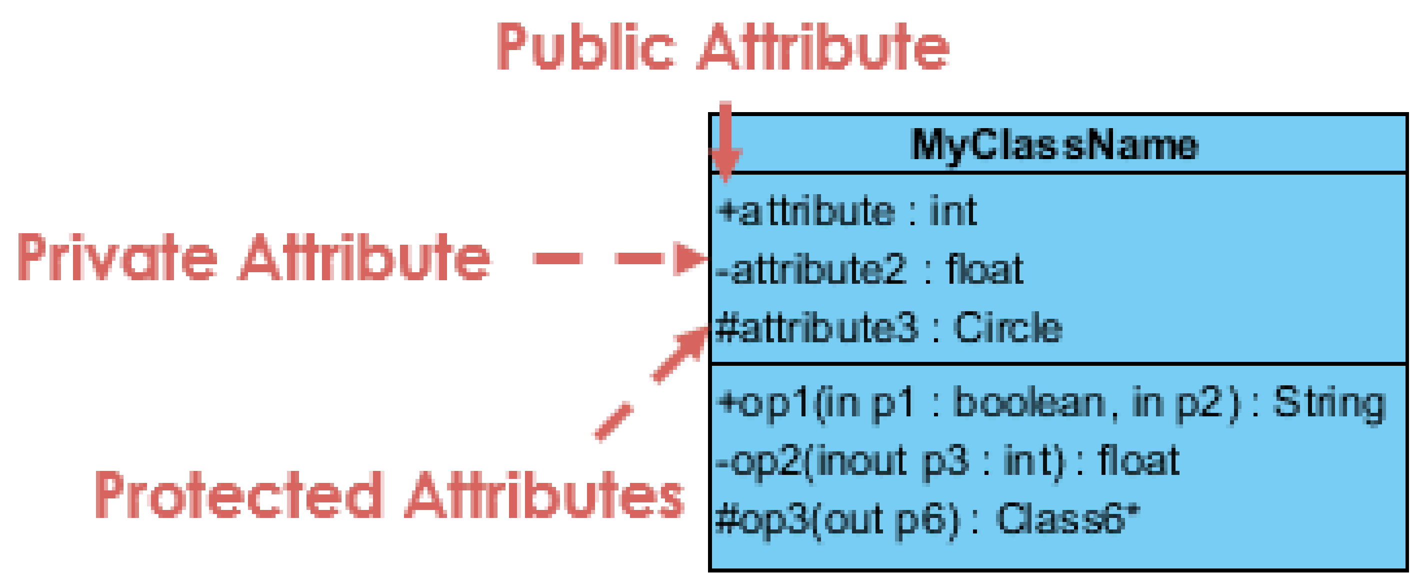
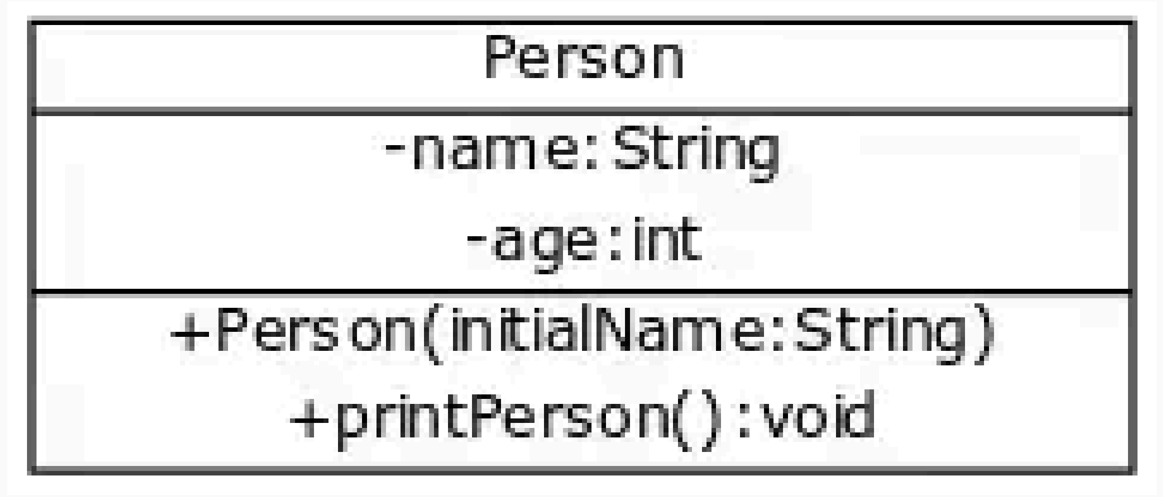
UML CLASS DIAGRAMS

