Object-Oriented Programming

Object-Oriented Development

Computer Science and Technology
United International College

Outline

- **Object-Oriented Development**
- **Objects vs. Classes**
- **UML**
- **Class Diagram**
- **Class Relationship**
- **Class Diagram → Java code**

Object-Oriented Development

- A popular software development method.
- Develop reusable systems.
- The concept started in 1968.
- Based on the use of single objects.

Classes

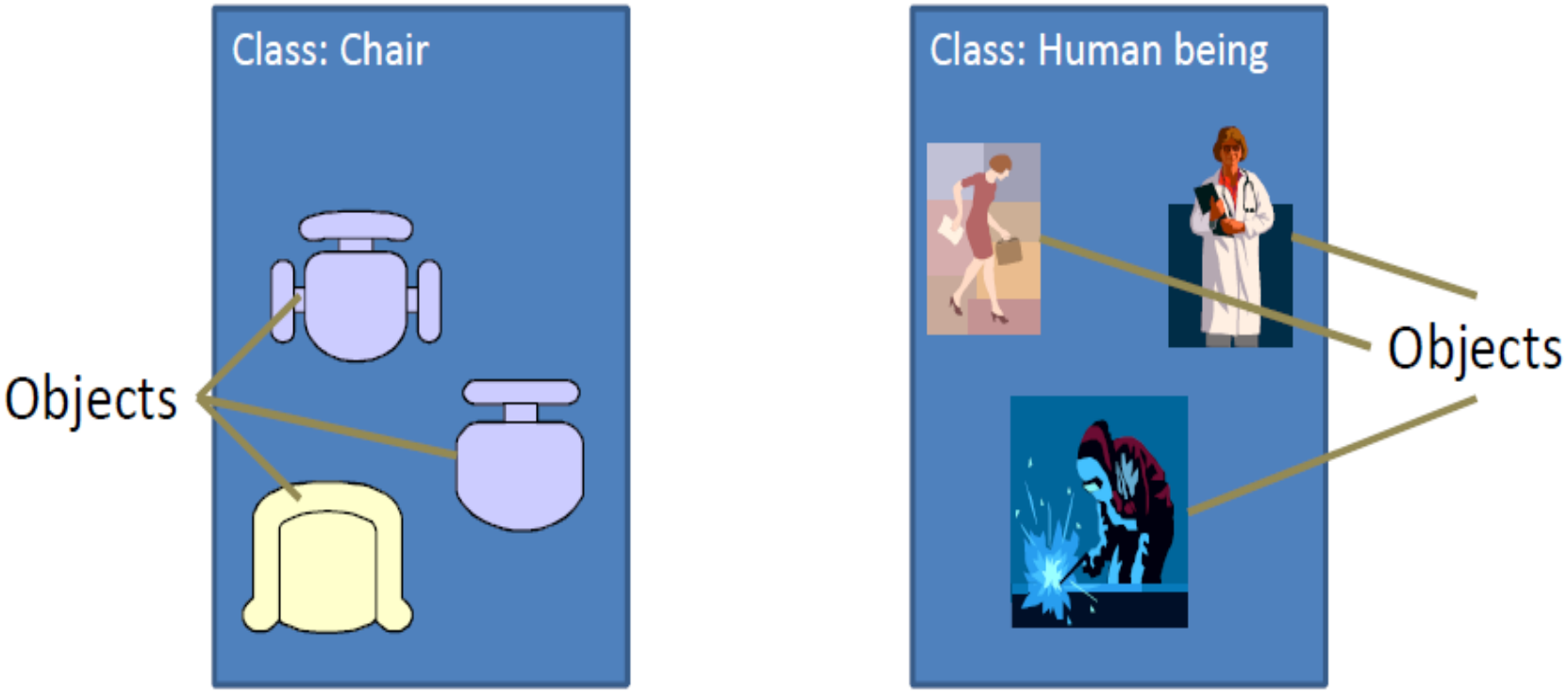
- A set of entities with similar attributes.

- A generalized description that describes a collection of similar objects.

- Example: Chair, Human, Students, Teachers, Books, etc.

Objects and Classes

- An object is an instance（实例） of a class.



Objects

- Object:
  - An entity;
  - Physical: a chair, a desk, a person;
  - Logical: a list, a stack, a rectangle.

- Objects have **state** and **behavior**.

  Example: Dog;
  - **State**: Color, Name, Breed ;
    - State is stored in **instance variables;**
  - **Behaviors**: Fetch stick, Drink water, Wag tail, Bark;
    - Behaviors are accomplished by **methods**.

Object-Oriented

- OO in one sentence: keep it **DRY**, keep it Shy and Tell the other guy.
  - DRY: Do not Repeat Yourself.
  - Shy: Should not reveal the information about itself unless really necessary.
  - Tell the other guy: Send a message rather than doing a function call.

- By Andy Hunt and Dave Thomas.

Benefits of OOP Approach

- **Modularity**:
  - The source code for an object can be written and maintained independently of the source code for other objects.
- **Information-hiding**:
  - By interacting only with an object's methods, the details of its internal implementation are hidden from the outside world.
- **Code re-use**:
  - If an object already exists you can use that object in your program. This allows specialists to implement / test / debug complex, task-specific objects, which you can then trust to run in your own code.

From Java Tutorial : https://docs.oracle.com/javase/tutorial/java/concepts/object.html

Object-Oriented Software Development Process

- OO Analysis: Requirement specification.
- OO Design: Architectural design.
- Object Design: Detailed design.
- **Object-Oriented Programming: Implementation.**

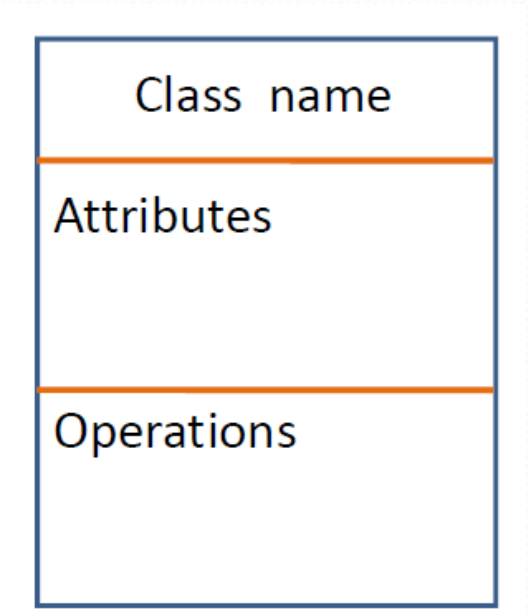
Our focus this semester!

