I. Please enumerate and shortly characterize technical Software Engineering Development activities.

Development activities include:

1. Requirements Elicitation
2. Analysis
3. System Design
4. Object Design
5. Implementation
6. Testing

1. Requirements Elicitation

During **requirements elicitation**, the client and developers define the purpose of the system. The result of this activity is a description of the system in terms of actors and use cases.

2. Analysis

During **analysis**, developers aim to produce a model of the system that is correct, complete, consistent, and unambiguous. Developers transform the use cases produced during requirements elicitation into an object model that completely describes the system. During this activity, developers discover ambiguities and inconsistencies in the use case model that they resolve with the client.

3. System Design

During **system design**, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams. Developers also select strategies for building the system, such as the hardware/software platform on which the system will run, the persistent data management strategy, the global control flow, the access control policy, and the handling of boundary conditions. The result of system design is a clear description of each of these strategies, a subsystem decomposition, and a deployment diagram representing the hardware/software mapping of the system. Whereas both analysis and system design produce models of the system under construction, only analysis deals with entities that the client can understand.

4. Object Design

During **object design**, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design. This includes precisely describing object and subsystem interfaces, selecting off-the-shelf components, restructuring the object model to attain design goals such as extensibility or understandability, and optimizing the object model for performance.

5. Implementation

During **implementation**, developers translate the solution domain model into source code. This includes implementing the attributes and methods of each object and integrating all the objects such that they function as a single system. The implementation activity spans the gap between the detailed object design model and a complete set of source code files that can be compiled.

6. Testing

During **testing**, developers find differences between the system and its models by executing the system (or parts of it) with sample input data sets. During unit testing, developers compare the object design model with each object and subsystem. During integration testing, combinations of subsystems are integrated together and compared with the system design model. During system testing, typical and exception cases are run through the system and compared with the requirements model. The goal of testing is to discover as many faults as possible such that they can be repaired before the delivery of the system.

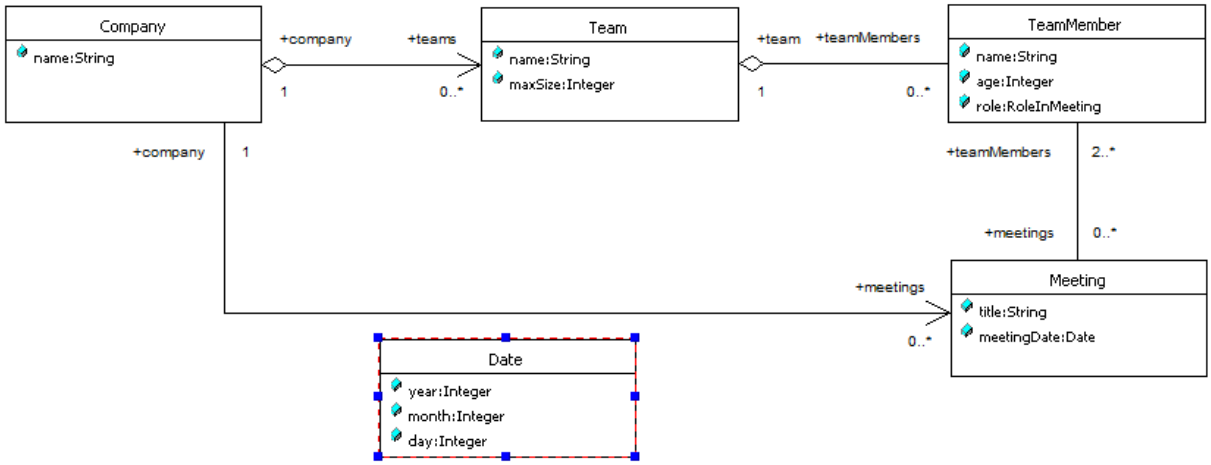
II. The employee of a company are organized in teams. Each employer is member of only one team. A team has a name and a maximum number of members. Team members may participate at different meetings organized by their company. A participant to a meeting may be organizer, speaker or ordinary participant. Each meeting has a title and is organized in a day.

a) Using UML, please specify a class diagram describing the structure modeling the above problem.

b) Using OCL, please specify appropriate invariants checking that:

- **the meetings attended by team members are the same with the meetings organized by the company**
- **each team member does not participate at many meetings in the same day.**

c) Using OCL, please specify an observer returning a team whose members participated at the largest number of meetings.