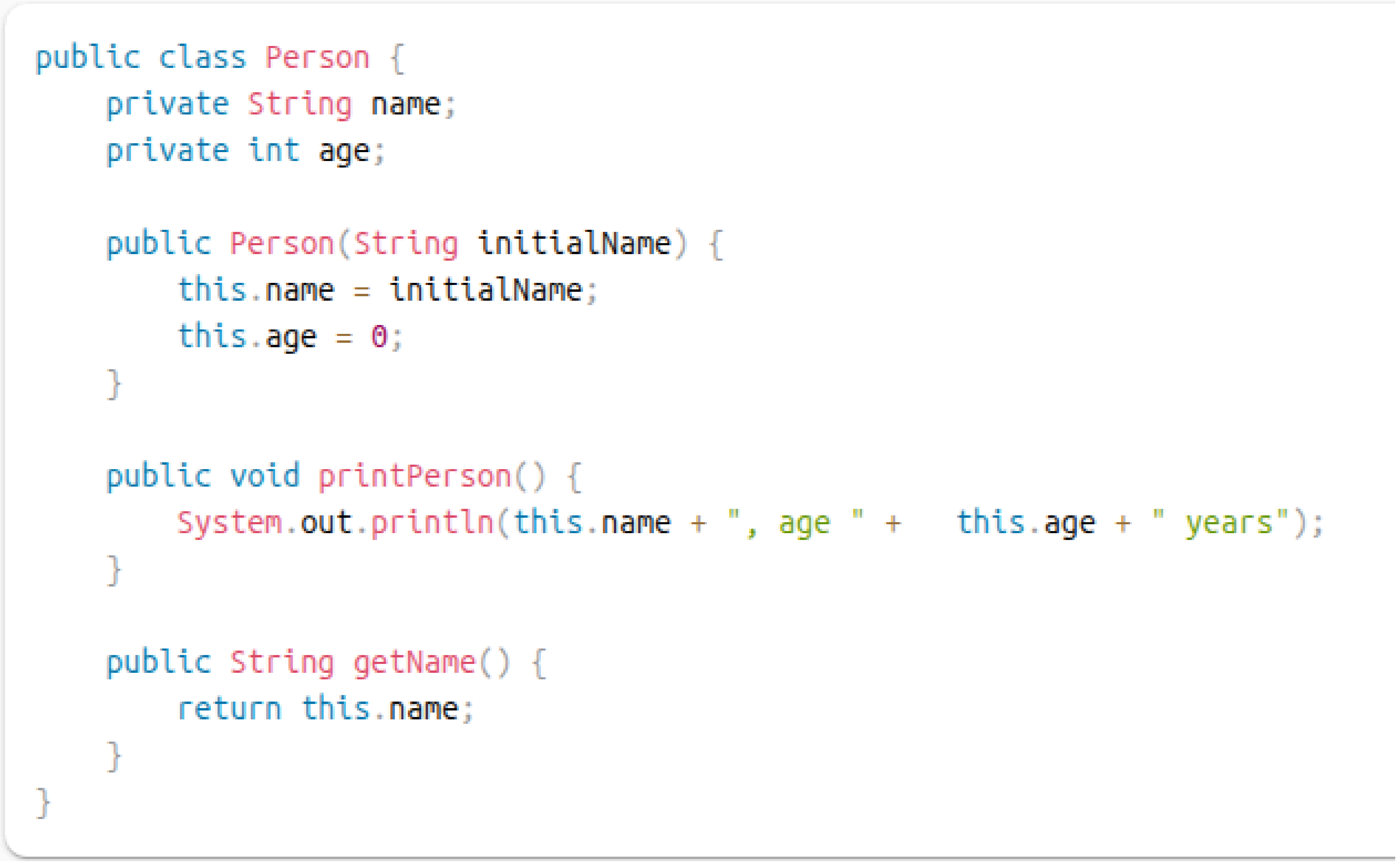
JAVA BOOTCAMP - 10th September, 2025



- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations

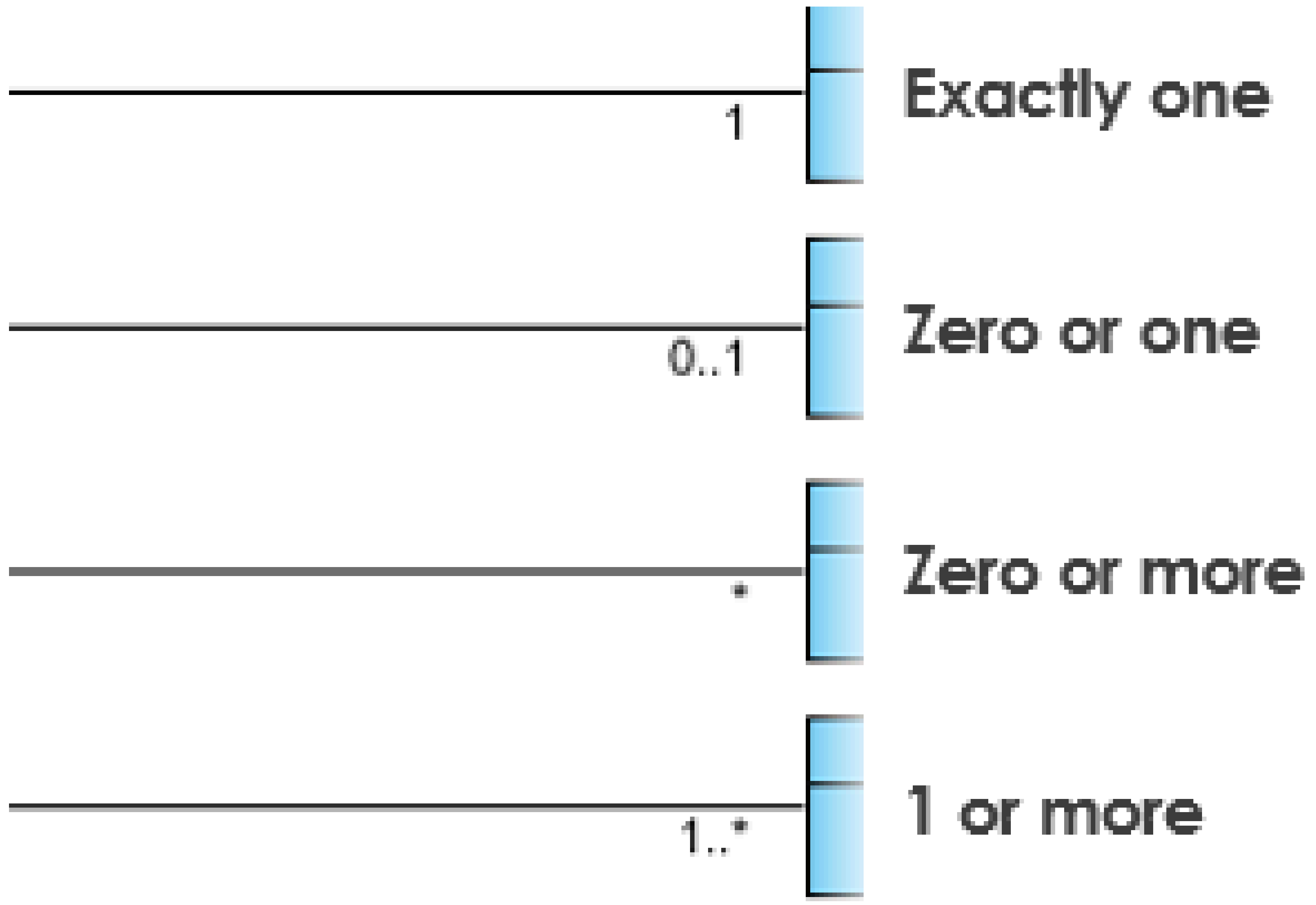
- Void methods have return type void
- Static methods/fields are <u>underlined</u>
- Constants are in ALL_CAPS



