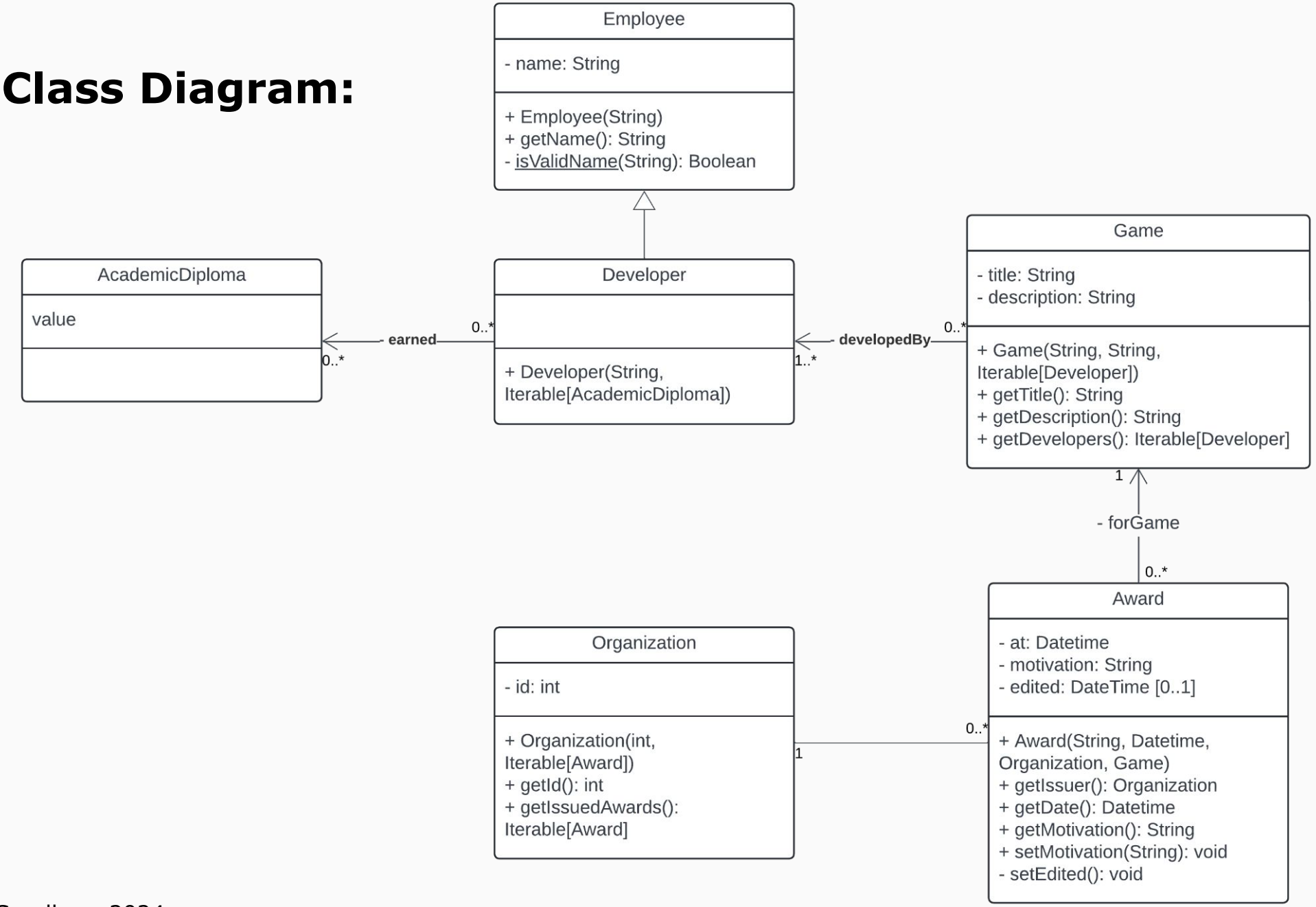
UML 2 CODE

Process, Responsibilities, Languages

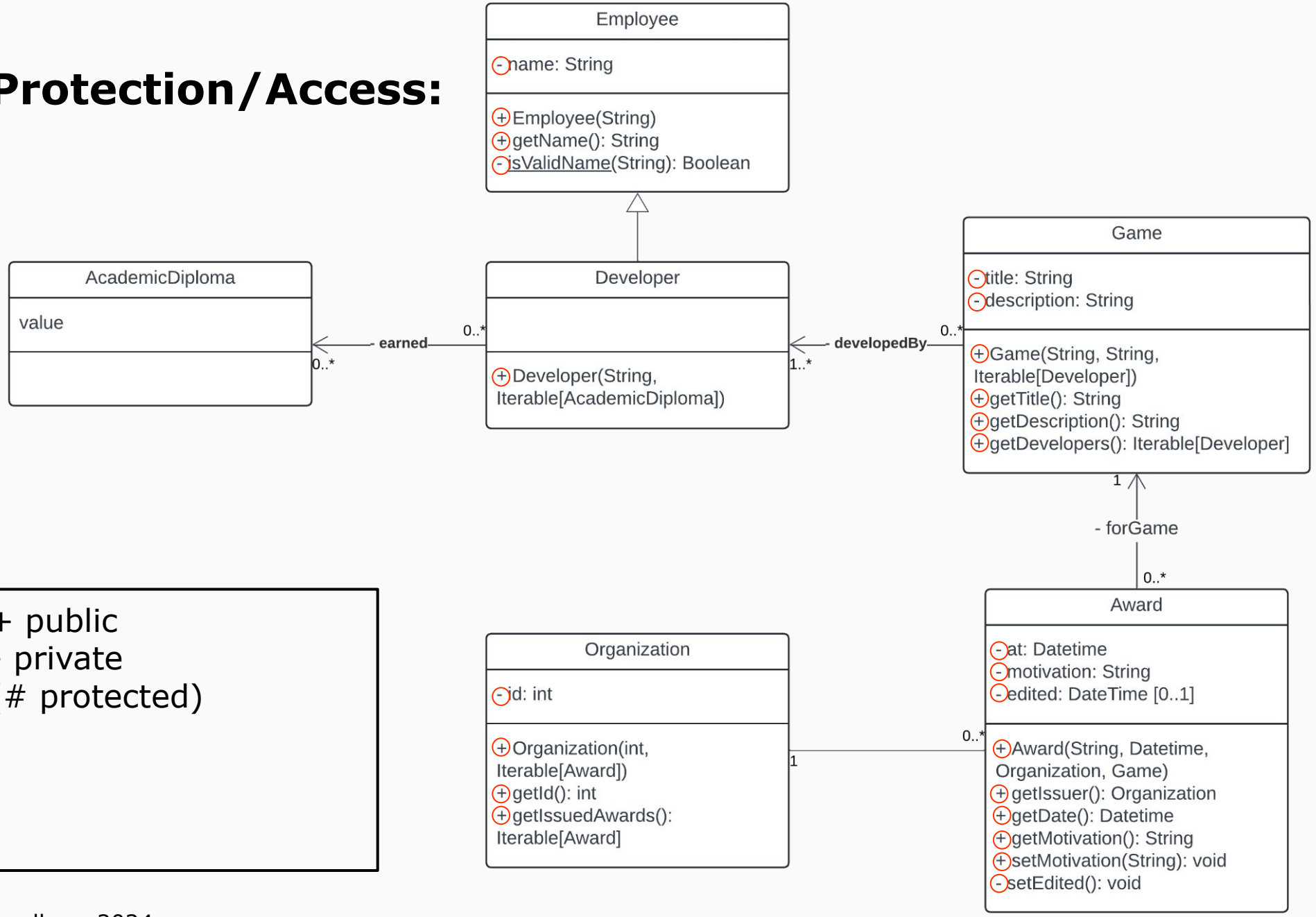
Objektorienterad analys och design (OOS)

Stockholms universitet

Class Diagram:

