OOP with Java - DB Assignment Briefing

COMSM0086

Dr Simon Lock & Dr Sion Hannuna

Before we begin

Thank you for your Blue feedback

Probably worth spending a little time addressing

Some of the more frequent comments...

Common Experience

Many new concepts to take in during first few weeks
Struggling to understand these in the time available
(Common experience, but not _always_ the case !)

Would like more practice using these new concepts

```
Structure of C practical exercises was:    aaa bbb ccc
Structure of Java exercises is more like:  abc abc abc
```

All core/essential materials have now been delivered
Practicals focus on applying and practicing concepts
(Plus a few extra 'value added' sessions on patterns)

Specific Requests and Responses

Can we have more sample solutions for exercises ?

Will work through a solution to OXO on Thursday !

Some extra challenges to push 'ambitious' students

Plenty of additional challenges in the assignments !

Can we have guidance on writing "good" test cases

Will run a session focusing on this topic next week !

Specific Requests and Responses

Little bit more practical lab contact time each week

In future we'll do less briefing, more practical time

I don't really find the briefing sessions that useful

In future we'll do less briefing, more practical time

That said, let's now do a briefing on the assignment

That is probably a fairly worthwhile use of our time !

The "DB" Assignment

Aim of this exercise is to build a database server...
...from the ground up !

A complex application for you to practice your Java
A chance to explore DB content from other units
As well as gain experience using a query language

This assignment WILL contribute to your unit mark
The weighting for this particular assignment is 40%

Overview of Server Operation

Your database server must operate as follows:

1. Receive incoming requests from a client
   (conforming to a standard query language)
2. Interrogate & manipulate a set of stored records
   (maintained as a persistent collection of files)
3. Return an appropriate message back to the client
   (Success or Failure, with data - where relevant)

Data Storage

A database consists of a number of 'tables'
Each 'table' consists of a number of 'columns'
Each 'table' contains 'rows' that store 'records'

The 'tables', their 'columns' and 'record' content should be stored as TAB separated text files

example-table.txt
example-table.tab

Record IDs

First (0th) column in a table is always called 'id'
Numerical value that uniquely identifies record/row
ID values are automatically generated by the server
The IDs act as primary keys for each record
Relationships between records use ID as reference
Note that the ID of a record should NEVER change
(or you risk breaking relationships !)
No "recycling" (don't reuse the IDs of deleted rows)

Communication

A minimal skeleton server is provided for you
(so you don't need to worry about networking)

Server listens on port 8888 and passes incoming messages to the 'handleCommand' method
Your task: implement 'handleCommand' internals !

For simplicity, assume only a single client connects
(i.e. there is no need to handle parallel queries)

Query Language

Clients communicate with server using "simple" SQL

- USE: changes the database we are querying
- CREATE: constructs a new database or table
- INSERT: adds a new record (row) to a table
- SELECT: searches for records that match a condition
- UPDATE: change existing data contained in a table
- ALTER: change the column structure of a table
- DELETE: removes records that match a condition
- DROP: removes a specified table or database
- JOIN: performs an inner join on two tables

Defining the Query Language

To help specify our simplified query language
We have provided an "augmented" BNF grammar
BNF

We hope that you appreciate this grammar
It was a significant challenge to create and refine !

There is also a "transcript" of typical queries
Provides examples of queries you might expect to see
transcript

Error Handling

Your parser should identify errors within queries:

- queries that do not conform to the BNF
- queries with "operational" problems (see workbook)

Your response to the client must begin with either:

- [OK] for valid and successful queries
  (followed by the results of the query)
- [ERROR] if there is a problem with the query
  (followed by a human-readable message)

Use exceptions to handle errors *internally*
However these should NOT be returned to the user

Testing

A command-line client has been provided for you
This is really just for demonstration purposes

Development should make use of automated testing
A template test script is found in the Maven project
Add all of your automated JUnit tests there

Testing should target the 'handleCommand' method
(Since this is where your code has been inserted)

Assessed Elements of Assignment

- Functionality: implement required SQL commands
- Error handling: dealing with erroneous queries
- Robustness: keeping server running at all times
- Flexibility: coping with variable white spacing
- Code quality: using appropriate structure & style

REMEMBER

You will only get marks for the code that YOU write

Avoid Collusion

This is an individual assignment, not group activity
Automated checkers used to flag possible collusion
If markers feel collusion has indeed taken place...
Incident is referred to academic malpractice panel

May result in a mark of zero for assignment
or even the entire unit (if it is a repeat offence)

Questions ?

SQL whitespace variability

As with any programming language (including Java) SQL can contain extra whitespace and still be valid
Your interpreter needs to be able to cope with this

For example, let's consider this INSERT statement:

```
INSERT INTO marks VALUES('Steve',55,TRUE);
```

How many variants are there with 1 extra space ?

all-possible-variants

generate-variants-script

Dealing with these can be a bit tricky !

Would you like a few extra pointers ?

(Warning: May Contain Metaphors)

How do you chop an onion ?

The "Computer Scientist" Approach

Select the newest, "most cutting edge" knife

Cut onion into two halves (binary chop)

Iterate through columns first (slice)

Iterate through rows next (dice)

After each cut, return chopped
onions into a heap (or stack)
(to keep working area clear)

How does that sound ?

Oh dear me NO !

You need to pull the onion out of the ground first !

Then remove roots (and any remaining soil)
Chop off the green leaves
Remove the outer layers of skin
Maybe even give it a wash
Throw it away if it looks rotten !

Only THEN try to chop it

The Problem

We had assumed that the onion was "ready to go"

But the farmer and seller have done a lot of work...

`Washing, Head and Tailing, Grading, Filtering etc.`

All before you get your hands on the onion

How does this relate to parsing SQL ?!?

Don't just "dive in" and start "slicing and dicing"
Pre-processing will save you a LOT of time & effort

We can do simple filtering / cleaning / scrubbing
In order to "normalise" the incoming commands
Get them into a standard size/shape - like onions !

If everything is similar and consistent...
It will make writing a parser a WHOLE LOT easier

BasicTokeniser