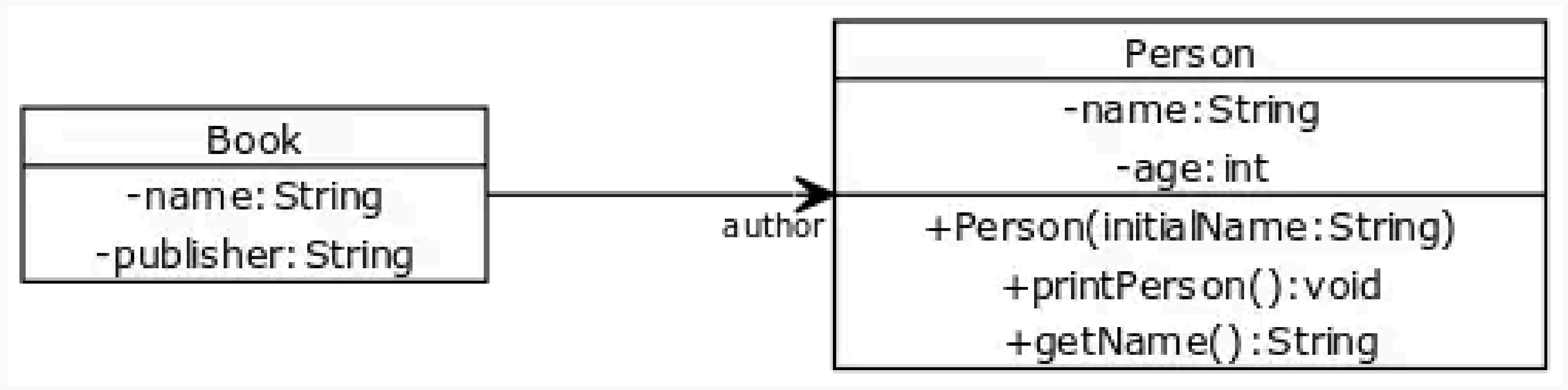


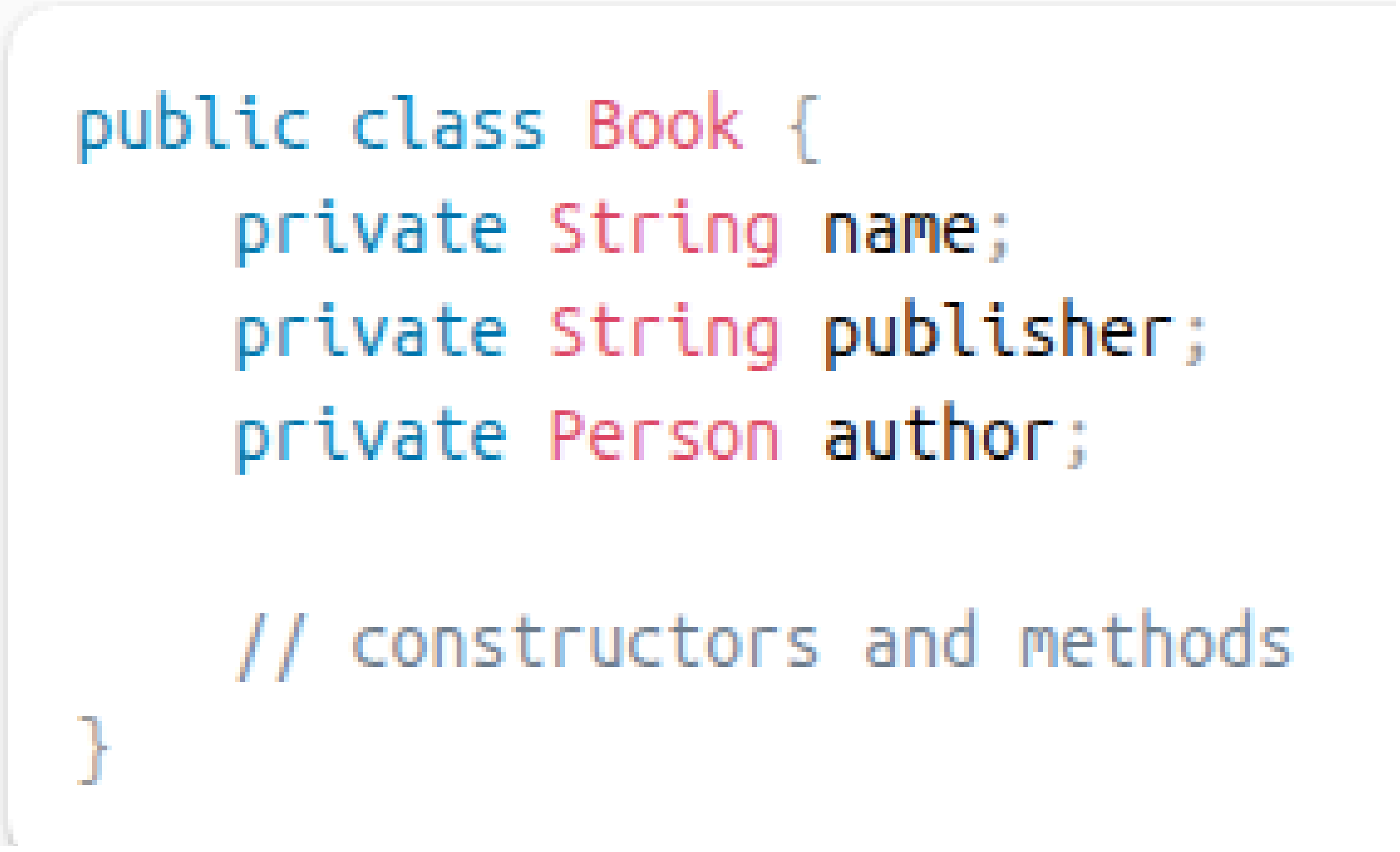
```java
public class Person {
    private String name;
    private int age;

    public Person(String initialName) {
        this.name = initialName;
        this.age = 0;
    }

    public void printPerson() {
        System.out.println(this.name + ", age " +   this.age + " years");
    }

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

Cardinality



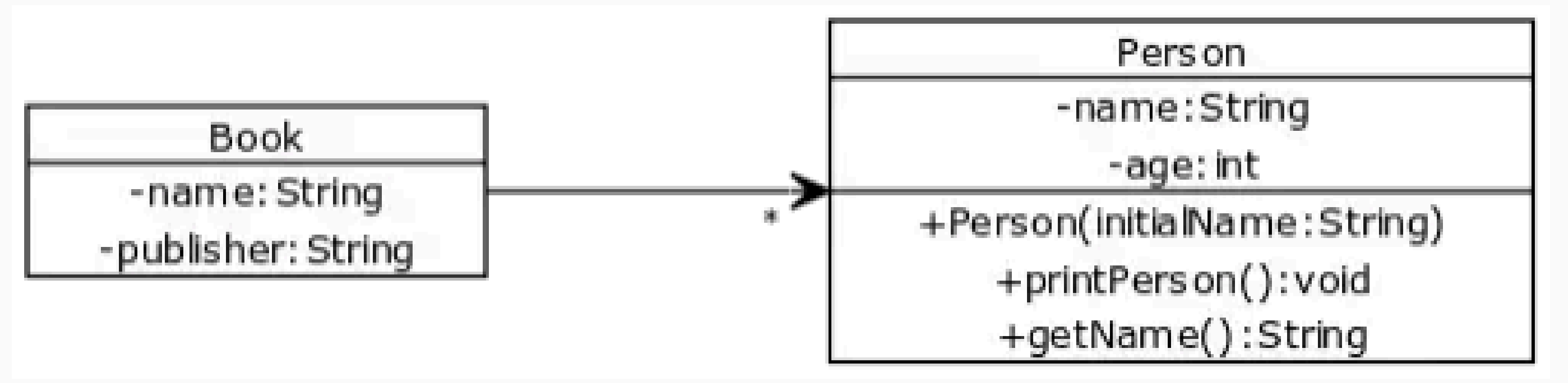
One to one

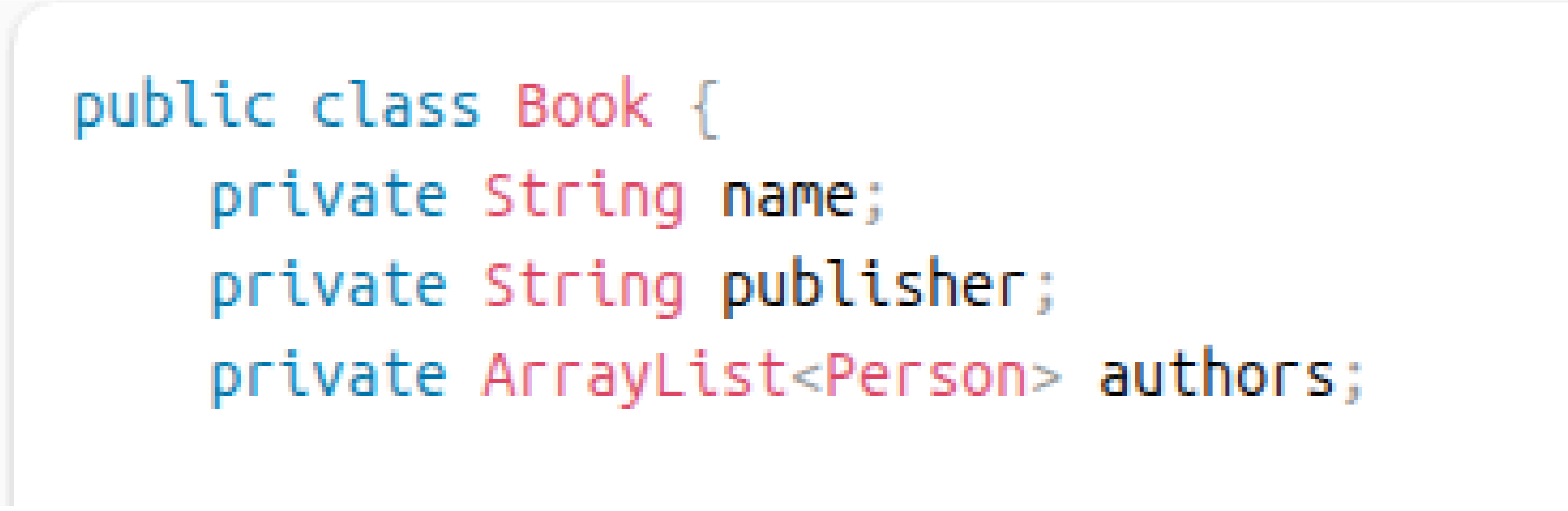




