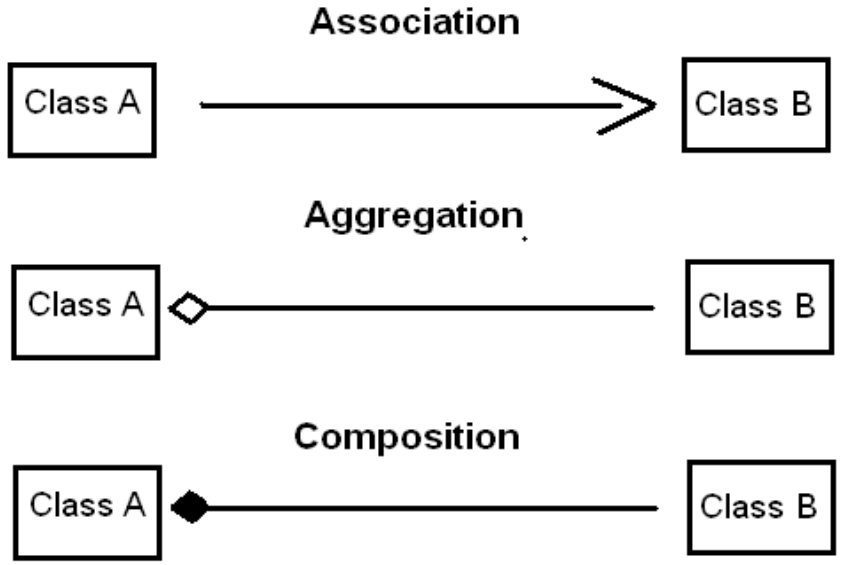
Java Programming AP Edition
U4C10 Object-Oriented Thinking

CLASS USE-RELATIONSHIPS

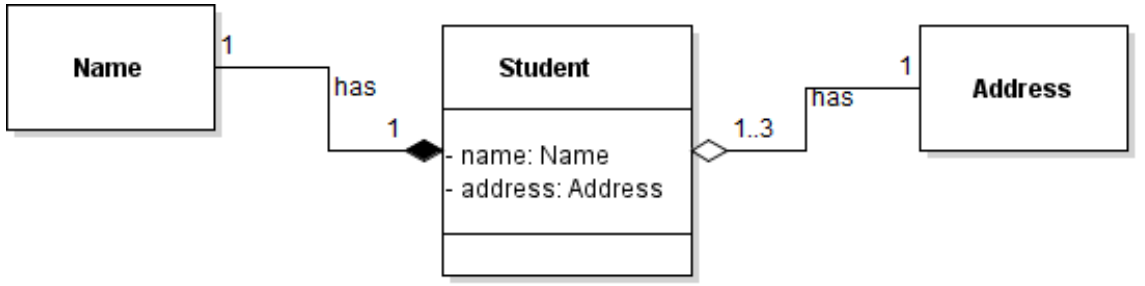
ERIC Y. CHOU, PH.D. IEEE SENIOR MEMBER

Class Relationships (use and inherit)

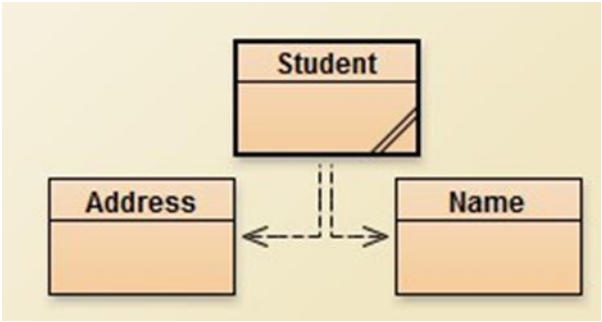
To design classes, you need to explore the relationships among classes. The common relationships among classes are *association*, *composition*, and *inheritance (later)*



Violet



BlueJ



These three relationship is use-relationship in BlueJ.

Association is general case. [many(one)-to-many(one)]

Aggregation is has-a relationship. (many-to-one/one-to-one)

Composition is exclusive has-a relation. (one-to-one)

Association

Association is a general binary relationship that describes an activity between two classes.

Association is a general binary relationship that describes an activity between two classes. For example, a student taking a course is an association between the **Student** class and the **Course** class, and a faculty member teaching a course is an association between the **Faculty** class and the **Course** class.

An association is illustrated by a **solid line** between two classes with an optional label that describes the relationship. (Sometimes an **arrow line** is used.) In Figure above, the labels are ***Take*** and ***Teach***. Each relationship may have an optional small black triangle that indicates the direction of the relationship. In this figure, the direction indicates that a student takes a course (as opposed to a course taking a student).