Unified Modeling Language

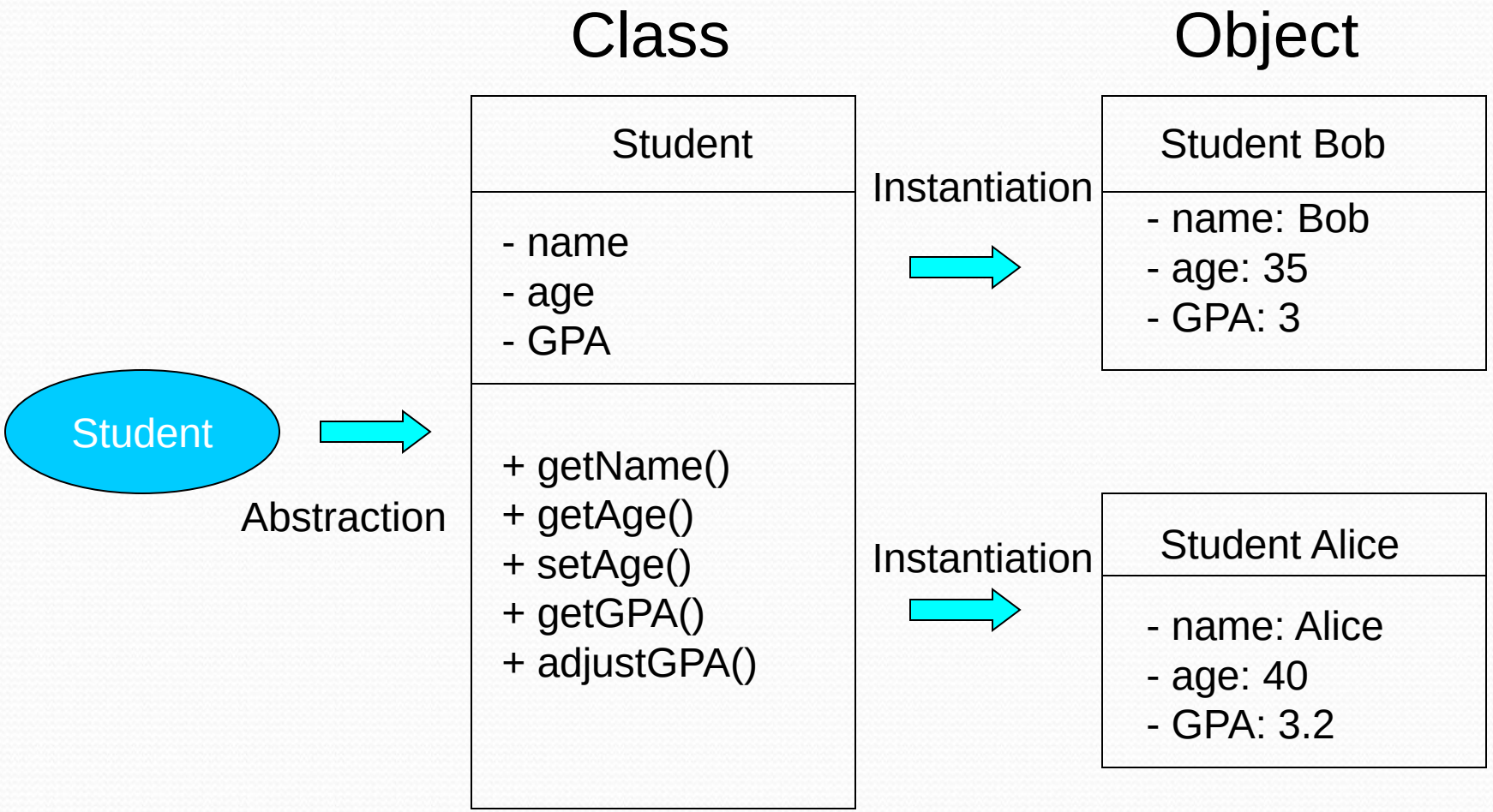
- The most popular diagrammatic notation used for Object-Oriented Development.
- Support from OOA (Object-Oriented Analysis) to OOP (Object-Oriented Programming).
- Consists of:
  - **Class diagrams;**
  - Sequence diagrams;
  - Use case diagrams;
  - Activity diagrams;
  - …

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

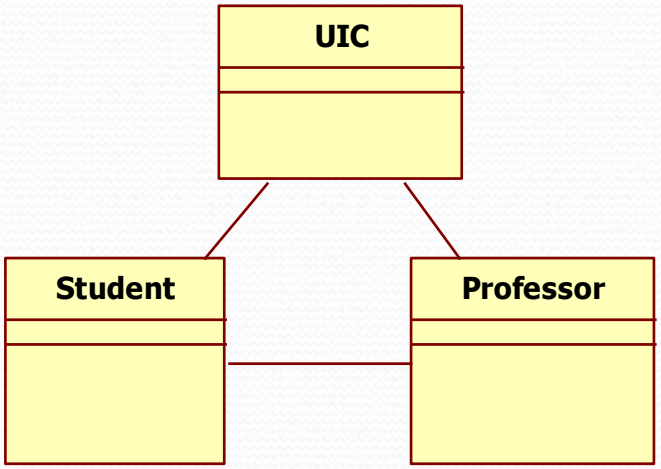

Class
Object
Student
Abstraction
Student
- name
- age
- GPA
+ getName()
+ getAge()
+ setAge()
+ getGPA()
+ adjustGPA()
Instantiation
Student Bob
- name: Bob
- age: 35
- GPA: 3
Instantiation
Student Alice
- name: Alice
- age: 40
- GPA: 3.2

Relationships between classes

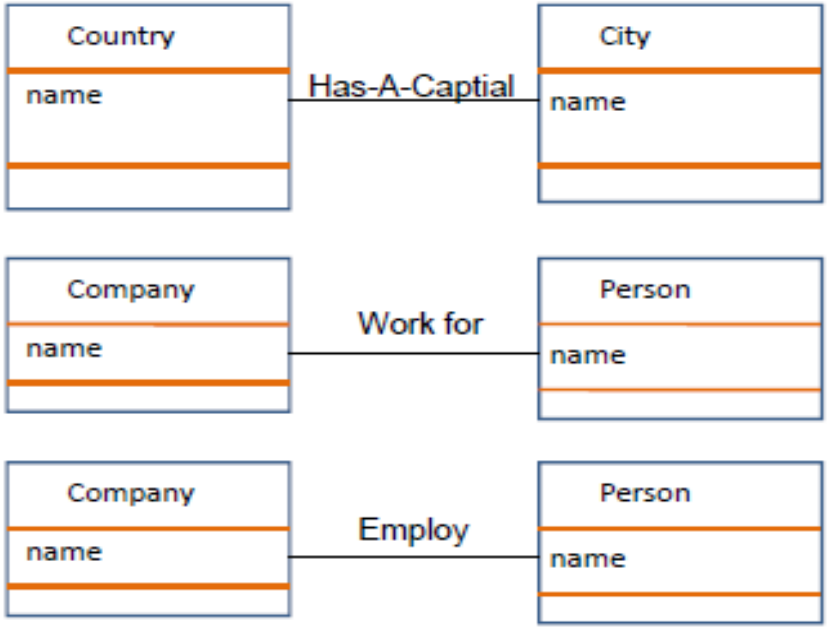
- Association (directional + Multiplicity).
- Aggregation.
- Composition.
- Inheritance.
- Polymorphism.

