OOP with Java - DB Assignment

COMSM0103

Dr Simon Lock & Dr Sion Hannuna

OXO Marking

We are just about ready to release OXO marking
Didn't want to do it today - need to focus on DB

Will release them early next week
Then run a debrief session (maybe Tuesday @ 3 ?)

Consensus marking worked really well
Nice broad coverage of implemented features
Felt a lot fairer than previous marking approaches

Good looking overall mark profile
(but more on that next week !)

That said...

We WON'T be using consensus marking for DB exercise

Why ?

A lot of people found it very stressful
(unfamiliarity and uncertainty can be distressing)

Some people have already decided they don't like it
(even before they have seen the marks ;o)
Often a losing battle to trying to change opinion

It took a long time to fix and run the tests !
(many tests to run against many students' code)

Feedback information is less than ideal
(test failure comments written by students)

Approach to marking next assignment

We will use the "old fashioned" marking approach

We have some test cases from previous years

Will dust them off & polish them up for use this year

About 50 tests (give or take)

Will need to keep these under wraps

(would make it too easy to just tick them off)

DB Assessed Exercise

Aim of this exercise is to build a database server...

...from the ground up !

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 35%

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 a TAB separated text files

example-table.txt
example-table.tab

Record IDs

The 1st column in a table is always called `id`

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 !)

Communication

A skeleton server has been 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

BNF

Error Handling

Your parser should identify errors within queries:
- queries not conforming 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

Assessed Elements of Assignment

- Functionality: implementing all SQL commands
- Error handling: dealing with erroneous queries
- Flexibility: coping with additional spacing
- Robustness: keeping the server running at all times
- Code quality: using appropriate structure and style
- Test cases: extent of testing features implemented

Any questions before we go on ?

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

Want a bit of help with that ?

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)

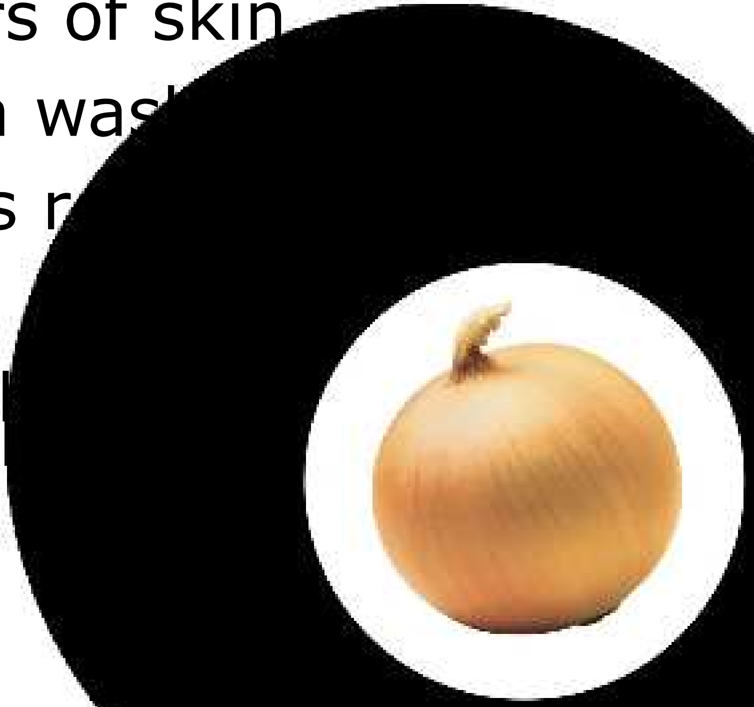
Chop off the green leaves

Remove the outer laters of skin

Maybe even give it a was

Throw it away if it looks r

Only then can you cl



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

Want to see my solution ?

