Object-Oriented Programming

UML and Class Relationship

United International College

Outline

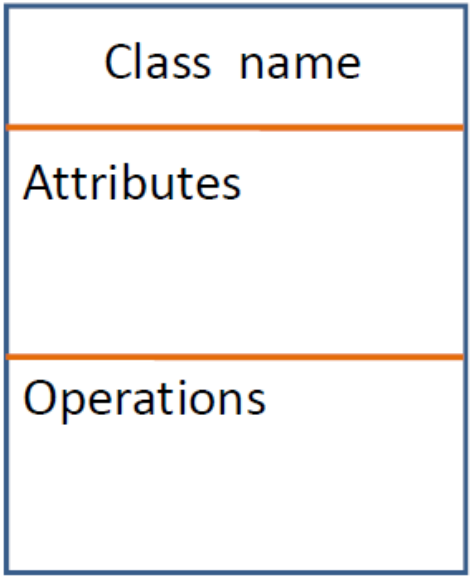
- **UML**
- **Class Diagram**
- **Class Relationship**
- **Class Diagram → Java code**

Unified Modeling Language

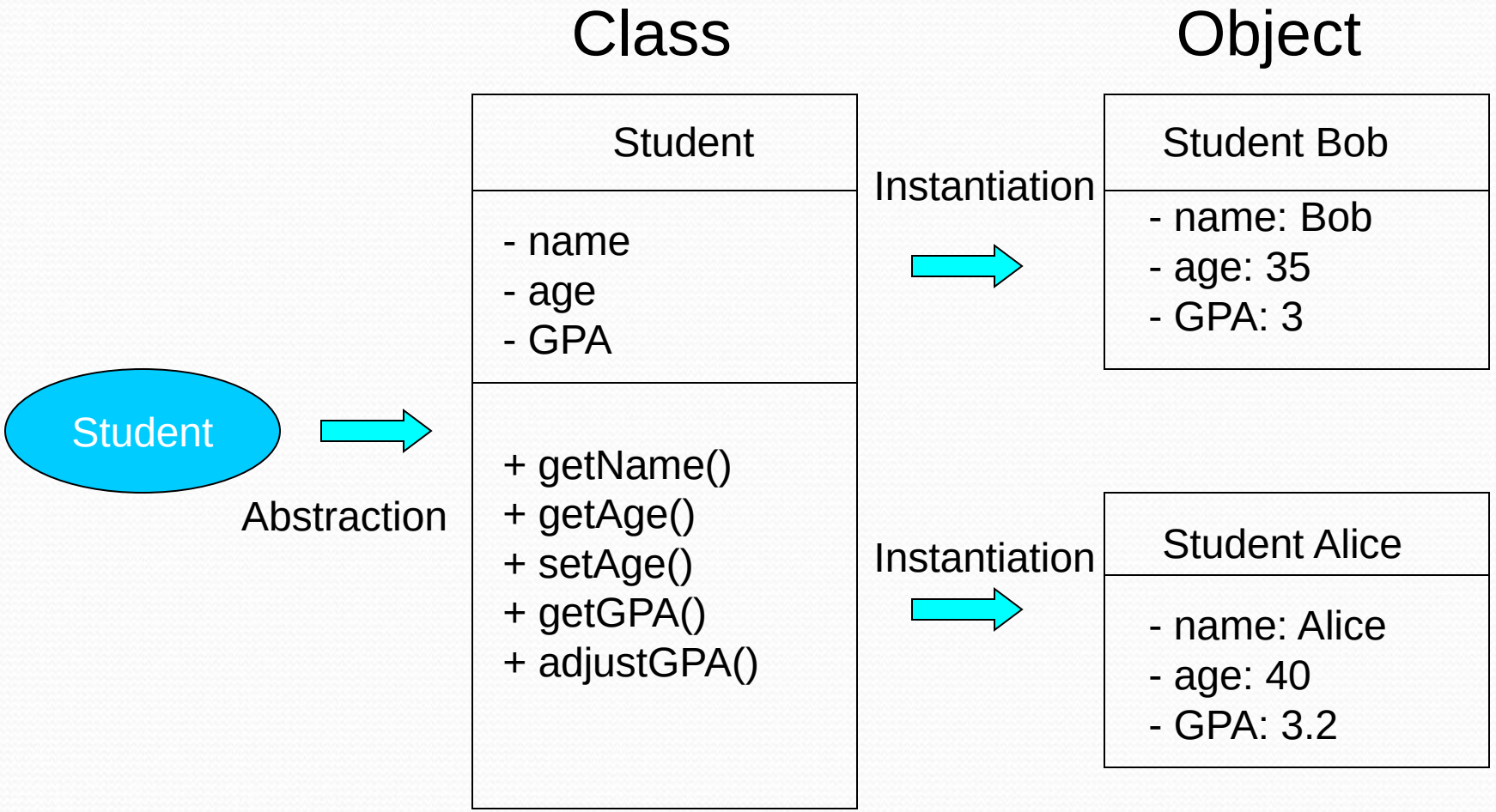
- The most popular diagrammatic notation used for Object-Oriented Development.
- Consists of:
  - **Class diagrams;**
  - Sequence diagrams;
  - Use case diagrams;
  - Activity diagrams;
  - …

Class Diagrams

- Describe the system in terms of classes and their relationships.
- Natural ways of reflecting the real-world entities and their relationships.
- Essential part in OO software Development.