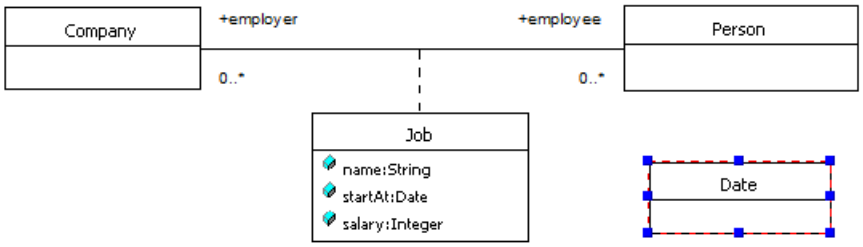



```
context Company
    inv teamAndCompanyMeetings:
        self.teams.teamMembers.meetings->asSet = self.meetings
```

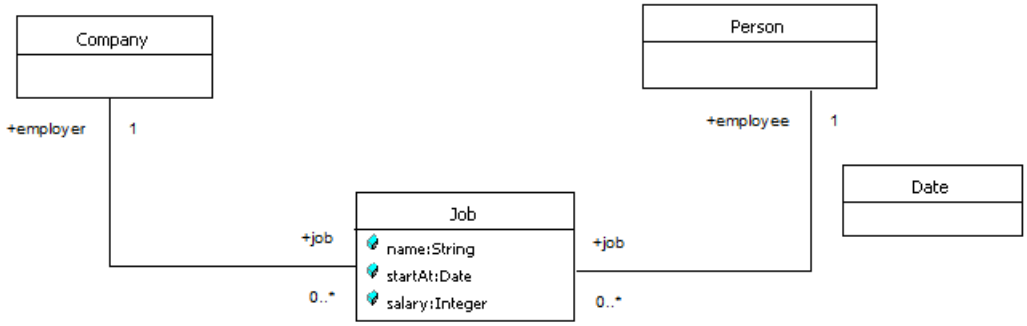
```
context Company
    def theTeamParticipatingAtMostMeetings:
        let aTeamParticipatingAtMostMeetings:Team =
        self.teams->collect(t:Team | (Tuple{a = t, b =
t.teamMembers.meetings->asSet->size}))->sortedBy( t:TupleType(a:Team,
b:Integer) | t.b)->last.a

context TeamMember
    inv nonConcurentMeetings:
        let ms:Set(Meeting)=self.meetings in
        ms->reject(m:Meeting | ms->isUnique(m | m))->isEmpty
```

III. Please represent the UML diagram equivalent with the following one and specify the Java code for managing the associations related to the classes Company and Job that can be automatically generated for managing the associations of this model.



The equivalent model is:



An excerpt of code generated using OCLE - Of course, using generic types is a preferred solution.

```
public class Company {
    public final Set getJobEmployee() {

        if (employee == null) {
            return java.util.Collections.EMPTY_SET;
        }
        return java.util.Collections.unmodifiableSet(employee);
    }
```

```
    public final Set getEmployee() {
        java.util.Set temp = new LinkedHashSet();
        if (employee != null) {
            Iterator it = employee.iterator();
            while (it.hasNext()) {
                temp.add(((Job)it.next()).getEmployee());
            }
        }
        return temp;
    }
    public final void addEmployee(Job arg) {
        if (arg != null) {
            if (employee == null) {
                employee = new LinkedHashSet();
            }
            if (employee.add(arg)) {
                arg.setEmployer(this);
            }
        }
    }
    public final void removeEmployee(Job arg) {
        if (employee != null && arg != null) {
            if (employee.remove(arg)) {
                arg.setEmployer(null);
            }
        }
    }
    public Company() {
    }
    public Set employee;

public class Job {
    public final Company getEmployer() {
        return employer;
    }
    public final void setEmployer(Company arg) {
        if (employer != arg) {
            Company temp = employer;
            employer = null;//to avoid infinite recursions
            if (temp != null) {
                temp.removeEmployee(this);
            }
            if (arg != null) {
                employer = arg;
                arg.addEmployee(this);
            }
        }
    }
    public final Person getEmployee() {
        return employee;
    }
```

```
public final void setEmployee(Person arg) {
    if (employee != arg) {
        Person temp = employee;
        employee = null;//to avoid infinite recursions
        if (temp != null) {
            temp.removeEmployer(this);
        }
        if (arg != null) {
            employee = arg;
            arg.addEmployer(this);
        }
    }
}
public Job() {
}
public String name;
public Date startAt;
public int salary;
public Company employer;
public Person employee;
```

IV. Please shortly characterize the blackbox and whitebox test cases.

Test cases are classified into **blackbox tests** and **whitebox tests**, depending on which aspect of the system model is tested. **Blackbox tests** focus on the input/output behavior of the component. Blackbox tests do not deal with the internal aspects of the component, nor with the behavior or the structure of the components. **Whitebox tests** focus on the internal structure of the component. A whitebox test makes sure that, independently from the particular input/output behavior, every state in the dynamic model of the object and every interaction among the objects is tested. As a result, whitebox testing goes beyond blackbox testing. In fact, most of the whitebox tests require input data that could not be derived from a description of the functional requirements alone. Unit testing combines both testing techniques: blackbox testing to test the functionality of the component, and whitebox testing to test structural and dynamic aspects of the component.