Introduction

UML (Unified Modeling Language) is a graphical language for modeling the structure and behavior of object-oriented systems. UML is widely used in industry to design, develop and document complex software. This page will focus on creating UML class diagrams, which describe the internal structure of classes and relationships between classes.

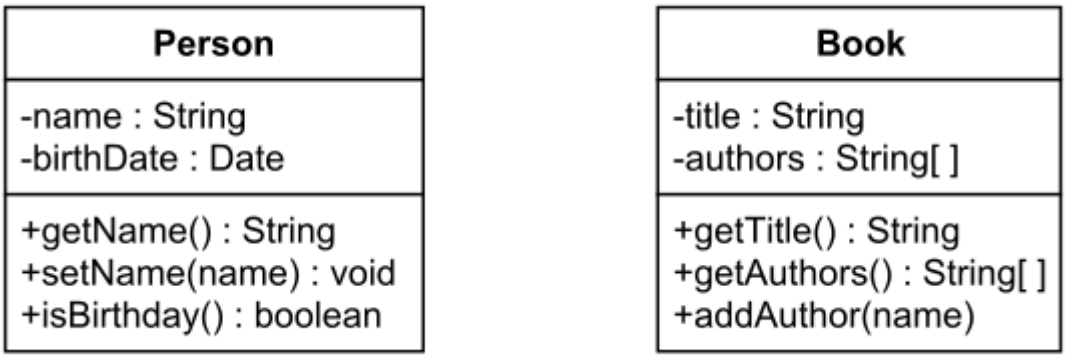
For additional information beyond the usual suspects (your textbook and Wikipedia), see UML Basics: The Class Diagram.

Classes

A class diagram contains a rectangle for each class. It is divided into three parts.

1. The name of the class.
2. The names and types of the fields.
3. The names, return types, and parameters of the methods.

For example, a Person class and a Book class might be modeled like this.



This indicates that a Person object has private fields named name and birthDate, and that it has public methods named getName, setName and isBirthday. A Book object has private fields named title and authors. A Book object also has public methods named getTitle, getAuthors and addAuthor.

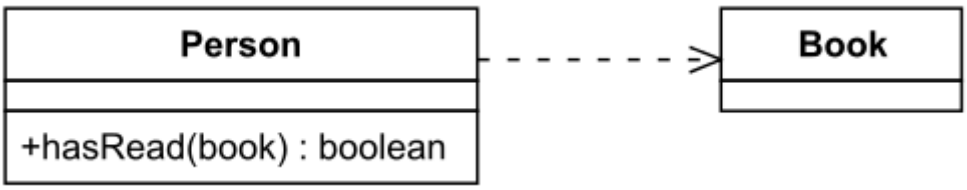
The examples below also model a Person class and Book class, but only shows fields or methods as needed for illustration.

Use Relationships

Often, objects and/or methods of one class use objects/methods from another class. For example, a person might read and/or own a book, and these relationships might be modeled in the UML diagram, so that they will be implemented in the corresponding program.

UML class diagrams include the following types of use-relationships, in order from weakest to strongest.

- Dependency: An object of one class might use an object of another class in the code of a method. If the object is not stored in any field, then this is modeled as a dependency relationship. For example, the Person class might have a hasRead method with a Book parameter that returns true if the person has read the book (perhaps by checking some database).



- Unidirectional Association: An object might store another object in a field. For example, people own books, which might be modeled by an `owns` field in Person objects. However, a book might be owned by a very large number of people, so the reverse direction might not be modeled. The *'s in the figure indicate that a book might be owned any number of people, and that a person can own any number of books.

- Bidirectional Association: Two objects might store each other in fields. For example, in addition to a Person object listing all the books that the person owns, a Book object might list all the people that own it.

- Aggregation: One object A has or owns another object B, and/or B is part of A. For example, suppose there are different Book objects for different physical copies. Then the Person object has/owns the Book object, and, while the book is not really part of the person, the book is part of the person's property. In this case, each book will (usually) have one owner. Of course, a person might own any number of books.

