Team 4: Rong Cui, Xuanxuan Ge, Ben Lin, Xinyi Tao
rongcui, xuanxuag, cblin, xinyit

17480/780 Final Project Proposal

0. API To Fix

Java Gitlab API - A wrapper for the Gitlab API written in Java.

1. Problems & Use Cases

- **Problem 1: Missing accessors for several necessary fields**
    - Some of the fields offered by the GitLab web API were not available for this wrapper API. For example, `importStatus` in `GitlabProject`, `closed_at` in `GitlabIssue`, `stat` in `GitlabCommit`. Some of the information might be important to clients, for example, clients might need to know the close date of an issue. This API should try to provide the same information that the original web API is providing.
    - Example use cases
        - Given a GitlabIssue object, the client might want to get the closed_at field to find out the date that an issue is being closed by using a getter inside of GitlabIssue, for example, `gitlabIssue.getClosedAt()`, but this method is not provided in `GitlabIssue` class, even though there is a private field `closedAt` inside the `GitlabIssue` class.
        - Sample code:

```
/* Get an issue's closed date */
String hostUrl = "gitlab.com";
String apiToken = "token";
GitlabAPI api = GitlabAPI.connect(hostUrl, apiToken);
GitlabProject project = api.getProject("namespace", "project");
List<GitlabIssue> issues = api.getIssues(project);
for (GitlabIssue issue : issues) {
	// getClosedAt() is not provided
	System.out.println(issue.getClosedAt());
}
```

- **Problem 2: Inappropriate failure handling**
    - Several methods in the class `GitLabAPI` return the modified object as return type, such as `updateUser()`, `updateLabel()`, `updateMergeRequest()`, `updateMilestones()`, `updateNote()`, and `updateProject()`. However when the operations fail, these methods do not return any error indicating the failure. This requires the client to double check if the returned object is null and then compare the updated fields with the new value.

- Violated principles:
  - M.3 Use appropriate parameter & return types
  - M.10 Don't make caller do the work twice
- Sample code:

```
/* Update the name of an existing project */
String hostUrl = "gitlab.com";
String apiToken = "token";
GitlabAPI api = GitlabAPI.connect(hostUrl, apiToken);
GitlabUser project = api.createProject("name", namespaceId, true, false,
                     true, false, true, true, visibility, "importUrl");
GitlabProject updatedProject = api.updateProject(project.getId(), "newName",
                               namespaceId, true, false, true, false, true,
                               true, visibility, "importUrl");
if (updatedProject != null && updatedProject.getName().equals("newName")) {
    System.out.println("Update succeeds.");
} else {
    System.out.println("Update fails.");
}
```

- **Problem 3: Long parameter lists in CRUD operations**
  - Methods in classes `GitlabGroup`, `GitlabLabel`, `GitlabMergeRequest`, `GitlabMilestone`, `GitlabNote`, `GitlabProject`, `GitlabSSHKey`, `GitlabUser`, `GitlabProject` are error-prone to be misused because they require long parameter lists and do not handle failure appropriately.
  - Violated principles:
    - M.6 Use consistent parameter ordering across methods
    - M.7 Avoid long parameter lists
  - E.g., `createProject, updateProject`
  - Sample code:

```
/* Create a gitlab project object and update it*/
String hostUrl = "gitlab.com";
String apiToken = "token";
int namespaceId = 1, visibility = 1;
GitlabAPI api = GitlabAPI.connect(hostUrl, apiToken);
GitlabProject project = api.createProject("name", namespaceId, true, false,
                        true, false, true, true, visibility, "importUrl");

GitlabProject updatedProject = api.updateProject(project.getId(), "newName",
                               namespaceId, true, false, true, false, true,
                               true, visibility, "importUrl");
```

2. How To Rectify

Java-Gitlab-API is a wrapper for the Gitlab API written in Java. Through this API, users can manage their Gitlab users, groups, projects, and issues such as build or commit. Note that this API doesn't support Git operations like pull or push, but it supports CRUD operations on Gitlab platform with authentication for components including Issue, Group, Label, Milestone, Namespace, Note, Project, Tag, User, and etc.

