Java Class Modeling Activity

Overview

In this activity, you'll create a new IntelliJ project and practice building real-world Java classes from scratch. You'll write classes that follow object-oriented principles, including:

- Using private fields
- Writing constructors that set all values
- Creating getters and setters

You'll model four classes. Each one builds on the skills you learned from the last. Take your time, write clean code, and think about how you'd use these classes in a real app.

Class 1: Author

Create a class named Author with the following private fields:

- name: String – Full name of the author
- birthYear: int – The year the author was born
- nationality: String – Country of origin
- alive: boolean – Is the author still living?
- netWorthMillions: double – Net worth in millions of dollars

What to do:

- Write a constructor that takes all the fields as parameters.
- Write getters and setters for each field (Do it by hand for a bit and then when you're comfortable use IntelliJ Alt+Insert).

Class 2: Publisher

Create a class named Publisher with the following private fields:

- name: String – Name of the publishing company
- location: String – City where the company is based

- yearFounded: int – Year the company started
- isIndependent: boolean – True if the publisher is not owned by a corporation
- revenue: long – Annual revenue in US dollars

What to do:

- Write a constructor that takes all the fields as parameters.
- Write getters and setters for each field.

Class 3: Book

Now you're going to model a class that uses other classes as fields. This is how we represent relationships between objects in Java.

Create a class named Book with the following private fields:

- title: String – Title of the book
- price: double – Price in US dollars
- author: Author – The author who wrote the book
- publisher: Publisher – The publisher of the book

Note: Author and Publisher are classes you already created. This is called composition — the Book class has an author and a publisher.

What to do:

- Write a constructor that takes all four fields as parameters.
- Write getters and setters for each field.

Class 4: Review – Constructor Overloading

Concept Focus: Multiple Ways to Create an Object

Sometimes in the real world, we don’t always have all the information when creating something. Think about writing a review for a book.

Question:
Have you ever left a quick 5-star rating on Amazon, without typing anything?
Now compare that to a detailed review where someone writes a whole paragraph.

That's two very different kinds of reviews. But both are valid. Should we really require all users to submit the same info?

This is where constructor overloading becomes useful.

Your Task: Create a Review Class

Create a class named Review with the following private fields:

- reviewerName: String – Name of the reviewer
- rating: int – A score out of 5
- comment: String – Optional comment or review text
- book: Book – The book that is being reviewed

Step-by-Step Instructions

1. Full Constructor
   Write a constructor that takes all four fields as parameters. This would represent a detailed review where someone enters their name, rating, and a written comment.
2. Now Think About This…
   What if a user just wants to rate the book without writing anything?
   Or they prefer to stay anonymous?

We shouldn't force them to enter a name or comment. That's where a second constructor comes in.

3. Create a Simpler Constructor
   Write another constructor that only takes:

- int rating
- Book book

This is for short-form, anonymous reviews.

4. Getters and Setters
   Write getters and setters for all fields. These allow other parts of the program to read or modify the values safely.

Discussion

Why might we want multiple constructors?

- Think about flexibility: Different users have different behaviors or preferences.
- Think about real-world systems: Sometimes data is partial.
- Think about code readability: It makes your class easier to use in different contexts.

What happens if someone passes in a rating that doesn't make sense (like -1 or 100)?

- Should your constructor guard against that?
- Could you add validation in the constructor or setter?
- Should you throw an exception? Or silently clamp the value?

Final Task

Once you're done, write a simple Main class and method where you:

- Create one Author and one Publisher
- Use those to create a Book
- Create at least one Review for the Book

This isn't just about syntax — it's about learning how to model real-world problems in code. Keep your code clean, organized, and clear.