Example

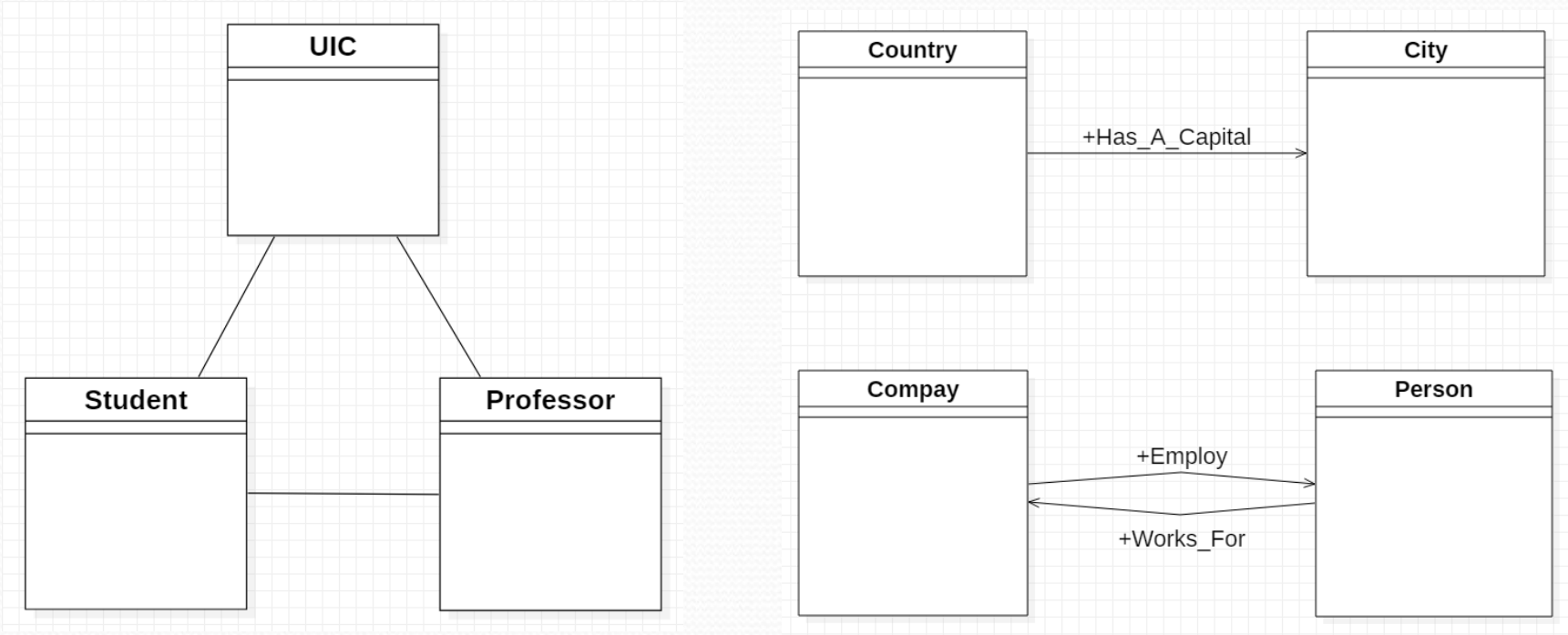

Class
Object
Student
Abstraction
Student
- name
- age
- GPA
+ getName()
+ getAge()
+ setAge()
+ getGPA()
+ adjustGPA()
Instantiation
Student Bob
- name: Bob
- age: 35
- GPA: 3
Instantiation
Student Alice
- name: Alice
- age: 40
- GPA: 3.2

Relationships Between Classes

- Association (directional + Multiplicity).
- Aggregation.
- Composition.
- Inheritance.
- Polymorphism.

Association(关联)

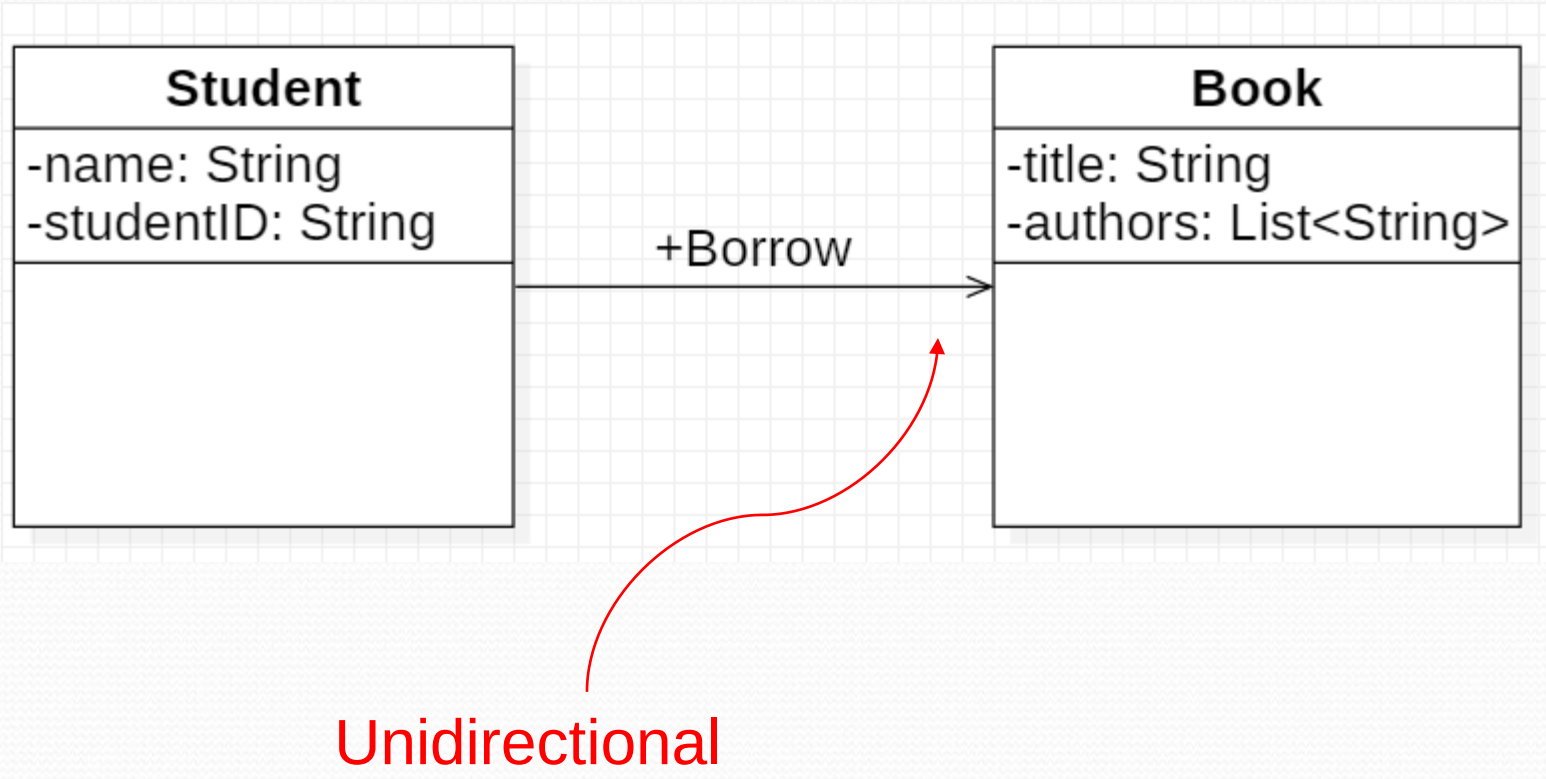
- Association represents a general binary relationship that describes an activity between two classes.
- A class is aware of and holds a reference to another class.
- Bidirectional or unidirectional.



