

"Lab 2: Inheritance and Polymorphism"

Introduce `LibraryItem`, `Student` and `Staff`; override fees; see static vs dynamic binding

Lab 2 of 9

Branch: week3-inheritance-polymorphism

About 2 to 3 hours

Guide: LMS Guide, Step 2

Objective

Refactor the flat Week 2 classes into **hierarchies**, then use **polymorphism**: one loop that behaves differently for each kind of item. You will also *observe* the difference between overloading (static binding) and overriding (dynamic binding).

Before you start

- Lab 1 merged. `main` is up to date.
- Lecture 3 notes read.
- Note one change: `Book` no longer owns `title` or `isBorrowed`. They move **up** into `LibraryItem`, and `isbn` is replaced by an `id`.

Git: start and submit

```
git checkout main
git pull origin main
git checkout -b week3-inheritance-polymorphism
```

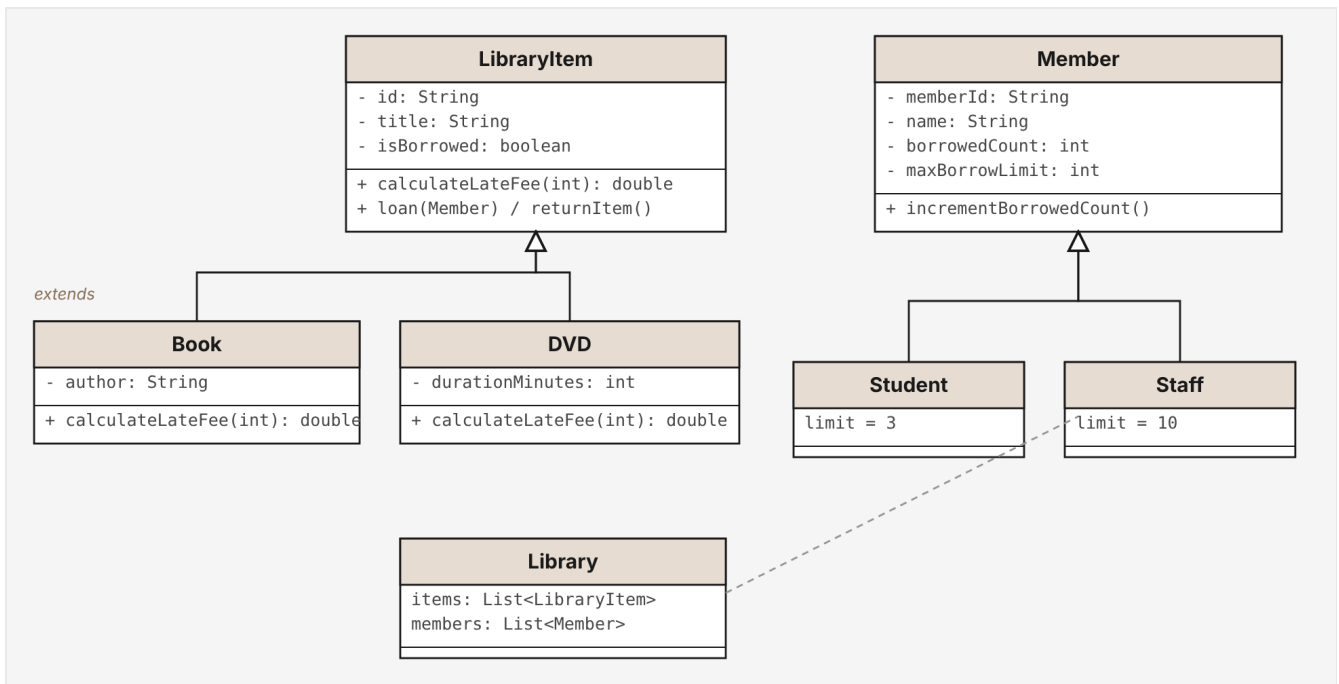
Commit small and often (see the *Git Support Guide*). When you finish:

```
git push -u origin week3-inheritance-polymorphism
```

Open a Pull Request titled **Week 3: Inheritance & Polymorphism**, fill in the template, and request a review from your lecturer.

Design first (10 minutes)

Draw the hierarchy first (`docs/uml-week3.png`):



Question to answer in your PR: *which relationships are is-a, and which one is has-a?*

Tasks

Task 1: LibraryItem (superclass)

Private `id`, `title`, `isBorrowed`; a constructor `LibraryItem(String id, String title)` that uses validated setters; getters; `loan()` -style helpers are not needed yet (just `borrow` state through `isBorrowed()` / `setBorrowed()`). Add

```
public double calculateLateFee(int daysLate) { return daysLate * 10.0; }
```

as the default rule, and a `toString()` that uses `getClass().getSimpleName()`, e.g. `Book{id='B001', title='Clean Code', borrowed=false}`.

Task 2: Book and DVD (subclasses)

- `Book` extends `LibraryItem`: adds private `String author` (validated), constructor `Book(String id, String title, String author)` starting with `super(id, title)`.
- `DVD` extends `LibraryItem`: adds private `int durationMinutes`, constructor `DVD(String id, String title, int durationMinutes)`, and **overrides** `calculateLateFee` with `daysLate * 25.0`. Mark it `@Override`.