Disclaimer: This isn't a "full parser" solution

It just splits a query into individual tokens

But it does illustrate the point I'm trying to make

Principles need to be applied more broadly

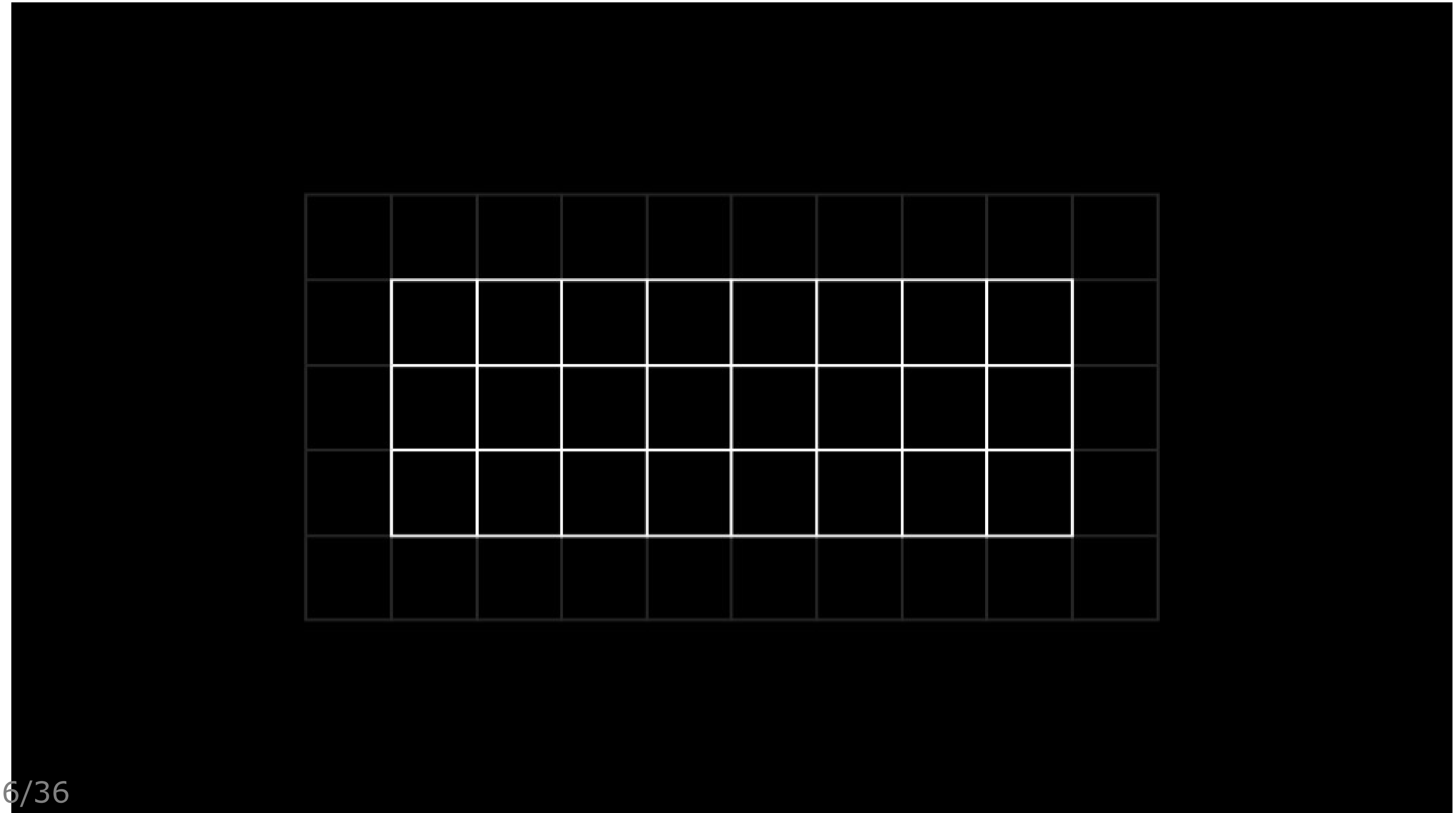
There will be exceptions and edge cases

Different SQL queries my operate differently

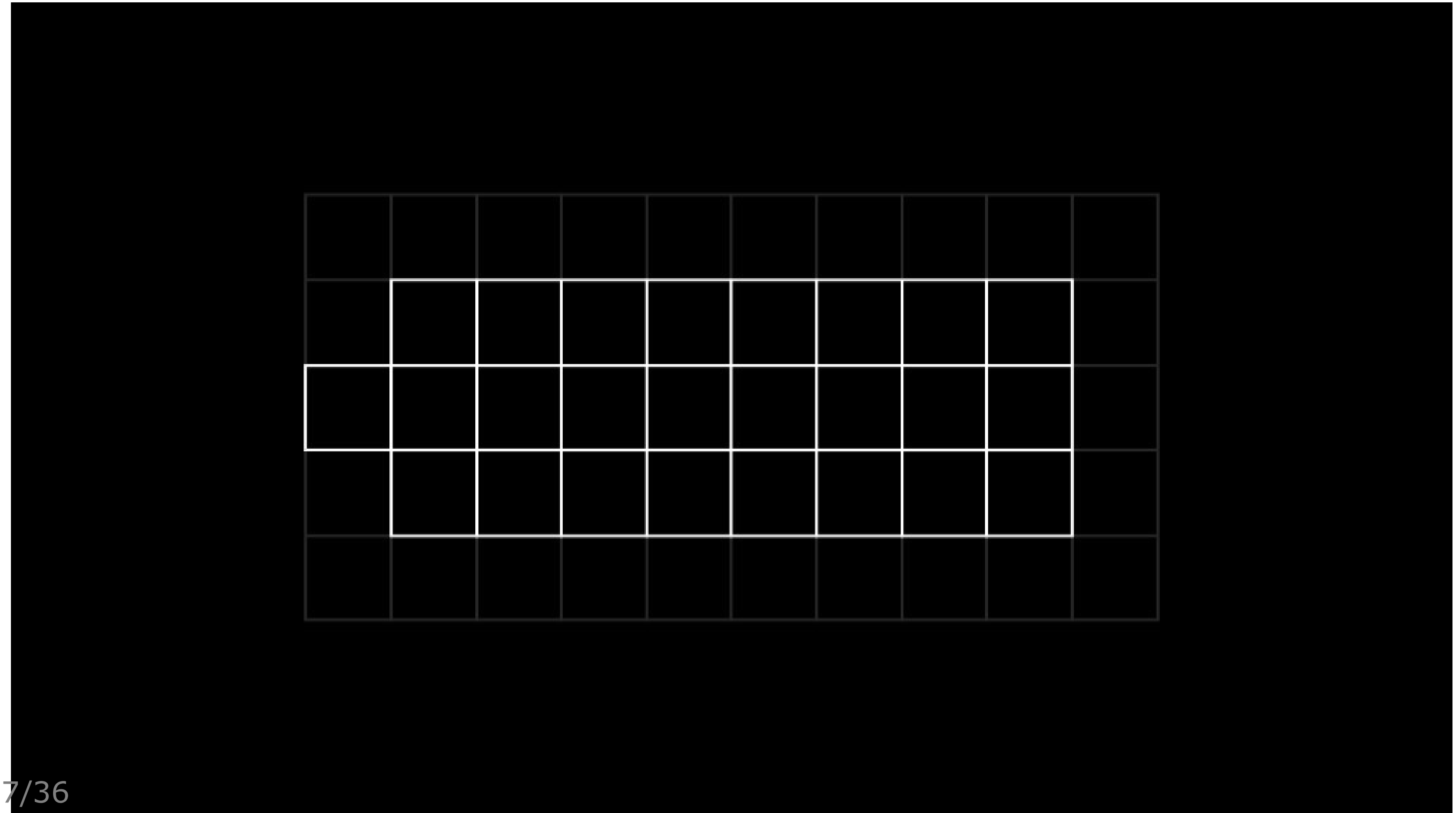
Tokeniser

"Completeness" of assignment brief

Features in assignment brief



Someone will ask: "What about this ?"



Frequently heard conversation

X: What about this feature ? What should we do ?

S: Well it's not something we considered...

X: What's the matter ? Don't you know the assignment ?

S: Well, yes, but we didn't want to put this in scope

X: We can't just ignore it - how should we handle it ?

S: [Being helpful] What do you think is appropriate ?

X: The system should report an error - it's obvious

S: OK, so do that then

Later on the discussion forum...

```
X: S said we must report an error in this situation
Y: What ? But that's not in the "specification" ?
X: Yes, the "specification" is incomplete
Y: I can't believe our code will be tested on this !
X: This is shifting the goalposts !!!

S: Err, hello everyone - this isn't a "core feature"
   We don't actually have any tests for it
Y: Well I'm going to implement it anyway
X: So am I
S: OK, so do we actually need to test this now then ?
   (to make sure it is being implemented correctly)
X&Y: This is SOOOOO unfair !!!!!!
```

As a consequence

Best not to get into discussion of marginal features

Consider the assignment materials as definitive:
If it's not in the brief / BNF / query transcript
It's not a core feature, so doesn't need implementing

If you really must fill a gap (for own satisfaction)
Do what you think is most appropriate