Association relationship in a UML diagram

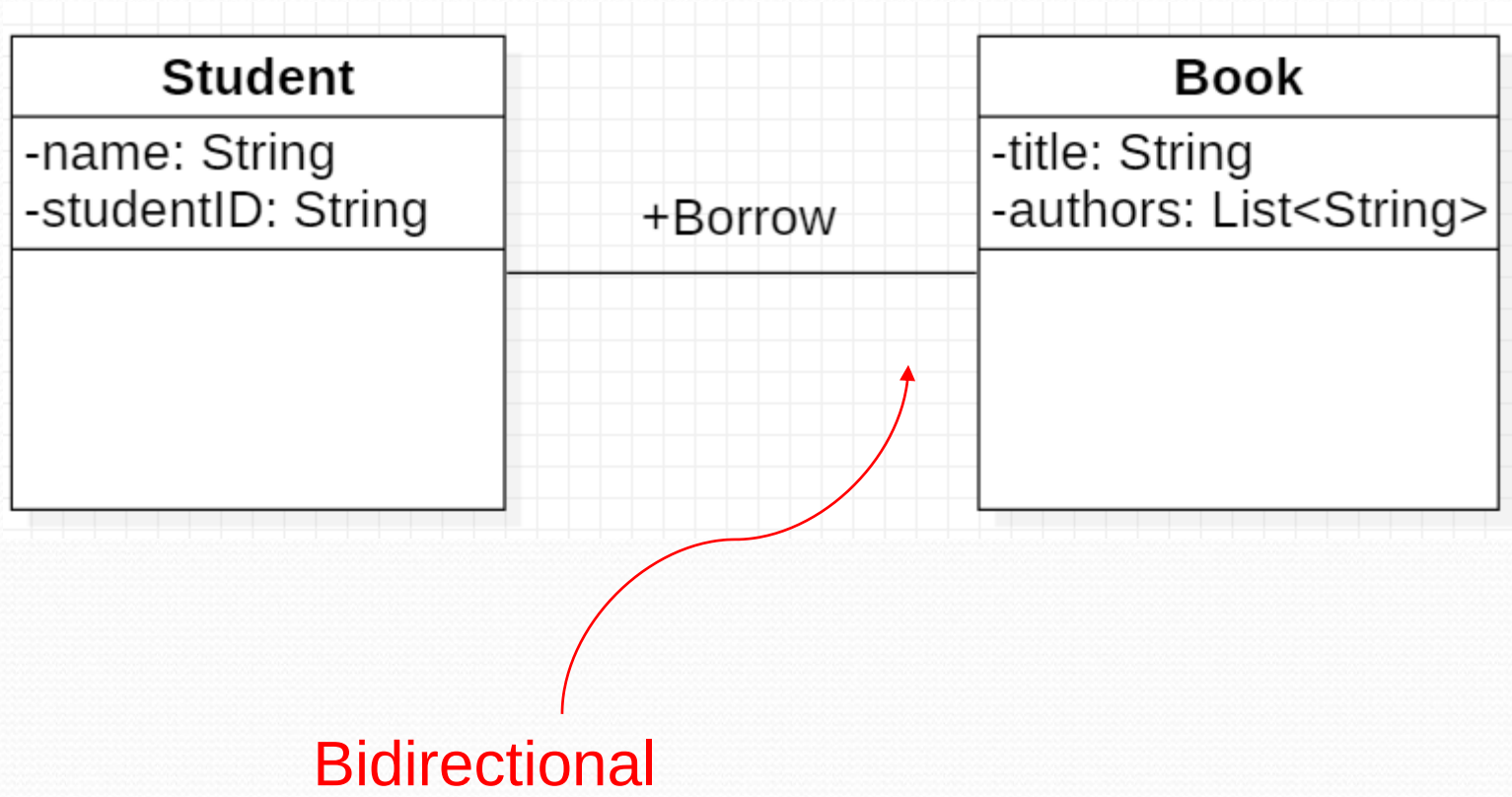
Unidirectional Association

- A student can query the books he/she borrowed but it is NOT possible to find which student the book is lent to.



