

Week 12 Practical Class Exercises

Supermarket Docket (Reprise)

Objectives

This week's exercises are based around the *Supermarket Docket* problem discussed in the lecture in week 5. The design process is the same, but we will use Java as the implementation language this time around. The main aim is to gain more experience with the design and organisation of object-oriented Java programs.

The Java classes `Product.java`, `DB.java`, `Docket.java` and `Test.java` for the initial version of the program are available from the course website.

Exercise 1 (Hash Table Database)

In the program developed in lectures, the database of products (`DB.java`) was represented as a simple list, `ArrayList<Product>`. To find a particular barcode we searched through the list from first to last. For a realistically large database, this kind of searching would be unacceptably inefficient — imagine looking up a name in a telephone directory that was organised in no particular order. Searching large sets of records is a very common application of computer technology, so a good deal of work has gone into developing data structures (such as trees and hash tables) that can be searched efficiently.

In COMP1110/1510 and later courses you may learn more about how these data structures behave. In this exercise, our aim is simply to *use* the hash table implementation provided in the standard Java 1.5 library `java.util.Hashtable` instead of a simple list. In essence, you will need to modify `DB.java` to replace the `ArrayList` type with a hash table, and modify the `find` and `add` methods to operate on the hash table rather than the list.

To begin, you will need to use the on-line Java API to find out about `java.util.Hashtable`, its constructors and methods. The documentation contains much more information than you will comprehend or need. An important part of this exercise is for you to practice sifting through the details to find the information that is relevant to your needs.

To help get you started, the *keys* of the hash table will be the barcodes and the *values* will be objects of class `Product`. Only `DB.java` should need modification.

Exercise 2 (Tracking Stock)

Get a fresh copy of the original Java classes for the basis of this exercise. Do not use the hash table version you developed in the previous exercise.

A natural extension of the supermarket product database is to include some information on the current level of stock for each product. That is, the `Product` class will include another field

representing how many items of that particular product the supermarket currently holds. Add such a field to `Product.java` and modify the constructor method accordingly.

If the *current stock* field is to remain consistent with reality, it should be decremented for each sale. Add a method to `Product.java` which reduces the current stock by one. What do you think should happen if the current stock falls below zero?

Add a method *Product sell(Integer barcode)* to `DB.java`, which is similar to the `find` method in that it searches for and returns the product with the given barcode, but also reduces its current stock field by one.

An obvious use of the current stock information is to generate orders from the supplier when the remaining stock of an item falls below a particular level. Add a method to `DB.java` which scans through the entire database and returns a list of all products with a current stock less than 12.

(Optional advanced activity: Consider how you might do this if the database was represented using a hash table as in the previous exercise. As you will see, how we represent search tables is an important choice.)

You will need to make some modifications to `Test.java` to test your work.