- Composition: In addition to an aggregration relationship, the lifetimes of the objects might be identical, or nearly so. For example, in an idealized world of electronic books with DRM (Digital Rights Management), a person can own an ebook, but cannot sell it. After the person dies, no one else can access the ebook. [This is idealized, but might be considered less than ideal.]

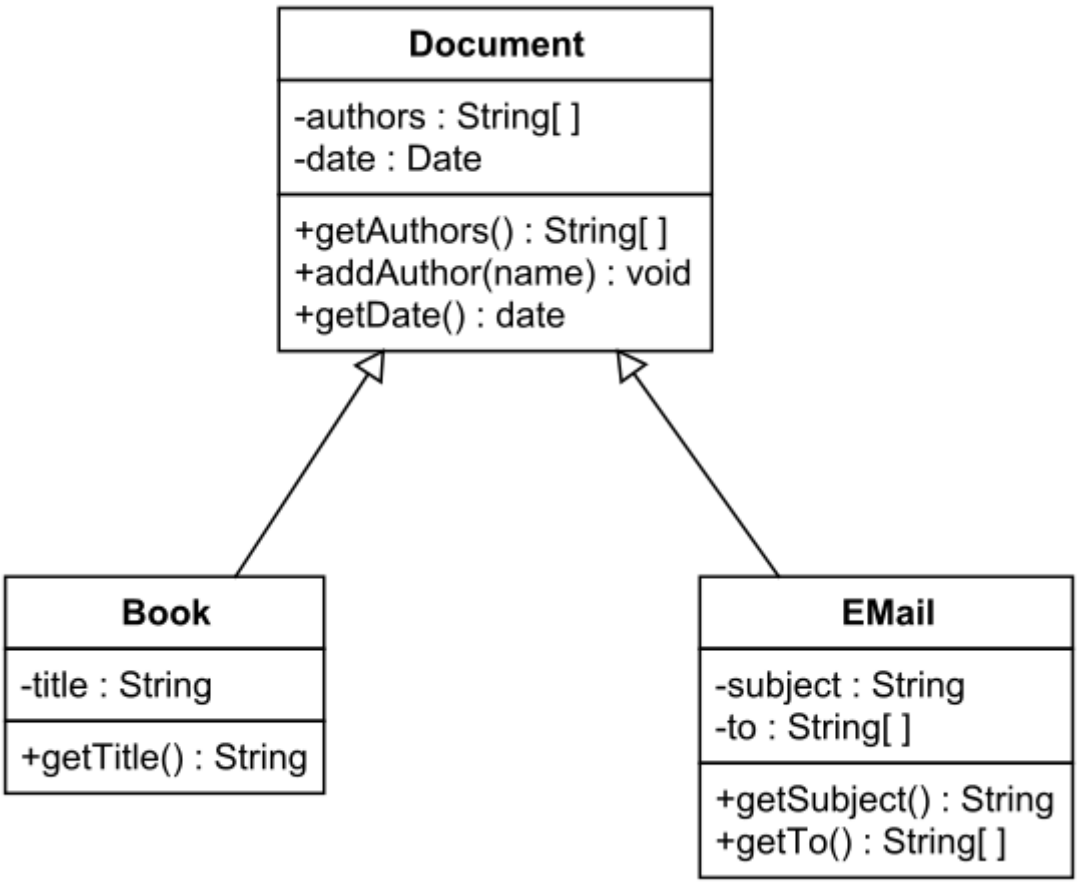
If you have difficulty distinguishing among association, aggregation and composition relationships, don't worry! So does everybody else!

Inheritance Relationships

The inheritance relationships in UML match up very closely with inheritance in Java.

- Generalization: A class extends another class. For example, the Book class might extend the Document class, which also might include the Email class. The Book and Email classes inherit the

fields and methods of the Document class (possibly modifying the methods), but might add additional fields and methods.



- Realization: A class implements an interface. For example, the Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.