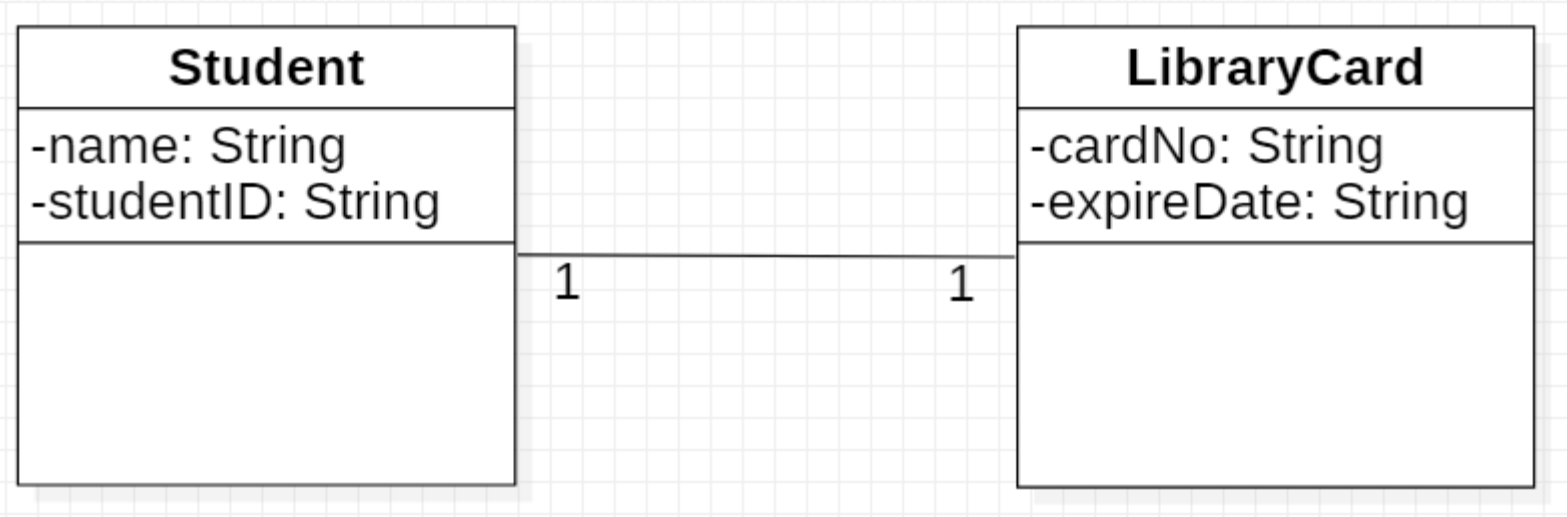
Bidirectional Association

- A student can query the books he/she borrowed and it is possible to find which student the book is lent to.



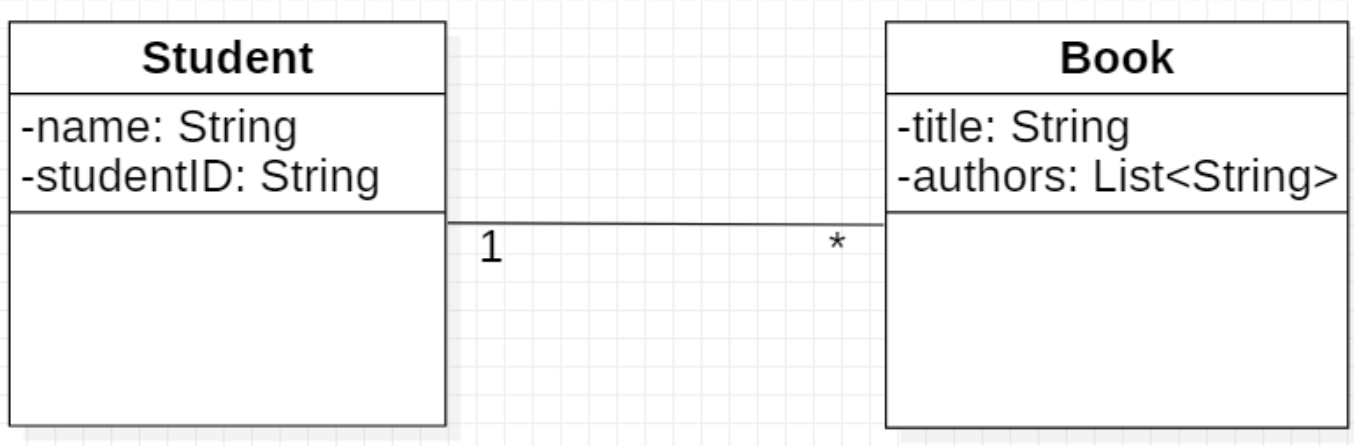
Multiplicity

- One student has only one library card, and one library card can only be owned by one student.

