Inventory Tracker Phase 3

Undo/Redo, Reports, Designing for Testability

Objective

In this phase you will design and implement Undo/Redo and Reports. This will give you experience implementing several design patterns that we have studied, including Layers, Command, Visitor, and Builder. You will also gain experience creating a testable design.

Design

Your design should cover the following areas:

1. Undo/Redo
2. Reports
3. Product Statistics Report Test Plan

Be careful to avoid code duplication in your design. Undo/Redo will be implemented in three different views. Rather than duplicating code unnecessarily, you should create common infrastructure that can be used in all three views. Similarly, there are five different reports to be implemented. You should look for ways to share common functionality between the different reports rather than duplicating code.

Undo/Redo

Design how your program will implement Undo/Redo in the following views:

- Add Item Batch View
- Remove Item Batch View
- Transfer Item Batch View

You should base your design on the Command pattern.

➢ <u>For this area you should provide:</u>

1) <u>UML class and sequence diagrams that show how you will implement Undo/Redo</u>
2) <u>Javadoc documentation for the classes involved in your Undo/Redo design</u>

Reports

Design several classes in your core object model that will implement the following reports:

- Expired Items
- Removed Items
- N-Month Supply
- Product Statistics
- Notices

Your design should use the Visitor and Builder patterns.

As in Phase 2, you should use the iText library to generate PDF files. HTML files are easy enough to generate without special library support.

As part of this phase you will design and implement an automated test suite for your Product Statistics Report implementation (see the next section). Therefore, your implementation of this report should be designed to support automated testing. In your design, explain how you will implement the Product Statistics Report so that you can run it on a variety of different test cases, and programmatically verify that the data in the report is correct.

- For this area you should provide:
  1) UML class and sequence diagrams that show how you will implement Reports
  2) Javadoc documentation for the classes involved in your Reports design

Product Statistics Report Test Plan

Create a test plan for your Product Statistics Report. Make a list of all of the test cases that you will implement to ensure that the contents of the report are always correct. Use the principles of equivalence partitioning and boundary value analysis when designing your test cases. Here are some examples of the kinds of test cases that you should include in your test plan:

Test 1: Test a report period that includes a transition from Standard Time to Daylight Savings Time

a. *The transition occurs in the middle of the report period*
b. *The transition occurs on the first day of the report period*
c. *The transition occurs on the last day of the report period*

Test 2: Test a report period that includes a transition from Daylight Savings Time to Standard Time

a. *The transition occurs in the middle of the report period*
b. *The transition occurs on the first day of the report period*
c. *The transition occurs on the last day of the report period*

Test 3: Test a product that was initially added to the system during the reporting period (to make sure that the days before the product was created are not counted in the product's statistics)

a. *The product was added in the middle of the reporting period*
b. *The product was added on the first day of the reporting period*
c. *The product was added on the last day of the reporting period*

Test 4: Test a product that was initially added to the system before the reporting period (to make sure that the days before the reporting period are not counted in the product's statistics)

a. *The product was added many days before the reporting period*
b. *The product was added the day before the reporting period*

Design Division of Labor

The work of documenting your team's design should be divided up as follows:

Undo/Redo – 1 person
Reports – 1 person
Product Statistics Report Test Plan – 1 person

If your group has 4 people, use the extra person as you wish.

Implementation

Implement your Undo/Redo and Reports designs.

Implement your Product Statistics Report Test Plan. Use JUnit to automate your tests. Put your test code in the `test-src` directory of your project (like you did in Phase 1). Use the `ant coverage` command to run code coverage reports, and augment your test cases until you achieve complete line and branch coverage on the code that implements your Product Statistics Report (or as close as you can get).

Make sure that your program compiles and runs successfully when invoked with the `ant run` command (which is how the TA will run your program). Make sure that all of your test cases compile and run successfully when invoked with the `ant test` and `ant coverage` commands (which is how the TA will run your test cases).

Use Subversion to share files with your teammates. Check in your code early and often, but don't break the build. Notify your teammates when you commit new code.

Remove errors reported by Checkstyle. For each phase, you have 5 free Checkstyle errors that you can ignore without penalty. For Phase 3 you can have a total of 15 unresolved errors. This is for cases where you disagree with Checkstyle, and don't want to follow its advice. Checkstyle will not be graded until after phase pass off. You have until then to remove Checkstyle errors. Your test cases do not need to pass Checkstyle.

Implementation Division of Labor

Undo/Redo – 2 people
Reports – 2 people
Product Statistics test cases – Everyone should help implement your test plan

Deliverables

- ☐ Group Design
  - UML diagrams (submit PDF file online)
  - Javadocs (submit zip file online)
- ☐ Group Implementation (send zip file to your testing team and the TA)
  - All Inventory Tracker functionality implemented (except Product Auto-Identification)
  - Product Statistics Report test cases implemented and working
  - All files checked into Subversion
  - Everything should build and run successfully

  - Remove Checkstyle errors
  - Create a zip file containing your `inventory-tracker.jar` file, the `lib/` directory, the `images/` directory, and any other directories or files needed by your program. It should be possible for a classmate or TA to unzip your zip file and run your program with the following command: `java –jar inventory-tracker.jar`
    - Use the ant script to generate your `inventory-tracker.jar` file rather than creating it by hand. This will ensure that it is created correctly.
    - Include the `lib/` directory (and everything in it) in your zip file because `inventory-tracker.jar` depends on the libraries in that directory
    - Include the `images/` directory (and everything in it) in your zip file because `inventory-tracker.jar` depends on the image files in that directory
    - Test your zip file on both Windows and Linux to ensure that it will work for others when they try to use it
  - **<u>Send your implementation zip file to your testing team and the TA</u>**
- ☐ Individual Phase Report / Exam (submit PDF file online)