Task 3: Member, Student, Staff

- `Member` gets a fourth field `maxBorrowLimit` (constructor `Member(String memberId, String name, int maxBorrowLimit)`).
- `Student` extends `Member` with limit **3**; `Staff` extends `Member` with limit **10**. Use a private static `final int` constant for the limit and call `super(...)`.

Task 4: Library with polymorphism

- Keep `List<LibraryItem> items` and `List<Member> members`.
- Add `printAllLateFees(int daysLate)`: one loop over `items` printing `title: fee`.

Task 5: overloading (static binding)

Add **three** overloads, each printing which one ran:

```
public void addItem(LibraryItem item) // prints: Added via addItem(LibraryItem): <title>
public void addItem(Book book)       // prints: Added via addItem(Book): <title>
public void addItem(DVD dvd)         // prints: Added via addItem(DVD): <title>
```

Task 6: instanceof

Add `streamIfPossible(LibraryItem item)`. If the item is a `DVD` (use `instanceof DVD dvd`), print Streaming "`<title>`" (`<n>` min)...; otherwise print `<title> cannot be streamed - it isn't a DVD`.

Task 7: Main

Create `Book b1`, `Book b2`, `DVD d1`; add them (declared types decide which overload); add a `Student` and a `Staff`; list; call `printAllLateFees(3)`; stream `d1` and `b1`.

Expected output

```
Added via addItem(Book): Clean Code
Added via addItem(Book): Effective Java
Added via addItem(DVD): Introduction to Algorithms
--- Late fees if 3 day(s) late ---
Clean Code: 30.0
Effective Java: 30.0
Introduction to Algorithms: 75.0
Streaming "Introduction to Algorithms" (120 min)...
Clean Code cannot be streamed - it isn't a DVD.
```

(Your list and member lines will also appear. Extra "extra step" messages in your overloads are fine.)

Checkpoints

Tick each before moving on. If a box stays empty, ask for help *now*; later steps build on it.

- ☐ `Book` and `DVD` do **not** declare `id`, `title` or `isBorrowed` themselves
- ☐ Every subclass constructor begins with `super(...)`
- ☐ `printAllLateFees` has **no** `if / instanceof`, yet prints 30.0 and 75.0
- ☐ You can explain why the same loop gave two answers
- ☐ `docs/uml-week3.png` committed

Experiments: break it on purpose

Learning happens when you predict, run, and compare. Write your **prediction** first, then paste the real compiler message or output into your PR description.

1. **Dynamic binding.** In `printAllLateFees`, print `item.getClass().getSimpleName()` next to each fee. Which class's `calculateLateFee` ran for the `DVD`, and who decided: the compiler or the JVM?

2. **Static binding.** Run: `java Book b = new Book("B9", "Test", "Someone"); LibraryItem same = b; library.addItem(b); library.addItem(same);` Which overloads print? It is the **same object**. Explain.
3. **Cannot find symbol.** In `Main`, declare `LibraryItem item = d1;` then call `item.getDurationMinutes();`. Paste the compiler message and fix it two ways.
4. Delete `super(id, title);` from `Book`'s constructor. Paste the message.

Stretch tasks (optional, no marks lost if skipped)

- Add `Magazine` with fee `daysLate * 5.0` and no new overload. Does `addItem(Magazine)` exist? Which overload runs and why?
- Add `Staff` late-fee discount: `Library.printFeeFor(LibraryItem, Member, int)` giving staff 50% off. Where should that rule live?
- Replace the `if/else` in `streamIfPossible` with a design where `LibraryItem` has a `describe()` method that subclasses override. Which version is better?

Common pitfalls

Symptom	Fix
constructor <code>LibraryItem</code> in class <code>LibraryItem</code> cannot be applied to given types	You forgot <code>super(id, title)</code> in the subclass
DVD fee prints 30.0	Missing or misspelt override; add <code>@Override</code> and the compiler will tell you
<code>title</code> has private access in <code>LibraryItem</code>	Use <code>getTitle()</code> , do not make the field <code>protected</code> just to get around it
incompatible types: <code>LibraryItem</code> cannot be converted to <code>Book</code>	Needs a cast after <code>instanceof</code> , or declare the variable as <code>Book</code>

What to submit

- ☐ PR **Week 3: Inheritance & Polymorphism** from branch `week3-inheritance-polymorphism`
- ☐ `docs/uml-week3.png`
- ☐ PR description: UML answer; predictions and real results for experiments 1 to 4
- ☐ Two review comments on a classmate's PR (from this week on)

Marking guide

Your PR is marked on four criteria. Each is scored 4 (excellent) to 1 (needs work) and weighted as shown.

Criterion (weight)	4	3	2	1
Works (40%)	All tasks run, expected output matches	Minor differences	Partly works	Does not compile or run
OOP design (30%)	Concepts of this week used correctly and deliberately	Mostly correct	Concepts present but misused	Concepts missing
Code quality (15%)	Clear names, small methods, no dead code, comments explain <i>why</i>	Mostly clean	Messy in places	Hard to read
Git and PR (15%)	Exact branch/PR name, small clear commits, full PR description, UML in docs/	One slip	Several slips	One giant commit or direct to <code>main</code>

Check Before you open the PR, read your own diff on GitHub as if you were the reviewer. Most deductions are things you would have spotted yourself.