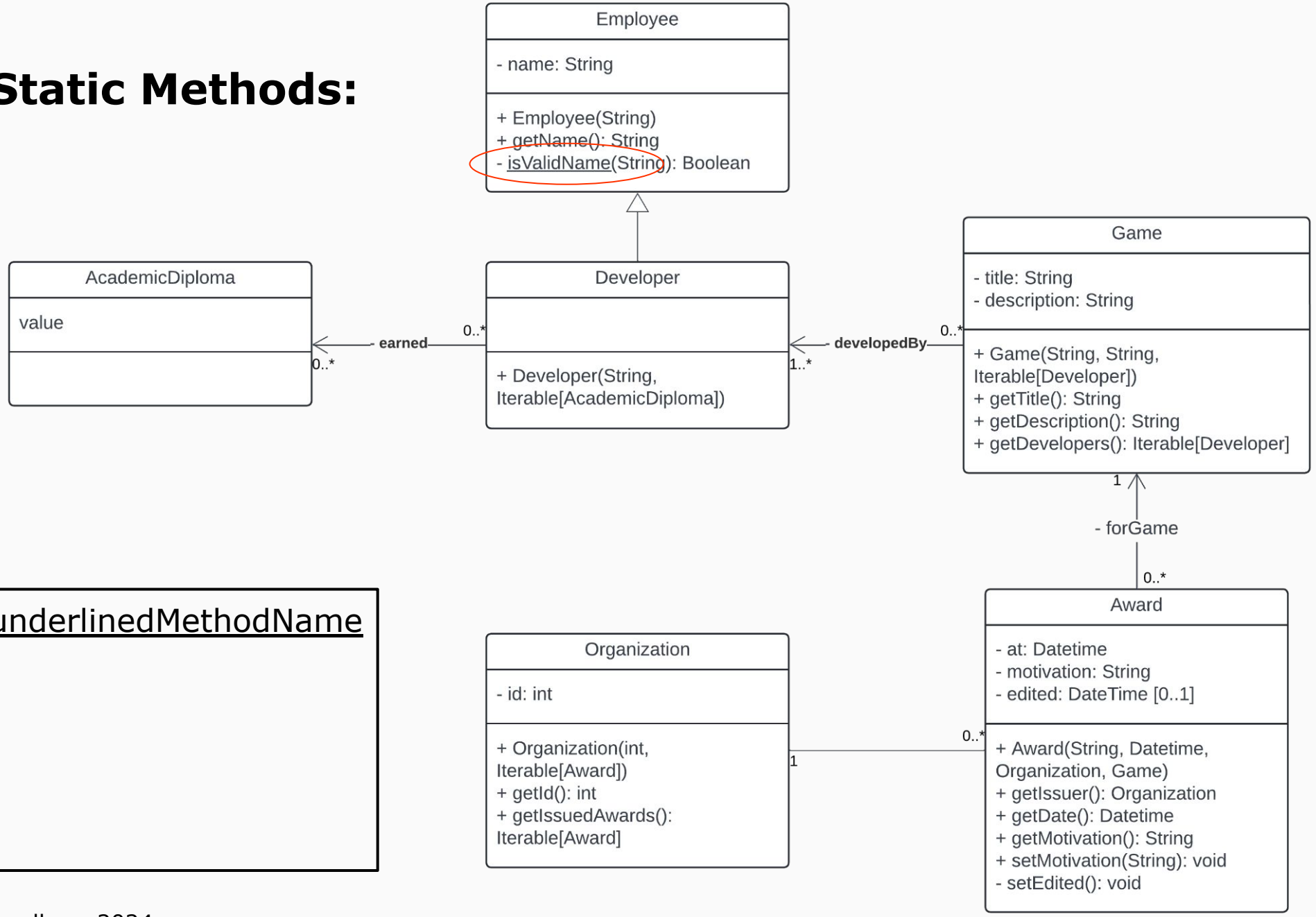


Protection/Access:



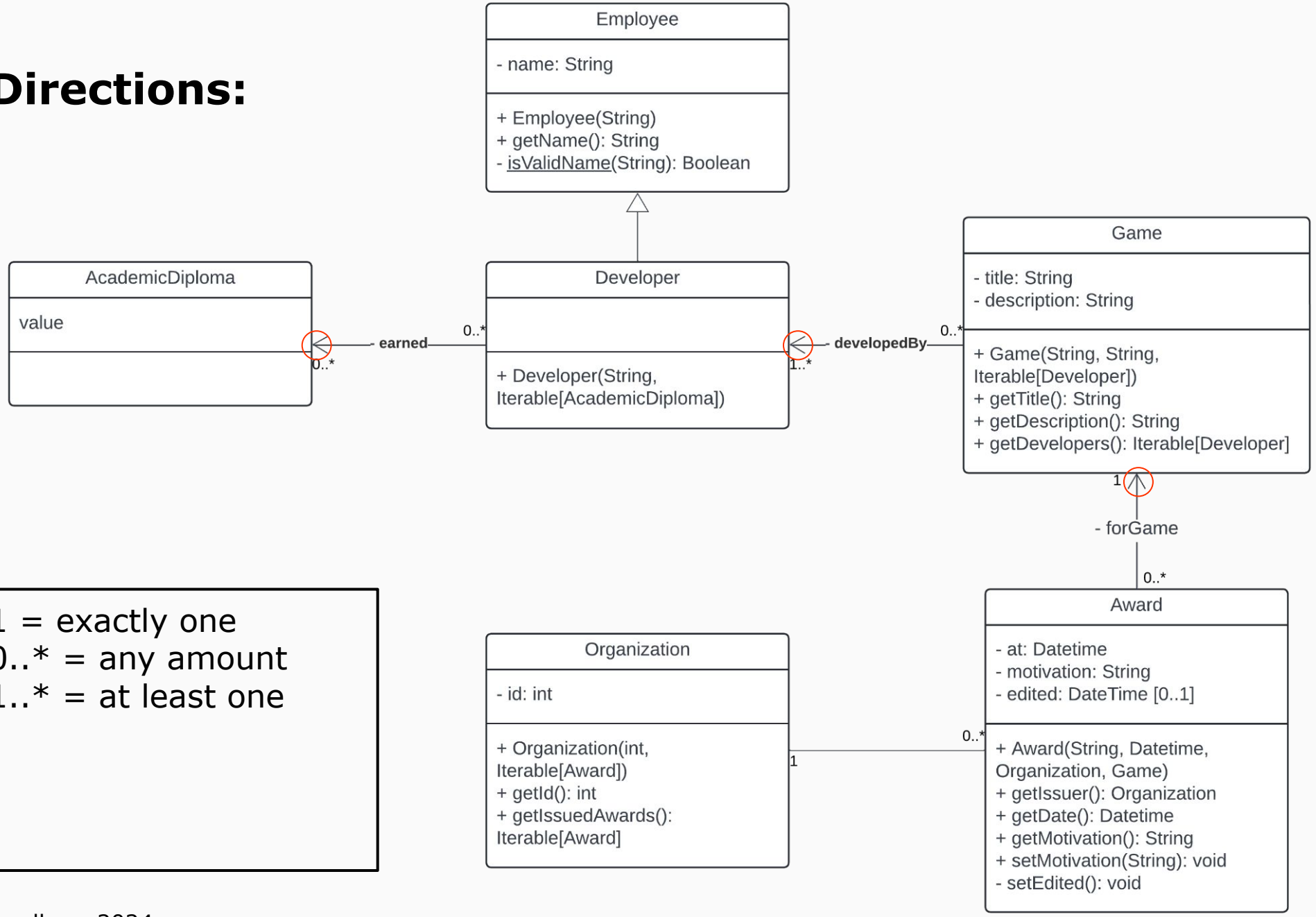
+ public
- private
(# protected)

Static Methods:



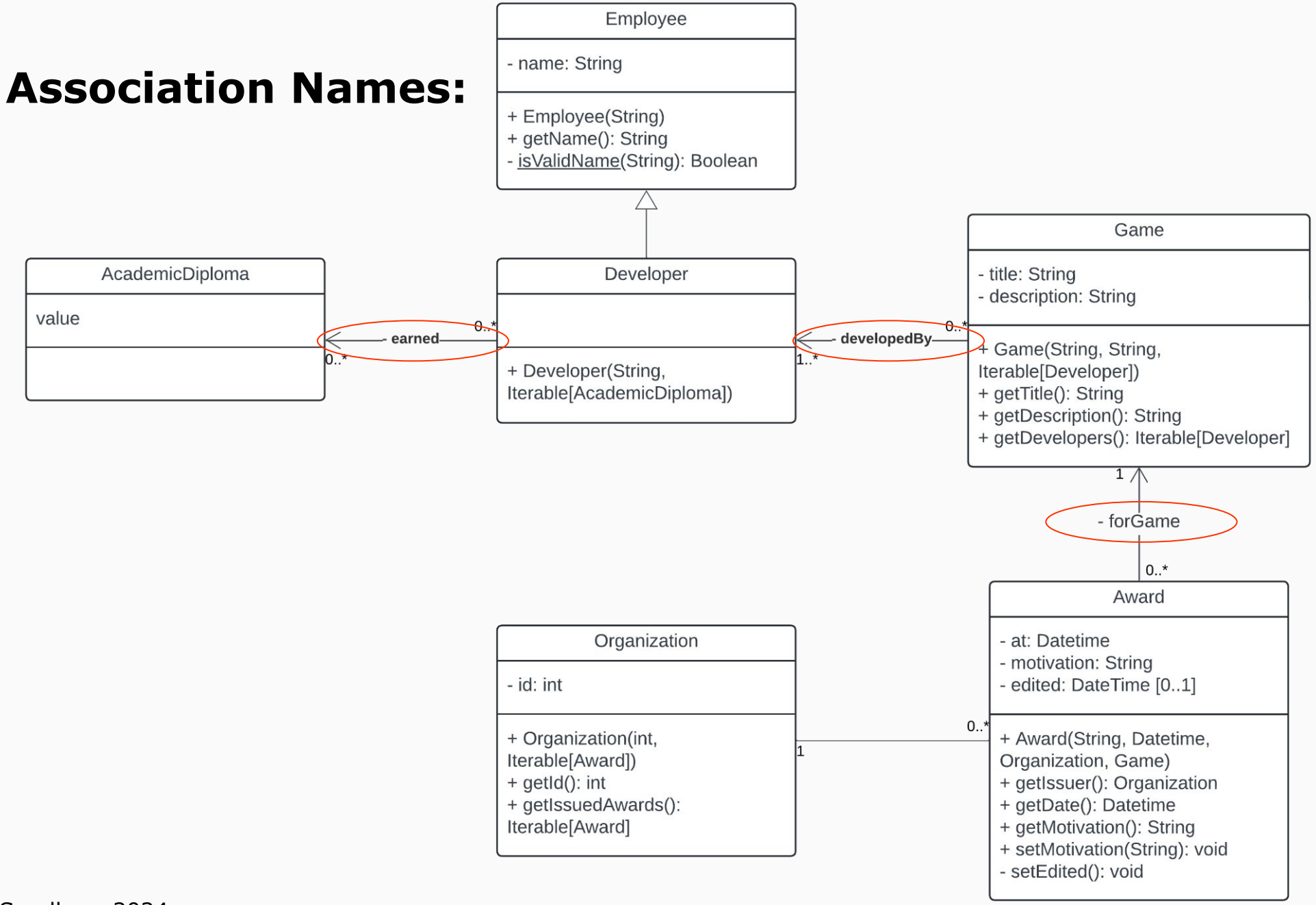
underlinedMethodName

Directions:

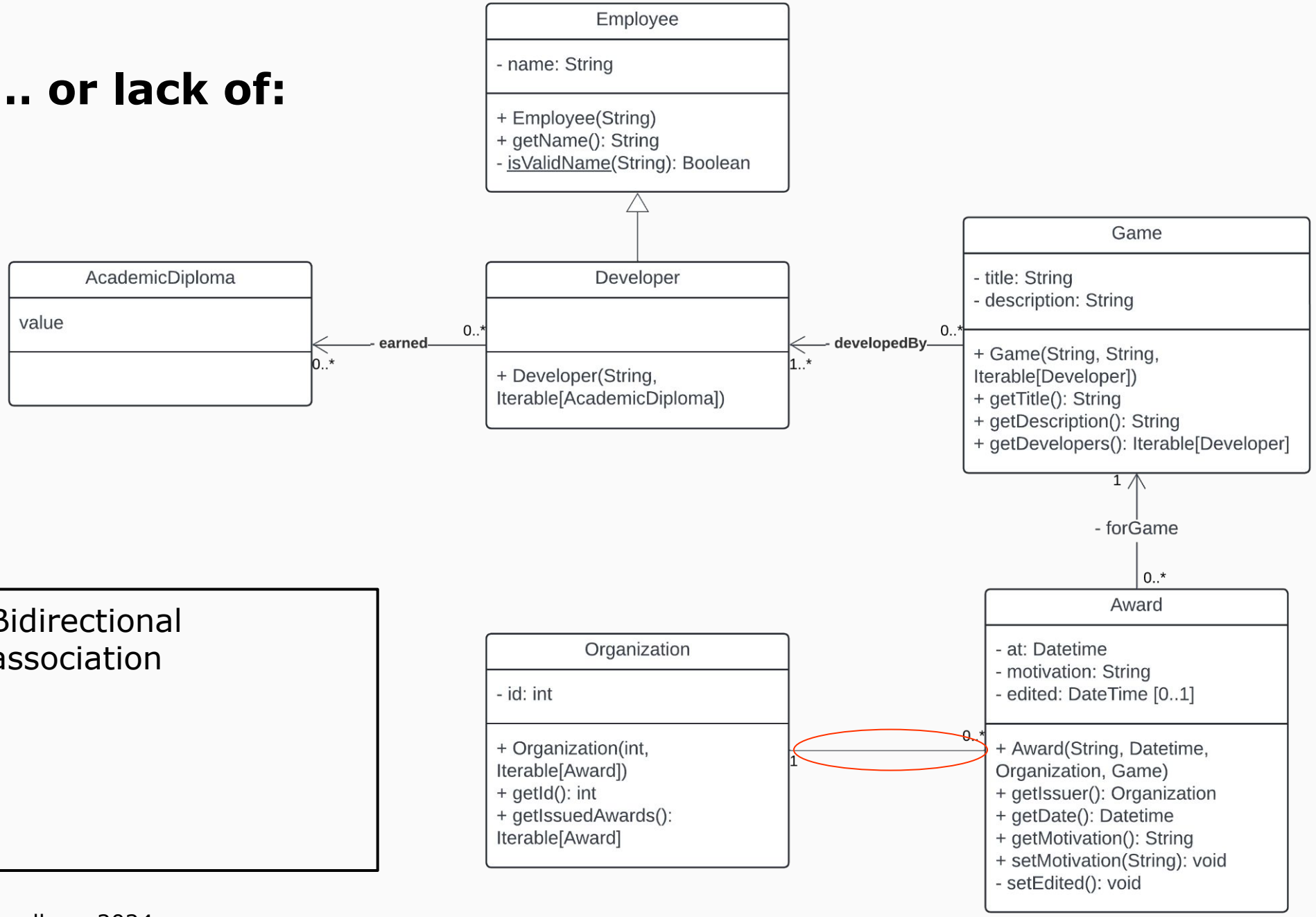


1 = exactly one
0..* = any amount
1..* = at least one

Association Names:

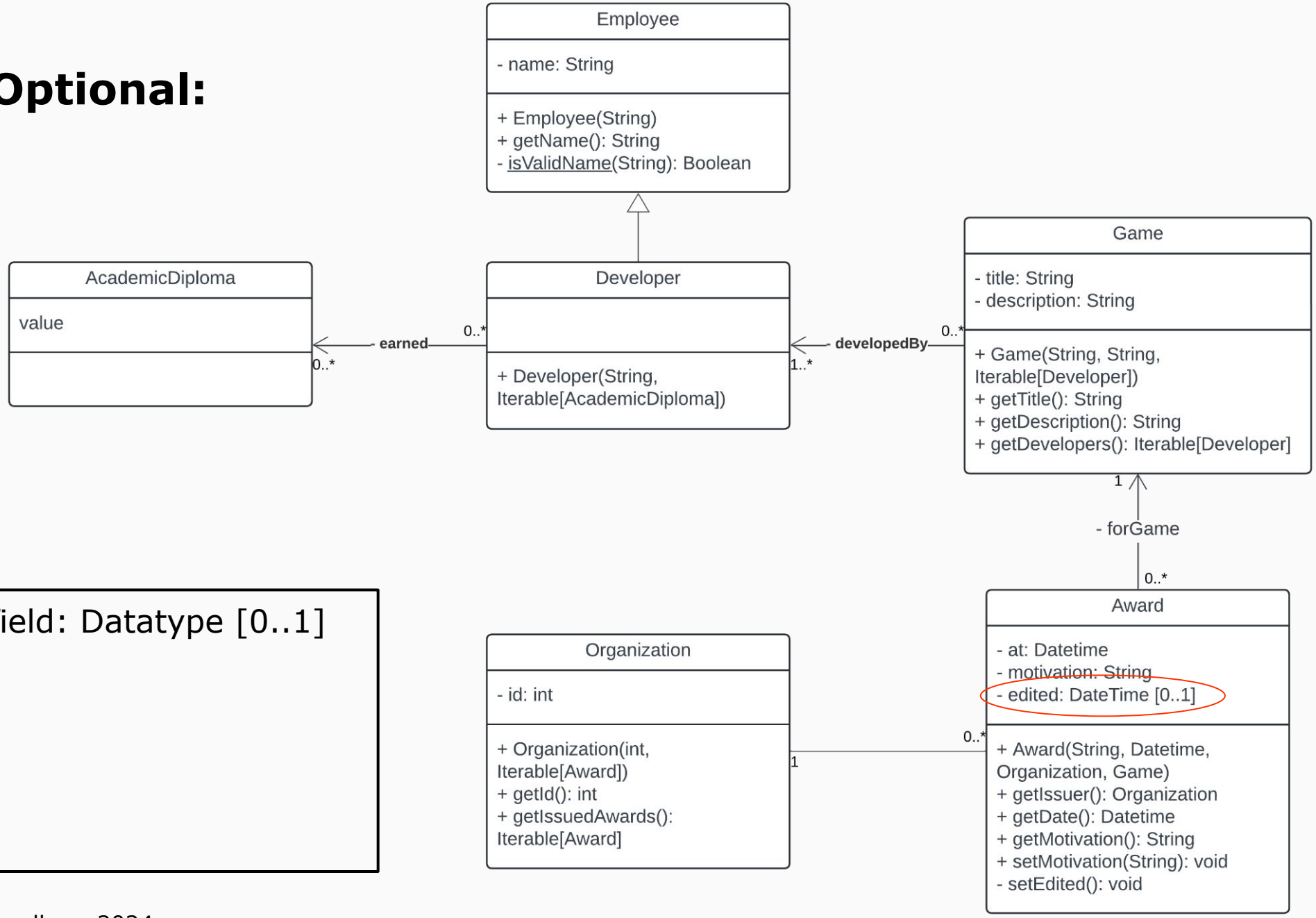


… or lack of:



Bidirectional association

Optional:

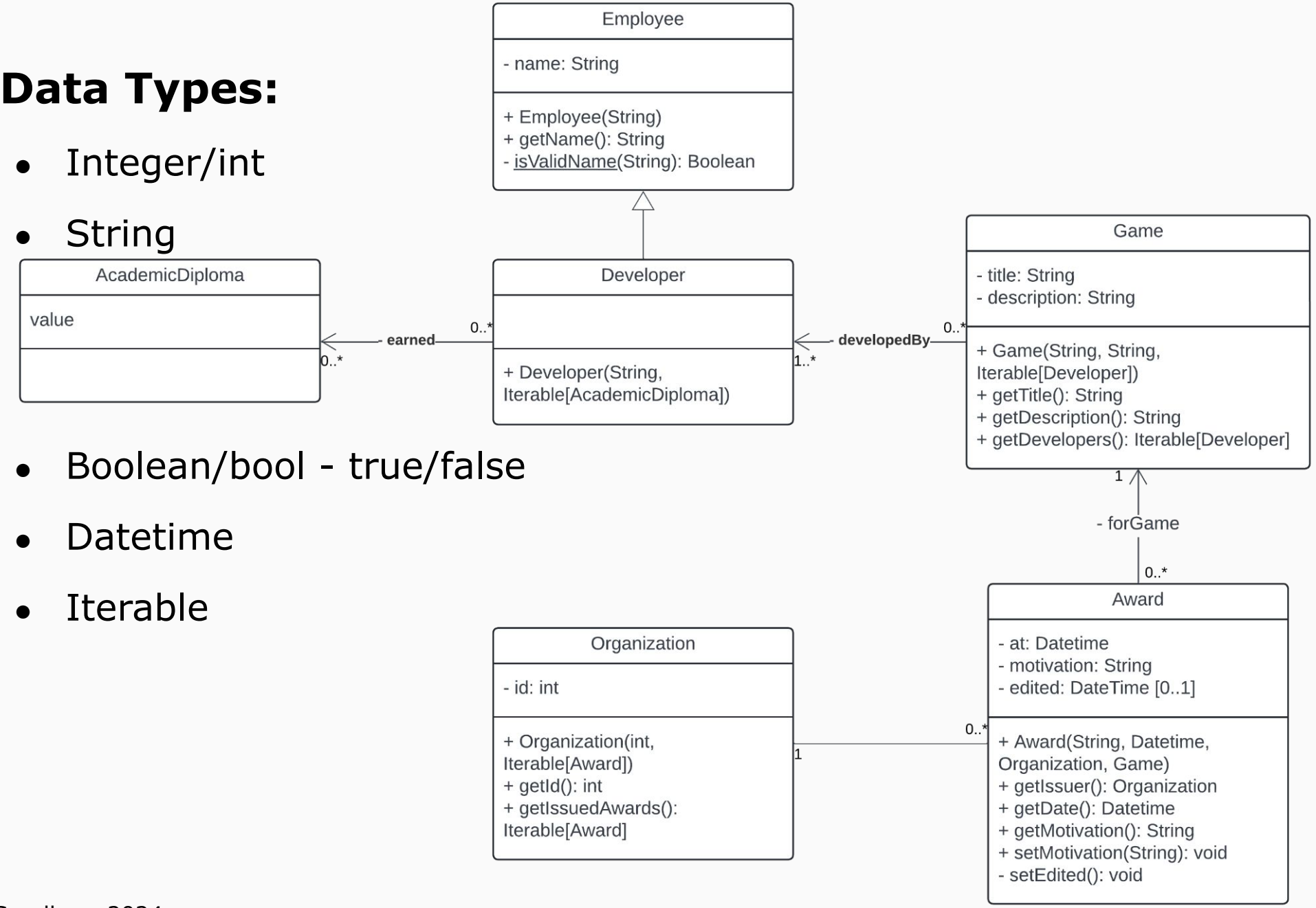


field: Datatype [0..1]

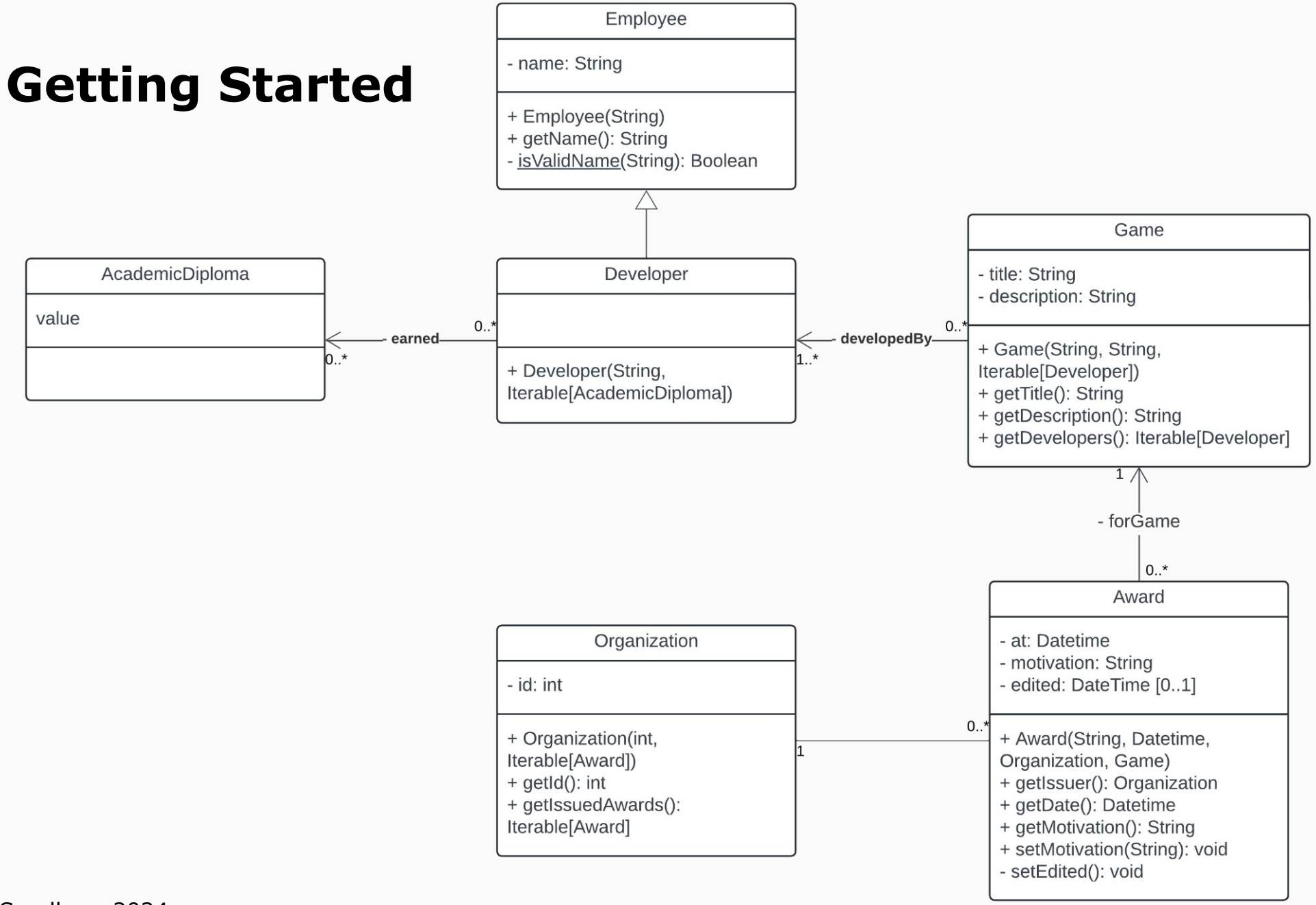
Stockholms universitet

Data Types:

- Integer/int
- String
- Boolean/bool - true/false
- Datetime
- Iterable

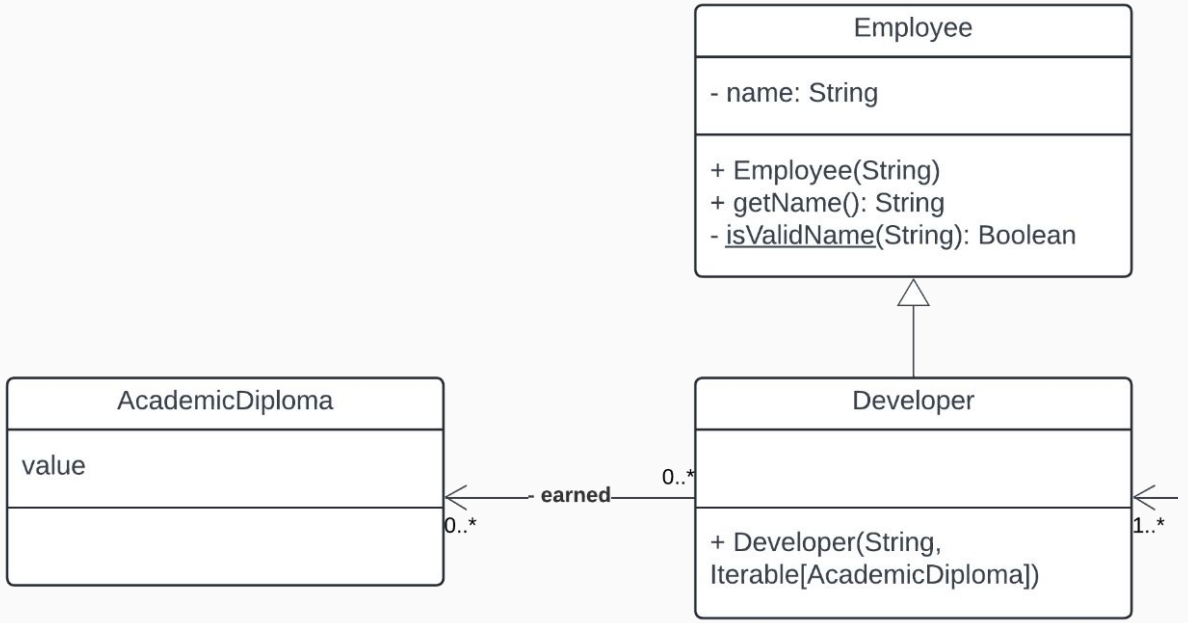


Getting Started



Getting Started

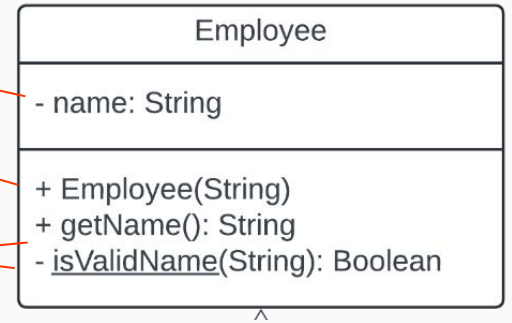
- One association →
- Given association name
- Inheritance
- Staticmethod



