Introduction

This task is created with intention to test your Java 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 readibility 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 8+
- jUnit 4+
- Framework or library that supports dependency injection
- Any source on the internet that may help you

Expected result:

- Java 8+ project
  - Solution hosted on GitHub (preferred) or compressed as zip file
  - Unit tests for the implementation
  - Please 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 private property policies to their customers.
System analysts found out that there will be a policy which will have objects (e.g. a House) and that objects will have sub-objects (e.g. eletronic devices such as TV).
One policy can contain multiple objects. One object can contain multiple sub-objects.
In this iteration, customer needs a functionality that calculates premium (a price that will be paid by each client that will buy this insurance) for the policy.

Premium is calculated by a formula defined in "Needed functionality" section.
In short - formula groups all sub-objects by their type, sums their sum-insured and applies coefficient to the sum. Then all group sums are summed up which gets us a premium that must be paid by the client.

No GUI is needed, policy data will be sent through the API directly to the methods that will be created.
No database is needed, functionality should not store any data. It should receive policy object, calculate premium and return result.

Preferred invocation of the functionality but may be changed if needed:

```
PremiumCalculator#calculate(Policy policy);
```

Needed functionality:

Please create functionality that calculates policy premium

In this iteration client stated that only risk types *FIRE* and *THEFT* will be calculated, however it may be possible that in near future more risk types will be added.
Make sure that it is easy to extend implementation for new risk types.

Premium calculation formula:

```
PREMIUM = PREMIUM_FIRE + PREMIUM_THEFT
```

- `PREMIUM_FIRE = SUM_INSURED_FIRE * COEFFICIENT_FIRE`
  - SUM_INSURED_FIRE - total sum insured of all policy's sub-objects with type "Fire"
  - COEFFICIENT_FIRE - by default 0.014 but if SUM_INSURED_FIRE is over 100 then 0.024
- `PREMIUM_THEFT = SUM_INSURED_THEFT * COEFFICIENT_THEFT`
  - SUM_INSURED_THEFT - total sum insured of all policy's sub-objects with type "Theft"
  - COEFFICIENT_THEFT - by default 0.11 but if SUM_INSURED_THEFT equal or greater than 15 then 0.05

Acceptance criteria:

- If there is one policy, one object and two sub-objects as described below, then calculator should return premium = **2.28 EUR**
  - Risk type = FIRE, Sum insured = 100.00
  - Risk type = THEFT, Sum insured = 8.00
- If policy's total sum insured for risk type FIRE and total sum insured for risk type THEFT are as described below, then calculator should return premium = **17.13 EUR**
  - Risk type = FIRE, Sum insured = 500.00
  - Risk type = THEFT, Sum insured = 102.51

Object entities:

Policy:

Policy can have multiple policy objects and each policy object can have multiple sub-objects.

Policy has 3 attributes:

Attributes	**Notes**
Policy number	e.g. LV20-02-100000-5
Policy status	e.g. REGISTERED, APPROVED
Policy objects	Collection of one or multiple objects

Policy object:

Policy objects can have multiple sub-objects and can be related only to one policy

Policy objects have 2 attributes:

Attributes	**Notes**
Object name	e.g. A House
Sub-objects	Collection of none or multiple sub-objects

Policy sub-object:

Policy sub-objects can be related only to one policy object and have 3 attributes:

Attributes	**Notes**
Sub-object name	e.g. TV
Sum insured	Cost that will be covered by insurance
Risk type	e.g. FIRE, THEFT