```
public class Book {
    private String name;
    private String publisher;
    private Person author;

    // constructors and methods
}
```

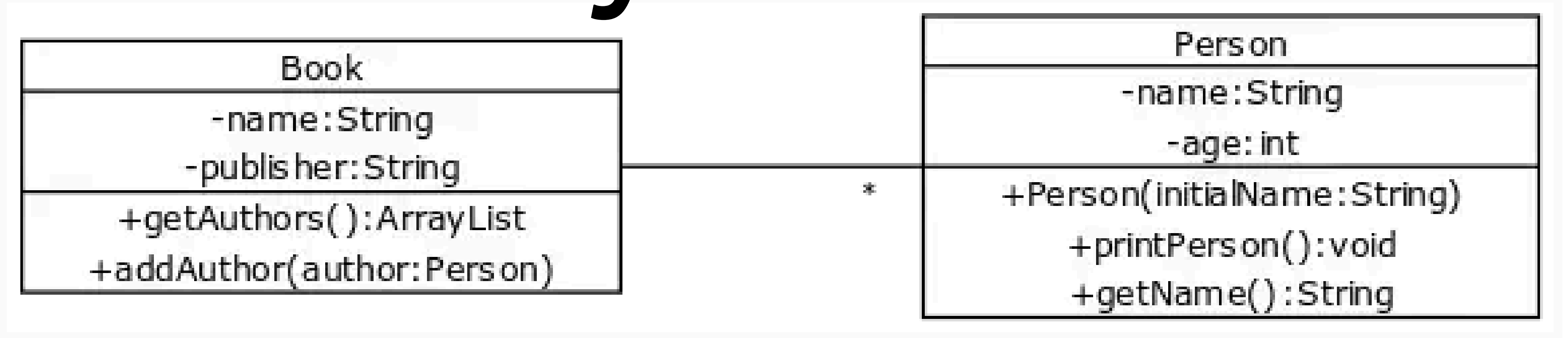
One to many





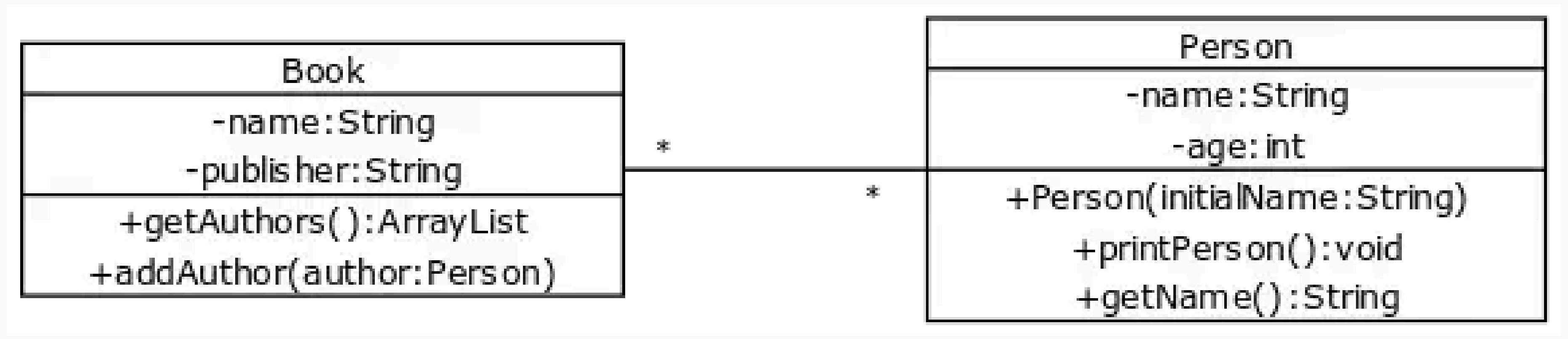
```
public class Book {
    private String name;
    private String publisher;
    private ArrayList<Person> authors;
```

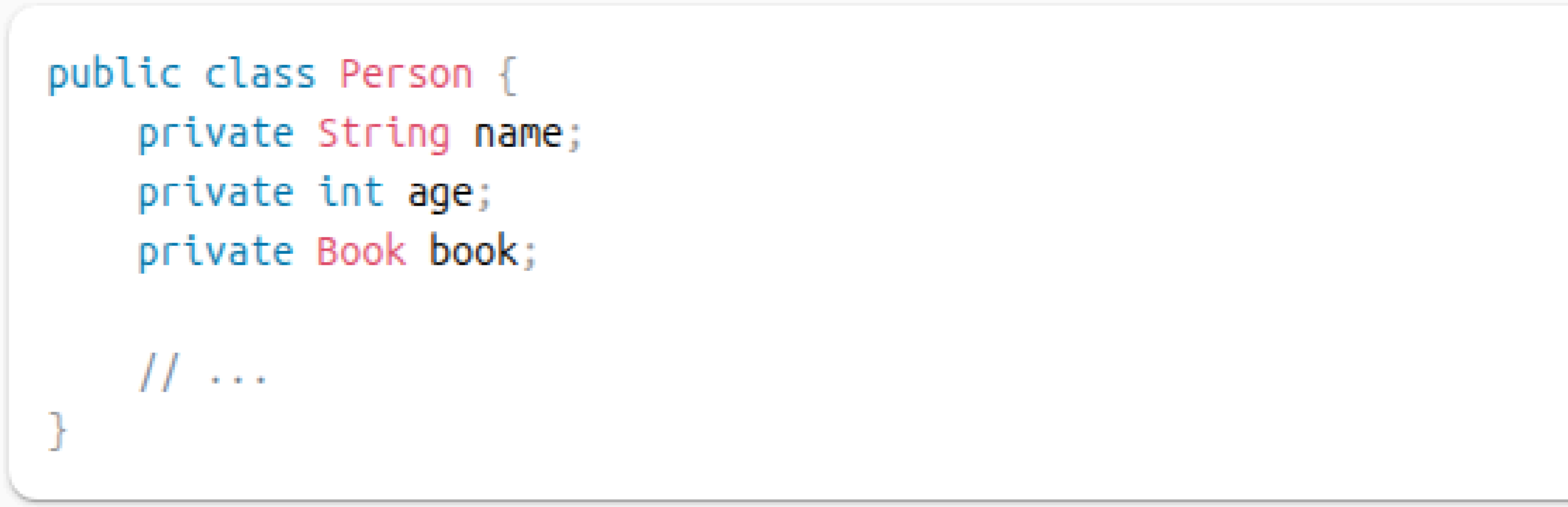
One to many



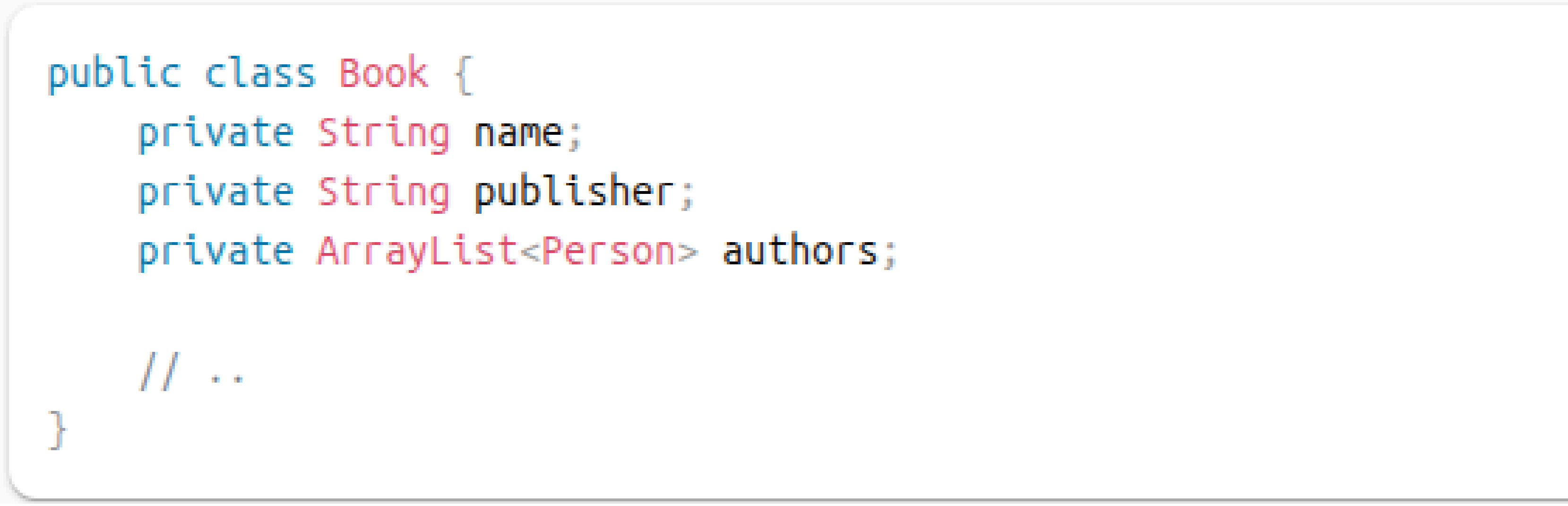
No arrow → both classes know about each other

Many to many

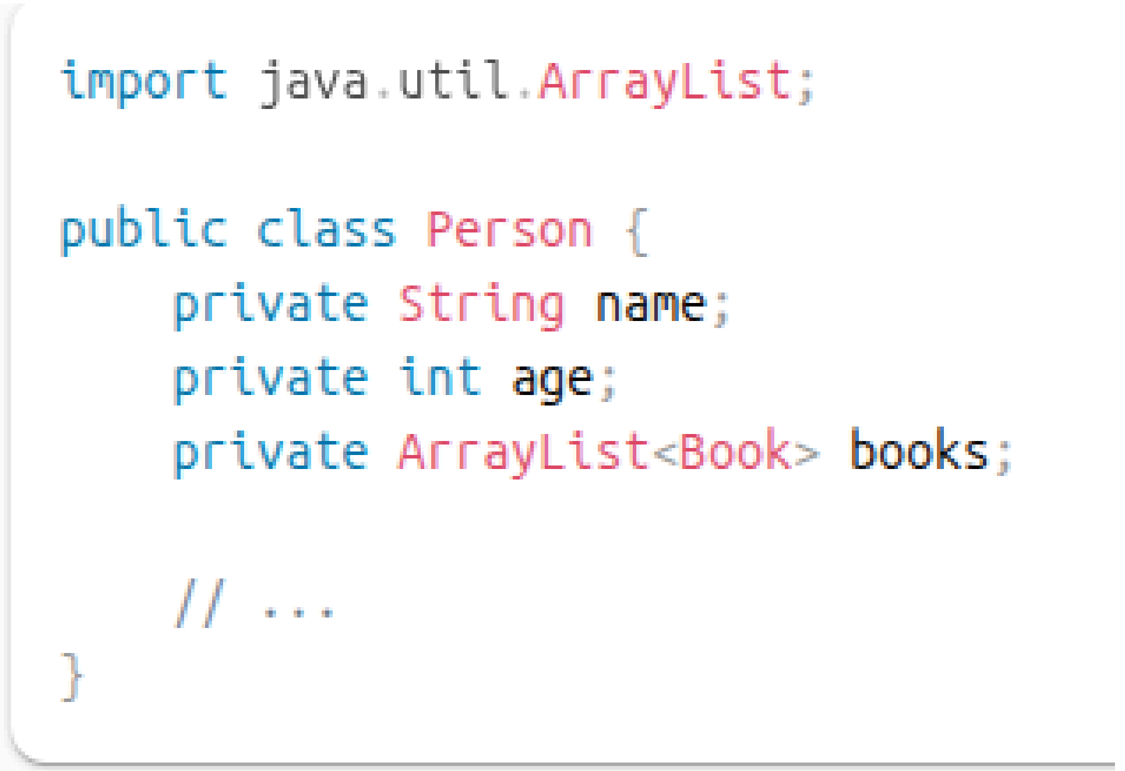