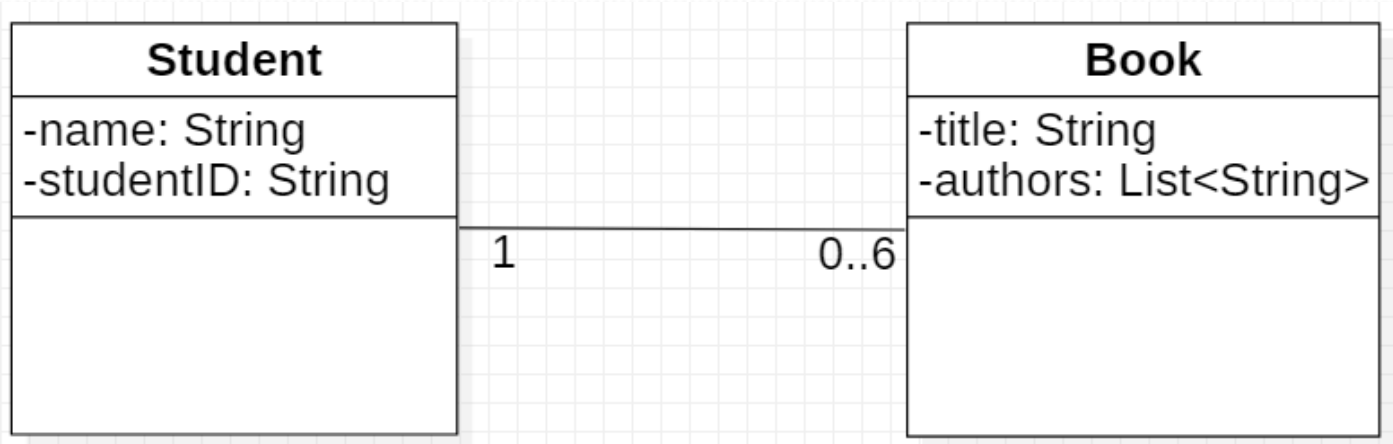


One to one relationship

Multiplicity



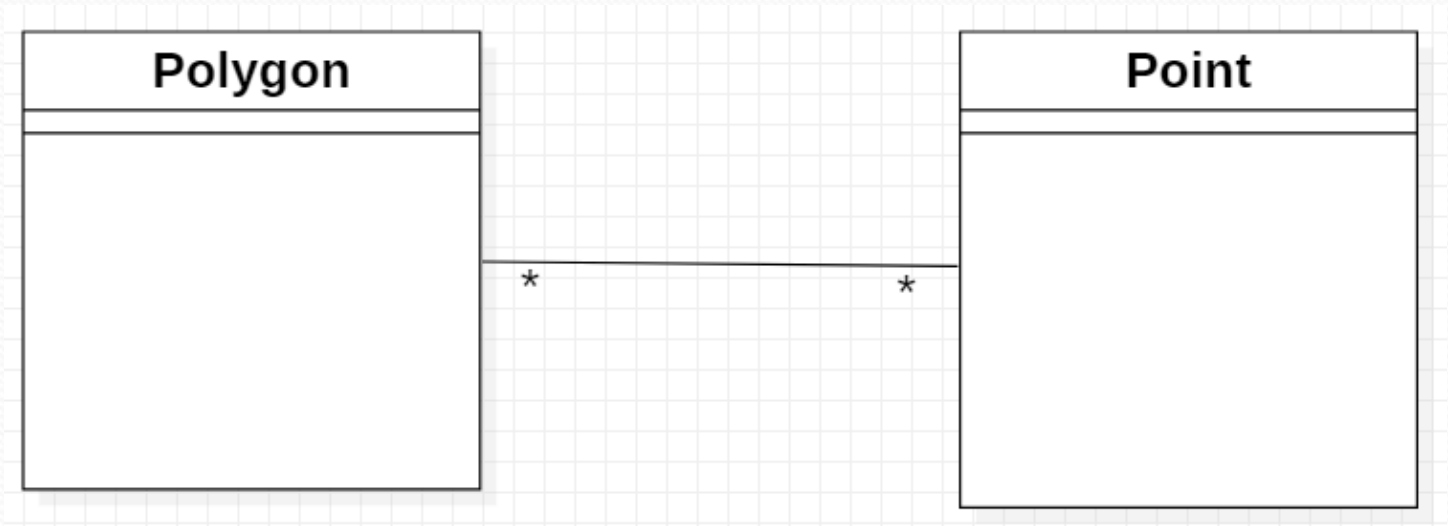
One student can borrow 0 or many books



One student can at most 6 books

Multiplicity

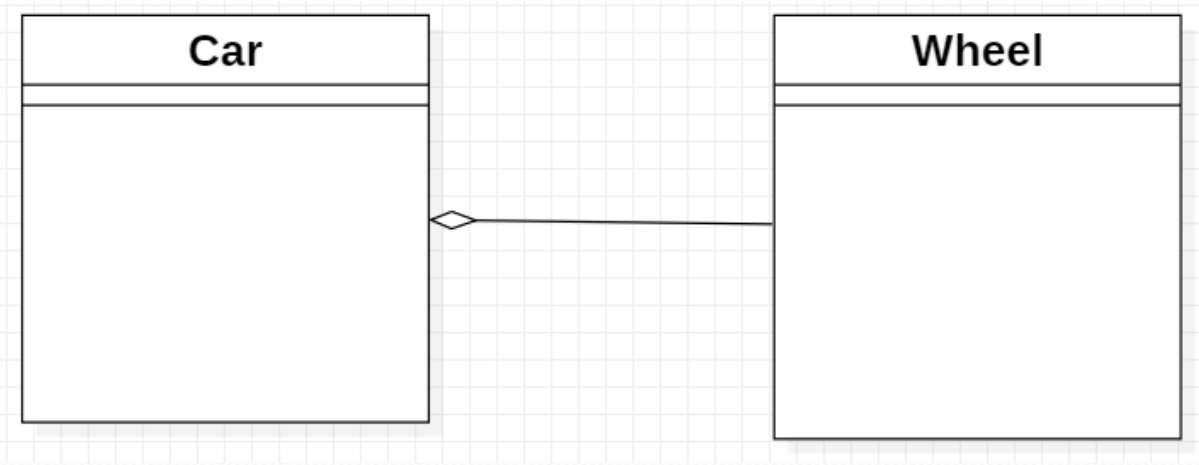
- One polygon has many points and one point can be in many polygons.



Many to many relationship

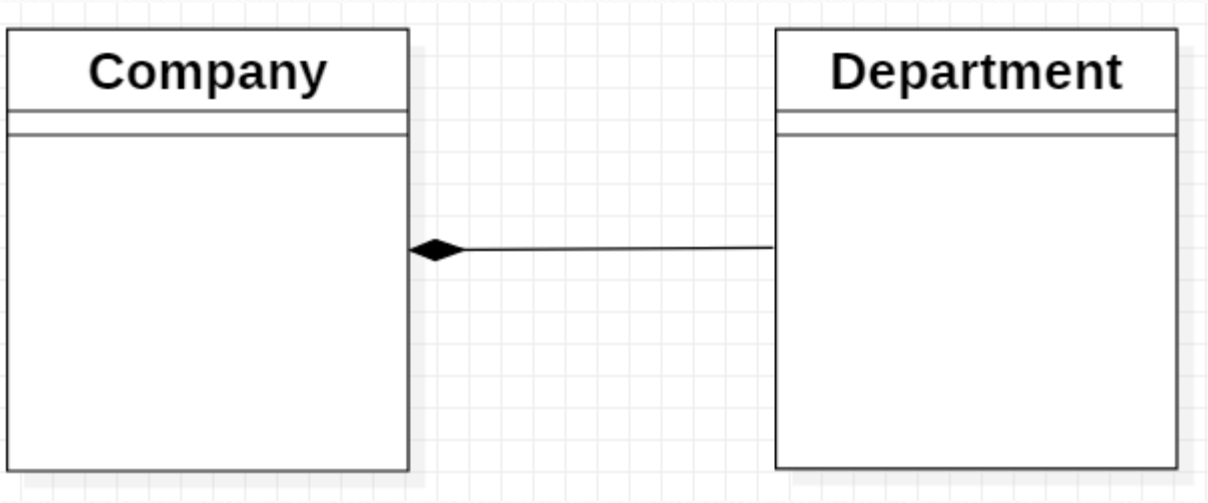
Aggregation

- A special type of association, which represents a "*Has-A*" relationship) .
  - E.g., College has Professors.
- Unidirectional association, i.e., one-way relationship.
- They may have different life cycles.