Multiplicity

placed at the side of the class to specify how many of the class's objects are involved in the relationship in UML.

A multiplicity could be a number or an interval that specifies how many of the class's objects are involved in the relationship.

The character * means an unlimited number of objects, and

the interval **m..n** indicates that the number of objects is between **m** and **n**, inclusively.

In Figure of previous page, each student may take any number of courses, and each course must have **at least five and at most sixty** students. Each course is taught by only one faculty member, and a faculty member may teach **from zero to three** courses per semester.

Association Relationship

```
public class Student {
  private Course[]
    courseList;

  public void addCourse(
    Course s) { ... }
}
```

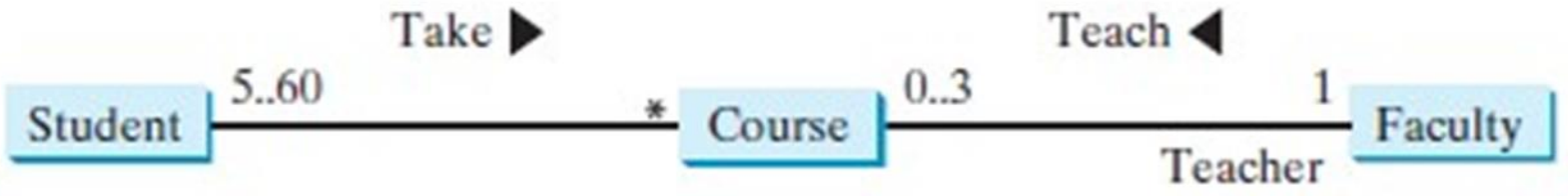
```
public class Course {
  private Student[]
    classList;
  private Faculty faculty;

  public void addStudent(
    Student s) { ... }

  public void setFaculty(
    Faculty faculty) { ... }
}
```

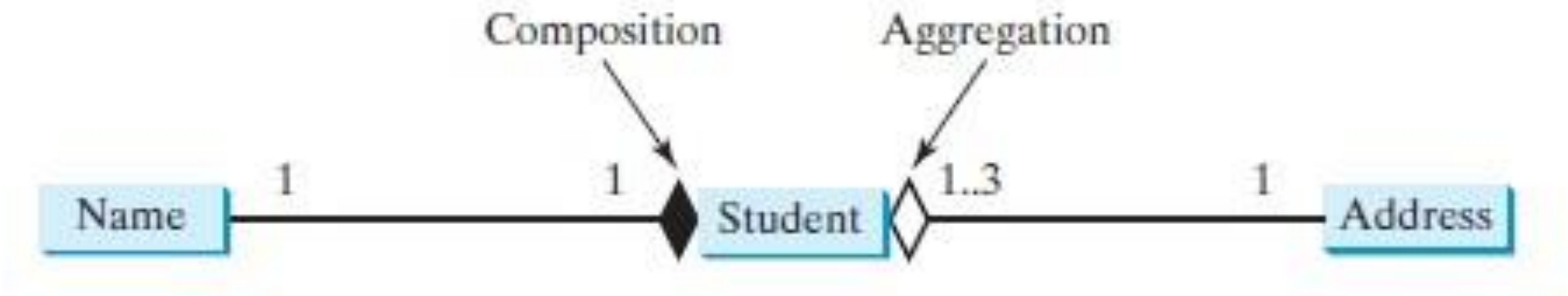
```
public class Faculty {
  private Course[]
    courseList;

  public void addCourse(
    Course c) { ... }
}
```



Aggregation or Composition

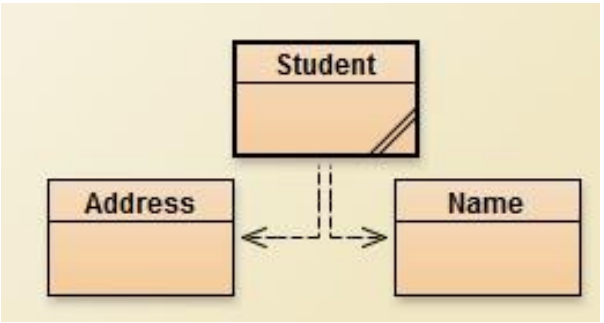
Since aggregation and composition relationships are represented using classes in similar ways, many texts don't differentiate them and call both compositions.



