# Advanced Object Oriented Programming – Major Assignment

# Academic Integrity

This is to be an INDIVIDUAL assignment. It will be assessed for similarity to other files submitted and if there is a strong resemblance, you will receive a grade of 0 and an academic misconduct. Please build your files independently and do not share your work with others.

We do use software that can check for non-trivial differences (In other words, don’t change a variable name and think it is somehow original work).

You should avoid using segments of code found on the internet. If you do choose to use any code segment(s) found on the internet, you must document where you found the code segment and describe in comments how it works. This must not account for more than 5% of your total code. A verbal review may also be used to assess if the submitted code is well understood. If you cannot explain your code, you will receive a grade of 0.

## Overview

The goal of this assignment is to demonstrate your ability to create interactive GUI’s using JavaFX, select appropriate data structures, demonstrate sorting, lambda expressions and the use of streams. For this assignment, you will be implementing a GUI that will allow a user to view the inventory of a company.

You are free to choose what types of products are in the inventory, but you will need to design your classes to support the concept of categories and products. For example, if it was an electronics store, you will need to have categories of products such as “Gaming Laptops”, “Mobile Phones”, “Televisions”, etc…

## Product

The product class should have the ability to store relevant information about the product. Examples could include the name of the product, a description, an image, a price and the number of units in stock.

The Product class should have the following:

* A constructor that configures a minimum of 4 instance variables
* All the regular get/set methods with validation for any set method (no validation required for a setImage() method)
* A method to sell 1 unit of the product. This method should validate that there are units to sell before it reduces the amount in stock by 1
* A toString() method to easily see the product name, units in stock and price

## Inventory

The inventory class should use a TreeMap to store Product objects by category. Hint: use the category name as the key and put a LinkedList as the value in the TreeMap.

The Inventory class should have methods that will:

* Return a List with all category names
* Return a List with ALL products
* Return a List with all products for a specific category
* A method to add a Product to a given category

## GUI

The GUI does not need to look exactly like the one shown below. If you can envision a cleaner look, then set it up, just ensure the functionality described below is in place.

When the GUI first launches, it should show all products in your inventory sorted, with 1 item selected and the image for the product displayed. In Figure 1 below, you can see the default sort order is priced high to low. In the ListView, the products are sorted most to least expensive. The first item is selected and that image is displayed.

The total amount of all inventory is displayed, but the category value is not updated because a category is not selected.

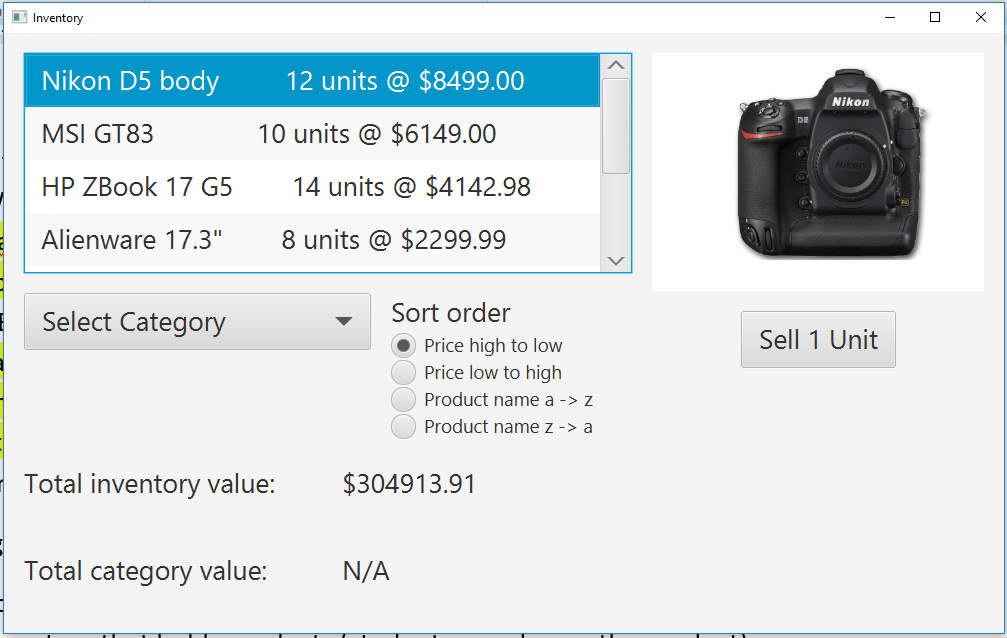


Figure 1

## Selecting a category

When creating a utility to select the category for this assignment, I want you to use the change listener through the controller class with a lambda expression. In other words, for full marks, do not call a method with the “on action” utility in scene builder.

If the user selects a category, the view should be updated by reloading the ListView (or whatever GUI object you choose) to load all the products for that category. In Figure 2 below, you can see that when the DSLR category is selected, the label for the total category value shifted from N/A to $101,988.00. That number comes from 12 units x $8499.

