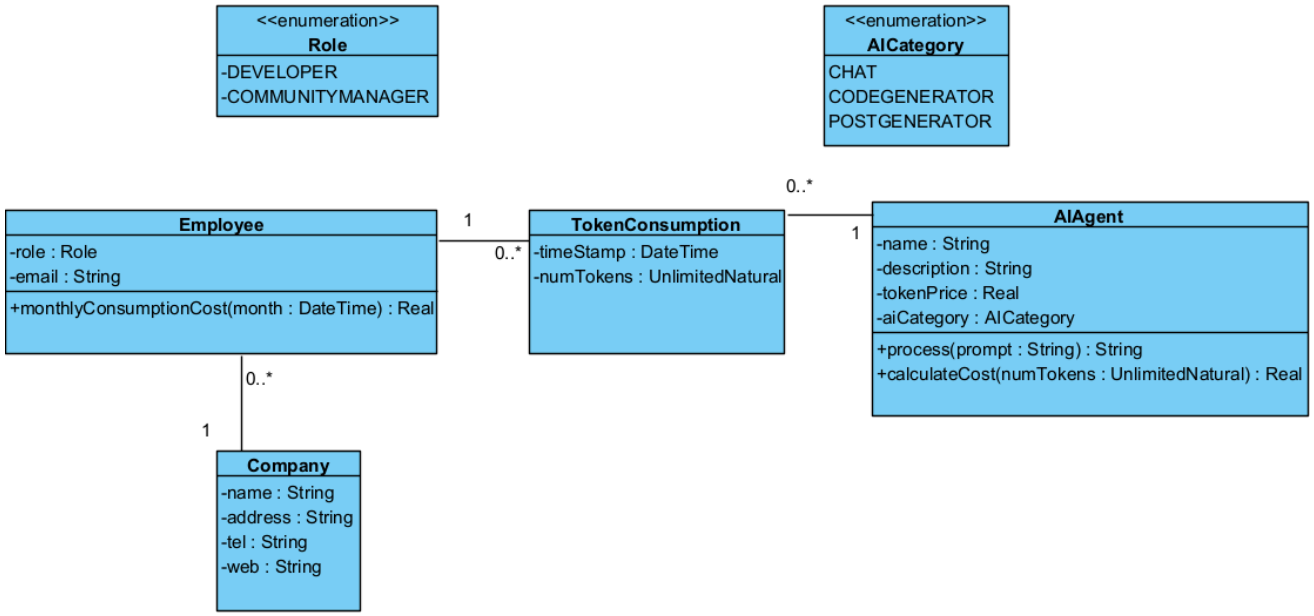
Software Design Patterns

Practice 2: Design and Responsibility Assignment Patterns (evaluated out of 50 points)

In this second exercise, we will add a series of functionalities to the “Guided AI Agents” platform that we modeled in the previous exercise. The class diagram was as follows:



Integrity Constraints

- A Company is identified by its *name*.
- An Employee is identified by their *email*.
- An AI Agent is identified by its *name*.
- Token Consumption is identified by timestamp and the employee's email.

Question 1 (17,5 points)

Statement

Now we want to implement a token quota management mechanism that each employee can consume monthly with automatic alerts and blocking. Furthermore, the quota policy and the notification channel (email, chat, etc.) must be fully configurable and easily extendable. That is, the system must easily allow adding new quota policies as well as notification mechanisms.

Imagine that each Employee has a monthly token quota (MonthlyRate) defined by the company. Every time they use an AIAgent, the tokens consumed are accumulated in their record..

A first quota policy to define is the *ThresholdPolicy*.

- When the employee reaches 80% of their quota, a pre-warning notification must be sent.
- If they exceed 100%, new agent executions must be blocked until the new period begins.
- The warning and blocking percentage rules, as well as the messages and notification channels, should be easily changeable (add new notification channels as well as new percentage calculation or quota limitation strategies) without modifying the basic consumption logic.

In addition, besides the quota management policy based on a single threshold (ThresholdPolicy), they also want us to add another policy that is incremental (IncrementalPolicy). This means that instead of just one warning and one blocking threshold, we will have several warning levels before reaching the block. We are asked to have a list of the following warning thresholds: 60% (mild warning), 80% (ordinary warning), 90% (urgent warning), and finally blocking at 100%.