```
public class Person {
    private String name;
    private int age;
    private Book book;

    // ...
}
```

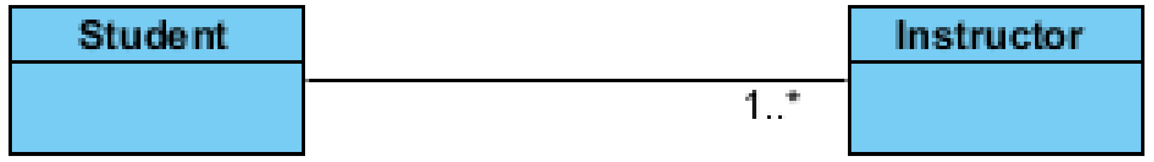


```
public class Book {
    private String name;
    private String publisher;
    private ArrayList<Person> authors;

    // ..
}
```

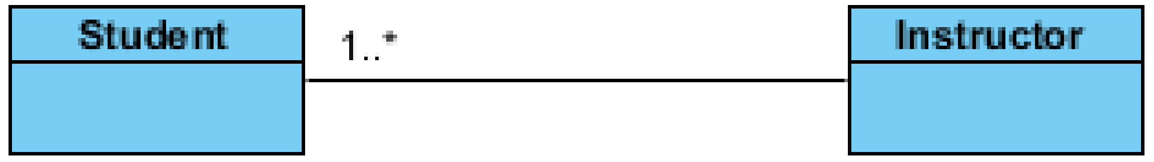


```
import java.util.ArrayList;

public class Person {
    private String name;
    private int age;
    private ArrayList<Book> books;

