First Task: API Automation

The following explains what was done to complete the first task:

1. Framework: The framework integrates RestAssured and Serenity for API testing, Cucumber for Gherkin-based BDD test cases, and JUnit as the test runner.

2. Configuration Management:

- Global variables, such as endpoints and request bodies, are stored in a `config.properties` file.

- `ResourceBundle` is used to load these properties dynamically.

3. Structure:

- Features are written in Gherkin syntax under `src/test/resources/features`.

- Step definitions implementing test logic reside in `src/main/java/com/automation/petstore/steps`.

- Test runner uses Serenity to enhance reporting and manage test execution.

4. Benefits:

- Serenity adds detailed, visually appealing test reports.

- Combines BDD readability with advanced reporting and RestAssured capabilities.

- Using a configuration file centralizes control over dynamic data and endpoints, improving maintainability.

The following mentions the different design patterns used in the automation, along with a brief explanation of their use within the Project:

1. **Page Object Pattern (Adapted for APIs):** The objective is centralize the logic related to interactions with endpoints in a single place (in this case, the properties in the config.properties file and the step definition methods).
    a. How it is applied:
        i. Endpoints and request bodies are stored in the config.properties file, avoiding repetition and improving reusability.
        ii. Interactions with the endpoints (e.g., POST, GET) are defined in the step definition methods (PetStoreSteps.java), acting as an intermediary layer between the tests written in Gherkin and the actual HTTP calls.
2. **Singleton Pattern:** The objective Ensure that the configuration file (config.properties) is loaded only once and is globally available.
    a. How it is applied:
        i. ResourceBundle.getBundle("config") creates a single instance of the properties that is reused throughout the framework.
3. **Factory Pattern:** The objective is dynamically generate request bodies as needed
    a. How it is applied: Although static request bodies were defined in `config.properties` in this case, a possible future enhancement would be implementing methods that dynamically generate these bodies based on required parameters (such as IDs or names).

4. **Separation of Concerns (SoC):** The framework clearly separates the responsibilities of each component:

- **Step Definitions:** Contain the test logic.

- **Gherkin Features:** Specify test scenarios in a declarative manner.
- **`config.properties` file:** Centralizes dynamic configurations.
- **Serenity and Cucumber:** Manage test execution and reporting.

In addition to the mentioned aspects, BDD is also used. Although not a design pattern in the classical sense, the framework follows the BDD approach.

Objective: Make tests understandable for all stakeholders by using Gherkin to write test scenarios in a readable format.

How it is applied:

- Tests in `petstore.feature` use the **Given, When, Then** format, with the logic implemented in the step definitions.