clase de acuerdo a su funcionalidad esperada.

4. Swing (2 puntos). Como se puede comprobar al ejecutar el código proporcionado, no está implementado todavía en la aplicación el componente JList que se aprecia en la captura.

a. Añadir un componente visual JList para conseguir completar la interfaz gráfica como se muestra en la captura y utilizar el método implementado en la Tarea 1 para que al iniciar la aplicación se visualicen todos los artículos (Items) disponibles en la base de datos en dicho JList. **(1 punto)**
b. Completar la funcionalidad de los botones "Añadir Item" y "Eliminar Item" para que al pulsar sobre ellos se añadan o se eliminen del objeto `currentOrder` de `gui.MainWindow` tantas unidades del artículo seleccionado en la lista de la izquierda como haya indicado el usuario en el campo de texto situado en la parte inferior. Se debe hacer uso de los métodos `addItem/removeItem` de la interfaz MyOrder y llamar al método `updateTable()` para que se reflejen los cambios en el JTable. **(1 punto)**

5. Serialización (2 puntos). Se pide implementar la funcionalidad de los menús `Pedido->Guardar Pedido...` y `Pedido->Cargar Pedido...` para escribir y leer, respectivamente, la orden de pedido actual utilizando el mecanismo de serialización de objetos proporcionado por Java. En concreto, en la clase `gui.MainWindow`:

a. Realizar los cambios necesarios para poder serializar correctamente la clase orders.MyOrder implementada en la Tarea 3. **(1 punto)**
b. Implementar el método `saveOrder(Order, File)` para escribir el objeto Order recibido en dicho método en el fichero pasado como parámetro. Si se produce algún error durante la serialización el método debe lanzar una excepción de tipo `OrderSerializationException`. **(0,5 puntos)**
c. Implementar el método `loadOrder(File)` que carga el fichero recibido como parámetro, y que retorna el objeto leído. Si existe algún error durante en la deserialización el método debe lanzar una excepción de tipo `OrderSerializationException`. **(0,5 puntos)**