Stockholms universitet

Employee

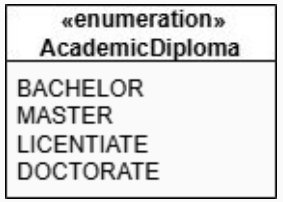
```java
public class Employee {
    private String name;

    public Employee(String name) {
        if (!isValidName(name)) {
            throw new IllegalArgumentException("Invalid name");
        }
        this.name = name;
    }

    public static boolean isValidName(String name) {
        return name != null && name.length() > 0 && name.split(" ").length > 2;
    }

    public String getName() {
        return name;
    }
}
```

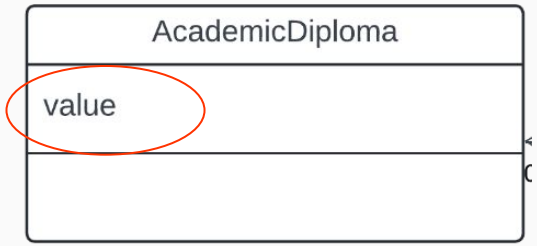


Stockholms universitet

AcademicDiploma

```java
public enum AcademicDiploma {
    BACHELOR,
    MASTER,
    LICENTIATE,
    DOCTORATE
}
```



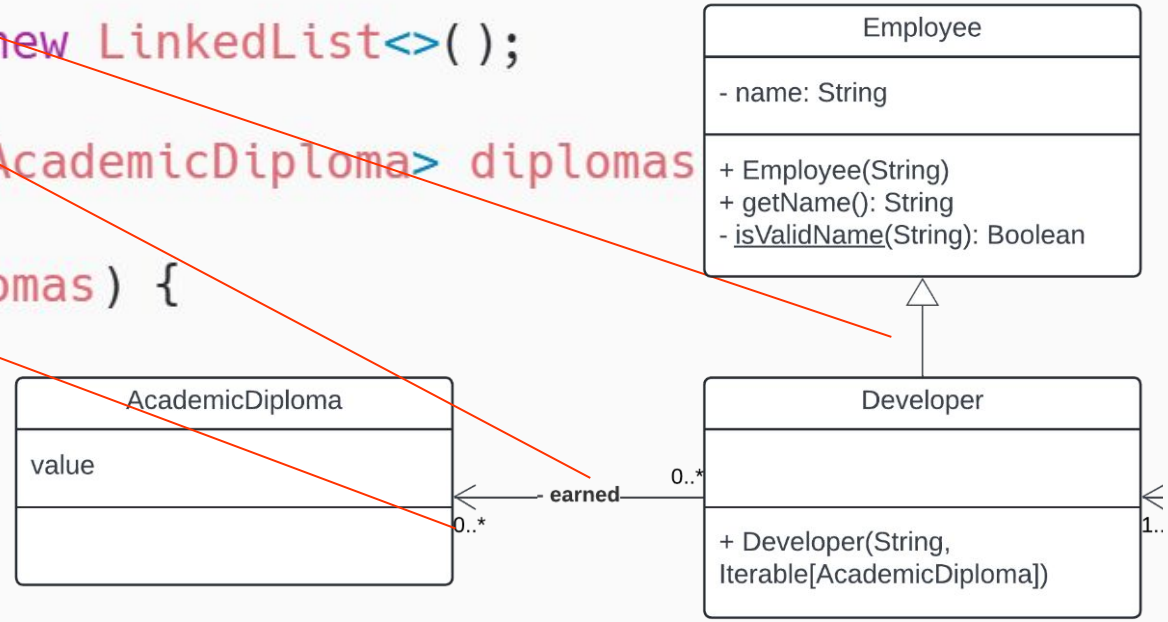


- Implementation detail
- Set of specified values
  - "Enum" - Enumeration
  - In practice usually "mapped" to an int, or the entire key (str) is stored
- Never/Rarely redefined → Recompilation

Stockholms universitet

Developer

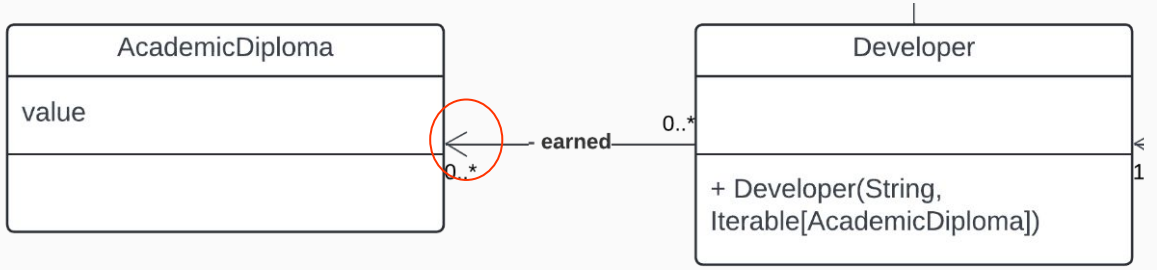
```java
import java.util.LinkedList;
import java.util.List;

public class Developer extends Employee {
    private List<AcademicDiploma> earned = new LinkedList<>();

    public Developer(String name, Iterable<AcademicDiploma> diplomas
        super(name);
        for (AcademicDiploma diploma : diplomas) {
            earned.add(diploma);
        }
    }
}

```



Stockholms universitet

AcademicDiploma ← Developer

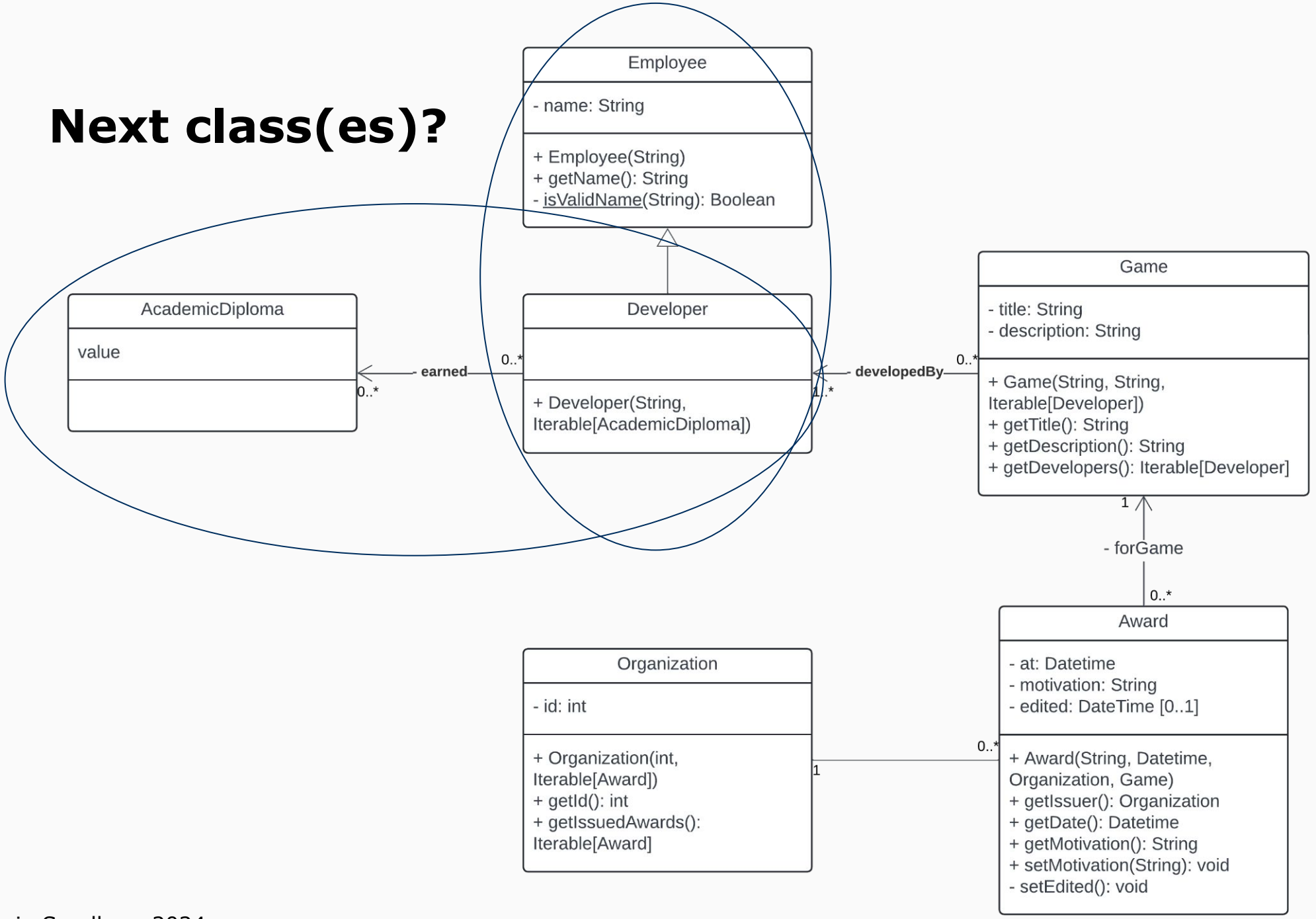
- Directed association!
  - Only developer keeps track of this association



