TP5 : Introduction to software traceability with Spoon

Exercise 1 : Logging with Spoon for profiling

Question 1

Create a simple application (with a CLI or a GUI) that allows you to :

1. create a user with an ID, name, age, email, and password.
2. provide a user with a menu through which (s)he can :
   - display products in a repository, where every product has an ID, a name, a price, and a expiration date.
   - fetch a product by its ID (if no product with the provided ID exists, an exception must be thrown).
   - add a new product (if a product with the same ID already exists, an exception must be thrown).
   - delete a product by its ID (if no product with the provided ID exists, an exception must be thrown).
   - update a product's info (if no product with the provided ID exists, an exception must be thrown) .

Question 2

1. watch the following videos :
   - Logging in Java
   - Logback vs SLF4J vs Log4J2 - what is the difference? Java Brains Brain Bytes
   - How to do logging in Spring Boot - Brain Bytes
2. choose a logging utility among the ones seen in the videos and play around with it a bit to understand its different facets. **Note**: The usage of Log4j2 or SLF4J is strongly recommended.

Question 3

Use Eclipse JDT or Spoon (recommended) to trace the code of your application using the logging utility chosen in the last question, such that the generated logs can be leveraged to create profiles of users as follows :

- a profile for those that mostly performed read operations on your repository.
- a profile for those that mostly performed write operations on your repository.
- a profile for those that searched for the most expensive products in your repository.

The definition of your profile's structure is up to you. The most important information to include is the user's info, and the marking features of your profile (*e.g., the read/write operations performed by a user for the mostly performed read/write profiles*). The storage format for your profiles and their associated data is also up to you. **Note**: The usage of JSON is strongly recommended due to its lightweight qualities and ease of serialization/deserialization to/from some persistence entity (e.g., file system, database, etc.).

Question 4

Define and execute a sequence of execution scenarios with different users to generate your logs. For example, you can create 10 users, and let each user execute around 20 different scenarios involving the above operations with different input values. Make sure that your scenarios are diverse enough to simulate a real-world experience to have properly crafted profiles at the end.

Question 5

Propose a way to parse the generated logs and extract the required information to construct your user profiles. Note that structured logs require the reification of an LPS and the definition of its construction and printing mechanisms (*e.g.,* `Timestamp: <date_time>, Event: <event_info>+, User: <user_info>+, Action: <method_info>+, etc.`). In this case, each part of the LPS' content can be built separately and then aggregated to form the resulting LPS (*think about the* `Builder` *Design Pattern*). You can also choose to store and display them as JSON files if you wish.

Exercise 2 : Logging with Spoon for debugging (*Bonus*)

Question 1

Use Eclipse JDT or Spoon (recommended) to trace the code of your previously-created application using the logging utility of choice, such that the generated logs can be leveraged to identify the most commonly occurring errors (*thrown and captured*). Make sure to distinguish the traces generated when an error is thrown from those generated when it is captured.

Question 2

Define and execute a sequence of execution scenarios with different users to generate your error logs. Make sure that your scenarios are diverse enough to simulate a real-world experience.

Question 3

Propose a way to parse the generated logs and extract the most common errors sorted in the descending order of their occurrence frequency. Make sure to indicate which user and what action caused the error and for what input values.