    // ...
}
```

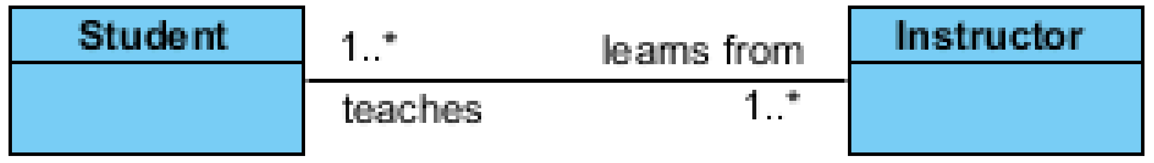
A single student can associate with multiple teachers:

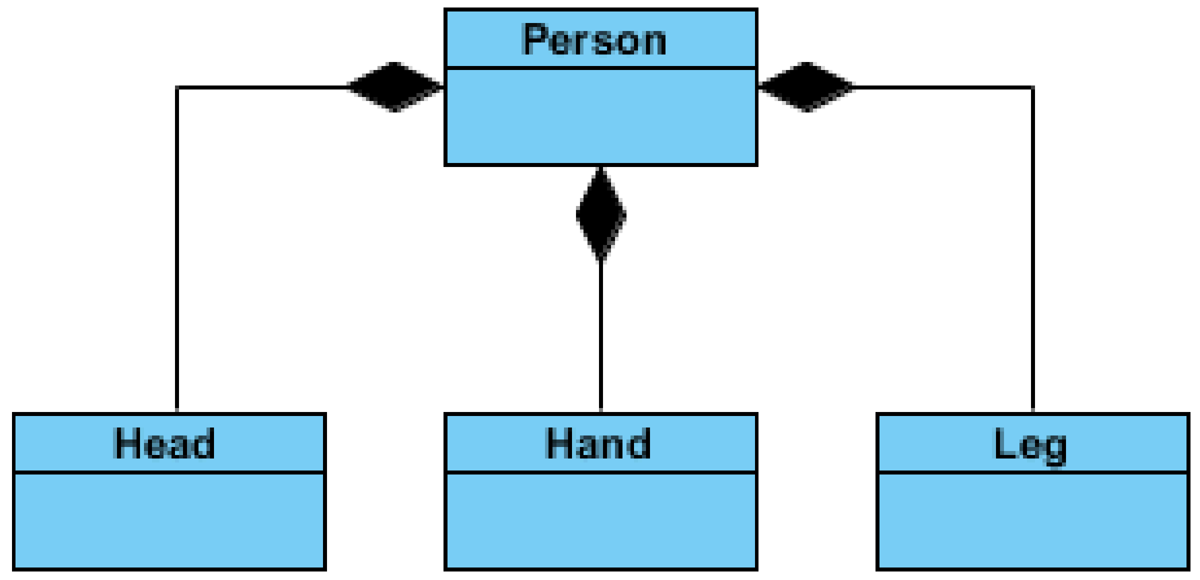


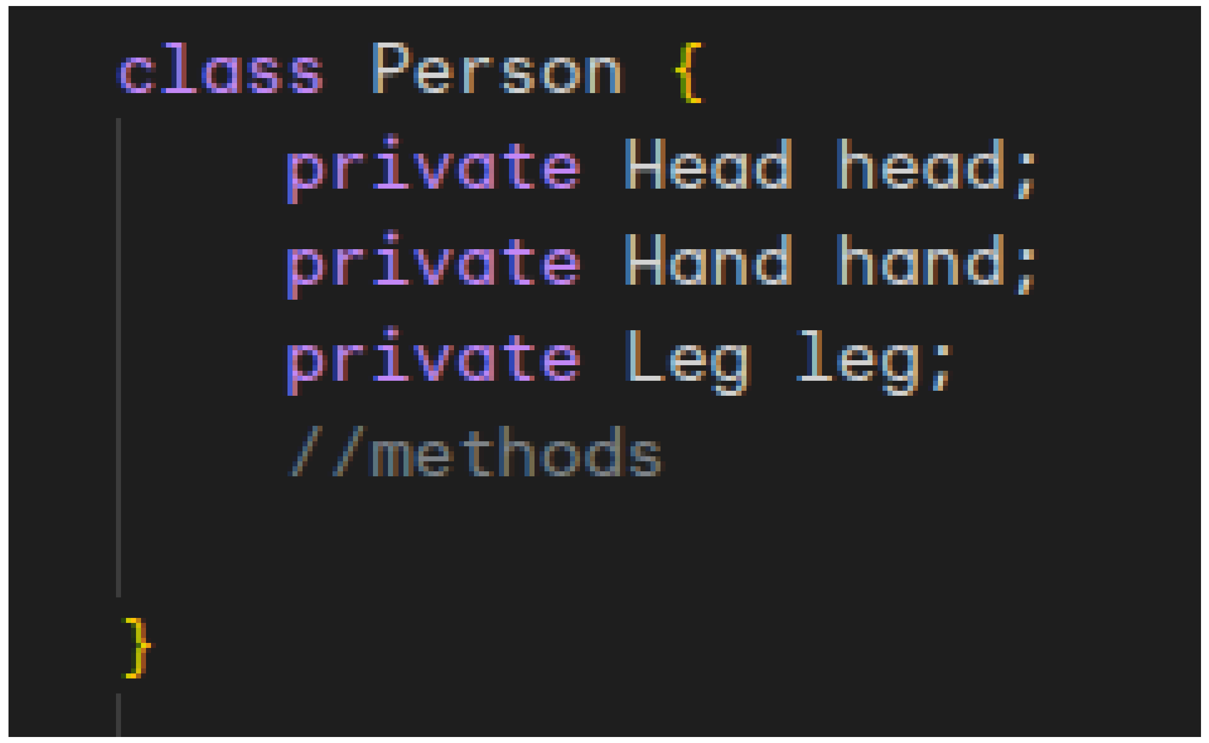
The example indicates that every Instructor has one or more Students:



We can also indicate the behavior of an object in an association (i.e., the role of an object) using role names.

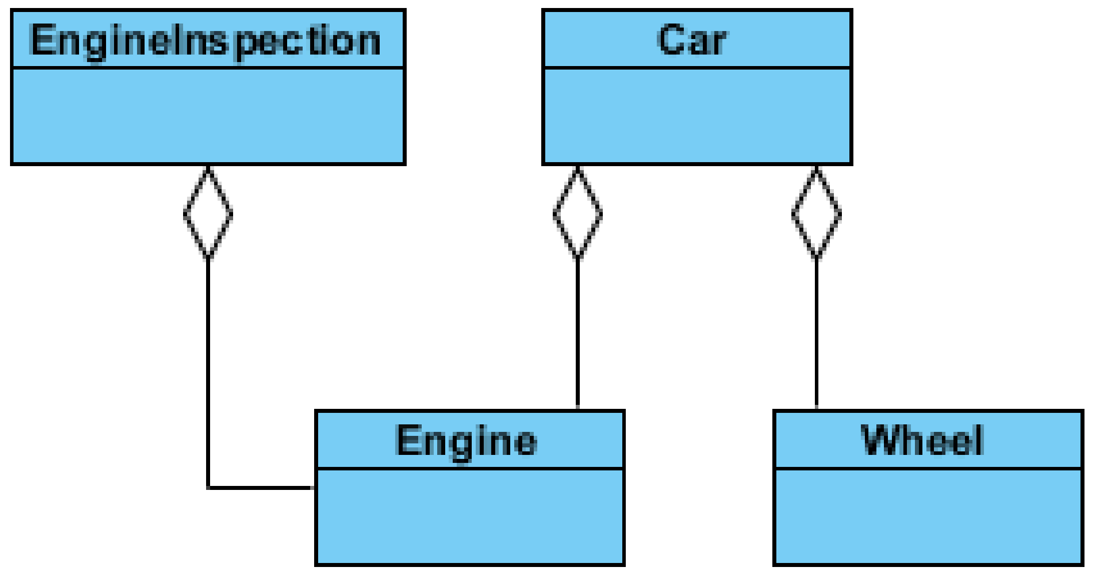


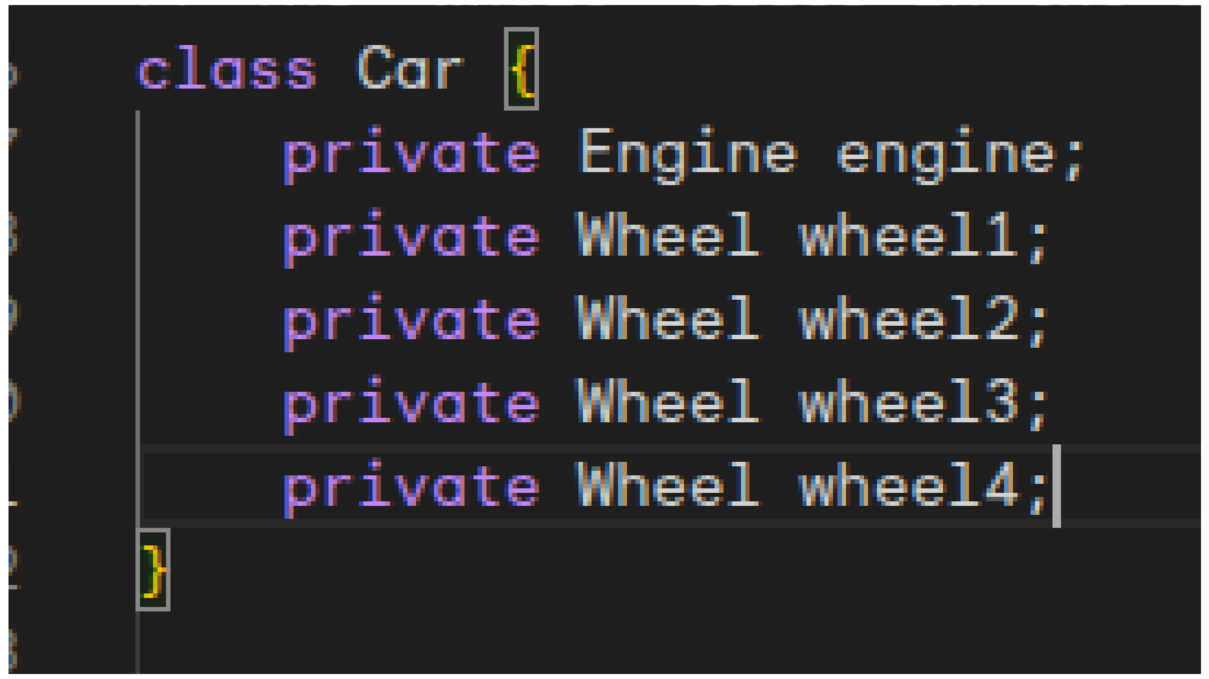