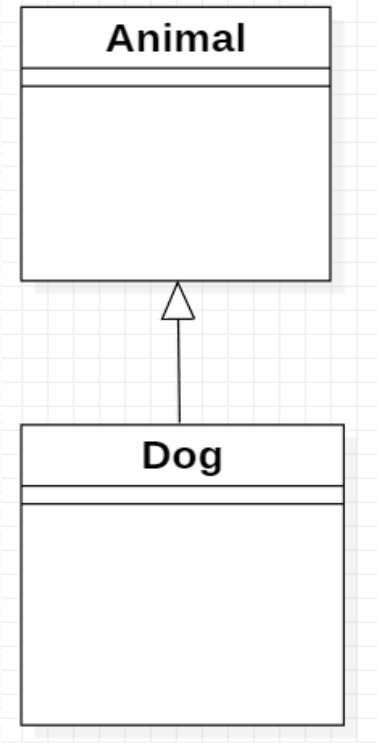
Composition

- Another type of the association
- Restricted form of aggregation, in which two objects are highly dependent on each other.
- Represents a "Part-of" relationship.
  - Life cycle of the ***part*** is dependent on the ***whole's*** *life cycle*



Inheritance

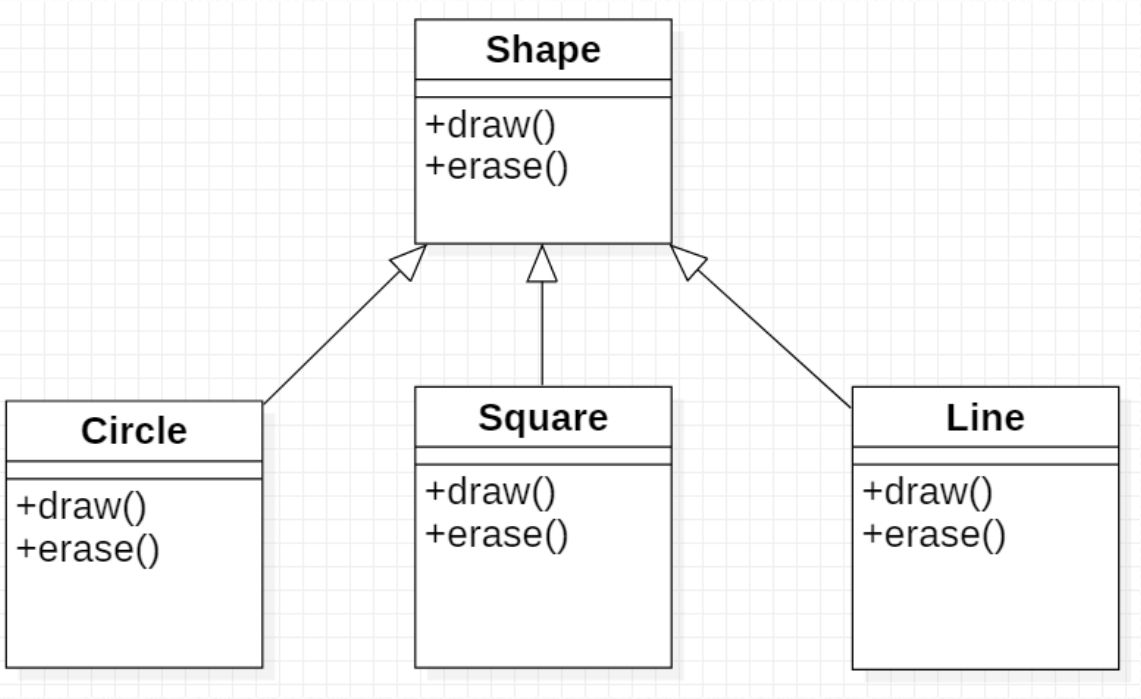
- Often referred as a “is-a” relationship.
  - E.g., a dog is an animal.
  - Animal is the superclass (base class, parent class).
  - Dog is the subclass (derived class, child class).



Inheritance relationship in a UML diagram

Inheritance

- Base class has more than one derived classes.
- When adding more classes, no need to touch code in other classes.

