SOFTWARE 2 PRACTICAL

EXAM SAMPLE QUESTIONS

Week 9 – Practical 9

For this practical, you should create a Java project. Then you should create a package **`project`**. The entire sheet should take less than 3 hours to complete. Note also that I have provided the signature of the methods, that is the formal parameters' name have been omitted.

Exercise 1: *The* `Worker` *class*

Implement the class `Worker` representing an employee in a department. An instance of the class `Worker` has:

1. the following attributes with package access modifier.
   - a `String uid` representing a unique employee ID,
   - a `String name` containing the name of the employee,
   - a list of the project ID he/she worked on as a `Set<String> projects`,
2. A public constructor `Worker(String, String)` having two parameters, the employee's ID and the employee's name in that order,
3. the following public methods
   - the accessor `Set<String> getProjects()` which returns the set of projects the employee worked on,
   - `boolean addProject(String)` which takes a project ID as parameter and adds the project to the list of projects the employee contributed to. The method must not add the project if it is already in the list and should return `false` (i.e. there are no duplicates in the list). The method returns `true` if the operation has been successful.
   - `boolean removeProject(String)` which takes a project ID as parameter and removes the project from the list of projects the employee contributed to. The method returns `true` if the operation has been successful, `false` otherwise (e.g. if the project is not in the list).

Exercise 2: *The* `Project` *class*

Implement the class `Project` representing a project done in a Department. An instance of the class `Project` has:

1. the following attributes with package access modifier,
   - a `String uid` representing a unique project ID,
   - a `String title` containing the title of the project,
   - `Set<String> collaborators` a list of the worker's ID who collaborated on the project,
2. public constructor `Project(String, String)` having two parameters, the project's ID and the project's title in that order,
3. and public methods:
   - the accessor `Set<String> getCollaborators()` which returns the set of the employees who worked on it,
   - `boolean addCollaborator(String)` which takes a worker's ID as parameter, adds the worker to the list of collaborators and returns `true` if the operation has been successful. The method must not add the employee if it is already in the list and should return `false` (that is there are no duplicates in the list).
   - `boolean removeCollaborator(String)` which takes a worker's ID as parameter, removes the employee from the list of collaborators, and returns `true` if the operation has been successful. The method must return `false` otherwise (for example if the employee is not in the list).

Exercise 3: *The* `InvalidIDException` *class*

Implement an unchecked exception `InvalidIDException`. The class should only contain two public constructors:

- `InvalidIDException()`,
- `InvalidIDException(String)`.

Exercise 4: *The* `Department` *class*

Implement the class `Department` representing a Department. An instance of the class `Department` has:

1. the following attributes with package access modifier,
   - a `String name` containing the name of the Department,
   - `Map<String, Worker> workers` to contain all workers from that Department,
   - a `Map<String, Project> projects` to contain all projects done in the Department.
2. A public constructor, `Department(String)` that takes the name of the department as parameter and initialise the other instance variable as empty maps.
3. and public methods:
   - `Worker getWorker(String)` which takes an employee's ID as parameter and returns the `Worker` instance with this ID. The method should throw a `InvalidIDException` if no such employee exists.
   - `Project getProject(String)` which takes a project's ID as parameter and returns the Project instance with this ID. The method should throw a `InvalidIDException` if no such project exists.
   - `Project createProject(String, String)` which creates and returns a new `Project` instance and adds it to list of projects done by the Department. The first parameter is the project ID and the second its title. The method must throw a `InvalidIDException` if a project with the same ID already exists.
   - `Worker createWorker(String, String)` which creates and returns a new `Worker` instance and adds it to list of employee working for the Department. The first parameter is the employee ID and the second her/his name. The method must throw a `InvalidIDException` if an employee with the same ID already exists.
   - `Project addCollaborator(String pID, String wID)` which add the employee identified by `wID` to the list of contributors for the project identified by `pID`. The method should return the updated project. The method must throw a `InvalidIDException` if the project and/or the employee do not exist.

Exercise 5: *Graph algorithm*

We are interested in analysing the collaboration network in the Department. The class `Department` has an implicit collaboration graph structure, where employees are nodes and edges are represented by collaboration of two employees on the same project. Figure 1 shows the implicit graph from a `Department` where the projects' collaboration table is given in Table 1.

Table 1: Collaboration table from a given Department.

Project ID	Employee ID
P1	Empl_A, Empl_B, Empl_D
P2	Empl_G, Empl_E
P3	Empl_E, Empl_H
P4	Empl_G, Empl_E, Empl_H
P5	Empl_A, Empl_B, Empl_C
P6	Empl_C, Empl_F, Empl_D

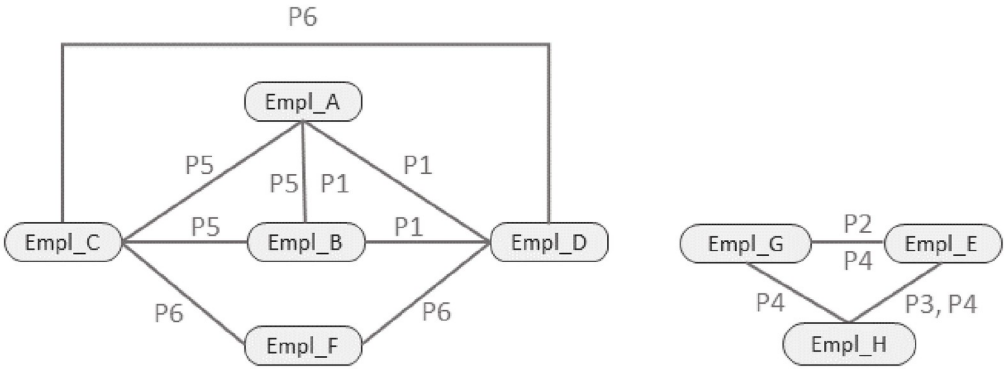

Figure 1: Implicit collaboration graph derived from the collaboration table shown in Table 1.

Implement the public method `Set<String> getConnectionCircle(String wID)` that returns the list of employees that are connected to the employee `wID` (i.e. there is a path between the two employees in the collaboration graph. For example, when considering the graph in Figure 1, `Empl_A` is directly connected to `Empl_C`, `Empl_B`, and `Empl_D`) and transitively connected to `Empl_F` via `Empl_C`. It should also be noted that `Empl_A` cannot be connected to `Empl_G`.

- Therefore, `getConnectionCircle("Empl_A")` should return the set `(Empl_C, Empl_B, Empl_D, Empl_F)`, whereas
- `getConnectionCircle("Empl_G")` should return the set `(Empl_E, Empl_H)`.

Hints: You can devise your own algorithm or adapt the Breadth-First-Search algorithm described in Algorithm (1).

Algorithm 1 Breadth-First-Search algorithm.

```
procedure BREADTHFIRSTSEARCH(Graph, root)
    for each node n in Graph do
        n.distance := ∞
    end for

    let Q be an empty queue Q
    root.distance := 0
    Q.enqueue(root)
    while Q is not empty do
        current := Q.dequeue()
        for each node n that is adjacent to current do
            if n.distance = ∞ then
                n.distance := current.distance + 1
                Q.enqueue(n)
            end if
        end for
    end while
end procedure
```

Figure 2: Breadth-First-Search Algorithm