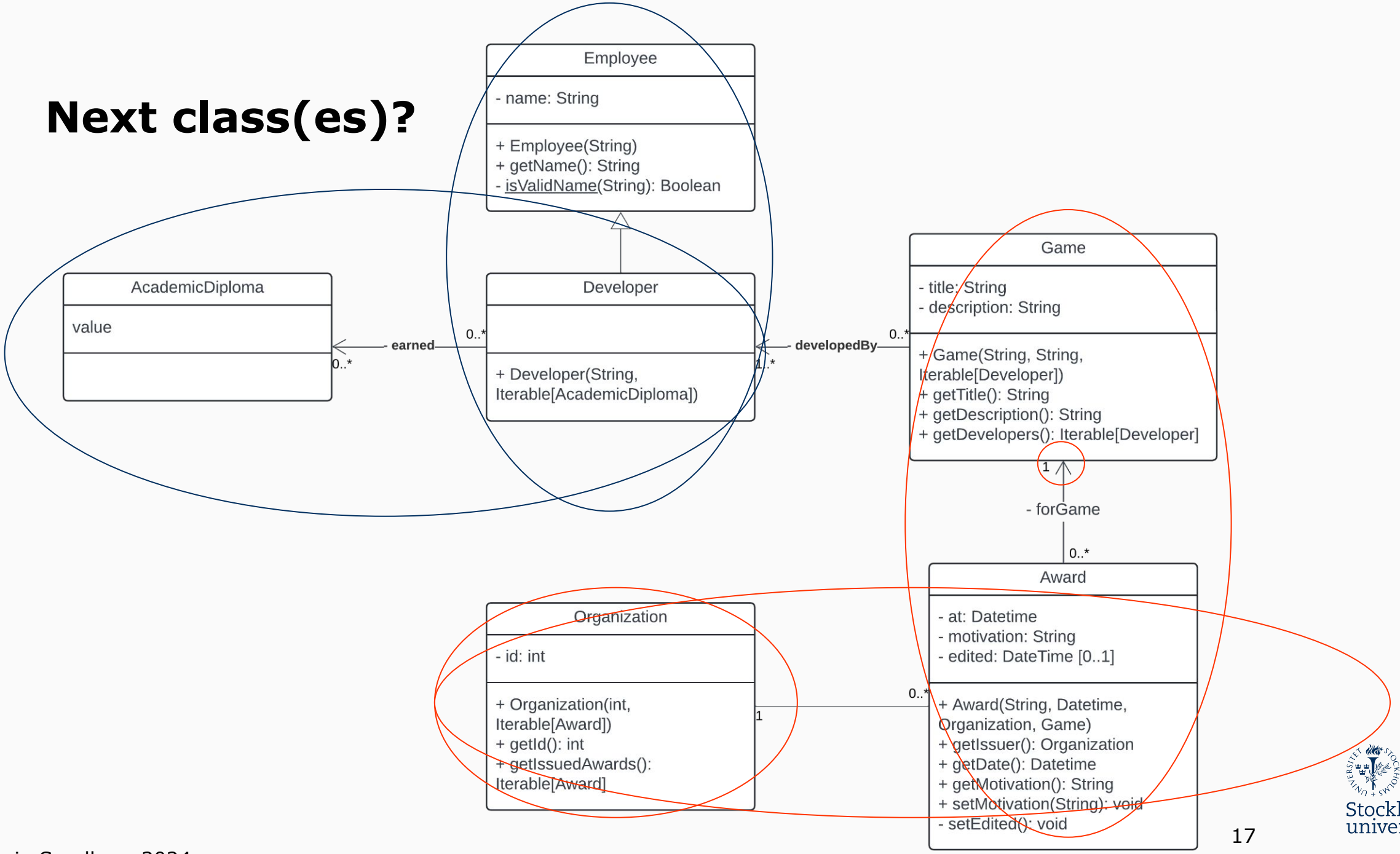
- Practical implications:
  - We can’t (without an external data structure) aggregate developers by expertise
  - “Which diplomas does this developer have?” → Cheap operation
  - “Which developers have the BACHELOR diploma?” → Expensive operation

Stockholms universitet

Next class(es)?

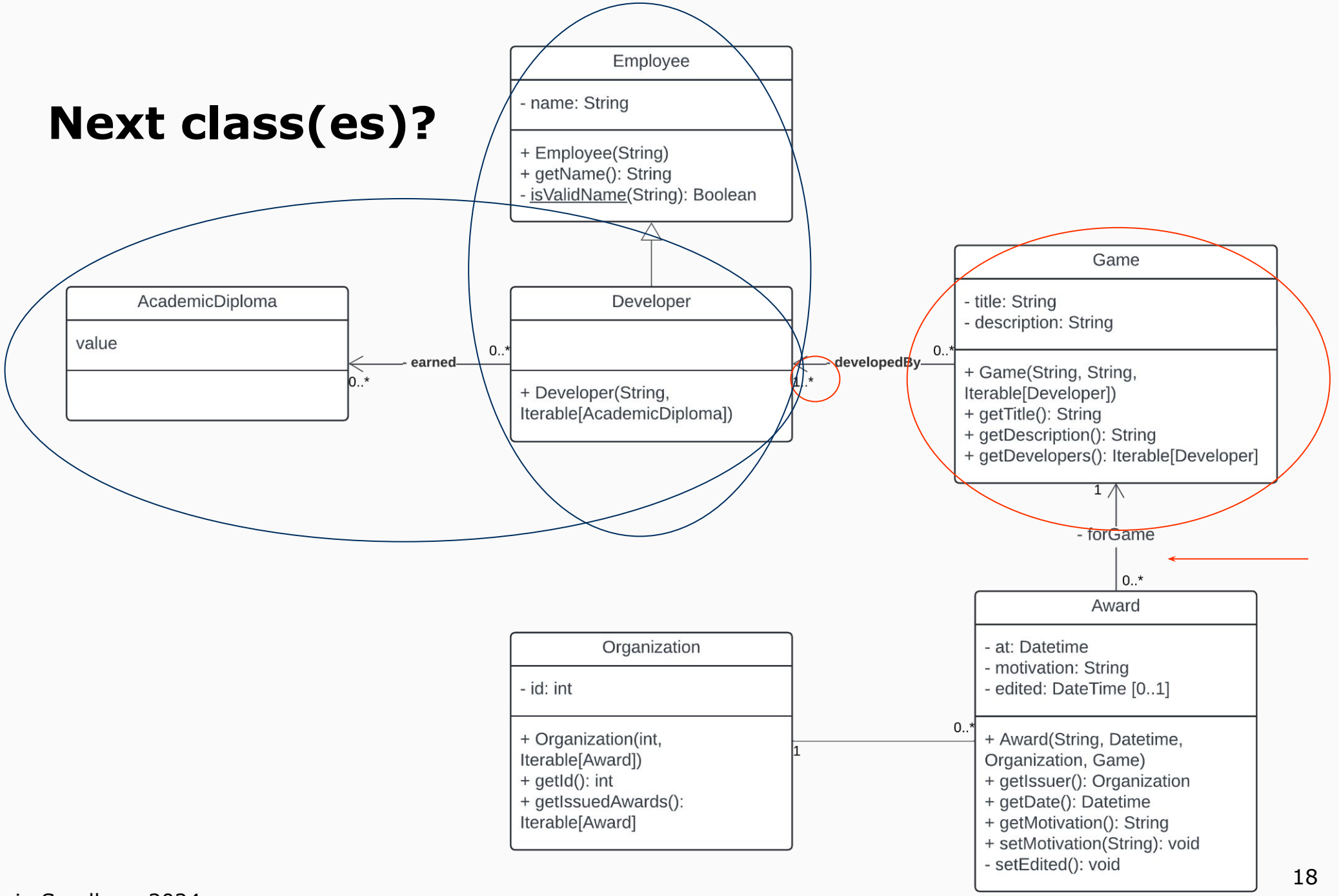


Next class(es)?