```
class Person {
    private Head head;
    private Hand hand;
    private Leg leg;
    //methods

}
```

Composition: has-a relationship (strong)

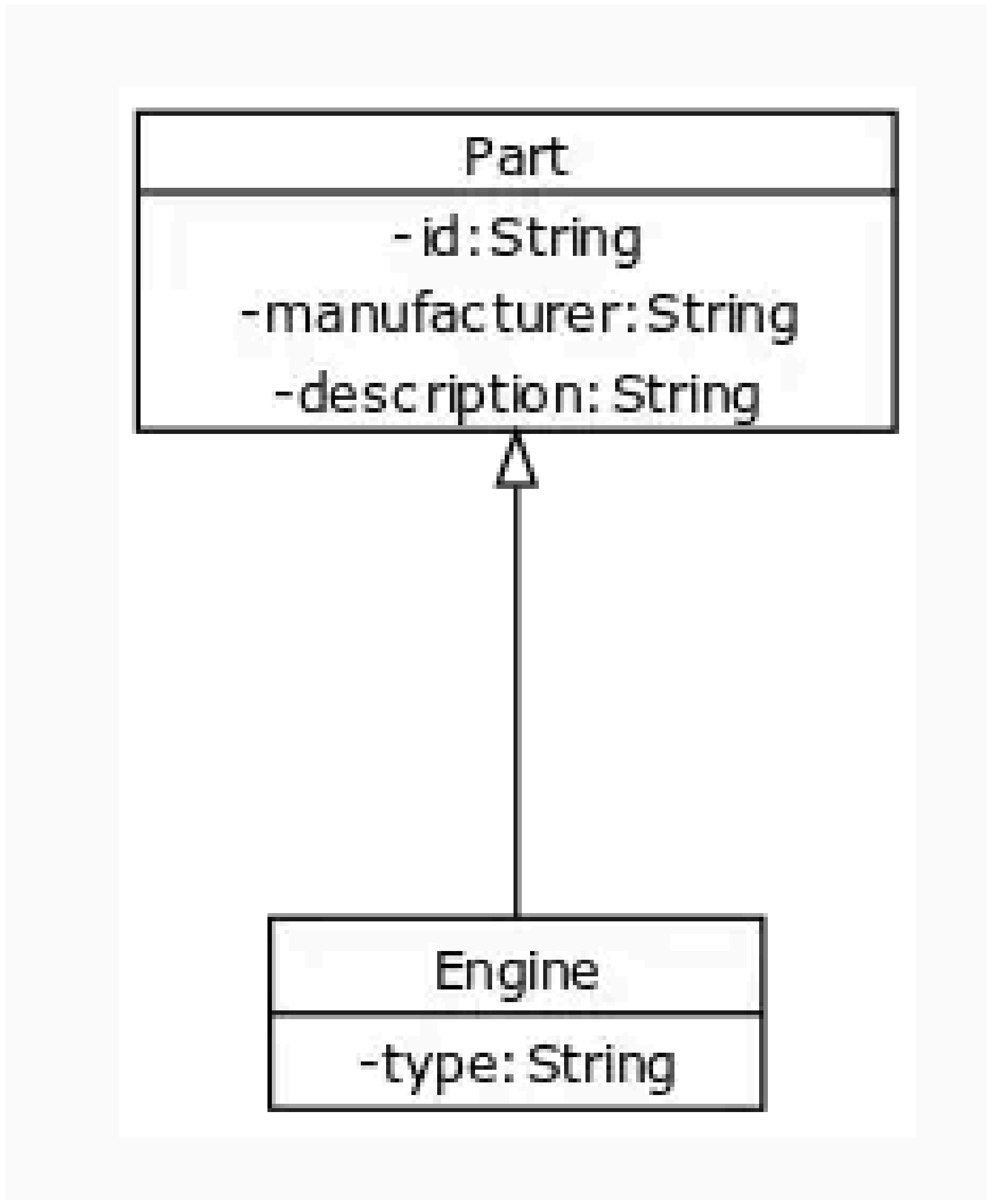


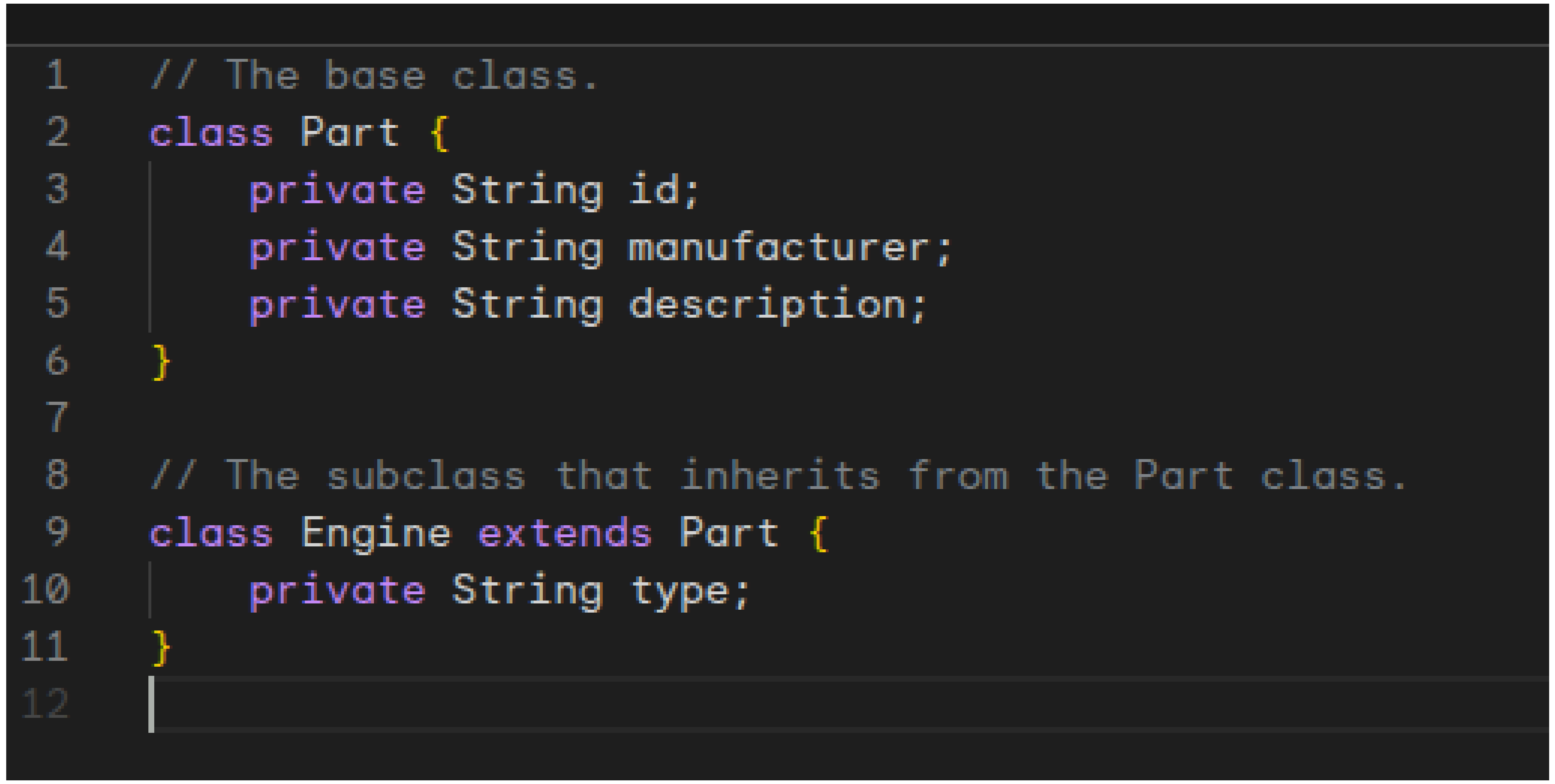