Student has a Name exclusively. (Composition)

Student has an address non-exclusively. (Aggregation)

Class Representation



An aggregation relationship is usually represented as a data field in the aggregating class. For example, the relationship in Figure of previous slide can be represented as follows:

```
public class Name {
  ...
}
```

Aggregated class

```
public class Student {
  private Name name;
  private Address address;
  ...
}
```

Aggregating class

```
public class Address {
  ...
}
```

Aggregated class

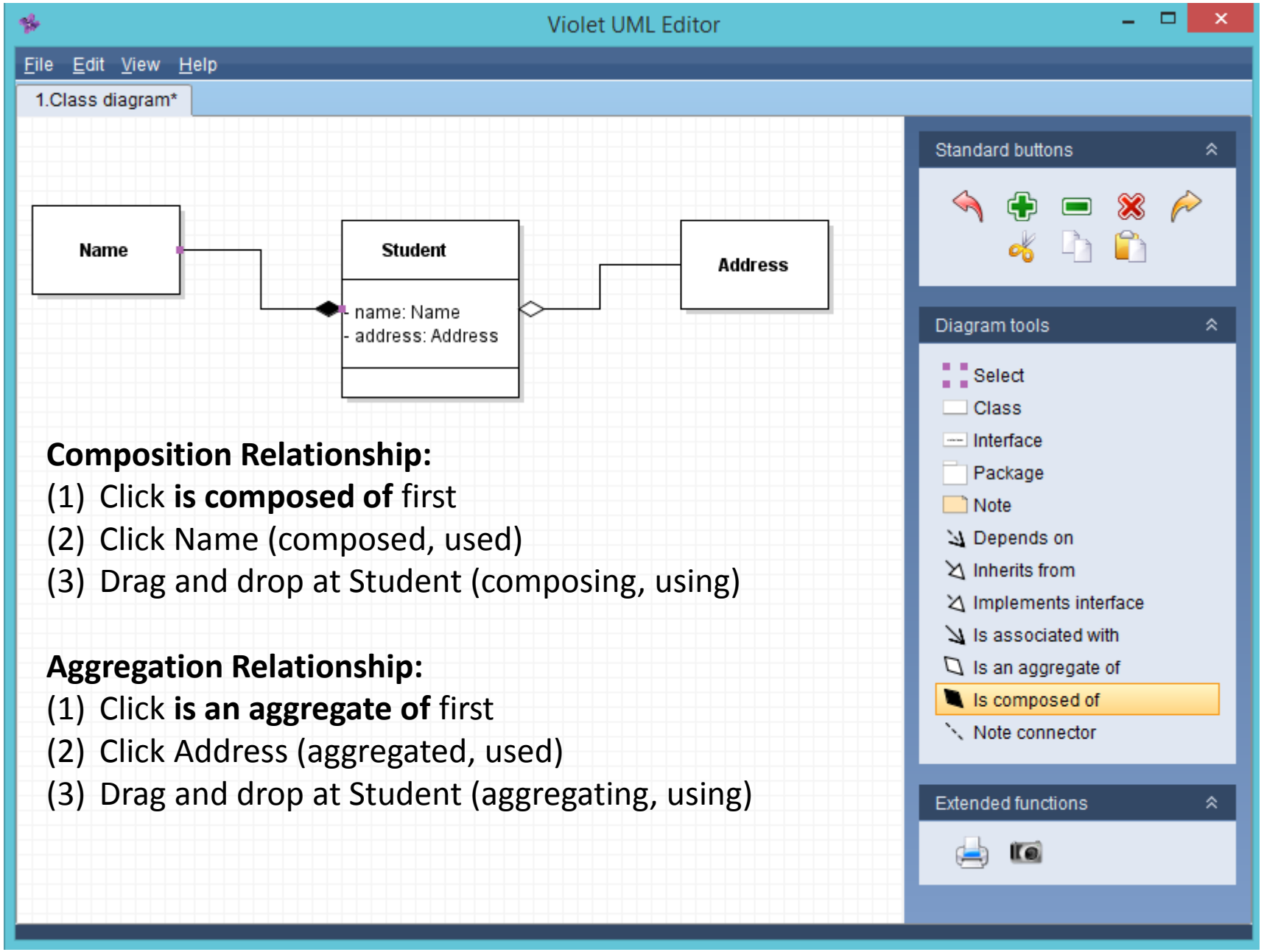
Aggregating: Using Aggregated: Used

Aggregation (has-a Relationship)

Aggregation is a special form of association that represents an ownership relationship between two objects. Aggregation models ***has-a*** relationships. The owner object is called an ***aggregating object***, and its class is called an ***aggregating class***. The subject object is called an ***aggregated object***, and its class is called an ***aggregated class***.

