UNIT 1. FINAL EXERCISE

FantasyCollectionFX

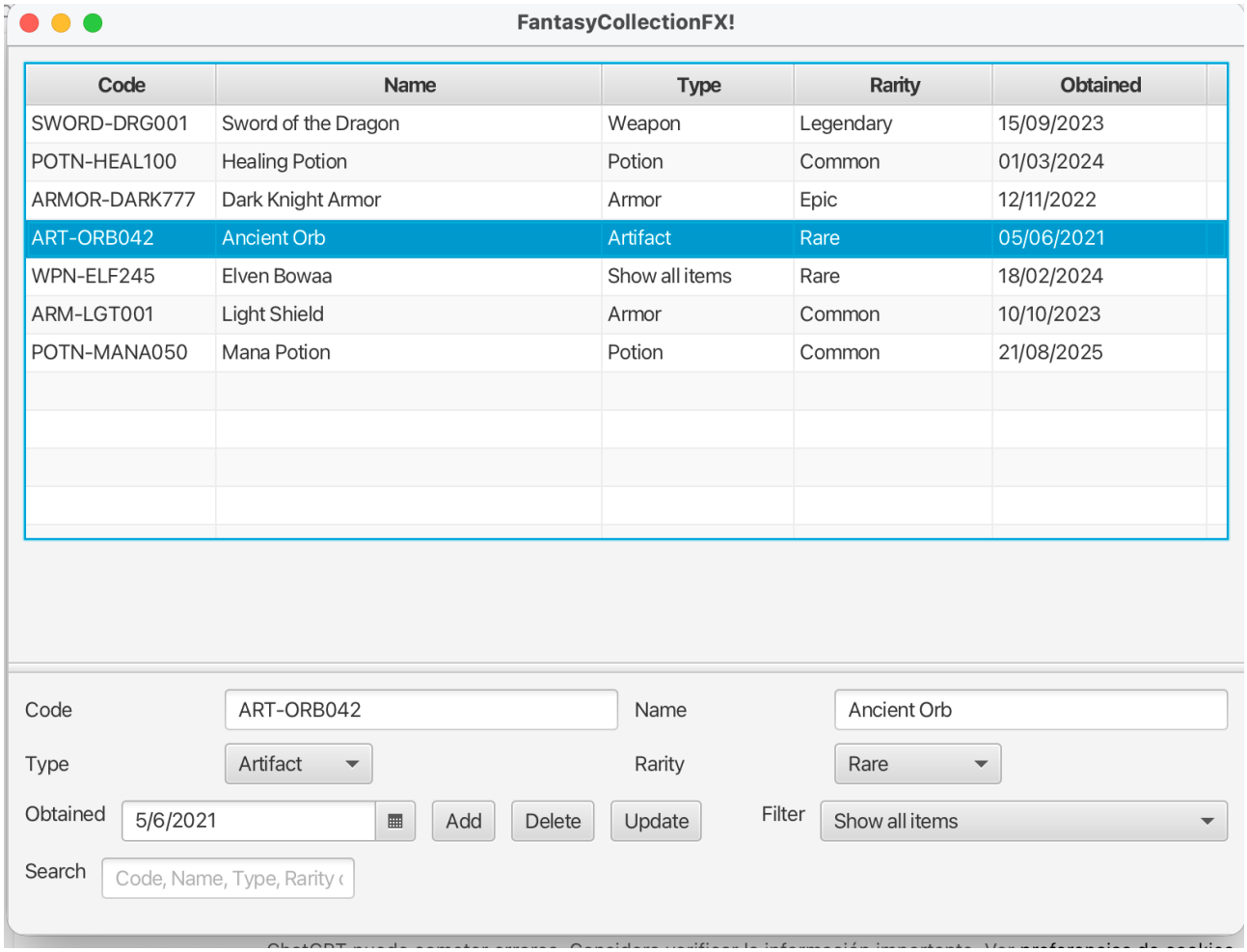
Service and process programming

IES SAN VICENTE

INDEX

1. Introduction and first steps

In the final exercise of this unit, we are going to implement a JavaFX application to manage a list of **fantasy items** collected in an RPG world. With this application, we will be able to filter items according to different criteria, search for them, and keep our geeky collection well organized.



1.1Setting up the project

We are going to create a JavaFX project called **FantasyCollectionFX**. Once the project is created, inside the *Source Packages* section, create the following packages and subpackages:

* **fantasyfx** package, which will be our main package with the JavaFX main class and controllers.
* **fantasyfx.model** package, to store our model (`Item` class).
* **fantasyfx.utils** package, to store some useful classes.

2. Class structure

Besides the JavaFX main application with the FXML file and controller (we will see them in next section), we are going to need some additional classes to store the information about the items.

