P2. Sprint 2

ASSIGNED: 29 February 2024
DUE: 11:59PM, Thursday, 4 April 2024

100 points

---

1. Objectives

- Assess mutation score of P1 test suite using PIT and improve it.
- Use static analysis tools (PMD and SpotBugs) and refactor the code to address issues.
- Use automatic test input generation tools (EvoSuite and Randoop).
- Implement the `RestController` class.
- Test `RestController` using Mockito to allow `DBConnector` to return different values.

---

2. Mutation Analysis Tool PIT

Update your pom.xml to make PIT target only classes and tests inside the 'model' package. Make sure the configuration of PIT inside your pom.xml looks like this:

```
<plugin>
   <groupId>org.pitest</groupId>
   <artifactId>pitest-maven</artifactId>
   <version>1.10.4</version>
   <configuration>
      <targetClasses>
         <param>edu.colostate.cs415.model.*</param>
      </targetClasses>
      <targetTests>
         <param>edu.colostate.cs415.model.*</param>
      </targetTests>
   </configuration>
</plugin>
```

The workflow CI will produce mutation score information, so the graders willget the report from there. However, you should also collect this information and **reflect** on it in your mutation score report. Moreover, you will use this information to improve the test suite to get a higher mutation score.

Create a file `reports/P2_mutation.md` in your repository. It should contain a coverage report showing in a tabular format, the mutation scores for the classes `Company`, `Project`, `Worker`, and `Qualification`. Remember to reflect on this table and lists tests you added to improve mutation

scores.

---

3. Static Analysis Tool

3.1. PMD

1. Install PMD.

```
<plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-pmd-plugin</artifactId>
     <version>3.13.0</version>
</plugin>
```

2. Run `mvn pmd:pmd pmd:cpd`

3.2. SpotBugs

Install SpotBugs from here.

Run PMD and SpotBugs and identify issues that you can address. Generate a screenshot for each tool that shows the issues before the code is refactored. Refactor your code to address the issues you selected. Generate another screenshot for each tool that shows the remaining issues and confirms that you addressed the issues you selected.

As you refactor, please don't make changes that can result in your code not compiling with ours. For example:

- Don't include print statements anywhere in your submitted code.
- Don't change the names/spellings/capitalization of methods, classes, enum types and values, and packages, and the types of parameters and associations.
- Don't add any public or protected methods that are not listed. Feel free to add private methods if you need helper methods. The reason for disallowing public and protected methods is that using them will prevent your code from compiling with our code.
- Don't add new classes/interfaces/enums.

The log files/screenshots for static analysis must be put in a document called reports/P2_sa.md.

---

4. Automatic Test Generation

4.1. Randoop

1. Download Randoop into the `tools` folder from https://randoop.github.io/randoop/ . Note that the `tools` folder should be at the same level as the maven `src` folder.

2. Create `tools/classes.txt` file containing the target classes:
   `edu.colostate.cs415.model.Qualification edu.colostate.cs415.model.Worker edu.colostate.cs415.model.Project edu.colostate.cs415.model.Company`
3. The classes for which you generated the tests must be listed in a file called `classes.txt`, which is in the same root folder as the `tools` folder.
4. Run `java -cp "tools/randoop-all-4.3.2.jar:target/classes" randoop.main.Main gentests --classlist=tools/classes.txt --time-limit=60`

   On Windows use `;` instead of `:`.

5. Capture a screenshot (or log) showing that you generated the tests.
6. Randoop may generate the test files in other folders. You must move the tests to appropriate folders corresponding to the packages and classes in the maven hierarchy.
7. Capture a screenshot (or log) showing that you could run these generated tests (maven must not fail).

4.2. Evosuite

1. Download Evosuite into the `tools` folder from https://www.evosuite.org/ . This is the same `tools` folder that contains the jar file for `randoop`.
2. Run `java -jar tools/evosuite-1.0.6.jar -prefix edu.colostate.cs415.model -projectCP target/classes`
3. Add dependency:

```
<dependency>
    <groupId>evosuite</groupId>
    <artifactId>evosuite</artifactId>
    <version>1.0</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/tools/evosuite-1.0.6.jar</systemPath>
</dependency>
```

4. Capture a screenshot (or log) showing that you generated the tests.
5. Evosuite may generate the test files in other folders. You must move the tests to appropriate folders corresponding to the packages and classes in the maven hierarchy.
6. Capture a screenshot (or log) showing that you could run these generated tests (maven must not fail).

The log files/screenshots for automatic test generation must be put in a document called reports/P2_atg.md.

---

5. Implementing RestController

You have already implemented the model classes. Now you will implement the controller class necessary to get ready for implementing the front end in the last sprint. The `RestController` class that you will implement feeds data to the front end using a `DBConnector` class, which could connect

to a persistent storage.

We have given you a sample `DBConnector` class that you should not touch. This class contains some canned data but in reality, it would actually retrieve data from a database. Since the `RestController` needs a `DBConnector`, to test the `RestController` you will "mock" a `DBConnector` in this sprint. Here are some important notes:

1. Do **not** modify existing code.
2. Do **not** change exception handling.
3. Do **not** change logging.
4. Do **not** add public methods, exceptions, and classes. Only add private methods to the `RestController`.
5. Use the provided endpoints as a guide to implement all other endpoints.
6. Update the methods getQualifications() and getQualification(String description) to return the actual data from the company object.
7. If a request cannot be satisfied, throw an exception. The exception handler will take care of the rest.
8. GET requests do not have a body.
9. When returning a DTO object, all fields must be initialized with the proper values.
10. When receiving a DTO object as the body of a request, not all fields are initialized. This is specified below by “Body required fields”.
11. DTO objects may contain a string field that represents a Qualification, or a Worker, or a Project. The string should contain their description or name. Do not use the toString, nor the JSON representation.

