PROJECT OVERVIEW

This DB app does some back-offices admin tasks on the company's products including:

- list all current products
- add a new product
- delete an existing product
- update the quantity in stock for a particular product.

It uses a general purpose programming language program (e.g., C#/ Java/…) to access your A2/A3 DB (with slight modifications) using batch-processing (to facilitate testing & output-capture for the demo deliverable). `Log.txt` output file contain: s 1) the input user request, 2) the resulting program-constructed SQL statements and 3) the query results (or update reassurance or error message) for each request in `A5UserRequests.txt` file.

PROGRAM STRUCTURE

Modular programming is expected, including these CLASSES (and others, as needed), each in a physically separate file:

- The **`main` program** is the overall controller which:
  - o sets things up, as needed
  - o displays a "BEFORE" picture of the relevant parts of the DB **
  - o controls the processing of the user requests (although it doesn't actually carry out the processing itself) *
  - o displays an "AFTER" picture of the relevant parts of the DB [using same procedure as step 2] **
  - o does any final cleaning up, as needed.
- The **RequestHandler** class contains individual methods to handle each type of user request. These methods construct the appropriate sqlString, based on the data provided to them, and echo both the request and the sqlString to the Log file.
- The **DbAccessor** class methods actually communicate with the DB server to retrieve or update the data, sending the resulting DB data or "reassurance message" to the Log file.

SOME NOTES

* Processing the user request file - A single transaction request is read in, then handled (by the appropriate requestHandler method), based on transCode (`L, A, D, U`), with control then returning to main's processing loop. Each request is handled independently from subsequent transactions – i.e., there is no add-loop or update-loop or …

Errors? - The user request data will not contain errors, per se, in terms of bad data types, bad record format, bad transCodes, etc. However, **Add** requests may have a DBS error – e.g., adding a product where there's already an existing ProductID – but Delete and Update requests will always be correct (to simplify this project).

** BEFORE & AFTER pictures both do:

```
SELECT * FROM Product ORDER BY ProductID
SELECT PONo, ProductID FROM PurchaseOrder ORDER BY PONo
SELECT * FROM OrderDetails ORDER BY OrderID, ProductID
SELECT SupplierID, Name FROM Supplier ORDER BY SupplierID
```

THE DATABASE

The DB is from A2 & A3 with a few modifications. This project focuses on Product table – but because of FK constraints, it also concerns Supplier, PurchaseOrder & OrderDetails tables.

Fix the DB schema:

- `default` values for Product table:
  QuanInStock: 20, ReorderPoint: 5, ReorderQuan: 20
- all PK/FK's are appropriately defined on the 4 tables of concern
  (as they should already have been in A2 & A3)
- use `CASCADE DELETE` for PurchaseOrder's FK (ProductID)

Fix DB loader: Insert 1 additional SUPPLIER (besides what's there already):

```
S540, NY Winters, 1903 Mich St., 2693871000, 60603
```

A "fresh copy" of the DB

- Manually (in SQL*Plus, using command files) fix those tables affected by this project so they're back like they ORIGINALLY were BEFORE A3 changed them. You'll also need to use the command file to RE-fix the tables AFTER you start testing your A5 code, so the actual DEMO starts with a fresh copy of the DB. [I don't need to see this "re-loading", per-se, in the demo. I'll see its results when looking at `main`'s BEFORE picture].
- `disable` the CheckQuan trigger set up in A3.

4 USER REQUEST HANDLERS

LIST All Products

Displays a "report" showing all products

- in sequence by ProductName
- with 1 product per line (with attributes in the same order as shown in the table)

This is not a fancy report, but use reasonable spacing across the page, including

- some spacing between fields
- vertical alignment of fields (use COURIER FONT when printing in NotePad or WordPad)
- make sure there's NO WRAP-AROUND

ADD A New Product

Inserts a new product (just 1)

Caller supplies:

- `ProductID` – if it already exists in the table, show:
  >> ERROR: existing ProductID instead of the reassurance message
  [NOTE: check DB table, do not keep any list of ProductID's in program itself]
- `ProductName` - enter as all CAPS
- `Price`
- `SupplierID`

Default values defined in the DB schema [as stated in DB section above]:
QuanInStock: 20, ReorderPoint: 5, ReorderQuan: 20

Reassurance message: >> OK, product added

DELETE A Product

Removes 1 product from DB (even if there's still current stock or orders)
    & MAY also result in tuples being deleted from
        PurchaseOrder table & OrderDetails table

Caller supplies: `ProductID`

Product table's ProductID is a FK in:

1) PurchaseOrder table
   - – DBS should automatically handle the deletion of the tuple(s) in this table because it's FK says CASCADE DELETE
2) OrderDetails table
   - manually handle this in the program itself:
     a) print the OrderID & Quantity (for this ProductID) in the Log file
        [NOTE: there may be 0 or 1 or >1 rows]
     b) delete the affected tuples in OrderDetails table
     c) THEN delete the tuple in Product table

Reassurance message: >> OK, product deleted

UPDATE A QuanInStock

Increases 1 Product's QuanInStock by the designated amount.

Caller supplies: `ProductID, AdditionalStock`

- NOTE: this is ADDITIONAL stock to be ADDED to QuanInStock.
It is NOT a NEW TOTAL QuanInStock

Reassurance message: >> OK, product updated

# PROGRAM STYLE/FORMAT # # # # # # # # # # # # # #

Good program style/format expected, including:
- modularization (classes/methods or functions/procedures)
- visual (physical) separators between modules (e.g., comment line of all *'s & blank lines)
- module size (method/procedure) <= 1 page (generally)
- `main` method/procedure shows the "big picture" of what's going on without doing much actual work itself - it calls other modules to do most of the detail work
- self-commenting code, including descriptive variable & descriptive module naming
- align/indent to visually show code's logic
- nesting no more than 2 (or so) levels deep (generally)
- appropriate amount of commenting for a simple program – e.g.,
  - an intro comment for the program as a whole, with at least your name, class, program name & very brief description of programs functionality
  - a top-comment on each physical file including at least your name, program name & very brief description of the class/module
  - comments on code which does something tricky or handles things in unusual way
  - don't manually comment detail things which are already self-commenting
- same "naming pattern" as DB itself does for variables for attribute storage
- same naming as design specs use

# WHAT TO HAND IN# # # # # # # # # # # # # # # # #

Packet to hand in (assembled in this order):
    Log file (printed out from NotePad or WordPad)
        With YOUR NAME(s) PRINTED ON THE TOP RIGHT CORNER
    MainProgram class
    RequestHandler class
    DbAccess class
    [other classes (?)]

DO NOT EDIT THE Log file - THAT WOULD BE "ACADEMIC DISHONESTY"
EXCEPT:
    -to make sure you're printing out in a fixed-width font (like Courier New)
    -and perhaps to use landscape mode and/or a smaller font so there's no wrap-around.