```
class Car {
    private Engine engine;
    private Wheel wheel1;
    private Wheel wheel2;
    private Wheel wheel3;
    private Wheel wheel4;
}
```

Aggregation: has-a relationship (weak)

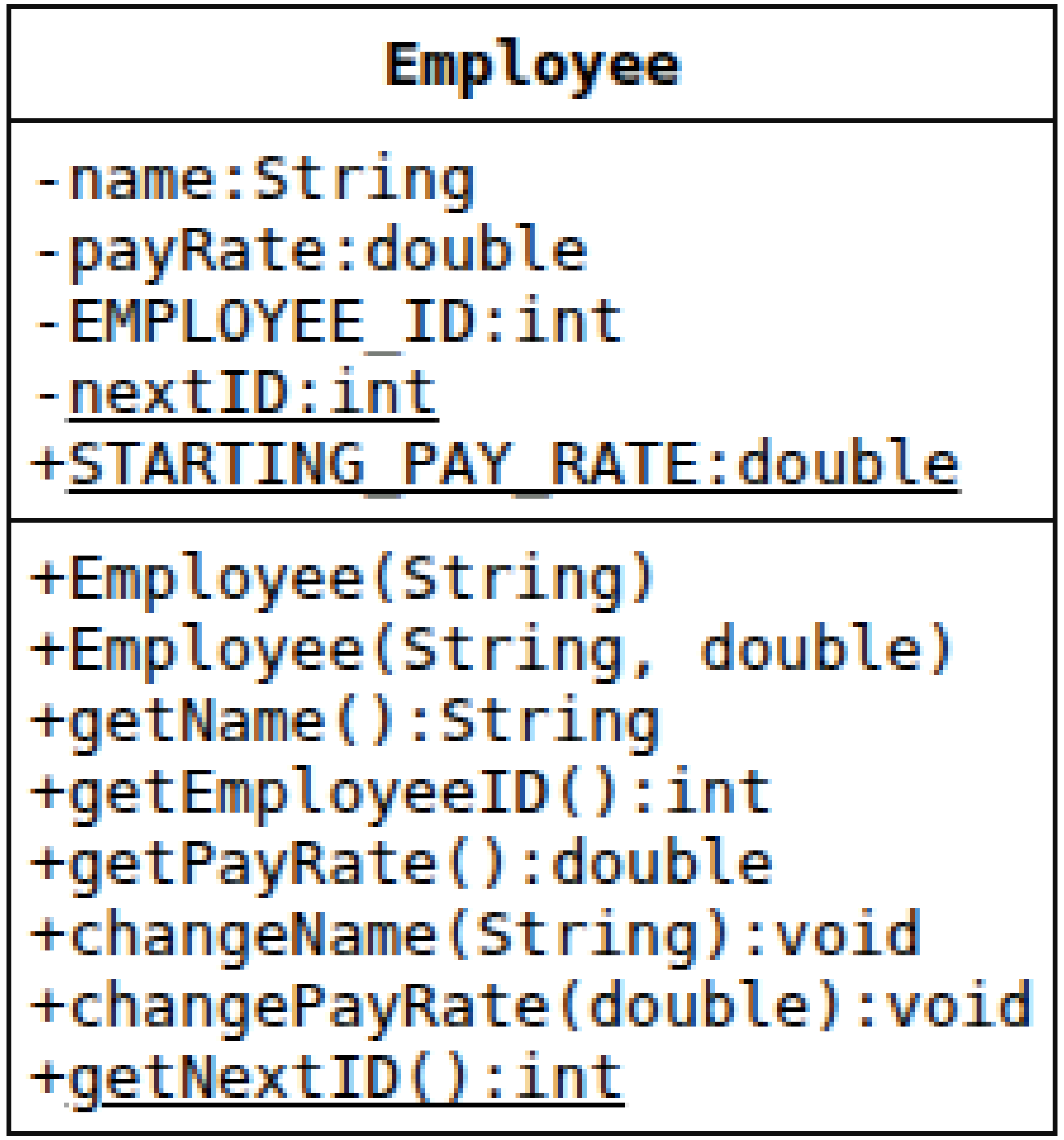
Inheritance: is-a relationship





```java
// The base class.
class Part {
    private String id;
    private String manufacturer;
    private String description;
}

// The subclass that inherits from the Part class.
class Engine extends Part {
    private String type;
}
```

Exercise: Implement this class in Java



```
Employee
-name:String
-payRate:double
-EMPLOYEE_ID:int
-nextID:int
+STARTING_PAY_RATE:double
+Employee(String)
+Employee(String, double)
+getName():String
+getEmployeeID():int
+getPayRate():double
+changeName(String):void
+changePayRate(double):void
+getNextID():int
```