# Assignment 6 C212 FA22

Last modified 2022-10-26 1:00 PM

## Learning Objectives

After completing this assignment, students will be proficient in modeling information with basic object-oriented design paradigms in Java. Given a scenario, they will be able to design and implement classes to model relevant details. They will also be able to write tests to demonstrate the quality and behavior of their work.

## Summary

The gist of this assignment document's contents are as follows:

- Program Description - Modeling scenarios with OOP
  - Author
  - Book
  - BetterBook
  - Testing for each
- Side notes on the goal of the assignment
  - The OOP iceberg
  - Embrace testing
- Rubric
  - Also visible on Canvas assignment page
- Changelog
  - View updates posted 1pm 10/26 described in this Canvas announcement

## Program Description

In this assignment you'll be writing three classes, which are meant to represent authors and books, as well as the programs to create instances and test the classes' functionality. You'll be given a description of the scenario being modeled – think of this like the client telling you what they want from your code – and from that description you'll have to extract the relevant information.

### Task 1 - Author

*For representing an author, there is some information we want to track: their name, email, and date of birth. We'll have a lot of authors in the system. An author's name and email should be able to change, but their date of birth should not. Furthermore, the Author class should be able to share (with any code that has access to an Author instance) this information about that author, as well as the ability to update the appropriate information. The author must have a name and a date of birth, but doesn't have to have an email address. If some other code wants to display information about the author, the author instance should be able to provide a String that looks like this:*

"Cay Horstmann (cay@horstmann.com), born 6/16/1959"

*Or, for authors who have no email address, it should look like this:*

"Ada Lovelace (no email), born 12/10/1815"

From this description, write a class, Author.java, which models an author. Identify the three instance variables, their types and access modifiers, and the methods that should be included in Author. Notice that we don't want authors to be able to "do" anything, like write books. For this assignment, ensure that information about an Author can only be retrieved via methods, not via its instance variables directly. Remember to override toString() for getting the complete information String about an Author.

As you write Author.java, also write a class TestAuthor.java, which contains only a main method. In this main method, create instances of Author and try out all the functionality on them. The purpose of this process is for you to convince yourself that Author works like you think it should. We will not tell you how much testing is required to get full credit on this part of the task. Alongside each test with some output, you must include either a comment or console printout accompanying it to show what the expected output is. Write enough testing to convince yourself it works beyond any reasonable doubt. But, since it's your first time writing substantive tests for classes, we'll hint that your tests should include answering (for yourself) the following questions, in addition to the obvious ones:

- Can I create multiple Authors?

- If I create multiple Authors, are they and their information distinct? Would changing some information on one Author change information on another?

## Task 2 - Book

*For representing a book, let's say that a book is written by exactly one author. A Book should know the Author that wrote it. It should also be able to store a title, price, and edition number. In fact, every book should have a title, price, edition number, and author, no matter what. The price should be able to change, but not the title, edition number or author – however, the book's author's email address or name might change, and the book should always be able to access the latest information about its author. Lastly, any code that has access to a Book instance should be able to get information about that book. If some other code wants to display complete information about a book, the book instance should provide a String that looks like this:*

"Big Java: Late Objects (2nd Edition), by Cay Horstmann (cay@horstmann.com), born 6/16/1959, available for $112.50"

From this description, write a class, Book.java, which models a book. Identify the instance variables, their types and access modifiers, and methods Book should have. Books also don't "do" anything yet; like Author, they just hold on to and display information. Remember to override and utilize toString() for getting the complete information String about a Book and its Author.

As you write Book.java, also write a class TestBook.java, which contains only a main method. In this main method, test Book just as you did with Author. Remember, the goal of testing is to convince yourself it works beyond reasonable doubt. Of course, you'll also have to use Author in TestBook, because Book depends on Author.

## Task 3 - BetterBook

*We made some assumptions about books that aren't realistic, and now we're going to make an enhanced book model that can do two things: a "better book" can have multiple authors, and a Book instance can be used to create a revised new "better book" instance with the same title and authors, but in the next edition number and at double the price, due to inflation. Creating a new edition of the book doesn't need any additional information besides what the book already knows about itself, of course. Since a "better book" can have a lot of authors, don't include the date of birth of each author, but do include their names and email addresses, for each one that has an email address. For example,*

"Elements of Charles Babbage's Analytical Machine (1st Edition), by Ada Lovelace and Luigi Federico Menabrea (lfmenabrea@carrierpidgeon.it)"

*As lovers of the written word, we have very strong opinions on the Oxford comma, and they're not good. We want books with multiple authors displayed in this way to not use it, meaning the names should be formatted like "A, B and C" instead of "A, B, and C".*

Begin by duplicating your work for Book.java into BetterBook.java. Adapt these enhancements into the class, including whatever changes need to be made to the **members** (a way of saying the variables and methods belonging to a class) of Book.

