14.75 / 20

Homework 05

Ohad Ragolsky 7381351, Gruppe 4

Aufgabe 1:

- 1st point: Instead of saying "Basic Programming 101 knowledge ;)", use a more professional tone.
- 2nd point: Clarify performance concerns. Rather than vaguely stating that the loop "might be slow," provide specific details
- 3rd point: detailed exception handling feedback:Replace the generic note on exception handling with actionable advice
- 4th point: Specify security risks, Rather than generalizing about potential security issues, focus on specifics
- To this Point: “Testing: I noticed a few tests included, which is good, but I think there could be more. Final”: Provide actionable test suggestions. Instead of just saying "add more tests," suggest what to test.
- Incorporate positive feedback. Highlight strengths to keep the review balanced and constructive

5

Aufgabe 2:

Equivalence Class Testing

Input Equivalence Classes:

1. Valid Inputs:
   - Total students fit into groups: totalStudents ≤ groupSize * availableGroups.
   - Some students remain unassigned: totalStudents > groupSize * availableGroups.
2. Invalid Inputs:
   - totalStudents < 0.
   - groupSize ≤ 0 or availableGroups ≤ 0.

Output Equivalence Classes:

1. Returns 0.
2. Returns number of unassignable students.
3. Throws exception.

In equivalence class testing, we do not partition using implementation specifics. Use constants only.

− 0.5

Boundary Value Testing

We identify boundaries for the inputs:

1. totalStudents: 0, 1, groupSize * availableGroups, groupSize * availableGroups + 1.
2. groupSize: 1, 2, a large value.
3. availableGroups: 1, 2, a large value.

Test Case Table

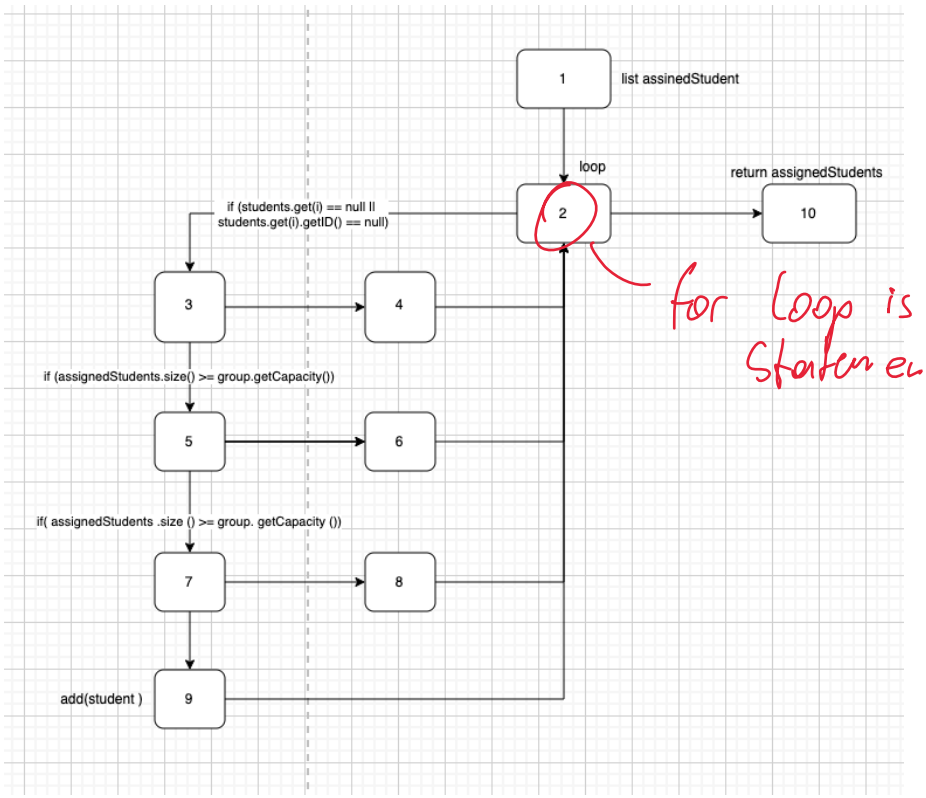
Test Case ID	totalStudents	groupSize	availableGroups	Expected Result	Notes
TC1	0	5	2	0	Valid case: totalStudents = 0.
TC2	-10	5	2	Exception	Invalid case: totalStudents < 0.
TC3	10	5	2	0	Valid case: exact capacity.
TC4	15	5	2	5	Valid case: excess students.
TC5	10	0	2	Exception	Invalid case: groupSize = 0.
TC6	10	5	0	Exception	Invalid case: availableGroups = 0.
TC7	10	-5	2	Exception	Invalid case: groupSize < 0.
TC8	10	5	-2	Exception	Invalid case: availableGroups < 0.
TC9	25	5	5	0	Valid case: more groups than needed.

Please have a look at how the tables in the lecture look like.
You only provide the bottom part.

- No boundary testing -1.5
- No distribution of equivalence classes into test cases -1.5

3.5

Aufgabe 3:



Interpretation of the Control Flow Graph

- **Node 1**: Initialize assignedStudents.
- **Node 2**: Start of for loop.
- **Node 3**: condition- if (students.get(i) == null || students.get(i).getID() == null).
- **Node 4**: Print "Invalid student..." and continue.
- **Node 5**: conditions - if (assignedStudents.contains(students.get(i))).
- **Node 6**: Print "Student already assigned" and continue.
- **Node 7**: conditions - if (assignedStudents.size() >= group.getCapacity()).
- **Node 8**: Print "Group is full" and continue.
- **Node 9**: Add students.get(i) to assignedStudents.
- **Node 10**: Return assignedStudents.