Association(关联)

- An association is a linkage between two classes.
- A class is aware of and holds a reference to another class.
- Often referred as a "has-a" relationship.
- Bidirectional or unidirectional.

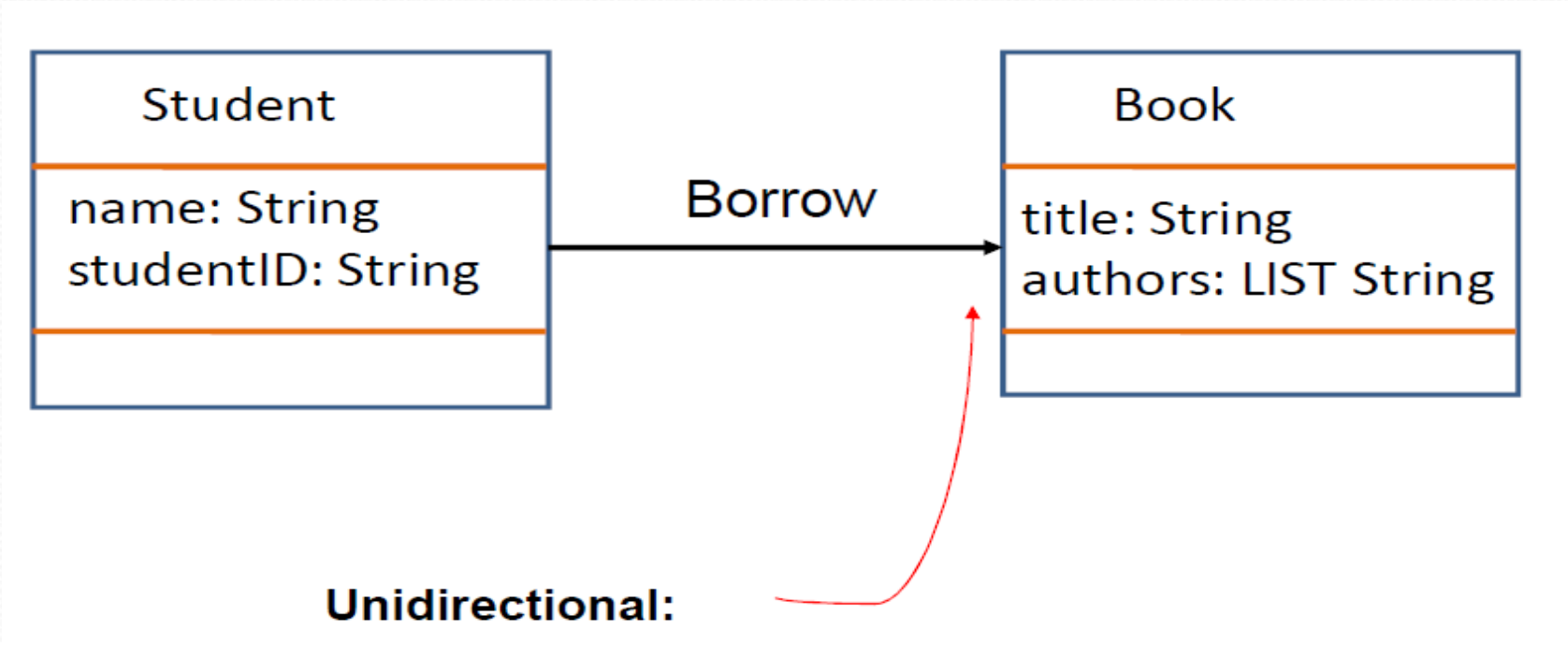


Association relationship in a UML diagram



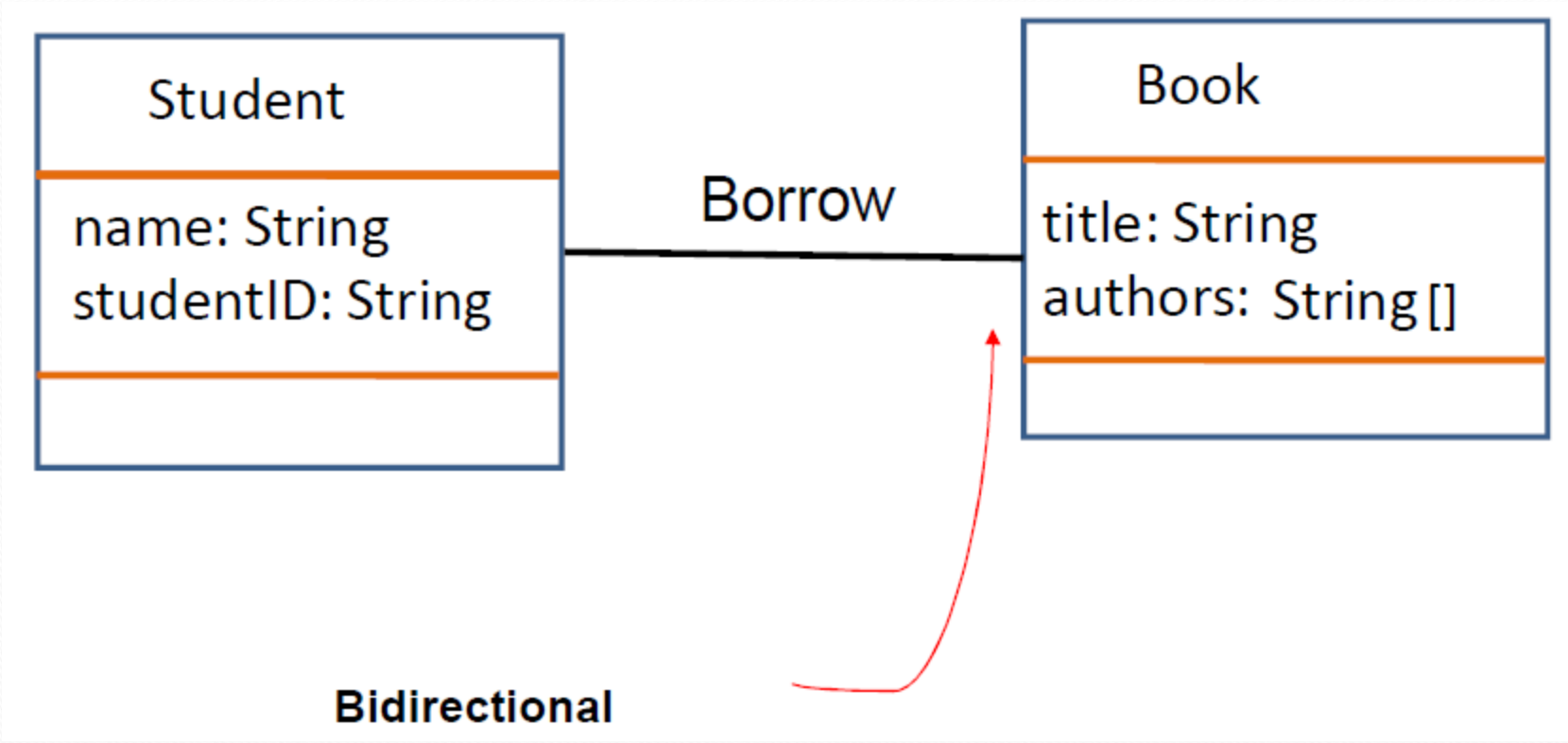
Unidirectional Association

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



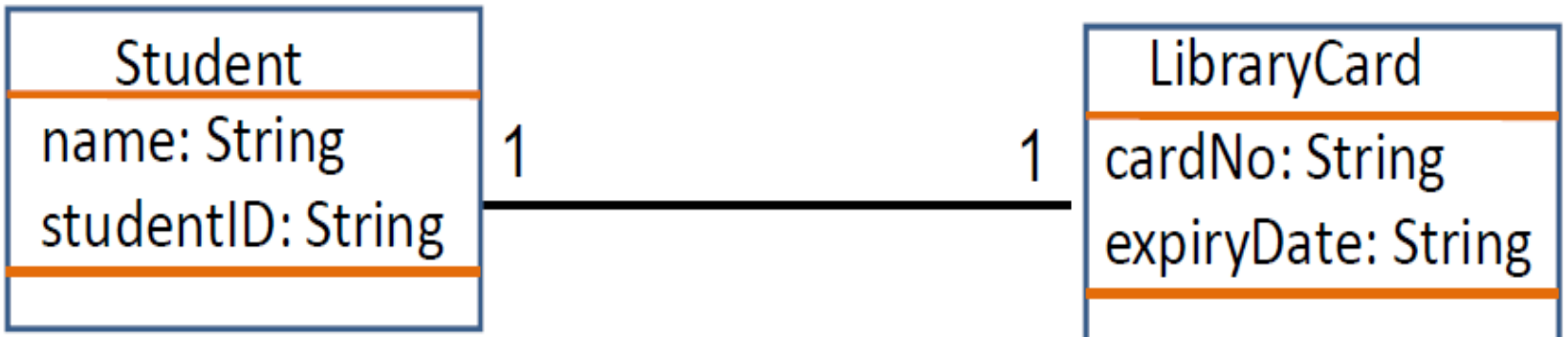
Bidirectional Association