Lastly, write a class TestBetterBook.java that tests BetterBook. You can focus your testing on this new functionality, rather than having to duplicate all the testing you wrote for Book. The same expectations for testing apply here.

## Side notes on the goal of the assignment

Assignment 6 deviates from previous assignments in the sense that you are not writing a program that boils down to an interactive script. That's not to say it is completely different: you have already learned how to represent information in a structured way, such as representing the contents of an image file with a 3D array. You've also had a taste of writing modular code: in A4, your method to make a move on a board didn't care whether it was a human or AI making the move. These are good things to know, and we introduce them because it helps you start thinking like a programmer, but these ideas are the absolute beginning of what object-oriented programming is. Which is to say, you have a long way to go from here! This should not discourage you – in fact, you can use this moment before you truly start learning OOP to think how you would solve these same problems that the paradigm's designers were faced with. For example, what if we wanted to make specific kinds of books that have unique functionality? Can we still treat all those sub-types of books uniformly? How much code would we have to write and rewrite? Why don't we keep a reference in each Author to their Books?

Testing always gets a bad rap from students when they're first introduced to it. It feels annoying to have to prove that something works when you could easily just look at the actual insides and see what it does, right? And plus, you wrote it, you already know what it does!

Writing tests for your code is giving yourself a tool. It is something that we as programmers use to make our jobs easier, not harder. There's a reason why all enterprise production codebases use test suites, and its not because their developers' instructors back in college told them they had to do it to get a good grade. If you're working on a big project and decide to change one little thing that ends up breaking it all, you'd want to know exactly when and where the problem came from. This is the motivation of unit testing, which we'll see in Lab09.

With that in mind, understand that we will never tell you to, for example, "write at least 3 test cases for each of this class's methods". Why? Because this is not how you should think about testing. It's not to check off a box on a rubric – it's for you. However, to possibly undercut our point here, in many upcoming assignments, it will be both something for you to use as a tool and something for you to check off on a rubric, as we get you in the habit of writing tests and writing tests well. So, on this assignment, get used to writing tests while the tests you write are still small and easy, so that you're ready for when the code you're testing gets more and more complex.

## Rubric

A rubric is also available on the assignment page on Canvas, but here is an outline of the point breakdown for different categories.

- (10pts) Code cleanliness, conventions, and comments
    - (10/10pts) for >95% correctness and all required comments
    - (7.5/10pts) for repeated minor mistakes or one egregious mistake, or for missing either overall or inline comments
    - (5/10pts) for code that is sloppy and hard to understand or is missing all comments
    - (<=2.5/10pts) for code that has no comments, is hard to understand, and consistently does not follow Java conventions
- (15pts) Author
    - (6pts) for instance variables
    - (1pt each) for instance variable of the appropriate types
    - (1pt each) for access modifiers on instance variables
    - (3pts) for constructor and its parameters
    - (3pts) for accessor and mutator methods
        - (-1pt) for each method with inappropriate return types, parameters, or access modifiers
    - (3pts) for toString method
        - (-1pt) if a null email address is not formatted correctly
- (15pts) TestAuthor
    - Sliding scale based on test coverage and completeness
- (15pts) Book
    - (3pts) for instance variables besides author, graded similarly to the above
    - (3pts) for author instance variable
    - (3pts) for constructor and its parameters
    - (3pts) for accessor and mutator methods
    - (3pts) for toString method
        - (-1pt) if the edition number isn't formatted correctly
        - (-1pt) if the price isn't formatted correctly
- (15pts) TestBook
    - Sliding scale based on test coverage and completeness
- (15pts) BetterBook
    - (10pts) for allowing multiple authors via an array or ArrayList
        - (2pts) for necessary change(s) to instance variables
        - (4pts) for necessary change(s) to constructor
        - (4pts) for necessary change(s) to toString
    - (5pts) for new edition method

        - (3pts) for method signature
        - (2pts) for method behavior
- (15pts) TestBetterBook
    - Sliding scale based on test coverage and completeness

Here's a rough guideline for grading with respect to testing:

- (0/15pts) for no testing whatsoever
- (5/15pts) for writing some but not much code that resembles testing, but doesn't seem clear what's being tested or why
- (10/15pts) for plenty of testing but the testing doesn't make sense or is missing corresponding expected outputs, OR tests which are reasonable and have corresponding outputs but aren't sufficient for demonstrating coverage
- (15/15pts) for plenty of well-thought out testing that demonstrates coverage of the testing and provides corresponding expected results

## Changelog

If any corrections, clarifications, etc. are made to the assignment after its release, they will be listed here. Any changes more substantial than simply fixing typos will get corresponding Canvas announcements. That way, you don't need to keep checking whether updates have occurred.

- 10/26 1pm: Updated expectations and rubric and provided clarifications and hints on various aspects of the assignment, described in more detail on Canvas