Project 2

CS 1302 - Fall 2022

In this project, you will be refactoring and extending the Inventory Management System. The starter currently implements (and tests) most of the features required for the system, and you will refactor the system to make use of inheritance and polymorphism. Therefore, your grade will be dependent on proper use of inheritance and polymorphism (as described in the appropriate tasks) as well as requiring the system to function as expected.

Grading Deductions

As you work, you should follow the expected feature branching style with appropriate commit messages, testing, and clean coding practices. You will lose one point from your final score for each of the following:

- Improper Commit Message: Each commit message that does not follow expected style (brief title, blank line, description).
- Unclean Code: Each Checkstyle warning (excluding test code).

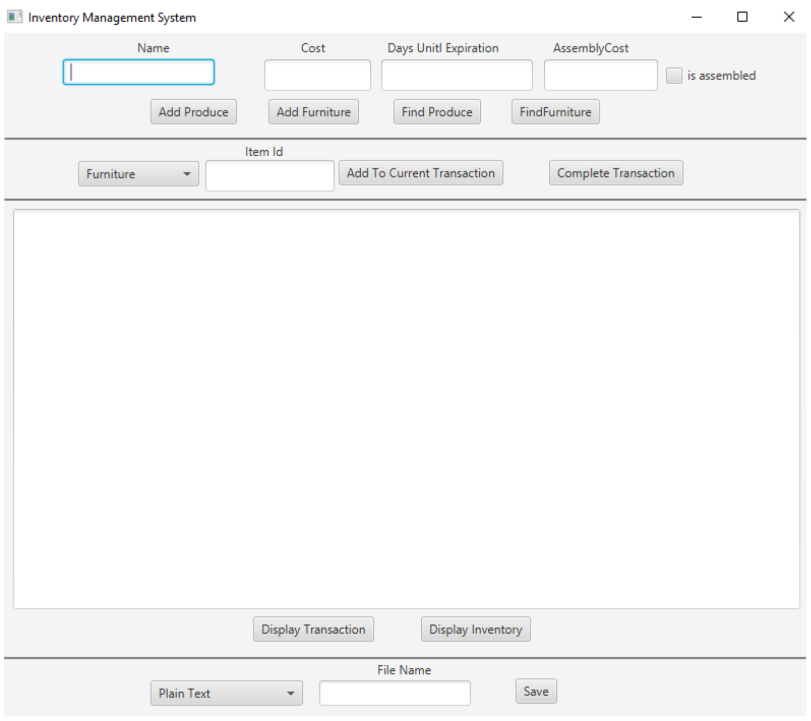
Failing to submit your project as a Git repository will result in a 10 point deduction.
Remember that all code, other than in the codebehind and simple getters/default constructors, must be tested. I will not always prompt you to add unit tests, you should add these tests as they are needed. Failure to test a method properly will result in points deducted from that task.

Submission

You should submit your project as a zip archive containing the project's Git repos.

Getting Started

1. Run the project and ensure that you can see the following GUI:



A - Establishing the `Product` Inheritance Hierarchy

1. Add a new abstract class named `Product` to the `edu.westga.cs1302.inventory_management.model.products` package.

2. `Furniture` and `Produce` should inherit from `Product`.

3. Add an abstract `getCost` method to `Product` (should have no parameters and return an `int`).

4. Move common methods/fields from `Furniture` and `Produce` to `Product`.

5. Ensure that all existing code works as expected.

B - Utilizing Polymorphism and `Products`

1. In `Transaction`…
    a. Replace the existing set of ArrayLists (for `Produce` and `Furniture`) with a single ArrayList that stores `Product`s.
    b. Replace the existing set of adder and getter methods with a single addProduct and getProducts method.
        i. You should add a new test class for the `addProduct` method.
        ii. You should remove the (now) unused test classes for `addProduce` and `addFurniture`.
    c. Update all other existing methods in `Transaction` to make use of the new field as appropriate.
        i. You may need to update existing tests as well.

