

Classes, Objects and Encapsulation

Private fields, validated setters, constructors, and your first LMS classes

Lecture + Lab 1

Branch: week2-encapsulation

Learning outcomes 1 and 3

What you will be able to do after today

- Explain **information hiding** and why fields should be `private`.
- Write **constructors**, **getters** and **setters**, and put **validation** in the setter.
- Explain why a constructor should call the setters.
- Build the first three LMS classes: `Book`, `Member` and `Library`, and test them from `Main`.

1. The problem: anyone can break your object

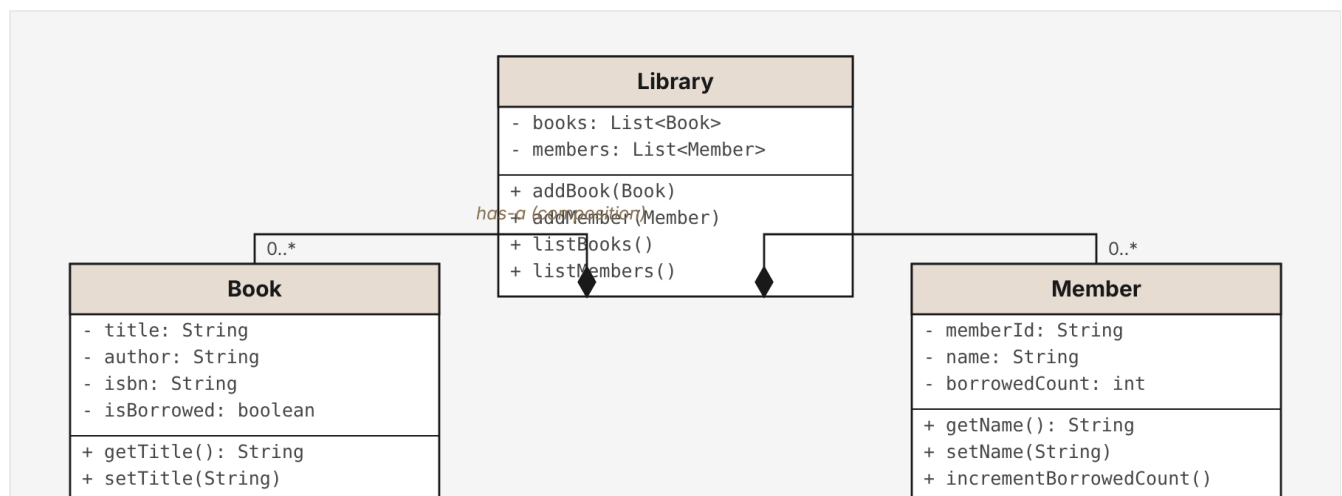
```
public class Book {
    public String title;           // public field
    public boolean isBorrowed;
}

Book b = new Book();
b.title = "";                    // nothing stops this
b.isBorrowed = true;             // the library never recorded a loan
```

A `Book` with an empty title is a **bug waiting to happen**, and every place that touches `title` is a place the bug can start. With public fields you cannot guarantee anything about your own objects.

2. Encapsulation in one sentence

Encapsulation = keep the data **private** inside the class and let the outside world use it only through **public methods** that you control.



Modifier	Who can see it
<code>private</code>	Only code inside the same class
<code>public</code>	Everyone
<code>protected</code>	Same package and subclasses (Week 3)

Rule for this course: every field is `private`.

3. Getters, setters and validation

A **getter** reads a field. A **setter** changes it, and is the right place to **reject bad data**.

```
public class Book {
    private String title;

    public String getTitle() { return title; }

    public void setTitle(String title) {
        if (title == null || title.isBlank()) {
            throw new IllegalArgumentException("Title cannot be blank");
        }
        this.title = title;
    }
}
```

`IllegalArgumentException` is Java's built-in way to say "you gave me a bad value". Week 5 shows how to handle it properly.

Tip Not every field needs a setter. `isBorrowed` is changed by *borrow* and *return*, not by anyone's free choice, so give it behaviour methods instead of a public setter. Hiding is also a design choice.

4. Constructors should use the setters

```
public Book(String title, String author, String isbn) {
    setTitle(title); // validation runs here too
    setAuthor(author);
    setIsbn(isbn);
    this.isBorrowed = false;
}
```

If the constructor assigned `this.title = title` directly, you could still create an invalid `Book`, and the validation in the setter would be useless.

```
new Book("", "Nobody", "000"); // throws IllegalArgumentException: Title cannot be blank
```

5. The Week 2 LMS

`Library` **has** books and members (this is called *composition*: a has-a relationship).

```
public class Library {
    private final List<Book> books = new ArrayList<>();
    private final List<Member> members = new ArrayList<>();

    public void addBook(Book book) { books.add(book); }
    public void addMember(Member member) { members.add(member); }

    public void listBooks() {
        System.out.println("--- Books in the library ---");
        for (Book b : books) System.out.println(b);
    }
}
```

`toString()` decides what `System.out.println(book)` shows:

```
@Override
public String toString() {
    return "Book{title='" + title + "', author='" + author + "', isbn='" + isbn + "', borrowed='" +
    isBorrowed + "'}";
}
```

Running it

```
Library library = new Library();
library.addBook(new Book("Clean Code", "Robert C. Martin", "978-0132350884"));
library.addMember(new Member("M001", "Nadeesha Perera"));
library.listBooks();
library.listMembers();

try {
    library.addBook(new Book("", "Nobody", "000-0000000000"));
} catch (IllegalArgumentException e) {
    System.out.println("Rejected invalid book: " + e.getMessage());
}
```

```
--- Books in the library ---
Book{title='Clean Code', author='Robert C. Martin', isbn='978-0132350884', borrowed=false}
--- Registered members ---
Member{id='M001', name='Nadeesha Perera', borrowedCount=0}
Rejected invalid book: Title cannot be blank
```

6. Common mistakes

Mistake	Why it is a problem
<code>public</code> fields	Anyone can break the object
Setter with no validation	Encapsulation in name only
Constructor assigns fields directly	Bypasses validation
Returning a mutable list straight from a getter	Callers can change your internal list
Forgetting <code>this.</code> when names clash	You assign the parameter to itself and the field stays <code>null</code>

7. Lab 1 preview (Week 2)

Build `Book`, `Member` and `Library`; every field private; every constructor uses its setters; test with a `Main` that includes one **invalid** input. Open the PR `Week 2: Classes, Objects & Encapsulation` from branch `week2-encapsulation`.

8. Self-check

1. What is the difference between a class and an object?
2. Why does the constructor call `setTitle(...)` instead of assigning the field?
3. A teammate makes `borrowedCount` public "to save time". Name two things that can now go wrong.
4. Which exception does the setter throw, and what is its message?