This API is, however, broken in many ways. Firstly, this API misses basic **Functionality** like getters for important information on the Gitlab platform. As use case 1 shows, users are not able to retrieve several fields of an object such as GitlabProject, GitlabCommit, and GitlabIssue. This violates F.3 and we should fix it to let it do what's common. Secondly, the CRUD operations for Gitlab related components like User, Project, Group are easy to be misused. As use case 2 & 3 both indicate, the methods have a long parameter list (M.7) and inconsistent parameter ordering (M.6) among overloaded methods (M.2). The return types of those methods serve little meaning (M.4) and can cause non-negligible difficulty in processing exceptions (E.3). Thus, the **Usability** of this API can become a great problem with these misleading utility methods. Thirdly, the **Readability** of this API is really bad. The major class GitlabAPI has more than 30 methods and can be hard to learn. Some method names and parameter names don't follow the Java naming convention (N.5) and they are sometimes incomprehensible (N.4). Finally, the **Quality** of this API can be doubtful because each time users have to wait for the query result for some time due to the status of long-live connection to Gitlab platform (Q.7), which depends on the network condition and the server status. Users are not guaranteed to get a valid result within reasonable time. To use this API, users have to learn how to deal with the method connect and its exceptional cases.

In order to fix the problems discussed above, we plan to replace the Java wrapper with a well-designed API substitute. There are three significant changes we are going to make.

One major change is to replace the class GitlabAPI with a new wrapper class Gitlab, which serves as the class with the highest level of all gitlab components. Currently, GitlabAPI is a major utility class. Although classes of gitlab components themselves do exist, they do not provide actual CRUD functionalities. Operations on most of the gitlab components have to be performed through the instance of GitlabAPI. We decided to organize all gitlab components as a hierarchical structure where the operations of each component is managed in the parent component. For example, through an instance of Gitlab, clients are able to create GitlabProject. Through an instance of GitlabProject, clients are allowed to create GitlabCommit, GitlabBranch and GitlabIssue. Also, these CRUD operations will return boolean values instead of a new instance to indicate whether the operation has succeeded, which resolves the problem of exception catches.

Another improvement is related to the exposure of GitlabSession class. The original GitlabAPI requires users to connect to and to start a session with Gitlab platform with some authentication methods (like password or API token) and get a GitlabAPI or GitlabSession instance to perform normal CRUD operations. This exposes either class GitlabAPI or class GitlabSession as a supertype of GitlabUser. We think it is better to merge the function of GitlabSession into the new Gitlab class because with the hierarchical structure discussed above, the instance extracted from the parent instance should be valid as long as the parent instance has valid authentication. It is unnecessary to expose GitlabSession to clients. This also allows us to refine the exception handling related to GitlabSession. Current API does not provide clients with appropriate ways to check if the session is alive. We will add relevant exception handling into the GitlabAPI class to solve that issue.

In addition, most of the CRUD methods in the current API have a long parameter list (> 6). All component classes have constructor methods and setters. That is, each class has their own fields and orders to specify. We plan to use Builder Pattern for all component classes to reduce the size of parameters required. Through making this change, we can reduce the effort that users have to make to learn to use this API due to the succinctness of this interface. Also, we avoid the problems of long parameter lists and inconsistent parameter consistency of those old CRUD methods since we take GitlabComponent as method receivers instead of input parameters (which can take dozens of parameters).

Besides the above changes, there are also some minor modifications we will make to improve the usability, quality, and readability of this API, including ensuring naming consistency and providing detailed documentations for the API.

3. Scope

75% Goal
GitlabAPI (Deprecate, and replace with another wrapper class Gitlab)
GitlabSession (Deprecate, and merge its functionality to class Gitlab)
GitlabProject
GitlabBranch

100% Goal
GitlabAPI
GitlabSession
GitlabUser
GitlabProject
GitlabBranch
GitlabCommit

125% Goal
GitlabAPI
GitlabSession
GitlabUser
GitlabProject

GitlabBranch
GitlabCommit
GitlabIssue