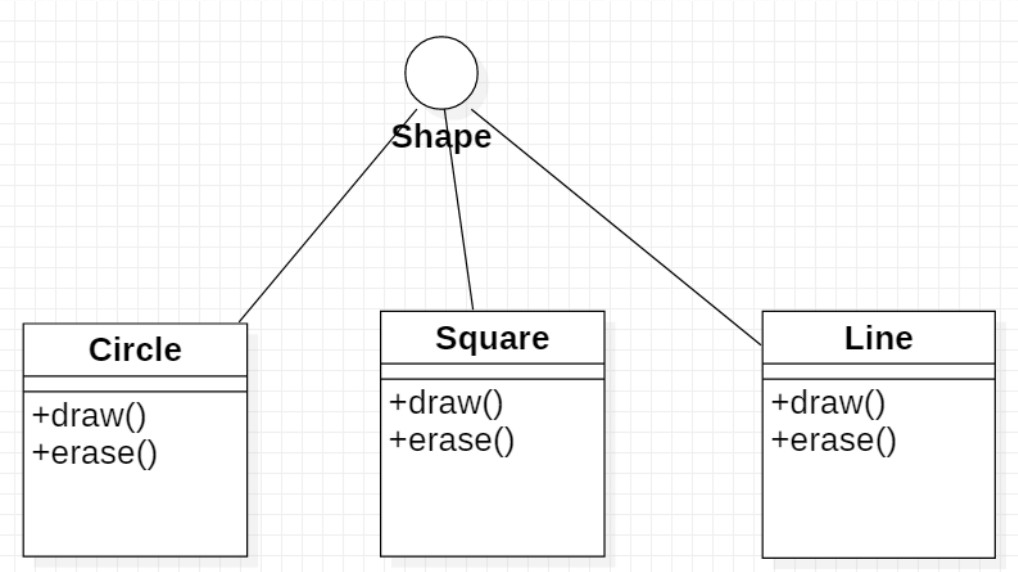


Polymorphism relationship in a UML diagram

Realization / Implementation

For interface

- In Java, an interface is not a class but a set of requirements for classes that want to conform to the interface:

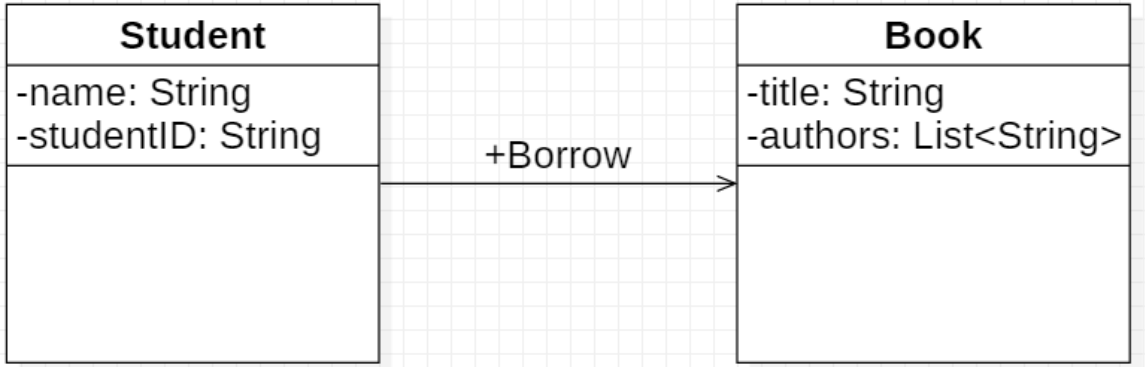


Implementation relationship in a UML diagram

Object-oriented Design

- Step 1: Given a problem, considering which class / object will exist in the problem domain.
- Step 2: Considering for each class / object, what fields and methods it should have.
- Step 3: Considering the relationships between different classes / objects.

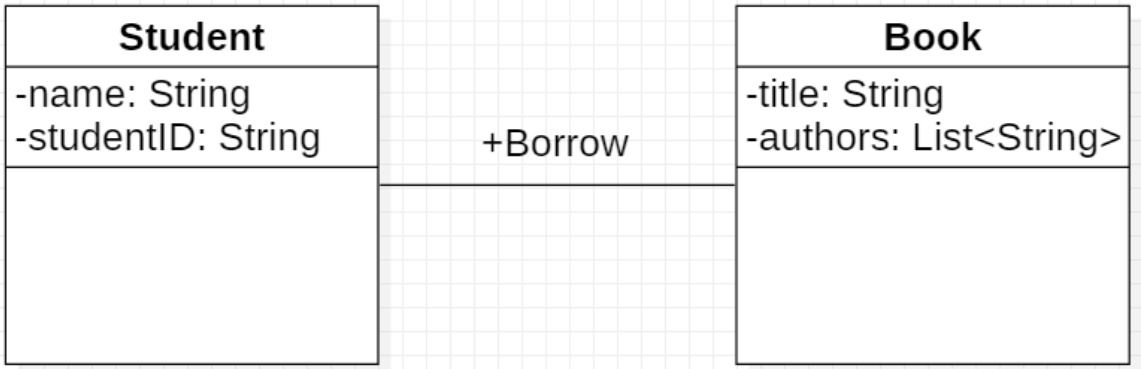
OOP: Class Diagram → Java Code




```java
/** */
public class Student {
        /** */
        public String name;

        /** */
        public String studentID;

        public Book myBook;

}
```

```java
/** */
public class Book {
        /** */
        public String title;

        /** */
        public LIST String authors;
}
```

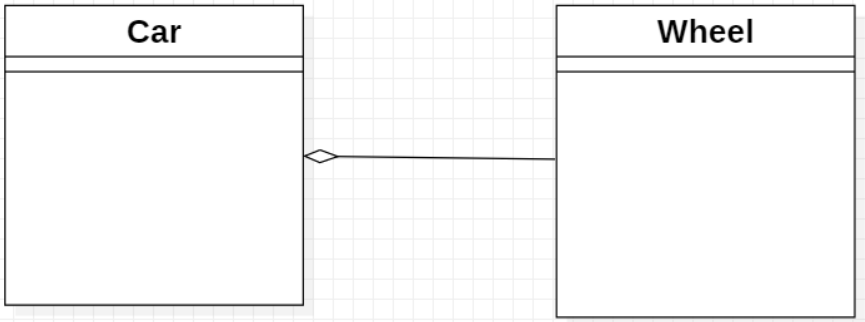
OOP: Class Diagram → Java Code




```
/** */
public class Student {
        /** */
        public String name;

        /** */
        public String studentID;

        public Book myBook;

}
```