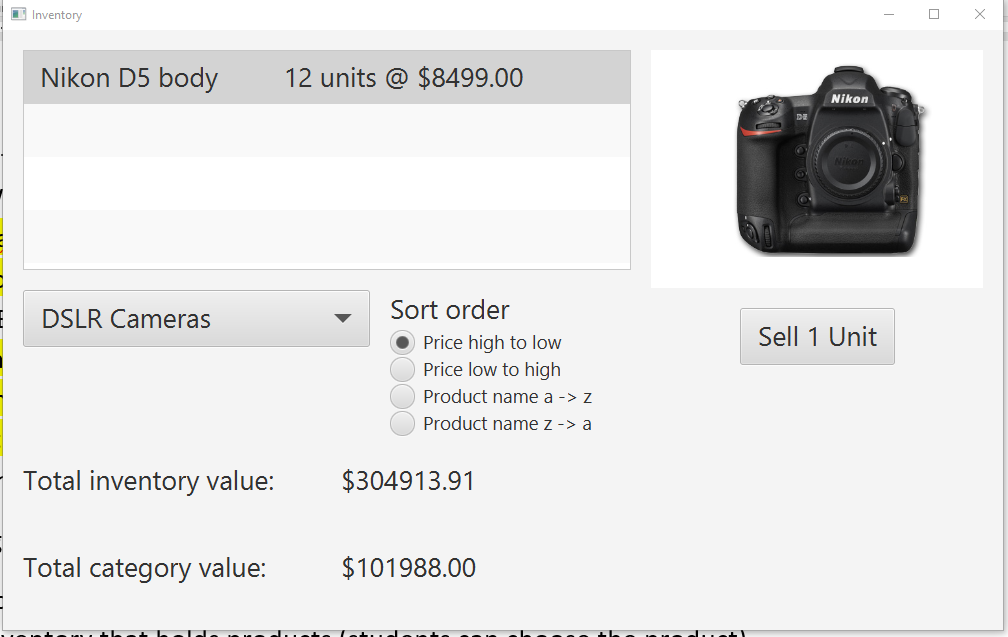


Figure 2

### Sorting

Your GUI should support sorting in at least 4 different ways (you can decide what type of sort most sense by your type of product). The sorting should be performed by using streams and the .sort() method. Note: if you put your Product objects in a TableView, the default sort features will not get marks for this part of the assignment. I want to see you demonstrate (and get a lot of practice) with streams. As you can see in Figure 3 below, the sort order is by product name (a-z).

If the user selects a new category, the sort order should be maintained.

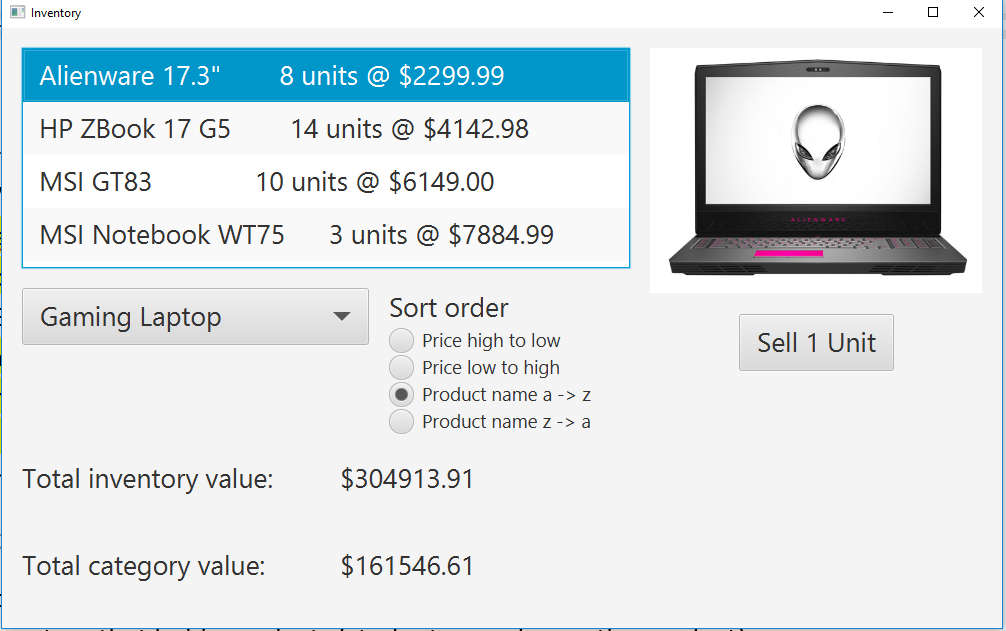


Figure 3

## Selling a product

If a product is selected, the user has the ability to click on the “sell” button. This should check if there is more than 0 of the selected Product available and then remove 1 unit from the inventory.

The units showing in the ListView (or other GUI object) should be updated with the new amount of units available, the total inventory and category values should also be updated.

## Labels showing $ amount for inventory

The labels showing the total amount of inventory and total amount for a given category should be updated by creating a stream that traverses the inventory and sums the price \* units in stock. You could also do this using a for loop, but for full marks, you need to use streams.