An object can be owned by several other **aggregating objects**.

If an object is exclusively owned by an aggregating object, the relationship between the object and its aggregating object is referred to as a ***composition***.



Composition Relationship:

(1) Click **is composed of** first

(2) Click Name (composed, used)

(3) Drag and drop at Student (composing, using)

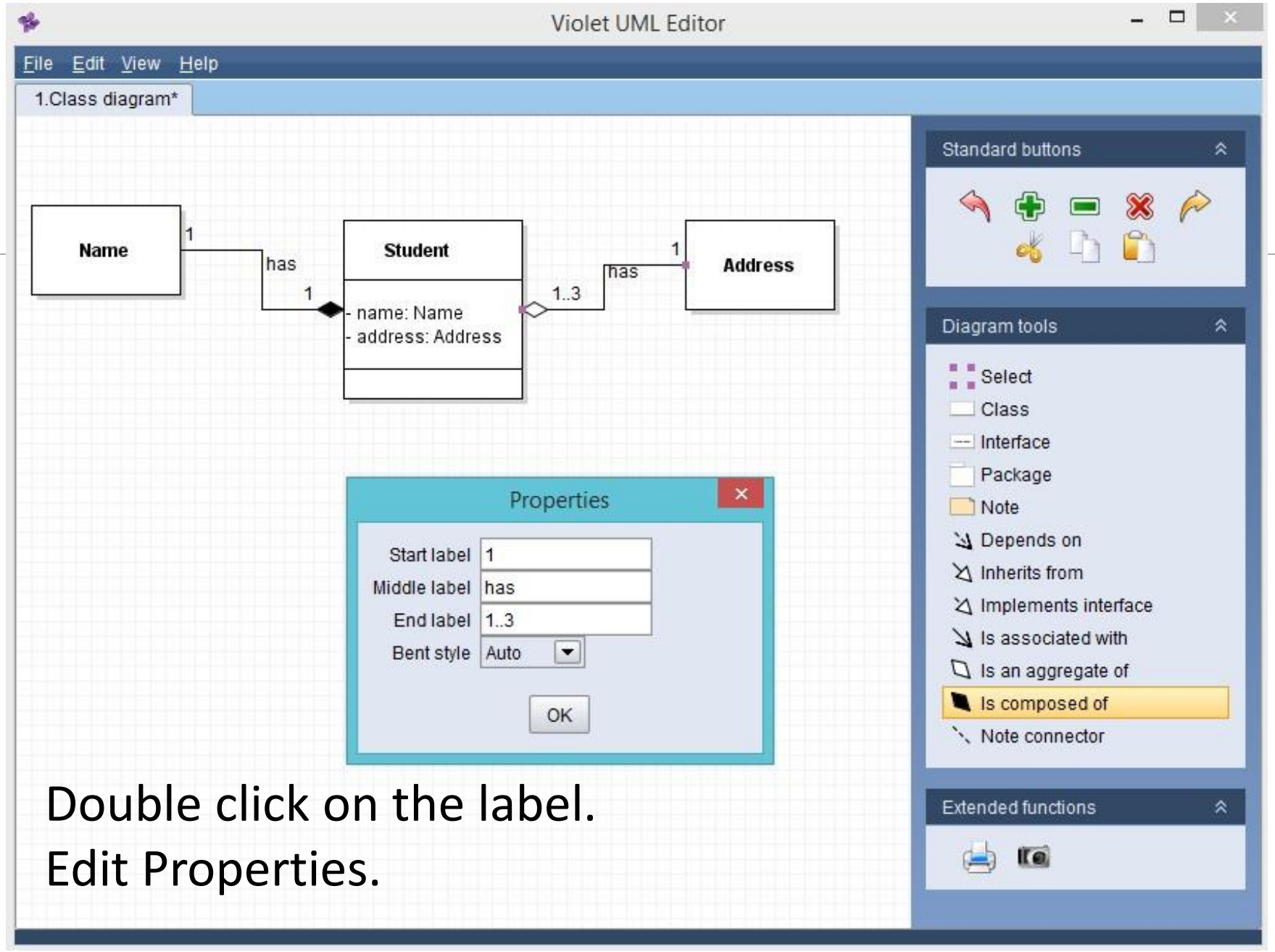
Aggregation Relationship:

(1) Click **is an aggregate of** first

(2) Click Address (aggregated, used)

(3) Drag and drop at Student (aggregating, using)

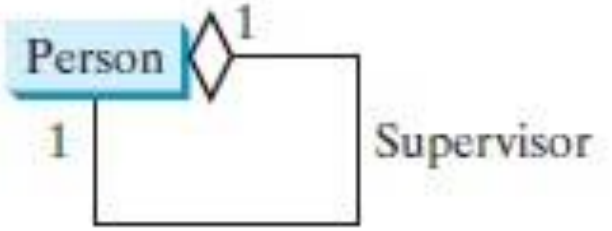
Labels



Double click on the label.
Edit Properties.

Aggregation Between Same Class

Aggregation may exist between objects of the same class. For example, a person may have a supervisor. **(Using itself once)**

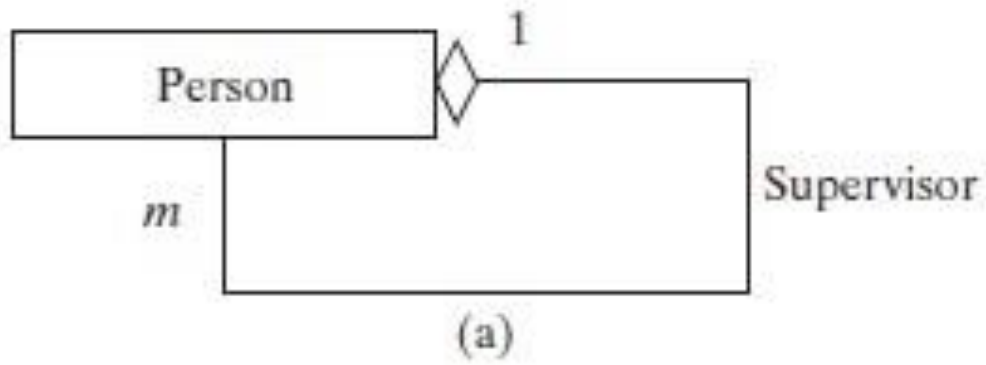



```
public class Person {
  // The type for the data is the class itself
  private Person supervisor;
  ...
}
```

Aggregation Between Same Class

What happens if a person has several supervisors?

(One class using itself many times)



(a)

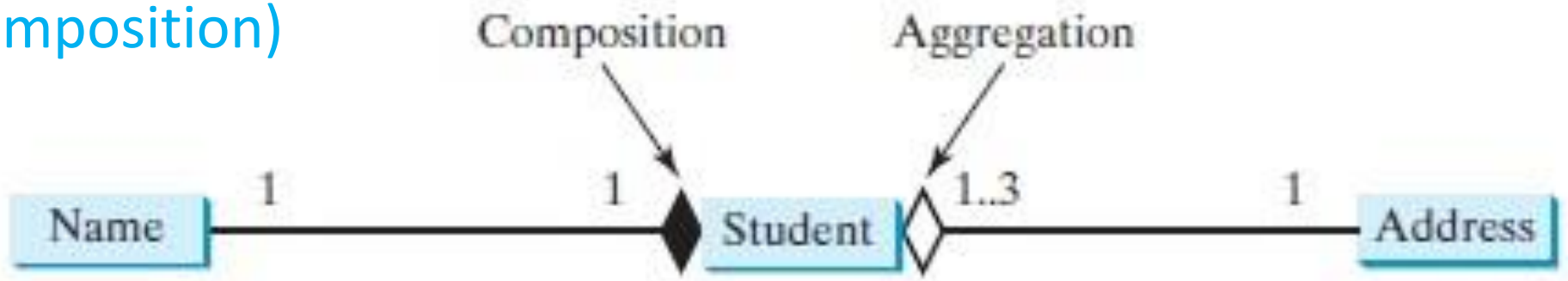
```
public class Person {
  ...
  private Person[] supervisors;
}
```

(b)

Object Composition

Composition is actually a special case of the aggregation relationship. Aggregation models *has-a* relationships and represents an ownership relationship between two objects. The owner object is called an *aggregating object* and its class an *aggregating class*. The subject object is called an *aggregated object* and its class an *aggregated class*.

Student has a Name exclusively.
(Composition)



Why analyze the multiplicity of relationship?

http://code.tutsplus.com/articles/sql-for-beginners-part-3-database-relationships--net-8561

Widely used in database design. When inner-join, outer-join are to be performed , analysis of these relationship is very essential.

It will be very important for data science.