MCP Interview Project (Java)

The goal of the project is to build a small interpreter for a JSON based DSL that performs some simple analytics on a data set. This must be java based and can either be implemented as a command line tool, or an HTTP endpoint which accepts the queries. You may use any libraries, frameworks or build tools you wish to accomplish this (or none at all).

You should aim to spend a couple of hours on the project, it's ok if you don't finish.

The Data

The data set is comprised of 3 small csv files, `securities.csv`, `attributes.csv` and `facts.csv`. There are 10 securities and 10 attributes, and for each security and each security, there is one fact. I.e. there is a one-to-many relationship between securities and facts, and attributes and facts.

Securities Schema

column	type
id	integer
symbol	string

Attributes Schema

column	type
id	integer
name	string

Securities Schema

column	type
securityId	integer
attributeId	integer
value	float

The DSL

A query in our DSL has the basic format:

```
{
  "security": <String>,
  "expression": <Expression>
}
```

The security property contains a single string, which is the symbol of a security the user wishes to evaluate an expression for.

The expression field contains a single expression, which the user wishes to evaluate for the chosen security. An expression contains an operator (the `fn` property), and the arguments to that operator (the `a` and `b` properties). The arguments to operators can either be the name of an attribute, a number, or another expression.

You only have to implement one operator, though it should be clear in your solution how it might be possible to extend the interpreter to include additional operators. Please choose one of the following operators to implement:

Operator	Arguments	Behaviour
+	a, b	Adds a and b
-	a, b	Subtracts b from a
*	a, b	Multiplies a and b
/	a, b	Divides a by b

Here are some example queries, demonstrating what each operator looks like and what the different parameters can be:

This one uses the `*` operator and makes use of one attribute name and an integer as its arguments:

```
{
  "expression": {"fn": "*", "a": "sales", "b": 2},
  "security": "ABC"
}
```

This one uses the `/` operator and makes use of two attribute names as arguments:

```
{
  "expression": {"fn": "/", "a": "price", "b": "eps"},
  "security": "BCD"
}
```

This one uses the `-` operator and the arguments are two expressions, which in turn use the `-` operator and attribute names as arguments:

```
{
  "expression": {
    "fn": "-",
    "a": {"fn": "-", "a": "eps", "b": "shares"},
    "b": {"fn": "-", "a": "assets", "b": "liabilities"}
  },
```

```
  "security": "CDE"
}
```