2. We have now added a new concern for our GUI. How do we serialize the products stored in a transaction? The current system expects that we can detangle the furniture and produce objects, but this is no longer possible. To correct this issue we will need to grow the public interface of the `Product` hierarchy.
    a. Add a new abstract method named `serialize` to the `Product` class.
        i. Accepts an `InventorySerializer` as input
        ii. returns a `String`
    b. Implement serialize in both `Produce` and `Furniture` by calling the appropriate method of the `InventorySerializer`.
    c. You should add Test classes for `Furniture::serialize` and `Produce::serialize`.
    Note: The methods called from `InventorySerializer` *have existing unit tests that completely test the behavior expected for* `Furniture::serialize` *and* `Produce::serialize`. *We redefine these tests to ensure that changes in expected behavior for* `InventorySerializer` *that violates expected behavior in* `Furniture` *and* `Produce` *should cause tests to fail for* `Furniture::serialize` *and* `Produce::serialize`.
    Note2: Since `Transaction`*s are no longer able to distinguish between* `Produce` *and* `Furniture`*, the serialization of* `Transaction`*s can't ensure that all* `Produce` *come before all* `Furniture`. `Product`*s will be serialized in the order that they are stored.*
    d. Alter `InventorySerializer::serializeTransaction` to call use `Product::serialize`.

3. In `view.MainWindow`, update existing behavior to utilize the updated public interface for `Transaction`.

4. Ensure all existing code works as expected.

C - Establishing the `Serializer` Inheritance Hierarchy

1. Add a new interface named `Serializer` to `edu.westga.cs1302.inventory_management.model.inventory_serialization`

2. `InventorySerializer` should implement `Serializer`.

3. Move `InventorySerialize::serializeInventoryToFile` to `Serializer`, make the method default, and delete the method from `InventorySerialize`.

4. Add an abstract method stub for the following methods to the abstract `Serializer` class:

    a. `serializeProduce`
        i. Accepts a `Produce` object as input
        ii. Returns a `String`

    b. `serializeFurniture`
        i. Accepts a `Furniture` object as input
        ii. Returns a `String`

    c. `serializeTransaction`
        i. Accepts a `Transaction` object as input
        ii. Returns a `String`

    d. `serializeInventory`
        i. Accepts an `InventoryManager` object as input
        ii. Returns a `String`

5. Add an abstract method stub for `Serializer::toString()`

6. Rename `InventorySerializer` to `PlainTextSerializer`. NOTE: You should use the rename refactoring tool in eclipse to complete this task.

D - Utilizing Polymorphism and `Serializers`

1. Change the type of the `MainWindow::fileType` field to `ComboBox<Serializer>`, but leave the initialization of items as it is (initially only contains and default set to be a `PlainTextSerializer`).

2. Change the type of the parameter for `Produce::serialize` and `Furniture::serialize` to `Serializer`.

3. Add a new class named `XmlSerializer` to `edu.westga.cs1302.inventory_management.model.inventory_serialization`

4. `XmlSerializer` should implement `Serializer`.

5. Test and Implement the `serializeXXX` methods to provide XML serializations of the appropriate objects. Do not override `Serializer::serializeInventoryToFile` in `XmlSerializer`. Below is the format for each type of object.

Produce Format

<Produce id=`id` name=”`name`” cost=`cost` expirationMonth=`expiration_month` expirationDay=`expiration_day` expirationYear=`expiration_year` />

Furniture Format

<Furniture id=`id` name=`name` cost=`cost` assembled=`assembled` assemblyCost=`assembly_cost` />

Transaction Format

```
<Transaction>
Produce-or-Furniture_items_each_starting_on_a_new_line
</Transaction>
```

Inventory Format

```
<Inventory>
Produce_items_each_starting_on_a_new_line
Furniture_items_each_starting_on_a_new_line
Transaction_items_each_starting_on_a_new_line
</Inventory>
```

6. Implement an `XmlSerializer::toString()` method that returns the text “`XML`”.

7. Update `MainWindow::initialization` to add an XmlSerializer to the ComboBox (default should still be plain text).