2.1 The Item class in model package:

Add a new class inside the *fantasyfx.model* package called `Item`. This class will have the following attributes:

- Code (a String, such as "SWORD-DRG001")
- Name (a String, e.g. *Sword of the Dragon*)
- Type (a String, e.g. *Weapon, Armor, Potion, Artifact)*
- Rarity (a String, e.g. *Common, Rare, Epic, Legendary*)
- Obtained date (a LocalDate)

Add two constructors:

- A constructor with the code only.
- A constructor with all the attributes.

Include getters and setters for each attribute.

2.2 FileUtils class

We will create a class called FileUtils in the *fantasyfx.utils* package. It will contain:

- static List<Item> loadItems()to load items from a text file.
- static void saveItems(List<Item>) to save a list of items into a text file.

The file will be named **items.txt** in the project root folder.

.2.1.1. Files of the project

The structure of the text file containing the items(items.txt) will be:

```
code;name;type;rarity;obtained_date
SWORD-DRG001;Sword of the Dragon;Weapon;Legendary;15/09/2023
POTN-HEAL100;Healing Potion;Potion;Common;01/03/2024
ARMOR-DARK777;Dark Knight Armor;Armor;Epic;12/11/2022
```

.2.2. Other useful classes

Although it is not compulsory, it may be useful to add some other classes. For instance, a class called *MessageUtils* (in the *fantasyfx.utils* package) to show different *Alert* messages. It could have these static methods:

- static void showError(String header, String message) to show error messages.
- static void showMessage(String header, String message) to show information messages.
- ...

You can add as many classes as you need inside this package.