It is required:

a) (5 points)Indicate which design pattern(s) you would employ to decouple the quota policy from the Employee logic as well as to manage notifications across multiple channels (Email, Slack, SMS, etc.). Remember that it is very important to allow for future extensions of both policies and channels.

b) (5 points) Justify your choice in detail, explaining how the pattern(s) will help meet the requirements of flexibility and extensibility

c) (7.5 points) Class diagram in UML format showing the design pattern(s) applied to solve the quota management only.

Show all the necessary classes, interfaces, and relationships in the application of the patterns. It will be essential to annotate the integrity constraints. It is not necessary to include the entire system diagram, only the new elements or modifications as well as those directly related to the application of the patterns.

Question 2 (7,5 points)

Statement

In quota notification management, we have several warning levels (60%, 80%, 90%) and blocking (100%) that always follow the same flow of steps:

1. Build the alert or error message.
2. Send it by email.
3. Log it.

Each level only differs in the specific content of the message text. You are asked to:

a) (4 points). How would you avoid duplicating the implementation of these three steps and allow each level to define only its specific message? Which design pattern(s) would you use? Justify your answer.
b) (3.5 points) In the future, the system is expected to grow and will consist of several subsystems (quota management, notifications, logging, among others). To simplify the interface used by the presentation layer (or an external client), we want to offer a single service that internally handles obtaining the employee, checking the quota, notifying if necessary, logging, and invoking the agent.

Which design pattern(s) would you apply to group and hide this complexity? Justify your answer.

Question 3 (15 points)

Statement

The project analysts also don't quite understand how the correct implementation of notification policies to users could be carried out, so that they receive both warnings when they have exceeded certain thresholds and are also notified when their monthly quota is exhausted.

When an employee reaches the warning threshold (for example, 80% of their monthly quota) or exceeds their quota (100%), only this employee should receive the notification through the channels they have configured (email, Slack, SMS, etc.).

In addition, the system must allow dynamically registering other recipients of the same notification without changing the basic logic: for example, the team leader, a monitoring service, or a global dashboard. Each time the event occurs (warning or blocking), all registered recipients must be notified automatically.

For the moment, the requirements we have are to allow quota warning and blocking notifications via:

- Email
- Slack

That is, when an employee reaches the warning limit(s), they can be notified by email, and if they reach the quota block, they could additionally be informed by message to their Slack.

It is required:

a) 5 points) Which pattern would you use to correctly implement the notification policy in the system? Justify your answer.
b) (5 points) Class diagram in UML format showing the design pattern(s) applied to solve the implementation of the notification policies.
c) (5 points) Write all the pseudocode for the Employee class method that executes the agent until a user is notified by email.

Show all the necessary classes, interfaces, and relationships in the application of the patterns. It will be essential to annotate the integrity constraints. It is not necessary to include the entire system diagram, only the new elements or modifications as well as those directly related to the application of the patterns.

Question 4 (10 points)

Statement

In the current system, Employee in the method executeAgent(agent: AIAgent) invokes AIAgent.processRequest(...). Now we want to be able to dynamically add extra behaviors to the execution of an agent, without modifying either the AIAgent class or the Employee logic. Here, functionalities such as these could be added:

- Caching: If the exact same request has been executed recently, return the stored result instead of calling the agent again (saving tokens).
- Detailed Logging: Record the date, employee, agent, and number of tokens used in each call to a file or database.
- Time Measurement: Calculate how long each call to AIAgent takes.

Each of these functionalities could be enabled or disabled at runtime and combined with each other (for example, first caching, then logging, then measuring the waiting time).

Se pide:

a) (2.5 points) Which design pattern(s) would you use for this requirement? Briefly justify your answer.

b) (7.5 points) Create a sequence diagram for the use case of executing an agent. In this section, you can assume that the system lacks a quota control policy or notifications. Specifically, you want to record both the start and end times of the call to the process(prompt: String) method of the AIAgent class in the log.