Risk Estimation Coding Challenge

I wrote a simple solution using the latest java LTS JDK: 17.

This document includes 3 main chapters:

- Proposed Solution
- Alternative Solution
- GIT Commands

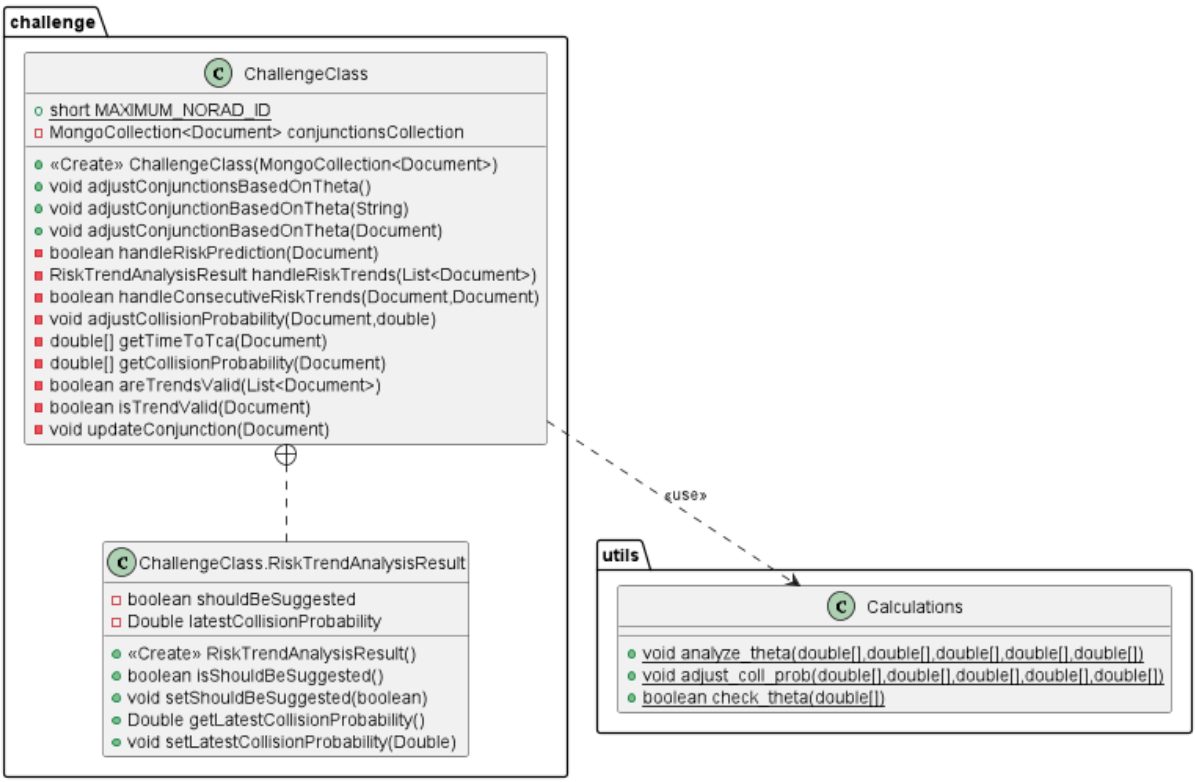
“Proposed Solution” talks about the solution I coded.
“Alternative Solution” describes an alternative solution that I would propose if I were to write actual production code, which is not my actual provided solution here because it’s slightly different than what was described as “Acceptance criteria” in the Challenge description.
“GIT Commands” contains the commands to push to a git repo

Proposed Solution

Assumptions:

- Everything has to be done in a single public class that directly uses the Calculations class with JNI functions
- The DB is accessed directly using the MongoDB java driver
- The risk_trend in “newest_risk_prediction” is a List<Document>: the API example seems to be wrong as it shows it as a List<List<Document>>.
- The “suggested” key’s value is changed only in newest_risk_estimation (and not newest_risk_prediction)
- The software must be single-threaded

UML:



The class “ChallengeClass” should be initialized by calling the constructor with the MongoCollection of conjunctions as the only argument.

The Challenge required me to write one method that would, for all satellites with norad Ids lower than 30000 and with bad theta values:

- Set the newest_risk_estimation to “Not suggested”
- Adjust the risk_trend and finally the newest_risk_prediction’s collision probability by using the “adjust_coll_prob” JNI function with the risk trends of newest_risk_prediction.

ChallengeClass, moreover, makes use of the “Document” class of MongoDB, but in production code I would possibly use a Conjunction class, which would contain all validation logic instead of it being in ChallengeClass.

