OOP with Java - DB Assignment

COMSM0086

Dr Simon Lock & Dr Sion Hannuna

DB Assessed Exercise

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

A complex application 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 assignment is 40%

Overview of Server Operation

Your database server should:

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'

Tables are stored in 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
IDs act as primary keys for a record
Relationships between records use IDs as references
Note that the ID of a record should NEVER change
(or you risk breaking relationships !)
No "recycling" (don't reuse IDs from 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 is to implement `handleCommand`

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

BNF Grammar

To help illustrate our simplified query language

We have provided an "augmented" BNF grammar

Hope you appreciate this - was challenging to create

BNF

There is also a "transcript" of typical queries

Provides examples of queries you might expect to see

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 in the maven project
Add all of your automated JUnit tests there

Testing must target the `handleCommand` method
Test content of responses, not exact formatting
(different people use different table layouts)

Agile

A key principle of Agile is providing value to client
Through *early* and *regular* delivery of features

Emphasis is on "Steady & Sustainable development"
(No "all-nighters", No "heroic effort")

Doing all the work just before deadline is not Agile !
(And really upsets team mates and team leaders)

It is important you gain experience of Agile working
You'll be assessed on "Agileness" of your process

GitHub

In order to get insight into your dev. process...
You will need to invite us to your GitHub repo

This will allow us to assess your WHOLE process
Not JUST marking the end product (the final code)

Investigate patterns of development work
Assess the accumulation of successful features

Identify occurrence of possible plagiarism
We have analysis tools to detect unusual behaviour

Your Process

You should commit and push to your repo regularly
After each coding session: "before you eat or sleep"

The master branch should *always* be operational
(that version of code should always be "runnable")

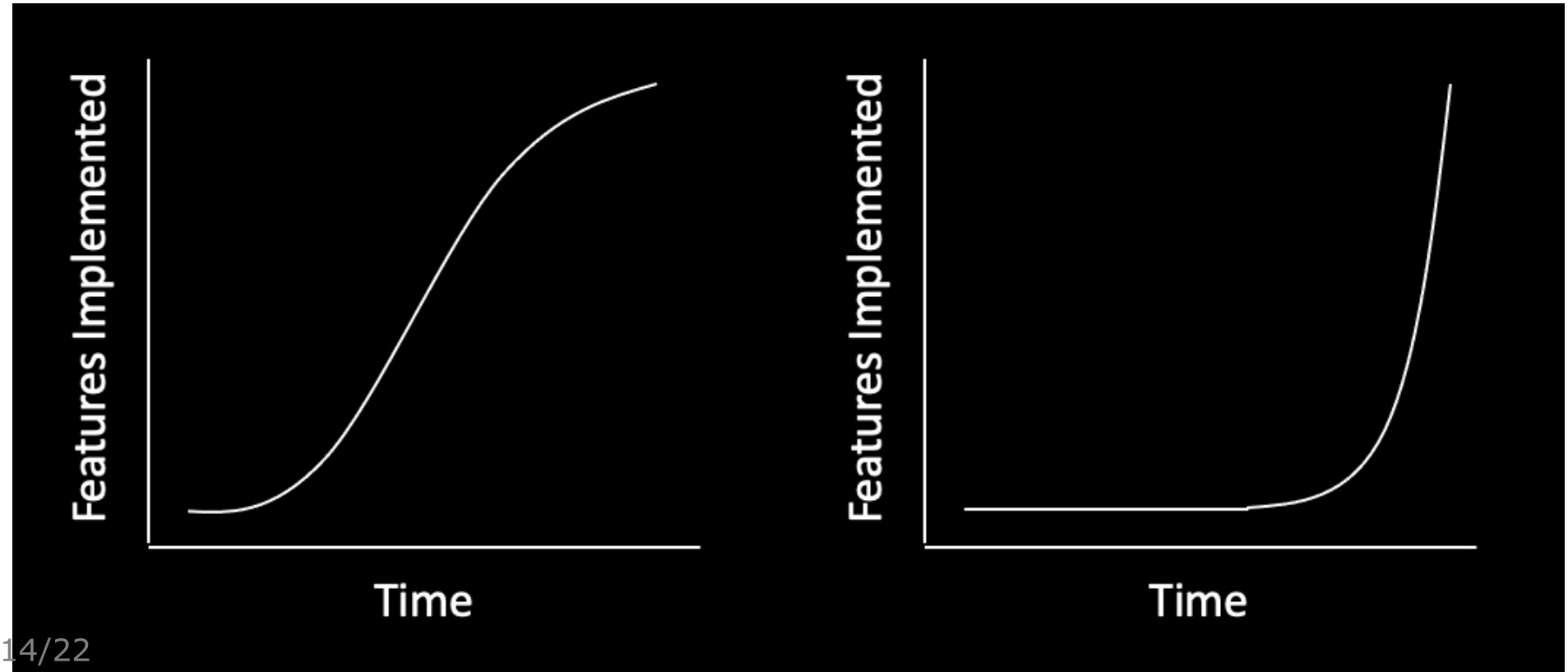
Your code has to be ready to run "out of the box"
(clone master branch and run server with maven)

What if you're working on version that doesn't run ?
Keep committing on a regular basis (track changes)
Only push commits to repo when code DOES run !

Accumulation of Implemented Features

Your aim is to achieve the graph illustrated on the left

And avoid situation shown in the graph on the right !



Assessed Elements of Assignment

- Functionality: implementing specified SQL commands
- Error handling: dealing with erroneous queries
- Robustness: keeping the server running at all times
- Flexibility: coping with variable white spacing

- Code quality: using appropriate structure and style

- Commit regularity: commit & push to your repo often
- Agile delivery: steady accumulation of features
- Repo content: only appropriate content on GitHub
- Cont. integration: keeping master branch operational

SQL whitespace variability

SQL can contain extra whitespace and still be valid
Just like any prog. language (including Java !)
Your interpreter needs to be able to cope with this

Let's consider the INSERT statement for example:

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

The script below generates a set of valid variants:

generate-variants-script

all-possible-variants

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 ?

Wrong !

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

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

Only then can you chop it

The Problem

We are assuming that the onion is "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 "standardise" incoming commands
Get them into a standard size/shape - like onions !

If everything is similar and consistent...
This will make writing a parser a lot easier

Preprocessing