Jama Supermarket Implementation Notes

The technical problem was a simple one, so I used it as an opportunity to demonstrate some of the technologies listed in the job listing. It uses Spring for dependency injection and injects either a simple Hashtable backed DAO (HashProductDao by default) or a database backed DAO (DbProductDao -db command line option). The DAO instances are created via factories defined for each DAO type (HashProductDaoFactory for creating a HashProductDao and DbProductDaoFactory for creating a DbProductDao) and these factories are exposed via an abstract factory (ProductDaoFactory) that creates instances of the DAO factories. The factory methods are implemented via Spring configurations.

The database mode uses Hibernate and MySql (a createdb.sql script to create the database and DB user is provided). The database will recreate itself each time the program is run unless a line is commented out of the hibernate config file after the first run (this is noted in the config file). I demonstrated unit testing with mocking in the NWEA demo so I focused instead on these preceding areas.

The Hibernate entities include a Product and an abstract Sale which is extended by the XForThePriceOfYSale. The table per class inheritance strategy was used. The Product and Sale entities are one-to-one for simplicity, although a many-to-many relationship would be more realistic.

Deletions of products aren't supported, although the default A, B, and C products can be supplied with new prices and sales (and existing sales can be removed) via the -newItems command line option which is how additional products are also added.

Command line example using the database, updating A, B, and C values, and adding an additional D product:
ACDABACABCCBD -db -newItemsA:10:4fpo3,B:20,C:30:3fpo2,D:40

A $10 (4 for the price of 3)
B $20
C $30 (3 for the price of 2)
D $40

4 A items
A total = 30 (4 at discount)

3 B items
B total = 60 (3 at regular price)

4 C items
C total = 60 (3 at discount) + 30 (1 at regular price) = 90

2 D items
D total = 80 (2 at regular price)

Total = 30 + 60 + 90 + 80 = 260

Usage: java Supermarket <ProductsString> [-db] [-newItems<<X1:X1Price[:YfpoZ]>,...,<Xn:XnPrice[:YfpoZ]>>]
	Where -db uses a database
	-newItems<<X1:X1Price[:YfpoZ]>,...,<Xn:XnPrice[:YfpoZ]>> is a comma separated list of colon separated triplets of Character item names, int item prices, and optional Y for the price of Z sales.