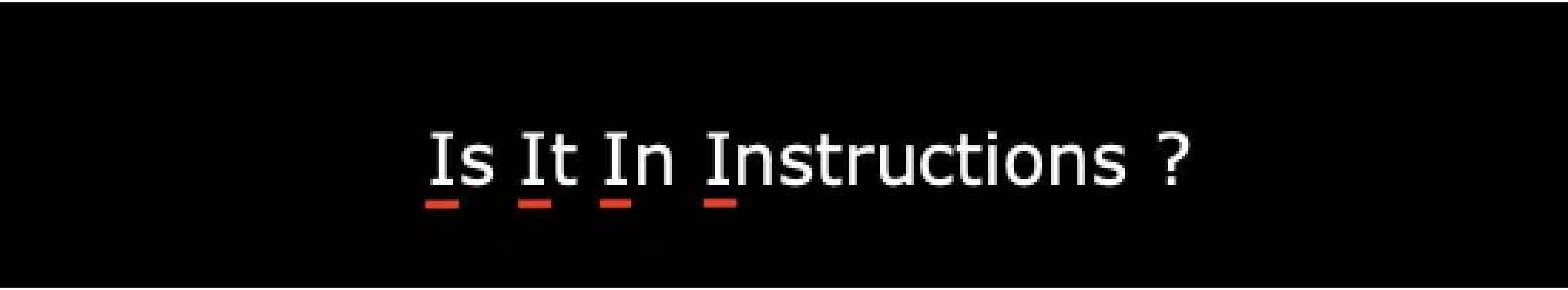
These marginal aspects will not be tested/assessed
(we could have with consensus marking but...)

Introducing a new concept

In answer to your question,

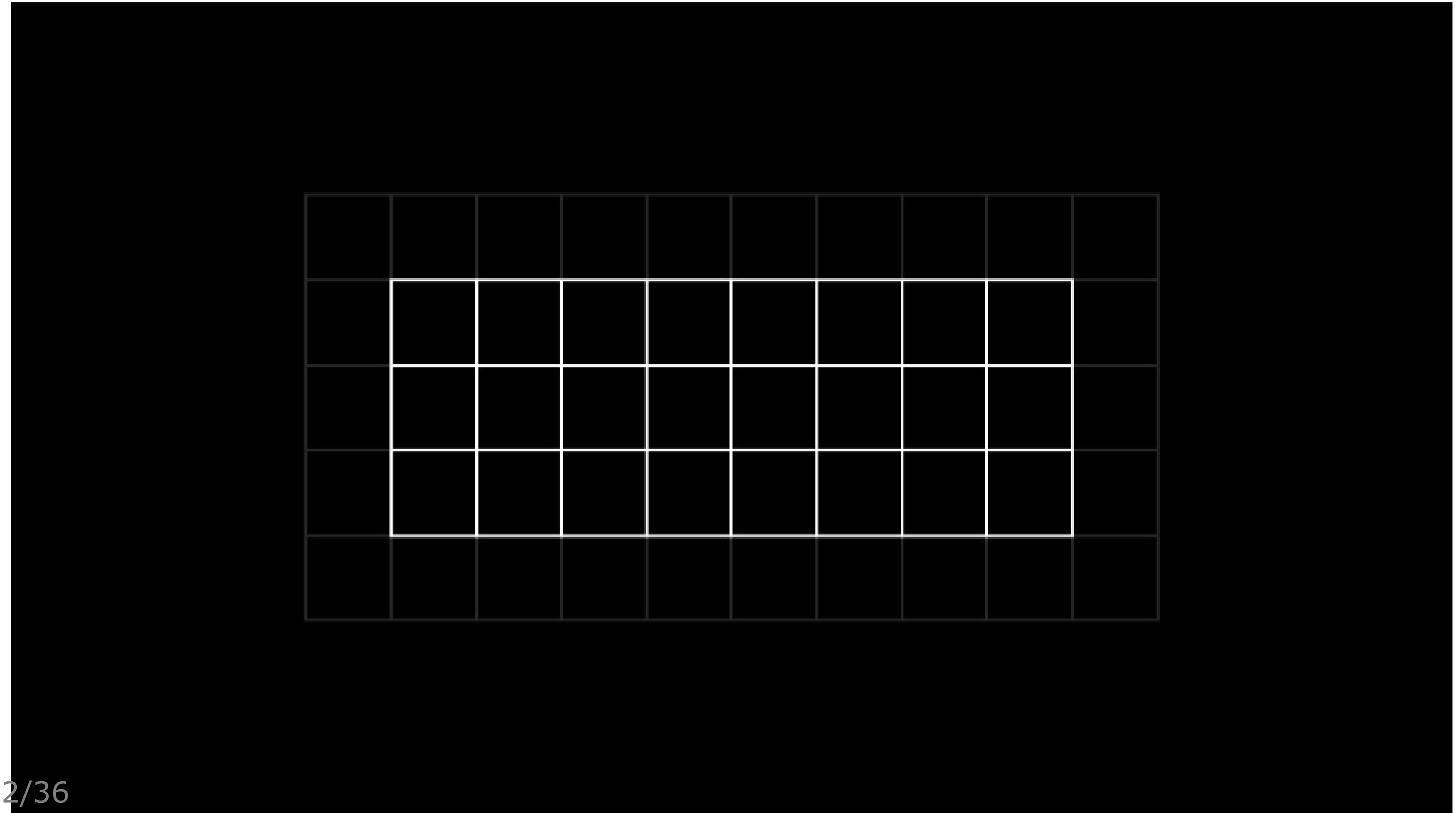
You might get the answer:

I4

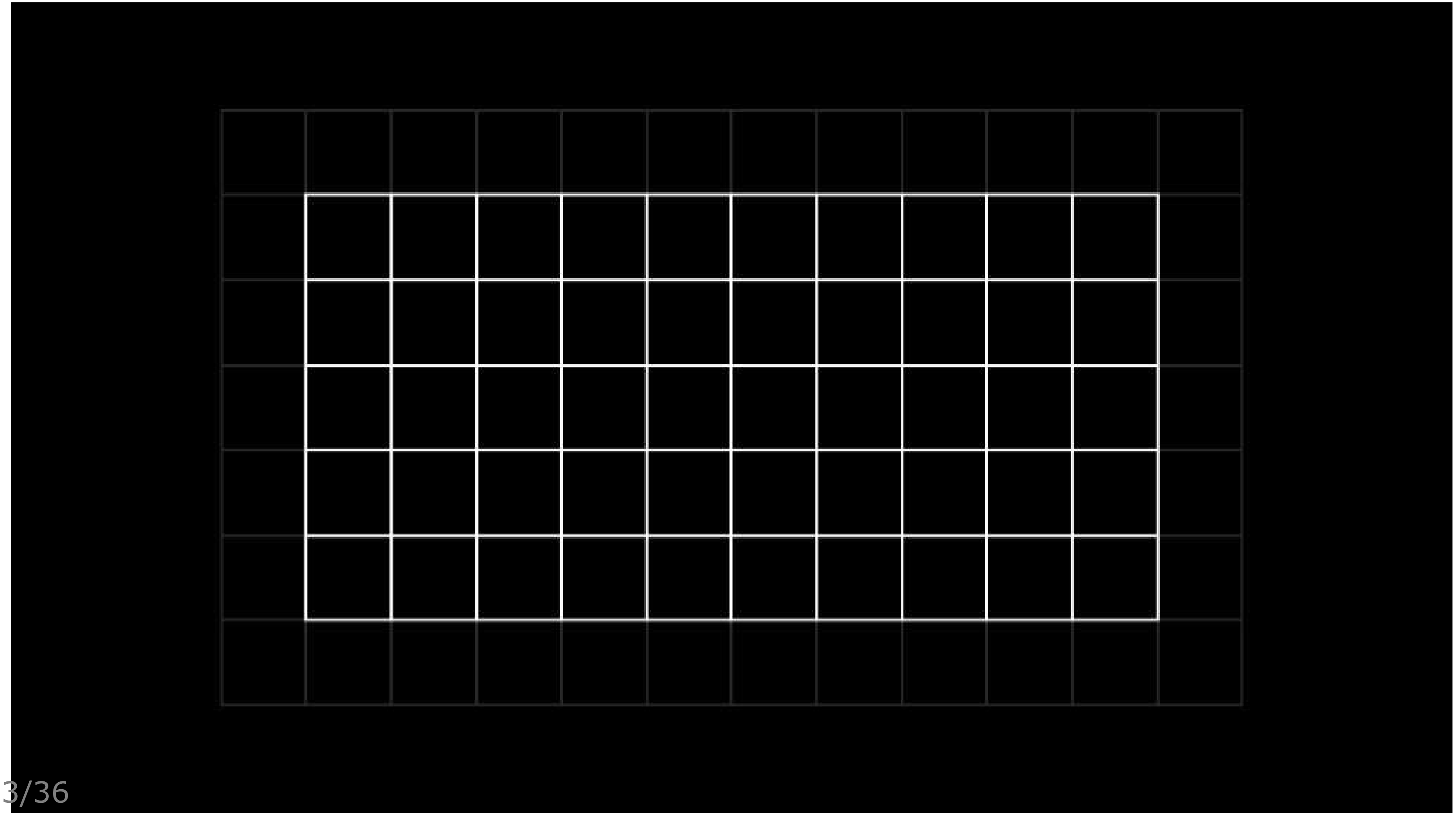


Is It In Instructions ?

