CS 635 Advanced Object-Oriented Design and Programming
Spring Semester, 2010
Assignment 3
Assignment Index
© 2010, All Rights Reserved, SDSU & Roger Whitney
San Diego State University -- This page last updated 3/26/10

In-Memory Database with Persistence

Due April 13

1. We will create an in-memory database for a video store inventory. The video story sells movie DVDs. Each movie has a name, price, unique id and a quantity. The store uses sequential integers for unique ids. We need to be able to

   - Add new movies.
   - Sell a movie in the inventory.
   - Add new copies of existing movies
   - Change the price of a movie
   - Find the price and/or quantity of a movie by either name or id.

   Create an Inventory class (you may need other classes) to keep track of the store inventory.

The problem with Inventory class as a database is that it does not persist. Once your program stops running all data is lost.

2. Use the memento pattern to copy the data in an Inventory object. Make the memento serializable so it can be saved in a file. Given an Inventory object and a memento you can restore the Inventory object to a previous state. Given that a program may have many references to the Inventory object you can't just replace the Inventory object with the memento.

So now we can periodically create and save a memento of the Inventory object. But the memento can become rather large. (Inventory is just an example. If we were doing this for real our data could be 100's of megabytes.) So we would not want to save it after each operation.

3. For each operation that changes the state of the Inventory object create a command. Make the commands serializable. Keep in mind that we don't want to serialize the Inventory object each time we serialize a command. Also when we deserialize a command object we will have it operation on the Inventory object that is in-memory, which is likely not to be the same Inventory object that the command first operated on.

Now every time we perform an operation on an Inventory object, we can create a command, perform the command and save the command to disk. This way we will have a history of all the operations. If our program were to crash we can recover the last state by first loading the last memento and then replaying all the commands done since the last memento was created. Since the commands will always be small, which is not the case with the Inventory object, saving it to disk each time will not be very expensive. After a while the number of commands may get very large. When this happens one can create a new memento, save it to disk and remove the old commands. Of course this needs to be done in a safe manner. That is one must make sure that the new memento is saved on disk before removing the old one and removing the old commands.

4. Create a proxy(s) for Inventory objects. For every operation that changes the Inventory object's state the proxy will create the command, perform the command and save the command to a file.

However having to create those commands each time we want to perform an operation can be annoying. The proxy should make this transparent to the client code.

5. Instead of creating proxy(s) you might be tempted to make the Inventory class create the commands, execute the commands and save them. Why is the proxy a better idea.

Grading

Item	Percent of Grade
Working Code	20%
Unit Tests	10%
Proper implementation of Patterns	60%
Quality of Code	10%