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



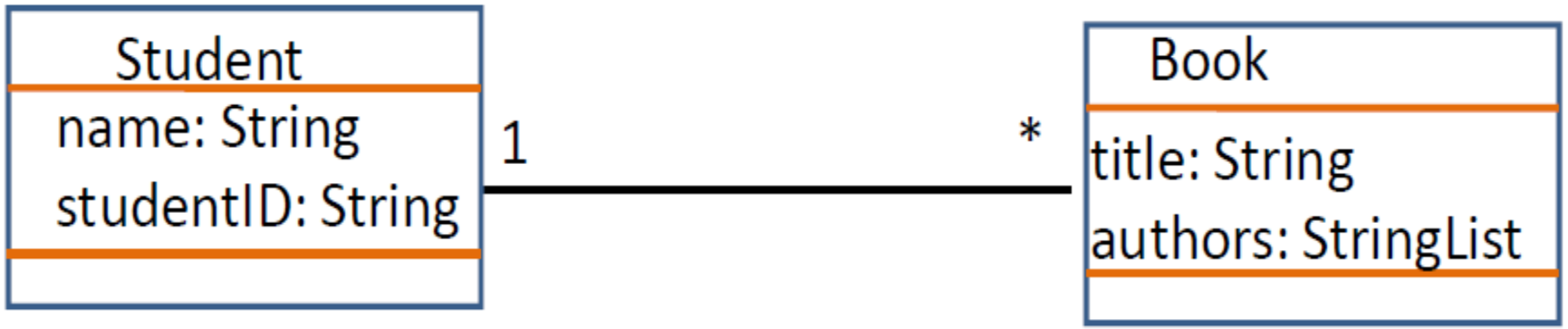
Multiplicity

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

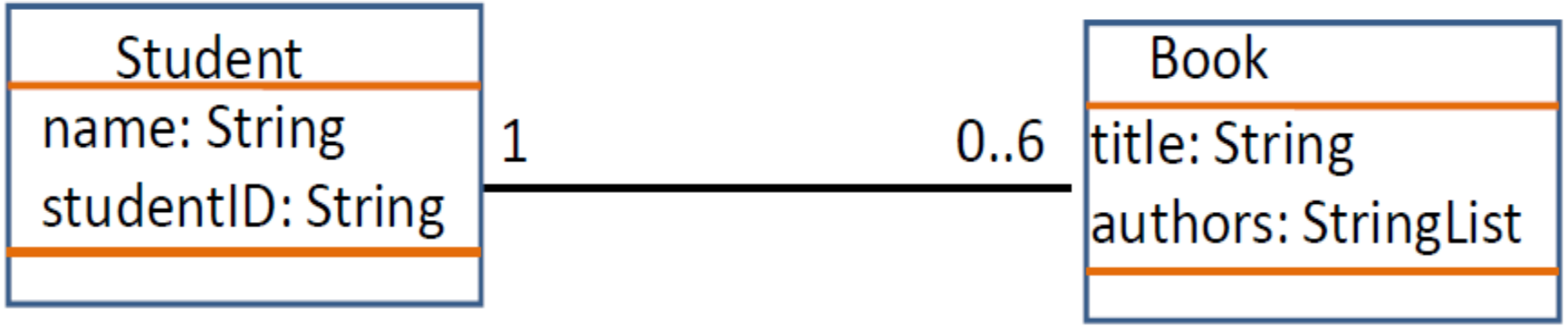


One to one relationship

Multiplicity



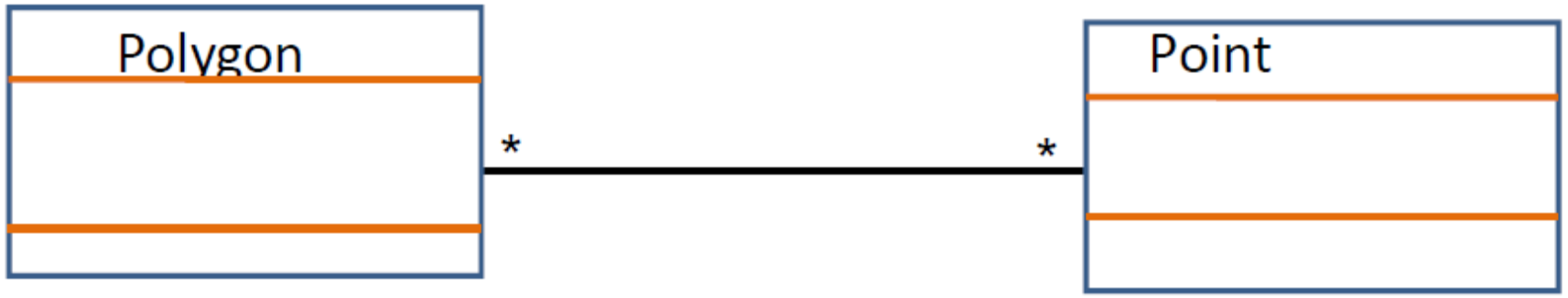
One student can borrow 0 or many books



One student can borrow at most 6 books

Multiplicity

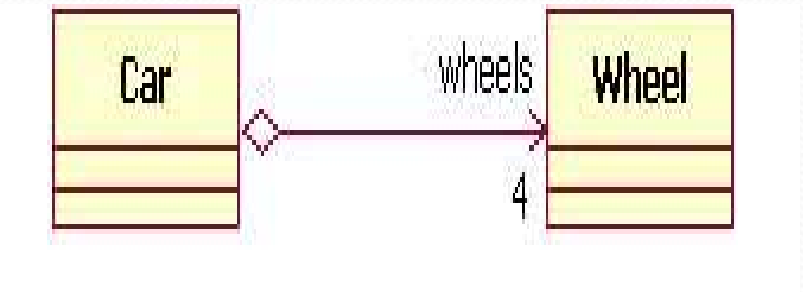
- One polygon has many points and one point can be in many polygons.

