Introduction

This task is created with intention to test your coding and analytical skills when you're at home, without stress and without any distractions. Do this task at your own pace.

Please read task description carefully and implement needed functionality. Write code by keeping in mind the best code writing and testing practices (we value clean code, clean architecture, good code readability very much). Make sure your code is tested and working according to task description. Write unit tests.

Things you may use to achieve the needed result

- Java 17+
- jUnit 4+
- Framework or library that supports dependency injection.
- Any source on the internet that may help you.

Expected result

Java 17+ project

- Solution hosted on GitHub (preferred) or compressed as zip file.
- Unit tests for the implementation.
- Use build tool such as Maven or Gradle.
- Implementation description.

Task description

Consider this description as a business task description in issue tracking system (e.g. Jira). All the analysis was done by system analysts and following description was created.

Insurance company wants to start issuing bicycle policies to their customers via brokers. REST service documentation swagger documentation is prepared and provided to brokers so they can start to develop frontend applications.

System analysts found out that there will be a policy which will have one or more objects (bicycles) that can have one or multiple risks. In this iteration, customer needs a functionality that calculates sum insured for risks and premium (a price that will be paid by client that will buy this insurance) for the policy.

Premium is calculated by a formula defined in "Needed functionality" section. In short – premium for each risk is calculated according to risk formula. All risk premiums are summed up which represents the object (bicycle) premium. Then all object premiums are summed up which gets us a premium that must be paid by the client.

REST API is needed, input data will be sent to it. No database is needed, functionality should not store any data. It should receive input data, calculate premium and return result.

New groovy rule engine must be created during this task. Risk sum insured and premium calculations must be done in separate groovy scripts for each risk type. To add new risk type in policy 2 groovy rules (sum insured, premium calculation rules) must be added to project, and it should be possible to pass new risk type in API request.

Needed functionality

Please create functionality that calculates policy premium. In this iteration client stated that only three risk types will be calculated – (THEFT, DAMAGE, THIRD_PARTY_DAMAGE), however it may be possible that in near future more risk types will be added.

Response

Object attributes must contain information about bicycle make, model, manufacture year and coverage.

Default values

See swagger documentation

Validations

- Bicycle must be newer than 10 years.
- Sum insured must be less than 10`000.

Risk insurance sum calculation formula

Theft

Risk sum insured = Object sum insured

Damage

Risk sum insured = Object sum insured / 2

Third party damage

Risk sum insured = 100

Risk premium calculation formula

Theft

premium = risk base premium * sum insured factor

Damage

premium = risk base premium * sum insured factor * bicycle age factor

Third party damage

premium = risk base premium * sum insured factor * risk count factor

Risk base premium

To get risk base premium data use riskBasePremium method from base script. Filter data by risk type.

Sum insured factor

To get sum insured factor data use sumInsuredFactorData method from base script. Filter data by risk sum insured. Use factor data to calculate factor by following formula:

factor = factorMax - (factorMax - factorMin) * (valueTo - sumInsuredActual) / (valueTo – valueFrom)

Risk count factor

To get risk count factor data use riskCountFactorData method from base script. Filter data by risk insured object risk count.

factor = factorMax - (factorMax - factorMin) * (valueTo - riskCount) / (valueTo – valueFrom)

Bicycle age factor

To get age factor data use getAgeFactorData method from base script. Filter data by object make, model and age. If there are no matching data repeat search by make and age. If there are no matching data repeat search by age only. Use factor data to calculate factor by following formula:

factor = factorMax - (factorMax - factorMin) * (valueTo - ageActual) / (valueTo - valueFrom)

Groovy rule engine:

Before groovy script execution provided base script must be set as base script. Scripts must return value which represents sum insured or premium.

Acceptance criteria

Request – see attached request.json

Response – see attached response.json

External resources

Swagger documentation – see attached swagger.json file.

Base script – see attached BaseScript.groovy file.