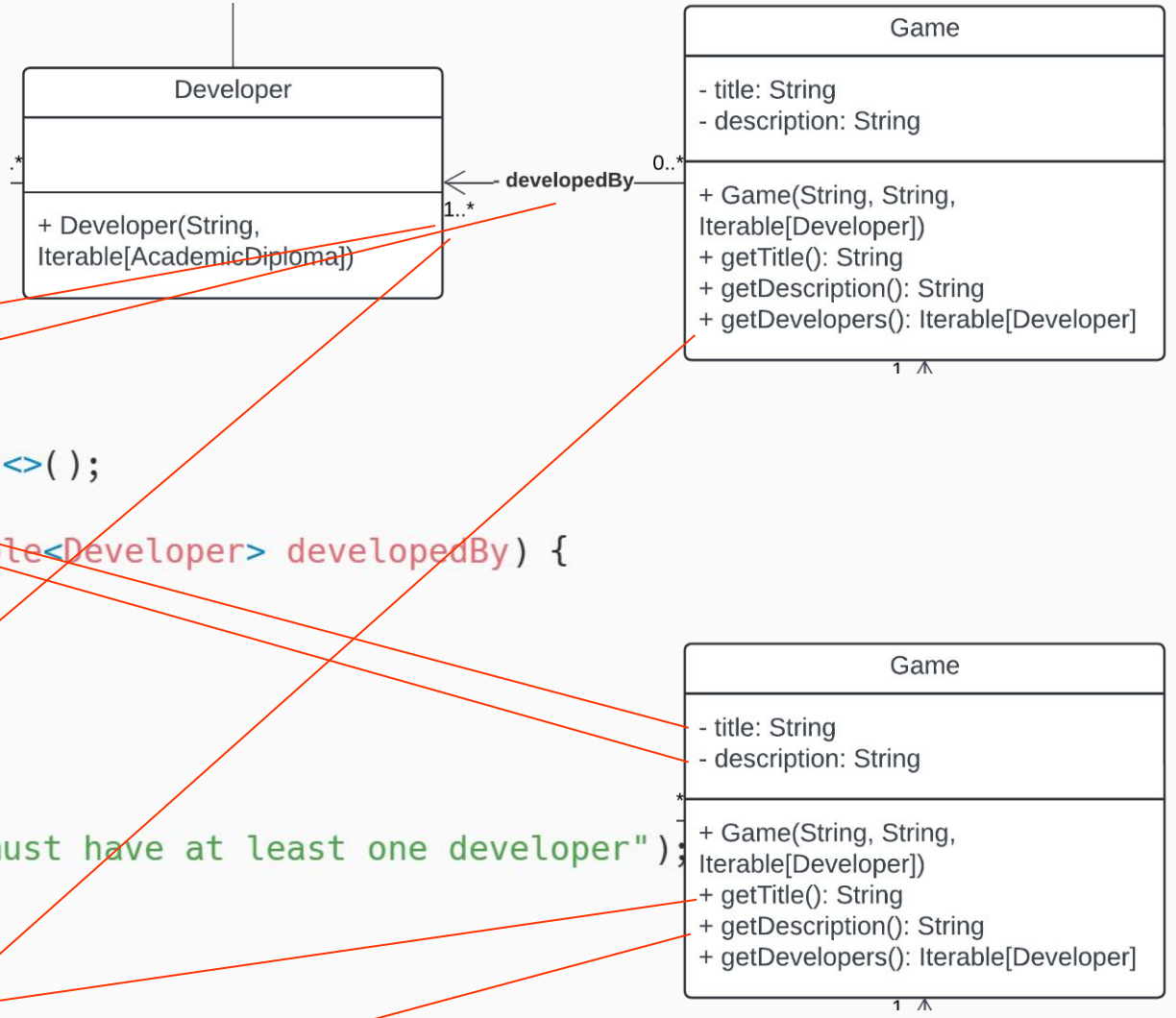
Stockholms universitet

Next class(es)?

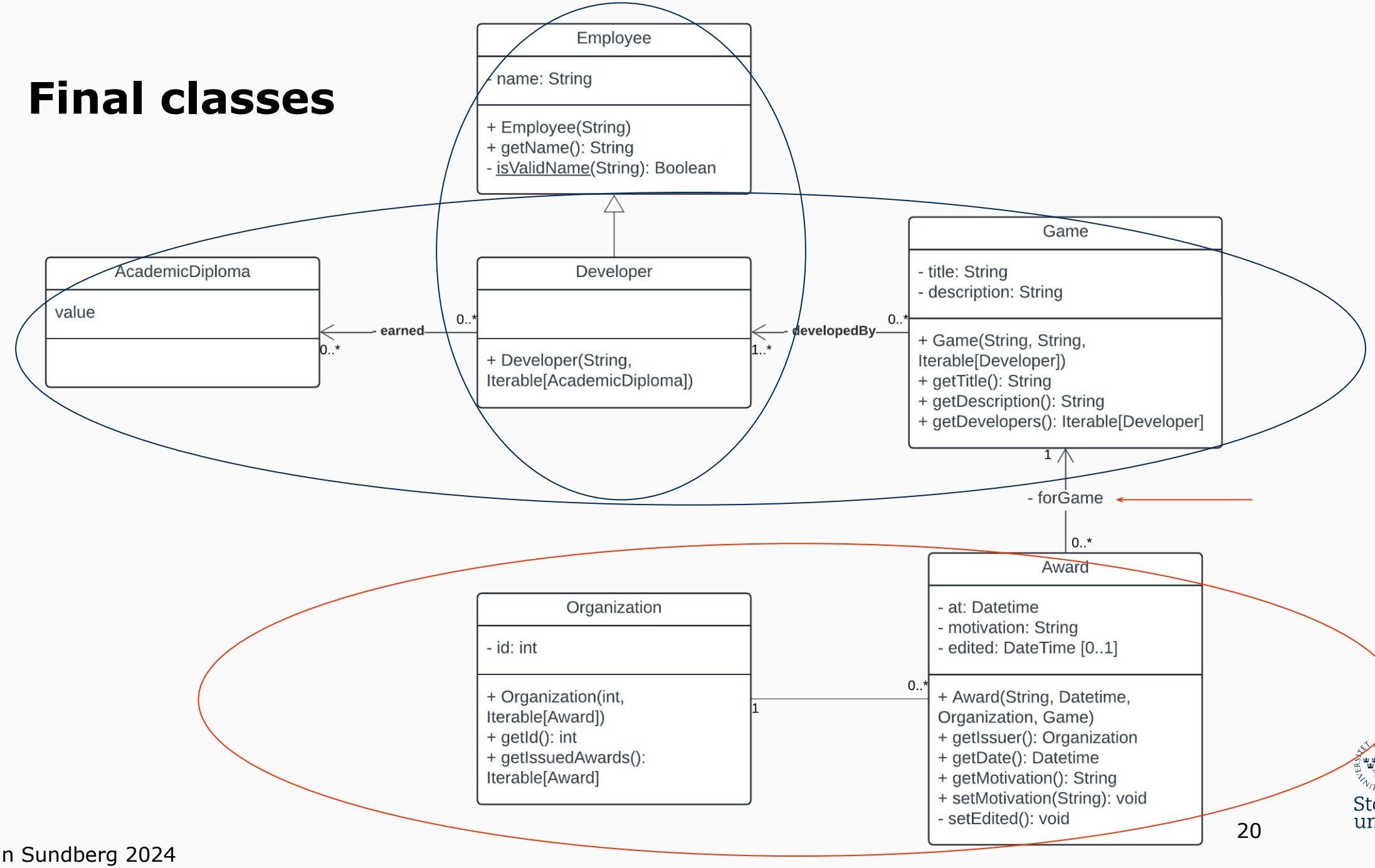


Game

```java
import java.util.LinkedList;
import java.util.List;

public class Game {
    private String title;
    private String description;
    private List<Developer> developedBy = new LinkedList<>();

    public Game(String title, String description, Iterable<Developer> developedBy) {
        this.title = title;
        this.description = description;
        for (Developer developer : developedBy) {
            this.developedBy.add(developer);
        }
        if (this.developedBy.size() == 0) {
            throw new IllegalArgumentException("A game must have at least one developer");
        }
    }

    public String getTitle() { return title; }

    public String getDescription() { return description; }

    public Iterable<Developer> getDevelopers() { return developedBy; }
}
```



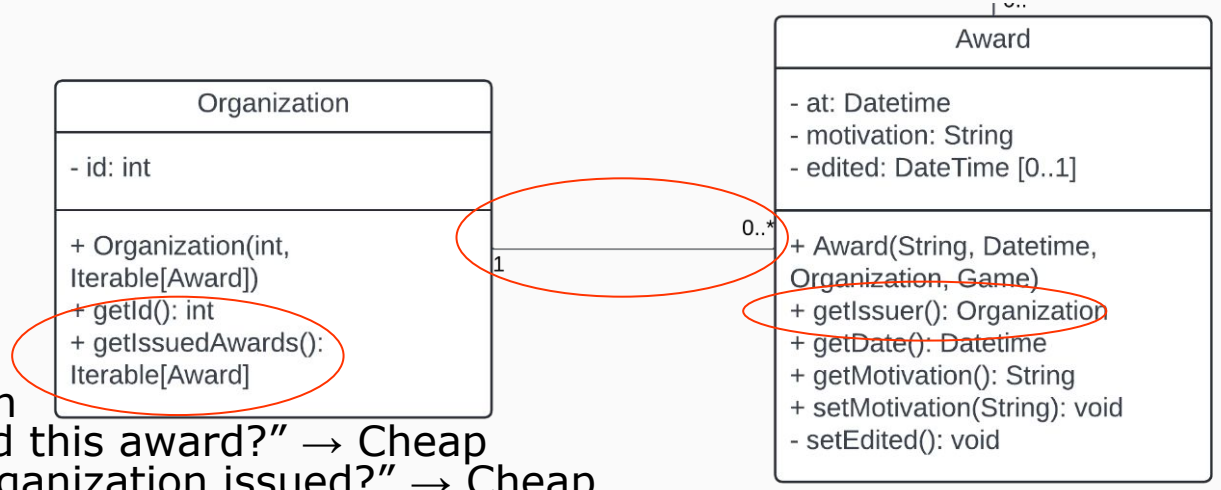
Final classes



Stockholms universitet

Bidirectional & Unnamed

- Bidirectional association!
  - Both classes keeps track of this association (2 references/pointers)