5.1. Provided Endpoints

```
Url: /
Request type: GET
Return type: String
Return value: Hello World!

Url: /helloworld
Request type: GET
Return type: String
Return value: Hello World!
```

5.1.1. Qualifications

```
Url: /api/qualifications
Request type: GET
Return type: JSON
Return value: QualificationDTO[]

Url: /api/qualifications/:description
Request type: GET
Return type: JSON
Return value: QualificationDTO

Url: /api/qualifications/:description
```

```
Request type: POST
Body type: JSON
Body value: QualificationDTO
Body required fields: description
```

5.2. Endpoints to be Implemented

5.2.1. Worker

```
Url: /api/workers
Request type: GET
Return type: JSON
Return value: WorkerDTO[]

Url: /api/workers/:name
Request type: GET
Return type: JSON
Return value: WorkerDTO

Url: /api/workers/:name
Request type: POST
Body type: JSON
Body value: WorkerDTO
Body required fields: name, qualifications, salary
Return type: String
Return value: OK
```

5.2.2. Project

```
Url: /api/projects
Request type: GET
Return type: JSON
Return value: ProjectDTO[]

Url: /api/projects/:name
Request type: GET
Return type: JSON
Return value: ProjectDTO

Url: /api/projects/:name
Request type: POST
Body type: JSON
Body value: ProjectDTO
Body required fields: name, qualifications, size
Return type: String
Return value: OK
```

5.2.3. Company

```
Url: /api/assign
Request type: PUT
Body type: JSON
```

```
Body value: AssignmentDTO
Body required fields: worker, project
Return type: String
Return value: OK

Url: /api/unassign
Request type: PUT
Body type: JSON
Body value: AssignmentDTO
Body required fields: worker, project
Return type: String
Return value: OK

Url: /api/start
Request type: PUT
Body type: JSON
Body value: ProjectDTO
Body required fields: name
Return type: String
Return value: OK

Url: /api/finish
Request type: PUT
Body type: JSON
Body value: ProjectDTO
Body required fields: name
Return type: String
Return value: OK
```

---

6. Testing RestController

In the previous iterations you tested every public method of the model classes. In this iteration you will write JUnit tests for every endpoint of `RestController`.

While testing `RestController`, use Mockito to make `DBConnector.loadCompanyData()` return different values for different companies, projects, qualitifications, and workers since currently `DBConnector` only has a small set of fixed values.

Here are some important notes:

1. Do **not** modify the `DBConnector` class.
2. Do **not** test the class `DBConnector` class.

---

7. Installing Dependencies and Updating Maven Target

7.1. Mockito

```
<dependency>
        <groupId>org.mockito</groupId>
```

```
        <artifactId>mockito-core</artifactId>
        <version>4.11.0</version>
        <scope>test</scope>
</dependency>
```

7.2. HTTPClient5

```
<dependency>
        <groupId>org.apache.httpcomponents.client5</groupId>
        <artifactId>httpclient5-fluent</artifactId>
        <version>5.1.3</version>
</dependency>
```

Find out more about HTTPClient 5 here.

Use `org.apache.hc.client5.http.fluent.Request` to test your `RestController`.

---

8. Grading criteria

Team scores will be based on both the product and the process. Individual adjustments will be made based on quality of contributions.

8.1. Team scores

- Process component (25 points)
    1. Pull requests
    2. Issues
    3. Branches
    4. Commits
    5. Commits to main (Should be none)
    6. How the work is distributed across the sprint (waiting to work until the end is bad)

- Product component (75 points)
    1. Submission of mutation score report (5 points): We will grade your mutation score report (`P2_mutation.md`).
    2. Use of mutation score report to improve test cases. (5 points) This should be described in your document (`P2_mutation.md`).
    3. Static analysis report (10 points): We will review your before and after report (`P2_sa.md`).
    4. Tests automatically generated should compile and run (10 points). We will grade your atg report (`P2_atg.md`).
    5. Quality of your implementation (20 points): We will test your implementation of `RestController` using our test cases.
    6. Quality of your test cases (15 points): We will use faulty versions of the `RestController` and assess the ability of your `RestControllerTest` test cases to detect these faults. We will also collect and use data about the coverage and mutation score from GitHub.
    7. Quality of your implementation and tests with respect to each other (5 points): We will run your code against your tests and we expect all your tests to pass.

8. Reflection report (5 points): We will grade your reflection (`P2_reflection.md`)

8.2. Individual Adjustments

Adjustments can be positive or negative up to 20 points and for reasons such as:

- low individual process score,
- bad peer reviews from teammates,
- gaming the system (e.g., purposefully increasing the number of commits without actually imlementing much functionality).

8.3. GitHub Actions

We use automated Github Actions to compile and tests the code that you push code to Github. You must never push broken code to the main branch, in other words you must never "break main". Github Actions are one of the tools that we use to calculate your grade. You are not allowed to add/modify/delete any GitHub action, workflow, or workflow run.

- If you modify a workflow file, we are going to see it in the repository git history, and your team is going to lose points.
- If you delete a workflow run, we are going to see that there is a missing workflow run for a specific 'git push', and your team is going to lose points.