Statement Coverage

```
public String attemptAssignToGroup(List<Student> students, Group group) {
    List<Student> assignedStudents = new ArrayList<>();                    // 1
        for (int i = 0; i < students.size(); i++) {                        // 2
            if (students.get(i) == null || students.get(i).getID() == null) {  // 3
                System.out.println("Invalid student or student ID");       // 4
                continue;                                                  // 5
            }
            if (assignedStudents.contains(students.get(i))) {              // 6
                System.out.println("Student already assigned");            // 7
                continue;                                                  // 8
            }
            if (assignedStudents.size() >= group.getCapacity()) {          // 9
                System.out.println("Group is full");                       // 10
                continue;                                                  // 11
            }
            assignedStudents.add(students.get(i));                         // 12
        }
        return assignedStudents;                                           // 13
    }
```

Test Case 1: testInvalidStudentId()

- Statement Coverage : Executes: 2, 3, 4, 5, 6, 18 -> 6/10

Test Case 2: testSuccessfulAssignment()

- Statement Coverage : Executes: 2, 3, 4, 8, 12, 16, 18 -> 7/10

Total Statements Coverage: 9/13 = 69.23%

To achieve 100% coverage, additional test cases are needed to execute:

- Line 7: Test duplicate students.
- Lines 9–11: Test when the group capacity is exceeded.

Branch Coverage

```
public String attemptAssignToGroup(List<Student> students, Group group) {   // Not a branch
        List<Student> assignedStudents = new ArrayList<>();                  // Not a branch
        for (int i = 0; i < students.size(); i++) {                          // Not a branch
            if (students.get(i) == null || students.get(i).getID() == null) {  // Branch 1 (true/fa
                System.out.println("Invalid student or student ID");         // Not a branch
                continue;                                                    // Not a branch
            }
            if (assignedStudents.contains(students.get(i))) {                // Branch 2 (true/fa
                System.out.println("Student already assigned");              // Not a branch
                continue;                                                    // Not a branch
            }
            if (assignedStudents.size() >= group.getCapacity()) {            // Branch 3 (true/fal
                System.out.println("Group is full");                         // Not a branch
                continue;                                                    // Not a branch
            }
            assignedStudents.add(students.get(i));                           // Not a branch
        }
        return assignedStudents;                                             // Not a branch
    }
```

Test Case 1: testInvalidStudentId()

- Branch Coverage: covered Branch Outcomes: 1 (Branch 1 → True). 1/6

Test Case 2: testSuccessfulAssignment()

- Branch Coverage: Covered Branch Outcomes: 3 (Branch 1 → False, Branch 2 → False, Branch 3 → False). -> 3/6

Branch Coverage: 4/6 = 66.67%

The **Branch Coverage** with the provided tests is approximately **66.67%**.

To achieve 100% coverage, we need additional tests to cover like:

- A duplicate student (Branch 2 → True).
- A group that is already full (Branch 3 → True).

Path Coverage

```java
public String attemptAssignToGroup(List<Student> students, Group group) {
    List<Student> assignedStudents = new ArrayList<>();
    for (int i = 0; i < students.size(); i++) {
        if (students.get(i) == null || students.get(i).getID() == null) {  // Path A
            System.out.println("Invalid student or student ID");
            continue;
        }
        if (assignedStudents.contains(students.get(i))) {                  // Path B
            System.out.println("Student already assigned");
            continue;
        }
        if (assignedStudents.size() >= group.getCapacity()) {              // Path C
            System.out.println("Group is full");
            continue;
        }
        assignedStudents.add(students.get(i));                             // Path D
    }
    return assignedStudents;
}
```

Test Case 1: testInvalidStudentId() - Covered Path A -> 1/4 -> 25%
Test Case 2: testSuccessfulAssignment Covered Path D -> 1/4 -> 25%

Total Path Coverage (A,D): 2/4 -> 50%

If you assume only one execution of the loop. -0.5

With only the two tests in the homework, the Path Coverage is 50%.
To achieve 100%, we need to:

- test for duplicate students (Path B).
- test for exceeding group capacity (Path C)

Condition Coverage

```java
if (students.get(i) == null || students.get(i).getID() == null) {  // Compound condition
    (assignedStudents.contains(students.get(i))) {                 // Single condition
    if (assignedStudents.size() >= group.getCapacity()) {             // Single condition
```

Test Case 1: testInvalidStudentId():

- Condition 1a (students.get(i) == null): True.
  - **True:** Covered by testInvalidStudentId().
  - **False:** Covered by testSuccessfulAssignment().
    Condition 1a is fully covered (2/2).

good.

- Condition 1b (students.get(i).getID() == null): Not evaluated (short-circuited due to ||).
  - **True**: Not covered.
  - **False**: Covered by testSuccessfulAssignment().
    Condition 1b is partially covered (1/2).

- Condition 2 (assignedStudents.contains(students.get(i))): Not reached.

- **True**: Not covered.
- **False**: Covered by testSuccessfulAssignment().
  Condition 2 is partially coveredv(1/2).

- Condition 3 (assignedStudents.size() >= group.getCapacity()): Not reached.
  - **Condition 3: assignedStudents.size() >= group.getCapacity()**
  - **True**: Not covered.
  - **False**: Covered by testSuccessfulAssignment().
    Condition 3 is partially covered (1/2).

Condition Coverage: 5/8 -> 62.5%
Additional tests are needed to cover:

- **Condition 1b → True**,
- **Condition 2 → True**
- **Condition 3 → True**.

As you counted it, yes.

6.25