Inventory Tracker Phase 4

Advanced Persistence, Product Auto-Identification

Objective

In this phase you will design and implement Advanced Persistence and Product Auto-Identification. This will give you experience programming with relational databases, and an opportunity to implement the Data Access Object, Abstract Factory, Chain-of-Responsibility, and Plug-In design patterns. This phase will also give you experience using polymorphism to create flexible, extendible designs.

Design

Your design document should cover the following areas:

1. Advanced Persistence
2. Product Auto-Identification

Advanced Persistence

Until now, your program has used Java serialization to persist the core object model to and from disk. The general approach has been:

1. At startup, load the core model from disk into memory
2. During execution, perform all operations on the in-memory core model (i.e., changes occur only in memory)
3. At shutdown, save the core model to disk.

You will now enhance your program to support both serialization and database persistence. At startup, the user will specify the type of persistence they want to use (serialization or database). Your program must be able to run using either type of persistence. If the user selects serialization persistence, your program should run just as it does now (as described above). If database persistence is selected, your program should run as follows:

1. At startup, load the core model from the database into memory
2. During execution, perform all operations on the in-memory core model, but also immediately propagate all changes made to the core model to the database (i.e., keep the in-memory model and the database in synch at all times).
3. At shutdown, nothing needs to be saved, because all core model changes have already been reflected in the database.

There are several design challenges here. First, since your program currently does not implement database persistence, you will need to do so. Second, you need to think hard about how you will make your program support both persistence modes in a clean way. To accomplish this, do the following:

1. Design the SQL database schema that you will use to persist your core object model in a relational database. Specifically, create a `.sql` text file containing the SQL commands for dropping and creating the tables in your schema.

2. Apply the Data Access Object design pattern to create a design that allows you to implement database persistence while keeping all database code (JDBC, SQL, etc.) out of your core object model.

3. Enhance your design to support both serialization and database persistence. (HINT: Use Polymorphism and the Abstract Factory design pattern)

- For this area you should provide:
  1) A `.sql` text file containing the SQL commands for **dropping** and **creating** the tables in your database schema.
  2) UML class and sequence diagrams that show how you will implement Advanced Persistence
  3) Javadoc documentation for the classes involved in your Advanced Persistence design

Product Auto-Identification

Until now, when the user entered a new product for the first time, they had to manually enter the product's description. You will now implement Product Auto-Identification. Rather than requiring the user to manually enter product descriptions, your program will search the Web to automatically determine the identity of a new product and fill in its description (if possible).

There isn't any one site on the Web that you can search to identify all possible products that a user might enter into Inventory Tracker. Rather, there are many sites that provide partial product databases that can be searched. Therefore, it is necessary to search many different data sources on the Web until one of them successfully identifies the product in question. It is also possible that none of the searched sites will be able to identify the product, and the user will have to enter the product's description manually. This problem can be solved effectively using the Chain-of-Responsibility design pattern.

Additionally, since the Web is very dynamic, new sites that provide product information will frequently become available, and old sites that we have searched in the past will disappear. This implies that the Product Auto-Identification feature should be configurable and extendible even after the software ships to customers. This problem is effectively solved using the Plug-In design pattern.

Create a design for implementing Product Auto-Identification. Your design should be based on the Chain-of-Responsibility and Plug-In design patterns. It must be possible to register new plug-ins through an external configuration file, without having to re-compile the entire program.

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

Design Division of Labor

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

Advanced Persistence – 1 or 2 people
Product Auto-Identification – 1 or 2 people

Implementation

Implement your Advanced Persistence and Product Auto-Identification designs.

Inventory Tracker must be able to run in either serialization or database persistence mode. The user should be able to run in database mode by specifying the `-sql` command-line option, or in serialization mode by omitting the `-sql` command-line option. (The TA will use the `ant run` command to run your program in serialization mode, and the `ant run-sql` command to run your program in database mode. Make sure that these commands work properly.)

Each person on your team should implement at least one plug-in that searches a Web data source for product descriptions. Also, it must be possible to register new plug-ins through an external configuration file, without having to re-compile the entire program.
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 4 you can have a total of 20 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.

Implementation Division of Labor

Everyone should help implement Advanced Persistence. It is important for everyone to get some hands-on experience doing database programming with SQL, JDBC, and the DAO design pattern.

For Product Auto-Identification, each person on your team should implement at least one plug-in that searches a Web data source for product descriptions.

Deliverables

- ☐ Group Design
  - UML diagrams, SQL database schema (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
  - All files checked into Subversion
  - Everything should build and run successfully, including your JUnit tests
  - 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
    - **Be sure to include all required database files**
    - **Be sure to include your plug-in registry configuration file**
    - Test your zip file on both Windows and Linux to ensure that it will work for others when they try to use it
  - **Send your implementation zip file to your testing team and the TA**
- ☐ Individual Phase Report / Exam (submit PDF file online)