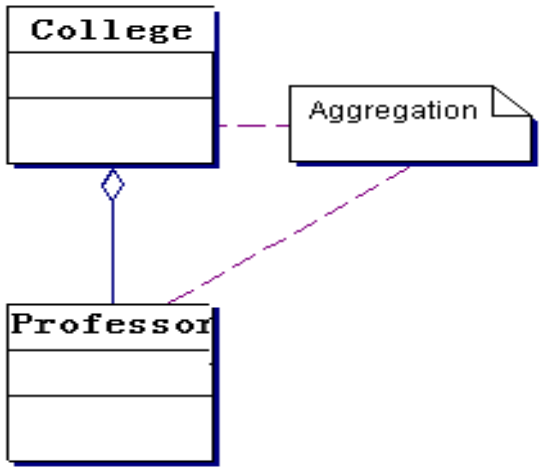


Many to many relationship

Aggregation

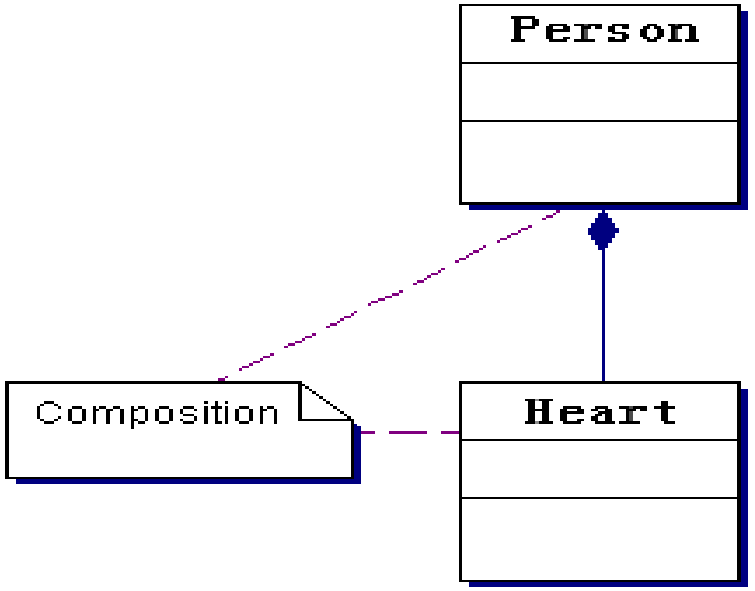
- A special type of association.
- Used to model a "whole to its parts" relationship.
- Also referred as a “has-a” relationship.
  - E.g.: College has Professors.
- They may have different life time.





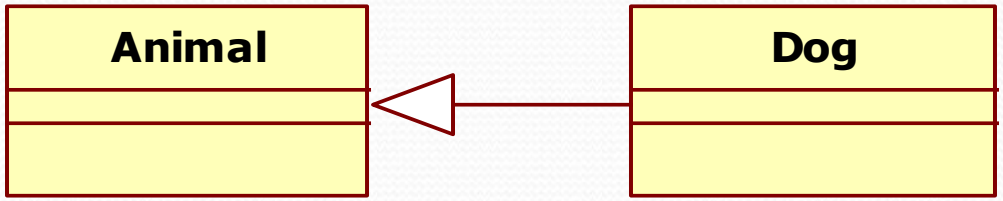
Composition

- Another form of the aggregation.
- Child class's instance lifecycle is dependent on the parent class's instance lifecycle.