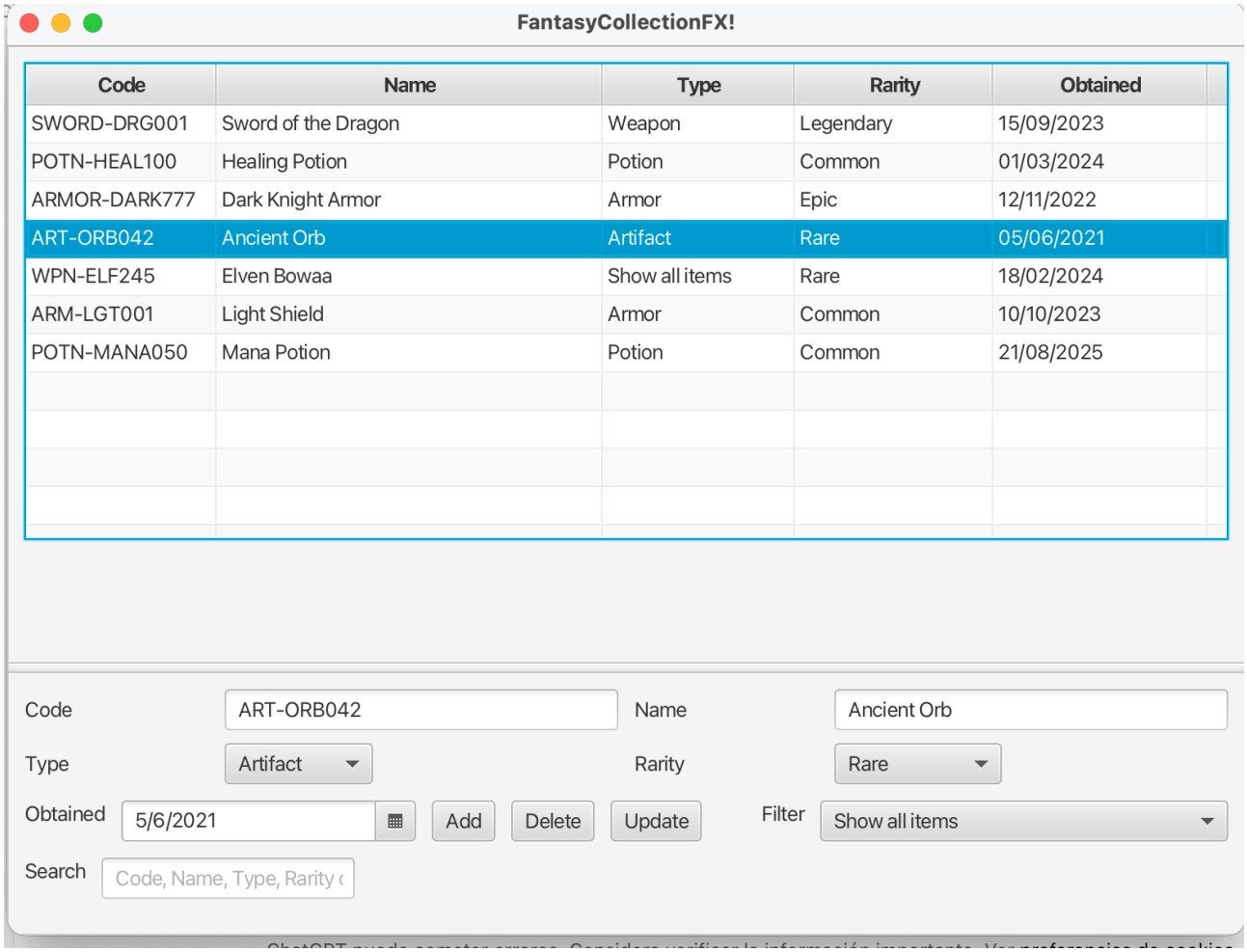
3. The JavaFX application

Let's create the JavaFX application classes. Follow these steps:

1. The JavaFX Main Application will be called **MainApplication** inside *fantasyfx* package.
2. There will be an FXML file called **main-view.fxml** with its associated controller (**MainController.java**). As the project is created using Maven, the fxml files will be in the resources directory and the java files will be in the *fantasyfx* package.
3. Make sure that your MainApplication class loads the contents from the FXML file.

3.2 Designing the main view

Use now Scene Builder to design the main scene and get an appearance similar to this:



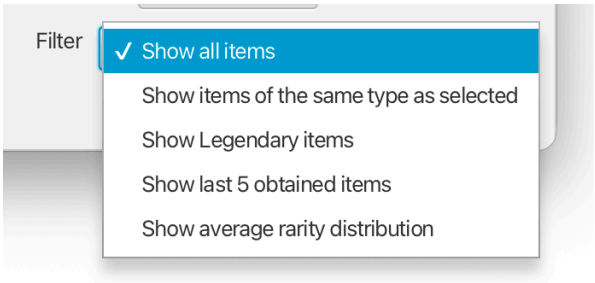
You can, for instance, use one gridPane to arrange general distribution, and in each cell of the grid a Vbox, Hbox or SplitPane.

Remember to set an ***fx:id*** to each element that may need to be accessed from the controller.

3.3 How should it work?

As soon as the application starts, it will need to load the file items.txt and place them into the left table of the view. Be careful with the *LocalDateTime*, you must show it with an appropriated format (remember, *tableViews* get the information from the getters). When selecting one of the books of the table, the *Textfields* of the down part of the view will be filled in with the information of the item selected. You can create a listener to the *Tableview* to do this.

With the *ComboBox* of the bottom you will select one of these options:



The first option consists of showing all the items on the *Tableview*.

The second option will show an Alert with the total number of items of the same type of the one selected in the *TableView*.

The third option will show in the *Tableview* only the items with the rarity = Legendary

`The next option will show the 5 most recent items based on obtained date.`
`And the last option will calculate and show the percentage of each rarity type (Common, Rare, Epic, Legendary).`

All these options will be implemented using streams.

`Add a text field at the bottom. When the user types, the application will show all items whose code, name, type, rarity, or date contains the typed text.` You can use the property of the text field called OnKeyReleased.

And of course, you will have to implement the options of the buttons for add, delete and update.

When finishing the application, the user will click on the X to close the window, the OnCloseRequest event will be released, and we will save on the file the items of the list with all the changes. We will only write on the file when closing the application.

4. Optional improvements

Besides the compulsory part of this exercise, you can try to implement this optional part.

4.1 Adding CSS styles (1 point)

Add a CSS stylesheet with some default styles for the main view, such as:

- Background color: choose among any color other than gray, black or white.
- Font color: according to the background chosen, it can be either white or gray.
- Every button must have a black background with white text color.
- Other styles that you may want to add.

5. Evaluation rules

5.1 Compulsory part

To get your final mark, the following rules will be applied, and you must **use streams and lambda expressions wherever you can.**

Item achieved	Points
Class structure (*model* and *utils* packages)	1
JavaFX application layout, similar to the one shown in previous figures. You must use the appropriate layout containers to arrange the controls properly in the view (HBox and/or VBox are very helpful).	1,5
Loading from the text files and writing in them using streams.	2
Showing items on the tableview	0,5
Implement add, delete and update options	1
Choose the elements regarding the ComboBox option for filter (streams)	1,5
Search text	0,5
Showing error messages whenever something can't be done.	1
Code documentation (Javadoc comments for every class and public method or constructor), cleanliness and efficiency.	1

5.2 About the optional improvements

If you decide to implement any optional improvement for this exercise, keep in mind that:

- If your compulsory part is lower than 10, you can get up to 10 points by implementing one (or many) of the optional improvements (up to 1 point per improvement)
- If your compulsory part is perfect (10 points), you can get up to 11 points by implementing ALL the optional improvements.

5.3 Final mark of the exercise

Remember that your grade will depend on the explanation and the additional program feature you must implement on the day of the correction.