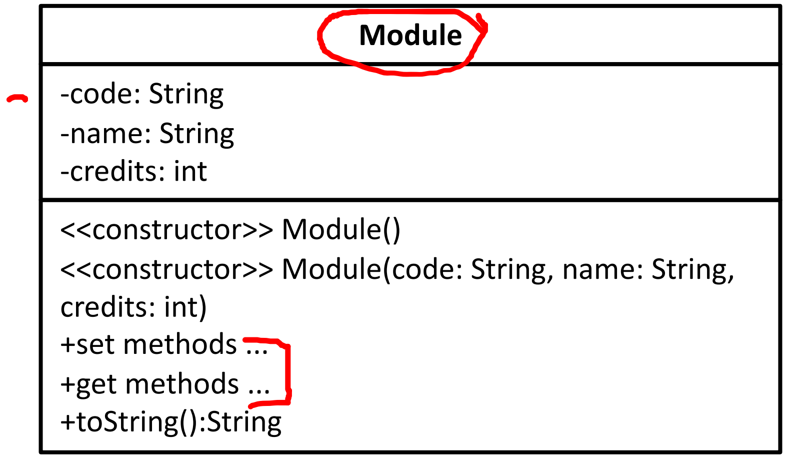
Association, Aggregation and Composition

- There are different types of relationships between classes
- We have already seen and made use of the inheritance or “is-a” relationship
- In this lesson we are going to look at:
  - Association – a description of an activity between classes
  - Aggregation and Composition which are two “has-a” relationships
- We will be revisiting our university people classes to do this,
- but first we need a Module class...

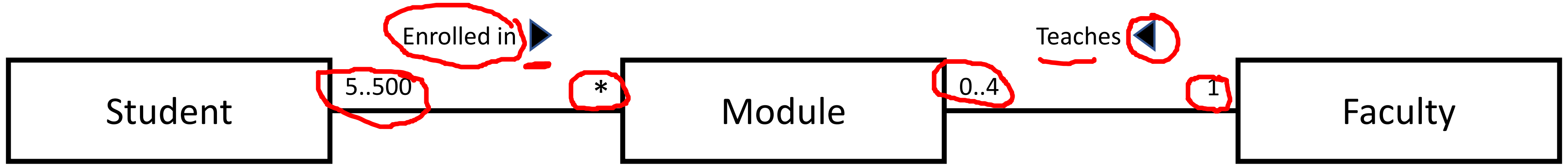

```
Module
-code: String
-name: String
-credits: int
<<constructor>> Module()
<<constructor>> Module(code: String, name: String,
credits: int)
+set methods ...
+get methods ...
+toString():String
```

CMPG 221

Association

- An association describes an activity between classes
- Lifecycles are independent, no ownership.
- In our university people setting we can have associations describing the following activities:
  - A student is enrolled in a module
  - A faculty member teaches a module
- There can also be multiplicities with the associations:
  - A student can be enrolled for any number of modules
  - A module can have between 5 and 500 students enrolled in it
  - A faculty member can teach between 0 and 4 modules a semester
  - A module is taught by one faculty member
- We use UML to show these associations and multiplicities as follows:

Association: UML diagram



- In this UML diagram connecting lines are used to show the associations
- Each association (activity) can have a name: Enrolled in and Teaches
- The arrowhead shows the direction of the activity, Faculty teaches the Module, not Module teaches Faculty
- The number of students enrolled in a module is specified by 5..500, while the * specifies that a student can be enrolled in any number of modules
- A module is taught by 1 faculty member and a faculty member can teach between 0 and 4 modules in a semester

Association: Java

- We use instance variables and methods to implement associations in Java
- In Student we need:
  - a list of modules – private Module[] moduleList;
  - a method to enroll in a Module – public void enrollModule(Module m) {...}
- In Faculty we need:
  - a list of modules – private Module[] moduleList;
  - a method to add a Module – public void addModule(Module m) {...}
- In Module we need:
  - a list of students – private Student[] classList;
  - a faculty member – private Faculty teacher;
  - a method to add a Student – public void addStudent(Student s) {...}
  - a method to set the teacher – public void setTeacher(Faculty f) {...}
- We will return to the actual implementation of association in the lesson on containers!

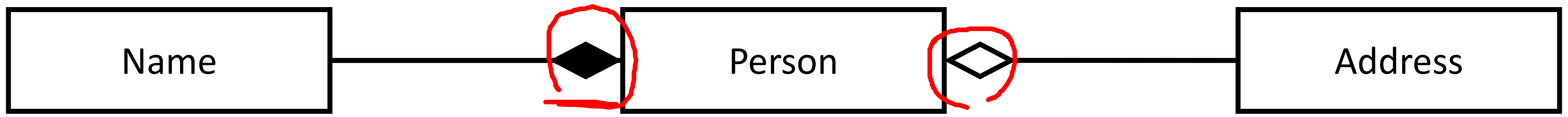
Aggregation and Composition

- These are ownership relationships – “has-a”
- In an aggregation relationship we have an owner and refer to the
  - aggregating class or aggregating object
- and a subject that is referred to as the
  - aggregated class or aggregated object
- If the existence of a subject (aggregated object) is dependent on an owner (aggregating object) then the relationship is a composition.
- Lifecycle dependency with composition is strong – when containing (owning) object is deleted the contained (owned) object is also deleted.

Association, Aggregation and Composition

	Association	Aggregation	Composition
Ownership	None	[✓ Yes]	[✓ Yes]
Lifecycle	Independent	Independent	Strongly dependent

Aggregation and Composition: UML diagrams



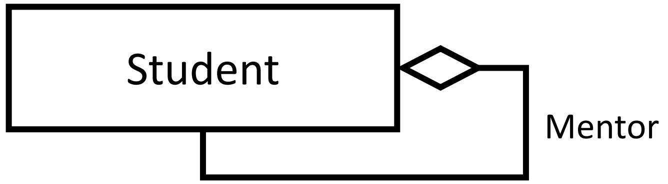
- A Person "has-a" Name, however a Name cannot exist without a Person
  - Therefore it is a composition relationship
  - denoted by the filled diamond arrowhead
- A Person "has-an" Address, however, an address can exist without a Person
  - Therefore it is an aggregation relationship
  - denoted by the empty diamond arrowhead

Aggregation and Composition: Java

- We use instance variables to implement the relationships of the previous UML diagram
- In Person we need:
  - an instance variable name – private Name name;
  - an instance variable address – private Address address;
  - their set and get methods –
    - public void setName(....) {name = new Name(...);}
    - public void setAddress(Address newAddress) {...}
    - public Name getName() { return name; }
    - public Address getAddress() { return address; }

Aggregation and Composition: UML diagrams

- An aggregation can exist between objects of the same class
- For example, a (senior) student can be a mentor to a (first year) student
- This is shown in the association diagram below
- The association "a student can have a mentor" is then implemented in Java by having an instance variable in the Student class as in –
  - private Student mentor;


Student
Mentor