Inheritance

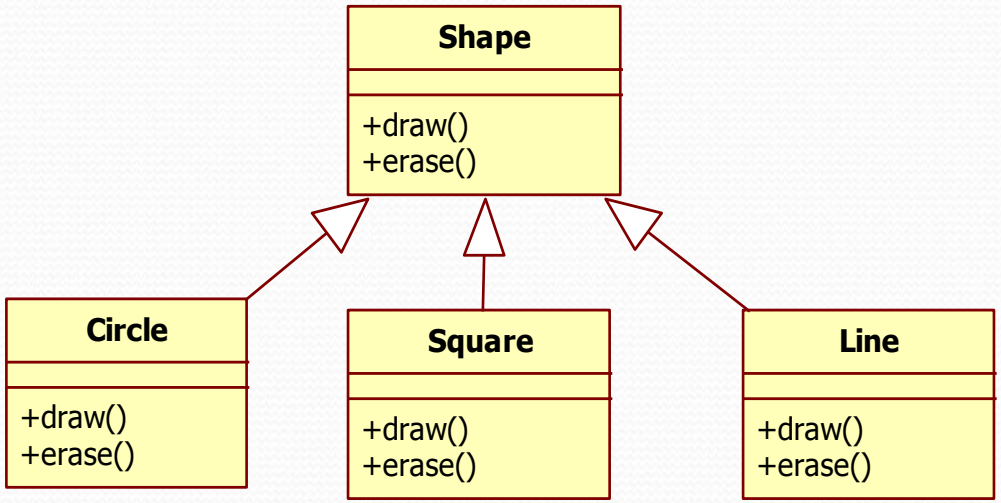
- Often referred as a “is-a” relationship.
  - E.g.: a dog is an animal.
  - Animal is the superclass (base class, parent class).
  - Dog is the subclass (derived class, child class).



Inheritance relationship in a UML diagram

Inheritance

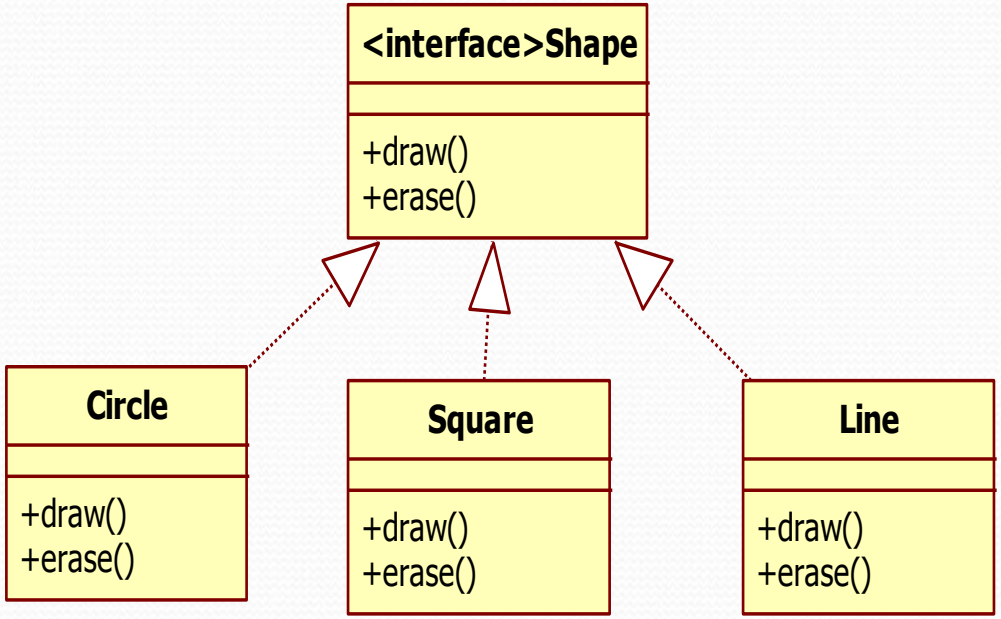
- Base class has more than one derived classes.
- Decide which draw( )/erase() to run at run time.
- When adding more classes, no need to touch code in other classes.