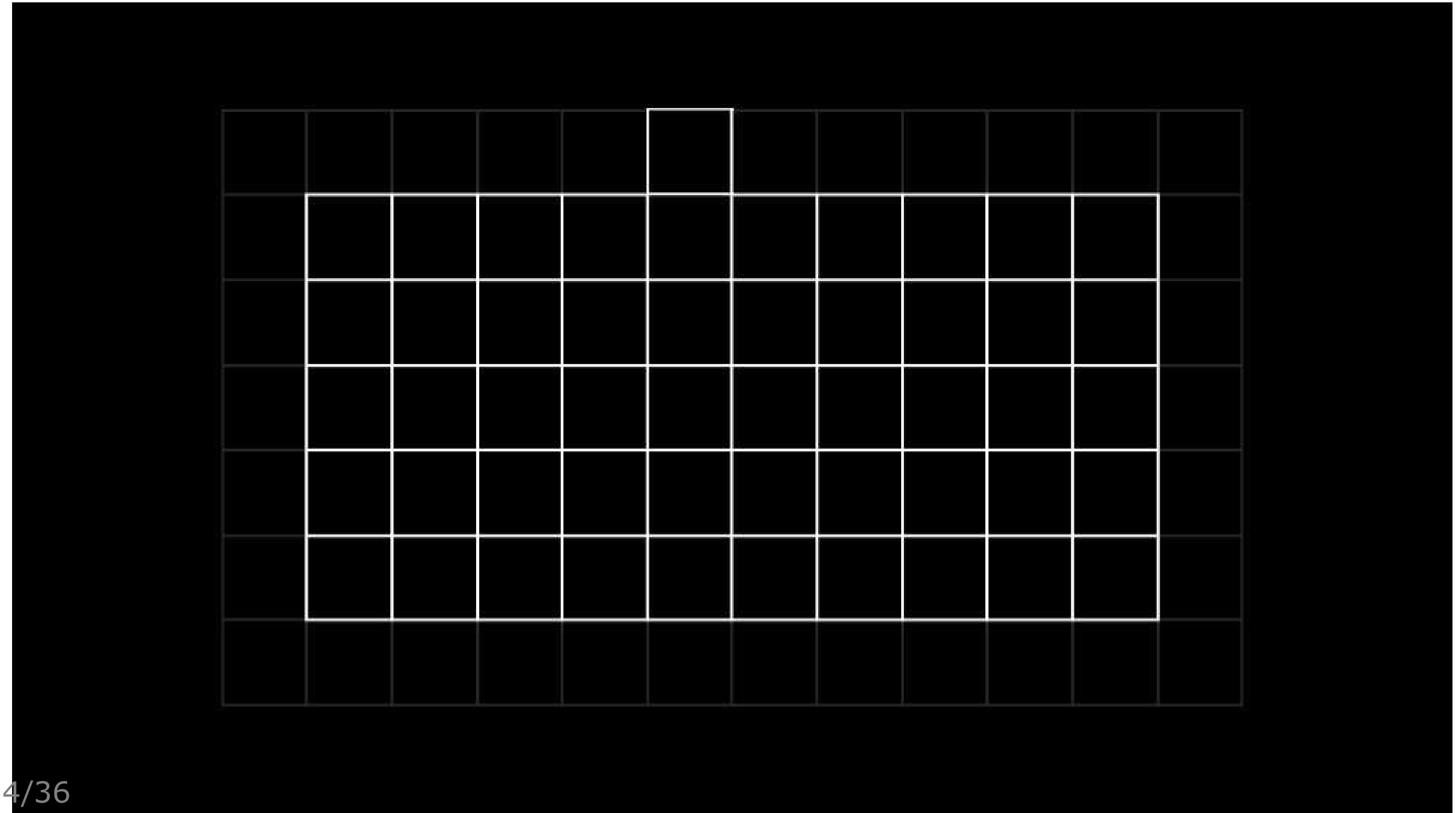
After each year...



After each year...



The following year...



Questions ?

Testing

A command-line client has been provided
This will allow you to manually test your server

Development should however use 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)