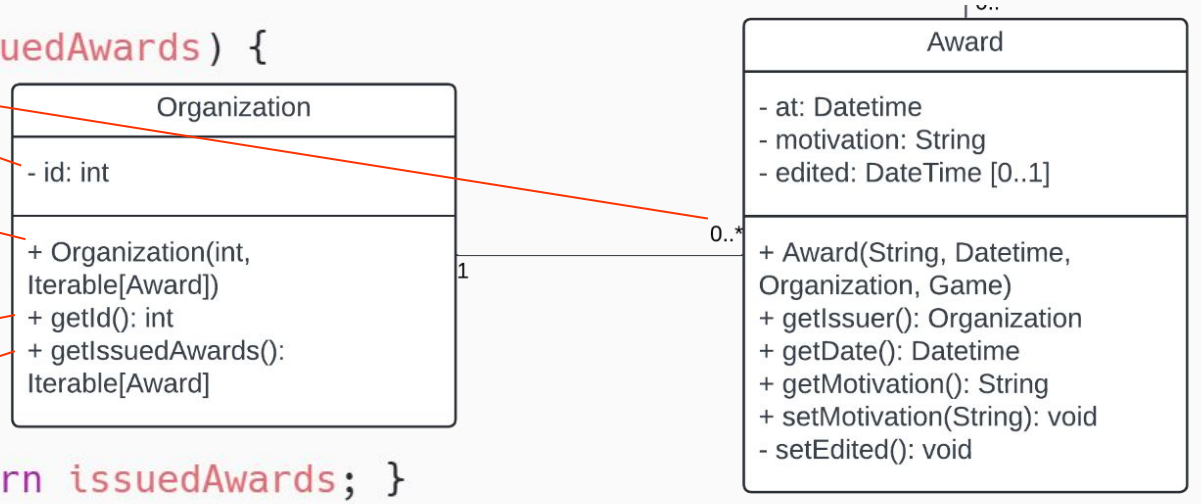


- Practical implications:
  - Easier to aggregate information
    - "Which organization issued this award?" → Cheap
    - "Which awards has this organization issued?" → Cheap
  - But:
    - More expensive in terms of memory usage
    - More complex to implement to ensure consistent state
- Unnamed
  - Up to the developer - but hints!

Stockholms universitet

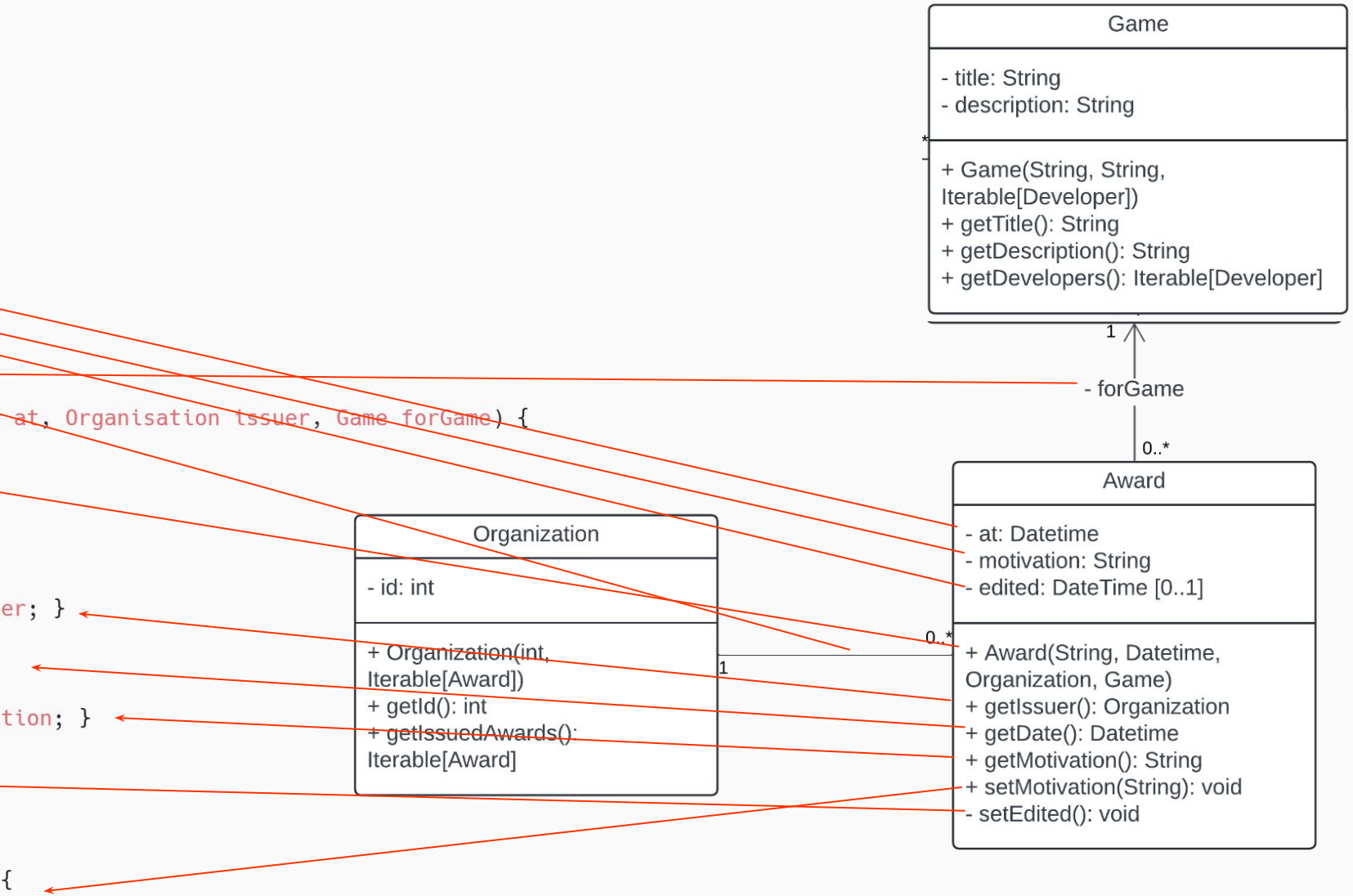
Organisation

```java
import java.util.LinkedList;
import java.util.List;

public class Organisation {
    private int id;
    private List<Award> issuedAwards = new LinkedList<>();

    public Organisation(int id, Iterable<Award> issuedAwards) {
        this.id = id;
        for (Award award : issuedAwards) {
            this.issuedAwards.add(award);
        }
    }

    public int getId() { return id; }

    public Iterable<Award> getIssuedAwards() { return issuedAwards; }
}

```



Award

```java
import java.time.LocalDateTime;

public class Award {
    private LocalDateTime at;
    private String motivation;
    private LocalDateTime editedAt;
    private Organisation issuer;
    private Game forGame;

    public Award(String motivation, LocalDateTime at, Organisation issuer, Game forGame) {
        this.motivation = motivation;
        this.at = at;
        this.issuer = issuer;
        this.forGame = forGame;
    }

    public Organisation getIssuer() { return issuer; }

    public LocalDateTime getDate() { return at; }

    public String getMotivation() { return motivation; }

    private void setEdited() {
        this.editedAt = LocalDateTime.now();
    }

    public void setMotivation(String motivation) {
        this.motivation = motivation;
        setEdited();
    }

}
```



Language Differences

Short look over programming languages

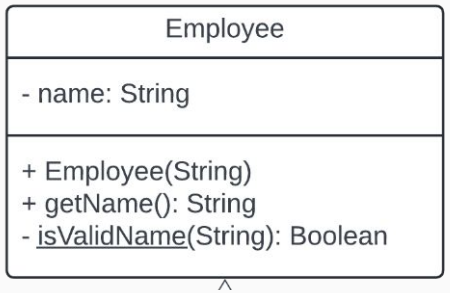
Stockholms universitet

Naming Conventions

camelCase, PascalCase, snake_case, kebab-case

```java
public class Employee {
    private String name;

    public Employee(String name) {
        if (!isValidName(name)) {
            throw new IllegalArgumentException("Invalid name");
        }
        this.name = name;
    }

    public static boolean isValidName(String name) {
        return name != null && name.length() > 0 && name.split(" ").length > 2;
    }

    public String getName() {
        return name;
    }
}
```

```python
class Employee:
    def __init__(self, name: str):
        if not Employee.is_valid_name(name):
            raise ValueError(f"Invalid name: {name}")
        self._name = name

    @staticmethod
    def is_valid_name(name: str) -> bool:
        return len(name) > 0 and len(name.split()) > 1

    def get_name(self) -> str:
        return self._name
```



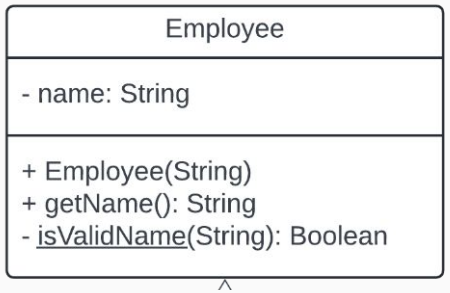
Stockholms universitet

Protection / Access Modifiers

Modifiers & Conventions

```java
public class Employee {
    private String name;

    public Employee(String name) {
        if (!isValidName(name)) {
            throw new IllegalArgumentException("Invalid name");
        }
        this.name = name;
    }

    public static boolean isValidName(String name) {
        return name != null && name.length() > 0 && name.split(" ").length > 2;
    }

    public String getName() {
        return name;
    }
}
```

```python
class Employee:
    def __init__(self, name: str):
        if not Employee.is_valid_name(name):
            raise ValueError(f"Invalid name: {name}")
        self._name = name

    @staticmethod
    def is_valid_name(name: str) -> bool:
        return len(name) > 0 and len(name.split()) > 1

    def get_name(self) -> str:
        return self._name
```



Stockholms universitet

Code & Material

- Java/Python code & slides available at https://github.com/Edwinexd/dsv-oos-uml-2-code

Stockholms universitet