Polymorphism relationship in a UML diagram

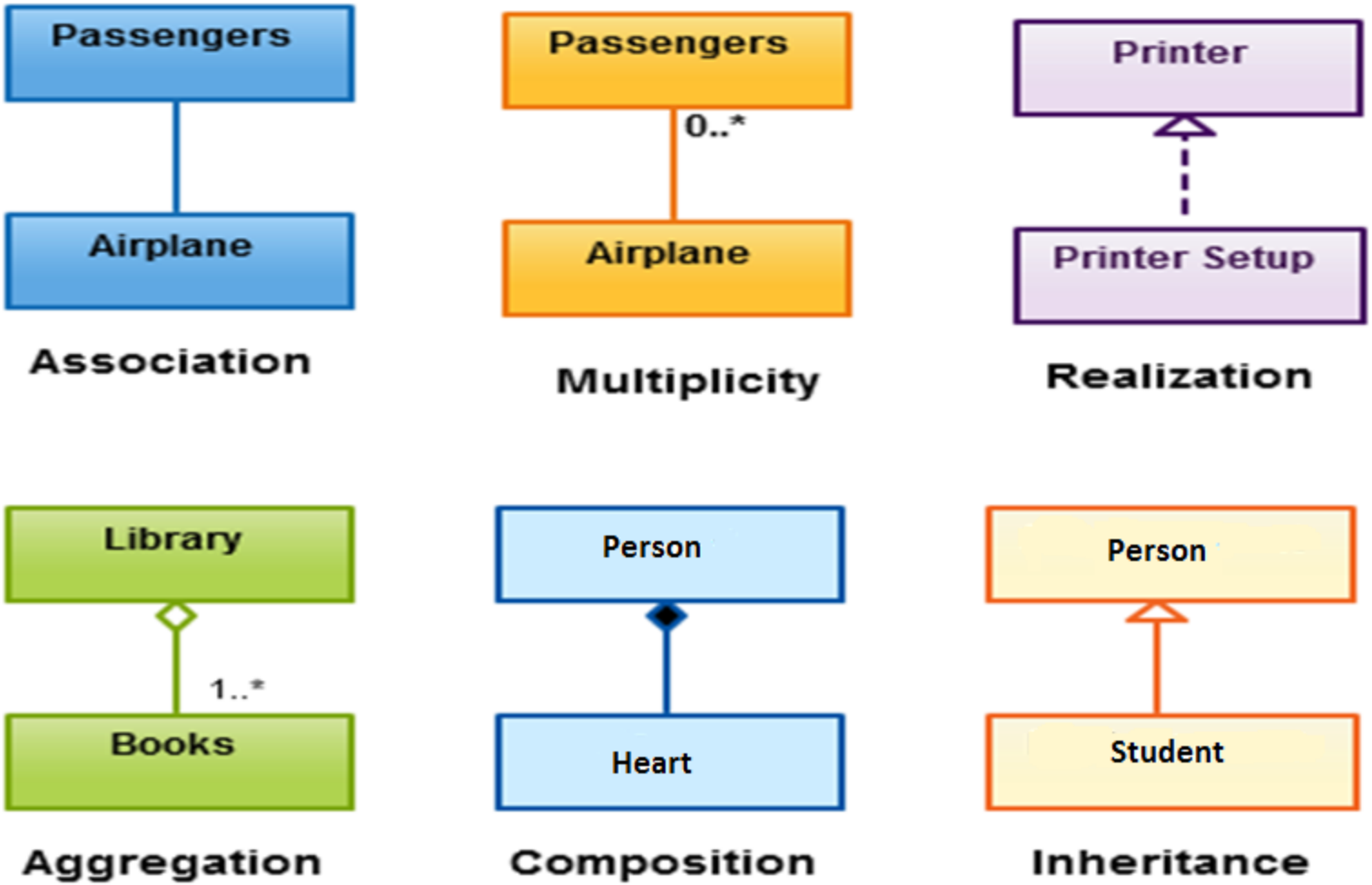
Realization / Implementation

- For interface:



Implementation relationship in a UML diagram

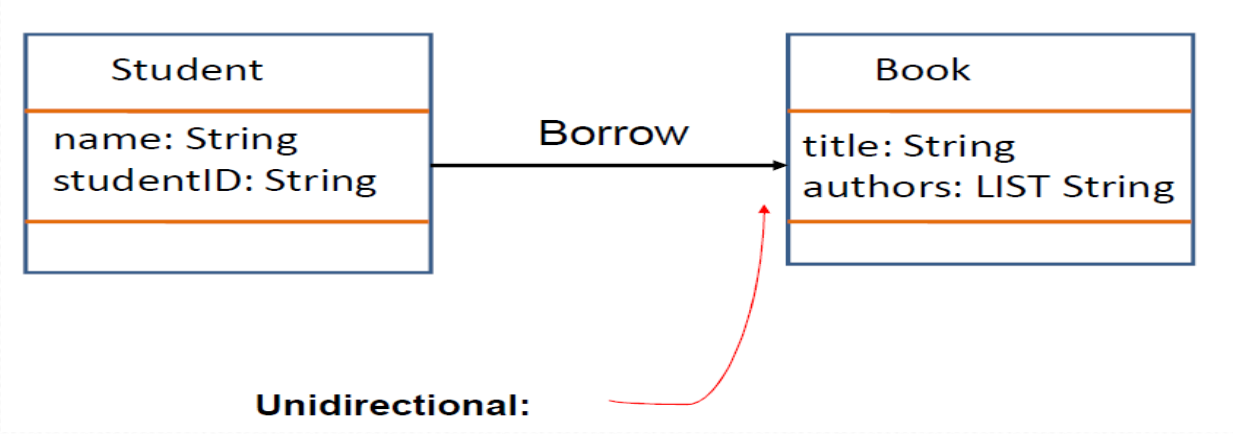
Relationship Examples



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.

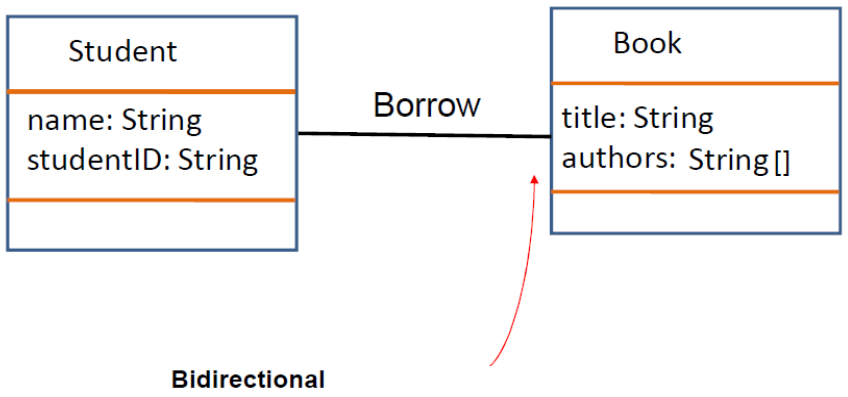
OOP: Class Diagram → Java Code




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

        /** */
        public String studentID;

        public Book myBook;

}
```

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

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

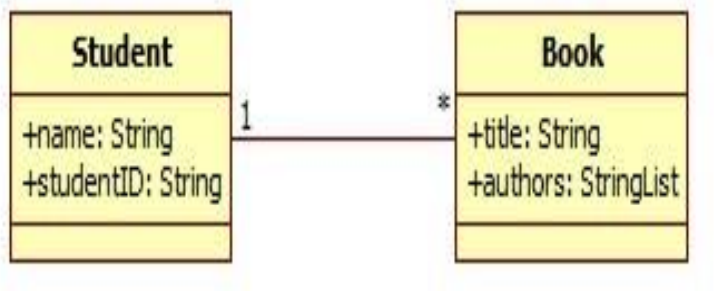
OOP: Class Diagram → Java Code




```
/** */
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;
}
```

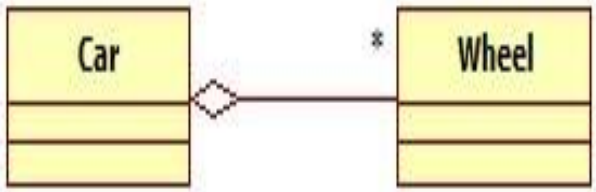
OOP: Class Diagram → Java Code




```
public class Student {
        public String name;
        public String studentID;

        public Book myBook;

        private numberOfBooks;
        public void maintainBooks(){
            ...
        }


}
```

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

        /** */
        public LIST String authors;

        public Student theStudent;
}
```

OOP: Class Diagram → Java Code

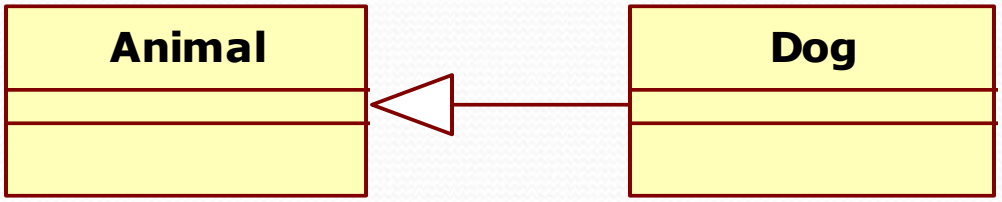



```
/** */
public class Wheel {
}

/** */
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 Departments();

     }

}
```