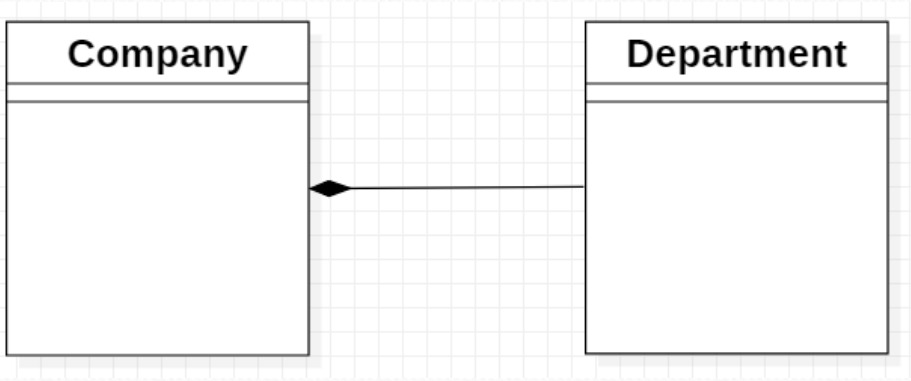
```
/** */
public class Book {
        /** */
        public String title;

        /** */
        public LIST String authors;

        public Student theStudent;
}
```

OOP: Class Diagram → Java Code



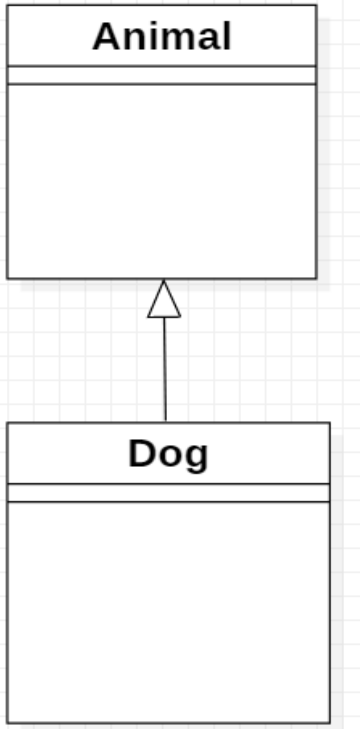

```
/** */
public class Wheel {
}

/** */
public class Car {

        public Wheel wheel;    //Car is aware of wheel
        Public Car(Wheel wheel) {  //Car needs wheel to exist
                this.wheel = wheel;

            }
    }
}
```

OOP: Class Diagram → Java Code




```
public class Department{
        ...
}

public class Company{
        public Department departments;

        public Company(){
        // must create departments before creating company

                departments = new Department();
        }
}
```

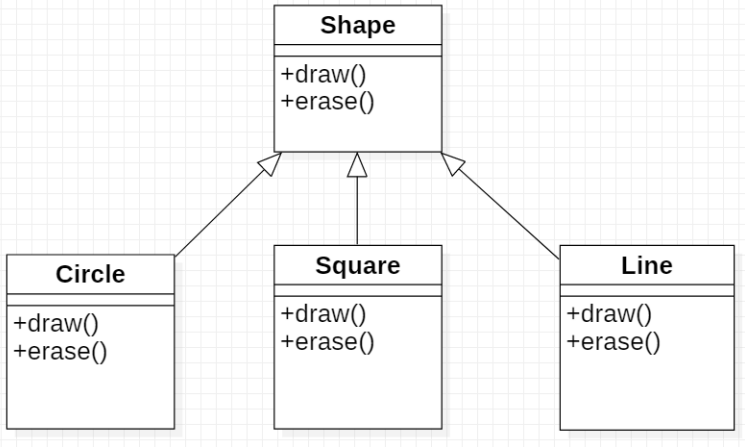
OOP: Class Diagram → Java Code




```java
/** */
public class Animal {
}
```

```java
/** */
public class Dog extends Animal {
}
```

OOP: Class Diagram → Java Code



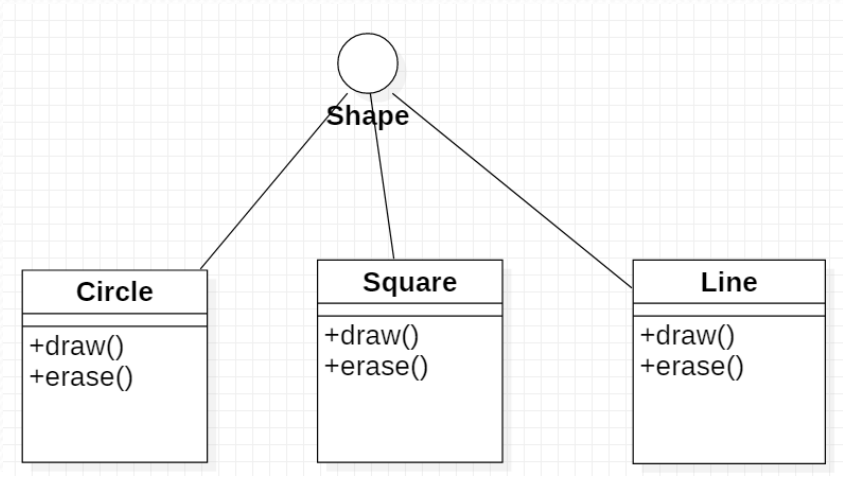

```
/** */
public class Shape {
        /** */
        public void draw() {

        }

        /** */
        public void erase() {

        }
}
```

```
public class Square extends Shape {
        /** */
        public void draw() {

        }

        /** */
        public void erase() {

        }
}
```

```
/** */
public class Line extends Shape {
        /** */
        public void draw() {

        }

        /** */
        public void erase() {

        }
}
```

```
/** */
public class Circle extends Shape {
        /** */
        public void draw() {

        }

        /** */
        public void erase() {

        }
}
```

OOP: Class Diagram → Java Code




```
/** */
public interface Shape {
        /** */
        public void draw();

        /** */
        public void erase();
}
```

```
public class Square implements Shape {
        /** */
        public void draw(){ ...}

        /** */
        public void erase(){ ...}
}
```

```
/** */
public class Line implements Shape {
        /** */
        public void draw { ...}

        /** */
        public void erase() { ...}
}
```

```
/** */
public class Circle implements Shape {
        /** */
        public void draw(){ ...}

        /** */
        public void erase() { ...}
}
```

Summary

- UML
- Class Diagram
- Class Relationship
- Class Diagram → Java Code