Faculty of Engineering and Information Technology
Computer Science Department
COMP2311 Project (Phase 1)

In this documentation, you will find all the information you need to develop your project.

<< في هذه الوثائق ، ستجد كل المعلومات التي تحتاجها لتطوير مشروعك >>

- The `Inventory` class in this project has lots of methods, but many of them are similar.

- The constructor should just make the category equal to the parameter value.

- In the two `newItem` methods, start with a call to the `findItem` method with the second argument `true`. If this returns `-1`, add a new Item or new Brand to the inventory. Make the code tight by chaining[1] the setQuantity and setPrice method calls after an anonymous new Item or new Brand instantiation in the add argument as the following examples:

  Example 1: inventory.add (new Item(type)).setQuantity(quantity).setPrice(price)
  Example 2: inventory.add (new Brand(brand, type)).setQuantity(quantity).setPrice(price)

- In the `setQuantity`, `setPrice`, `getQuantity`, and `getPrice` methods, start with a call to the `findItem` method with the second argument `false`. If this **does not** return `-1`, set or get the quantity or the price in or from the object at that location. To do this, you'll have to use `ArrayList`'s `get(<index>)` method. If `findItem` returns `-1`, in each one of these **get** methods, do the following:

  1. In the one-parameter `getQuantity` method, **return -1**.
  2. In the two-parameter `getQuantity` method, **return 0**.
  3. In either of the `getPrice` methods, return **`Double.NaN`**[2].

- Start the `update` methods with a call to the `findItem`, as in the set methods above, and call the appropriate `update` method in the `Item` class only if `findItem` does not return `-1`.

- In the **`findItem` methods**, you have to create at least two variables : `index`, and `itemsFound` where the `index` is initialized to `-1` and `itemsFound` to `0`. Then, loop through the entire

[1] Method Chaining is the practice of calling different methods in a single line instead of calling other methods with the same object reference separately. For more information visits https://www.geeksforgeeks.org/method-chaining-in-java-with-examples/

[2] NaN is a numeric data type value which stands for "not a number". For more information visits https://www.baeldung.com/java-not-a-number

`inventory`, and whenever there is a match between the specified type or the specified brand and specified type, set `index` to that object's index, and increment `itemsFound`.

- In the three-parameter of the `findItem` method, you'll have to make the comparison with something like the following:

```
item = inventory.get(i);
if (type.equals(item.getType()) &&
  item instanceof Brand &&
  brand.equals(((Brand) item).getBrand()))
```

- After the loop, if `itemsFound` is zero and `warningIfFound` is `false`, print a "cannot find" error message. If `itemsFound` is not zero and `warningIfFound` is `true`, print an "already exists" error message. In the two-parameter method, if `itemsFound` > 1, print a "found more than one brand" message. In either method, if exactly one match was found, return its `index`. Otherwise, return −1.

- In the `stockReport` method, use the following format:

```
<type> - in stock: <quantity>, price: $<price using #.##>
...
<brand> <type> - in stock: <quantity>, price: $<price using #.##>
...
<type> - in stock: <quantity>, price: $<price using #.##>
Total value: $<total value using #.##>
```

For example, **stock reports** can be found in the document (Project 1_Phase 1, page 3).

- To get and display the brand name from instances of the `Brand` class, you'll have to include something like the following:

```
if (item instanceof Brand){
  System.out.print(((Brand) item).getBrand() + " ");
}
```

- For the most part, the `Item` class methods and the `Brand` class methods should be easy. The only non-trivial activities are in the `update` methods in the `Item` class. When the parameter is an `int`, the method should increase the quantity by this parameter. When the parameter is a `double`, the

method should multiply the price by 1 + this parameter. (A negative value should automatically decrease the quantity or price.)

- Lastly, the `Comparable` interface implemented by the `Item` class is based on the `price`. Therefore, when comparing two items, the one with the highest price is considered to be the larger item.

Further assistance:

- warningIfFound : Boolean variable may you need to use to print error messages on the screen. In the findItem method, this variable will be used as a second argument. See the following example:

```
if (itemsFound == 0 && !warningIfFound) {
   message = "Cannot find " + brand + " " + type;
}

if (itemsFound > 0 && warningIfFound) {
   message = brand + " " + type + " already exists";
}
```

Best of Luck