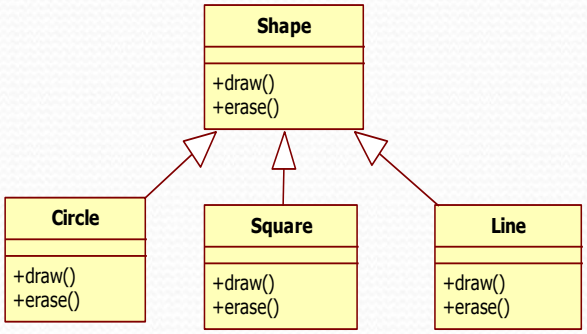
OOP: Class Diagram → Java Code




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

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

OOP: Class Diagram → Java Code



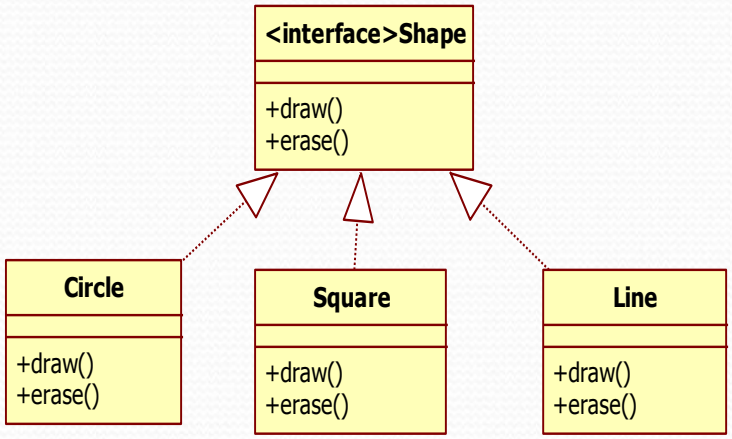

```
/** */
public class Shape {
        /** */
        public void draw() {

        }

        /** */
        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();
}
```

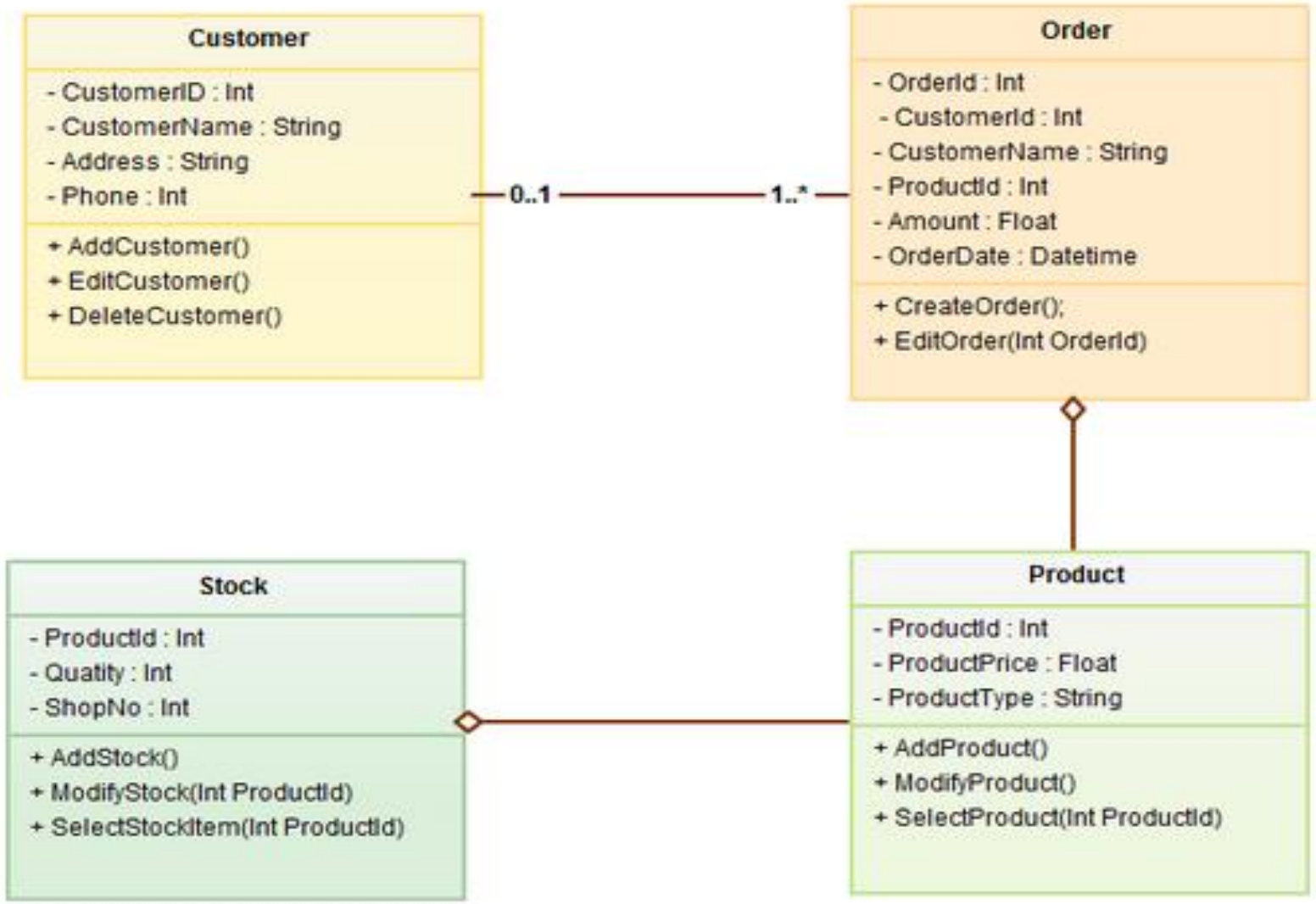
```
/** */
public class Line implements Shape {
        /** */
        public void draw();

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

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

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

What can be classes?

- Software Requirement Specification.
- Find all nouns in the SRS.
- Remove the following nouns:
  - Duplicates;
  - Unrelated;
  - Vague or general nouns;
  - Dependent nouns, which should be attributes;
  - Interface, which is about other system interacting with the system.

Class Diagram for Order Processing System



Summary

- Objects vs. Classes
- UML
- Class Diagram
- Class Diagram → Java Code
- Software development process