## Building your project

My recommended order of building your assignment is:

1. Build the class diagrams (You do not need to hand these in, but it can help with the design process)
2. Create a private repository on GitHub that will store ALL files required for your project to run. ***Continuously*** use this repository and update it every time you add a method or small group of working functionality.
3. Create the Product and Inventory classes (remember to upload to GitHub once complete).
4. Build a simple GUI to at least load products. You are free to just “hard code” the inventory in your controller class. You do not need to connect to a DB. If you wish to use a DB, be sure to include the driver file in your GitHub submission. The DB should be remotely accessible (everyone received an account on AWS this semester as an option)
5. Add the ability to select a category and reload the list in the view
6. Add the ability to sort. Make sure to test the default sort on initial load, then test when changing categories and ensure it works consistently.
7. Add the ability to sell a selected item. Test to ensure the total inventory, category value and the display showing how many units are in stock all go down to reflect 1 product being sold

You should submit the class diagrams and a link to your private GitHub repository. ***This is an individual assignment, all work MUST be your own. I should be marking “your” code and NOT seeing MY code.***

## Rubric

|  | Level 0 | Level 1 | Level 2 | Level 3 |
| --- | --- | --- | --- | --- |
| Create a Product class | The Product class is not created or has an error preventing the code from compiling | The Product class is created and compiles, but it starts with a lower case letter. | The Product class is created, it compiles, it starts with an upper case letter and stores a minimum of 4 attributes of the class. | All of level 2, all variables & methods have good datatypes and naming conventions |
| Create an Inventory class | The Inventory class is not created or has an error preventing it from compiling | The Inventory class is created and it has a TreeMap to hold categories and Products | All of level 1 and aligns with professional Java style, naming conventions, indents, spacing, etc… | All of level 2 and has methods for Return a List with all category names  ->Return a List with ALL products  ->Return a List with all products in a category  -> A method to add a Product to a given category |
| JavaDoc Comments | Little to no comments | There are a few comments, but most methods are not covered | Every method in the model class(s) has a comment, but they are not in the Javadoc format | Every method in the model class has a Javadoc comment. |
| GUI launch class | A launch class is not provided | A launch class is provided and will trigger the GUI to run. Note: the GUI must show on the screen for this mark. |  |  |
| GUI Professional look | Objects are not lined up and / or text is truncated | The GUI uses at least 5 different visual GUI classes | Level 1, plus the GUI has a clean, professional look to it. |  |
| Controller-load inventory items | The controller does not have a way to load an Inventory object that contains Products | The controller can display preloaded Product objects, but they are not part of an Inventory | All of level 1, plus the GUI will populate a ComboBox (or some other object) to select the categories | All of level 2 and there is at least 3 categories and 7 products loaded into the view. The products are sorted, the first product is selected and the image of the product is showing |
| Update the inventory value | There is no utility to calculate and/or display the total inventory value | The total inventory value is updated, but the $ amount is either incorrect or does not use a stream to calculate the total | The total $ amount label is updated with the correct amount. The total was calculated using streams. |  |
| Update the category $ total | There is no utility to calculate and/or display the category inventory value | The category inventory value is updated, but the $ amount is either incorrect or does not use a stream to calculate the total | The category $ amount label is updated with the correct amount. The total was calculated using streams whenever a new category is selected. |  |
| Selecting a category | There is no utility to select a category | When the category is selected, the products shown is updated to just the products for that category | All of level 1 and the sort selection is correct. (i.e. if it was alphabetical, the list will be displayed alphabetically) | All of level 2 and the first item in the list is selected and the image is updated to the product selected |
| Sell a unit | There is no utility to sell a unit | If a product is selected, a unit can be sold, but the GUI does not update. The code protects from selling more units than you have in stock. | Level 1 plus the view of the products shows the updated units available. | Level 2 plus both the total inventory value and category values decrease by the value of the product |
| Sorting | There is no utility to sort the products | There is at least 4 sort options (see examples above) | Level 1, plus the sort order is maintained whenever the GUI display of the product list is updated. | Level 2 plus the sort is being performed by a stream and the .sort method. |
| Submission | A GitHub link was not submitted or it created a 404 error because JaretWright was not made a collaborator (or it is not a private repository). The project is sent with any form of compression other than zip.  Code requires a local SQL database and the sql files are not provided. | A zip file was sent with the project and/or a local SQL DB is required. The complete SQL commands are sent in a file and documented in the comments of the submission. |  | A GitHub link to your private repository was submitted. JaretWright was made a collaborator on the repository |

A bonus of 3 marks will be granted if after creating an object, it is added to a list and the user is taken to a new Scene showing all of the objects created. An example is shown below.

## What to hand in November 30 (prior to 11:59 pm)

Submit your Product and Inventory classes by sending the link to your private GitHub repository. If you have other files in the project that is fine. I just want to ensure that your models make sense and that you have started the project. This will be marked as a lab. Remember to make JaretWright a collaborator in GitHub for the repository. This will count as a lab and give you opportunity to get my feedback prior to the final submission.

## What to hand in December 9 (prior to 5 pm)

Submit your GitHub link to your private repository containing ALL necessary files to upload in Intellij. Remember to make JaretWright a collaborator in GitHub for the repository.