Since I was not sure about the required inputs for the method, I decided to write the following two methods instead

- adjustConjunctionsBasedOnTheta() → Adjusts conjunctions on all the conjunctions in the provided MongoCollection (supposing that it contains only the conjunctions that have to be analyzed and adjusted, otherwise it would be both risky and potentially complicated to use)
- adjustConjunctionBasedOnTheta(String conjunctionId) → Adjusts the conjunction with the provided conjunctionId

Methods description:

Both methods, internally, do the following:

- Filter out all the conjunctions that are not valid, e.g. a conjunction that does not have the key “newest_risk_prediction” or “newest_risk_estimation is invalid, or a conjunction that has at least a satellite with norad id higher or equal to 30000 is invalid.
- Check for the risk_trend key and verify that it is actually valid, which means that at least 2 consecutive risk trends are present and no values are NaN or null
- For each consecutive risk trend, calculate theta by calling the “analyze_theta” JNI function, then check for bad theta values by calling the “check_teta” JNI function
- If check_teta returns false, it means the value is problematic, hence newest_risk_estimation is set to “not suggested” and the “adjust_coll_prob” JNI function is called and its result is replaced in the second analyzed risk_trend

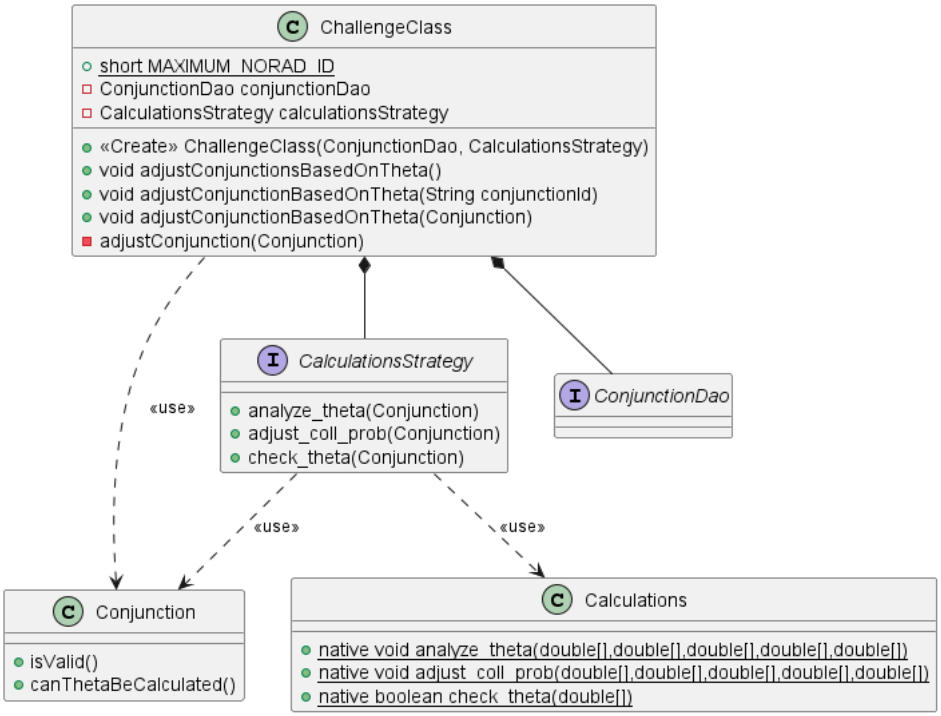
Tests:

I enjoy Test Driven Development, hence I created a couple simple tests in ChallengeClassTest just to make my code runnable in some way, even if missing the implementation of Calculations’ native functions.
The tests make use of PowerMockito to mock the Calculations JNI functions and make use of the real MongoDB (without mocking it) and are pretty straightforward: they mock the Calculations JNI functions and verify whether the MongoCollection has been updated according to the mocked results of the Calculations functions. Using PowerMockito seemed the most straightforward way to test ChallengeClass, however in production code I would probably prefer changing the code a bit to make it more testable: more on this in the “Alternative Solution” chapter.

Alternative Solution

Here I would like a different solution to the problem which I haven't used as the proposed solution because I felt like it diverged too much from the acceptance criteria: it asked to create a single class which used directly the Calculations class and MongoDB.

But if the requirements of directly using the Calculations class and MongoDB is removed, then one could use CalculationsStrategy and ConjunctionDao:



One of the advantages would be better testability and code reusability: regarding the former, there's no need to use PowerMock anymore and Mockito will be good enough, which could be used to mock both CalculationsStrategy and ConjunctionDao.

If needed, the same approach could also be extended to the filters used in ChallengeClass, for example if we want to filter out any NoradIDs higher than 100,000.

Moreover, if the software does not need to be single threaded, some multi-threading code could be written to speed up the adjustments of multiple conjunctions.

GIT Commands

The following git commands can be used to add an origin, adding all required files to the staging area, committing them and finally pushing them, assuming that git credentials are already properly set up.

$origin_url and $branch should be replaced, respectively, by the https url of the remote one would want to push to and by the branch one would like to push to.

```
# Create empty git repo (if needed)
git init

# Add a remote (if needed)

git remote add origin $origin_url

# Add challenge/src and challenge/pom.xml to the staging area
git add challenge/src challenge/pom.xml

# Commit with message (always needed)
git commit -m "Added Challenge Solution"

# Change/rename branch (if needed)
git branch -m $branch

# Pushes to the remote branch (-u needs to be used only the first time)
git push -u origin $branch
```