

"Lab 3: Abstract Classes and Interfaces"

Abstract `LibraryItem`, add `Loanable` and `Reservable`, `Magazine`, and make `Member` `Comparable`

Lab 3 of 9

Branch: week4-abstraction-interfaces

About 2 to 3 hours

Guide: LMS Guide, Step 3

Objective

Turn `LibraryItem` into a proper **abstract** class, define **interfaces** as contracts (`Loanable`, `Reservable`), add a third item type, and make members **sortable** with `Comparable`.

Before you start

- Lab 2 merged.
- Lecture 4 notes read.
- Plan: `LibraryItem.calculateLateFee` stops having a body. Every subclass must write its own.

Git: start and submit

```
git checkout main
git pull origin main
git checkout -b week4-abstraction-interfaces
```

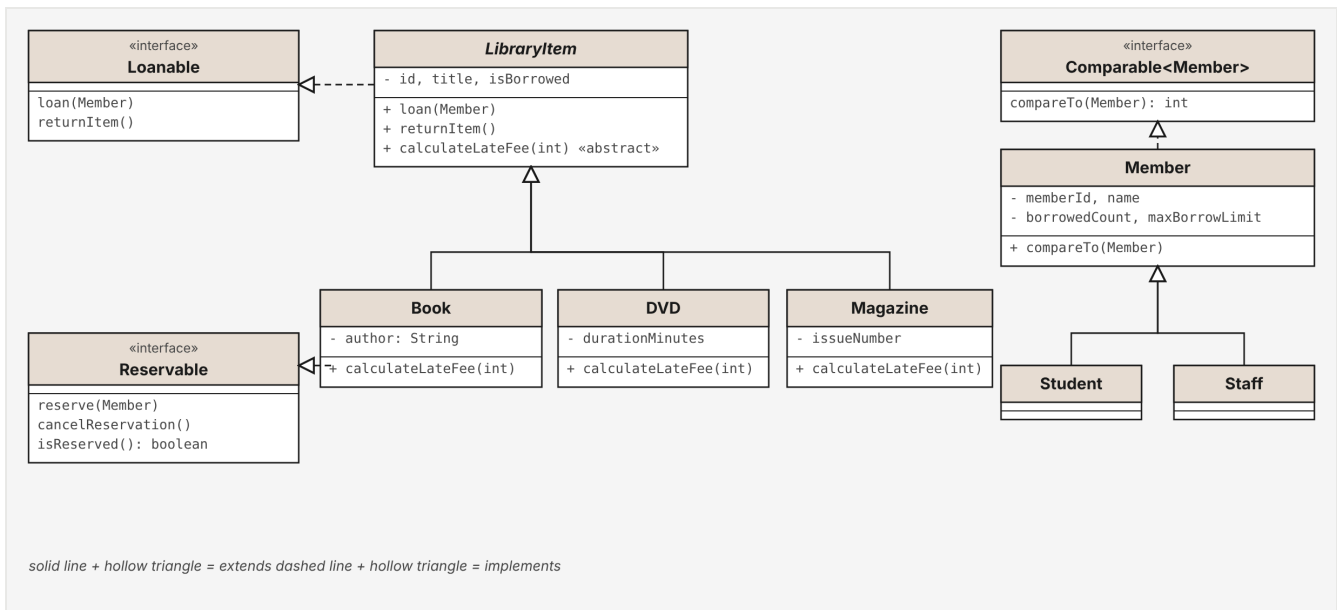
Commit small and often (see the *Git Support Guide*). When you finish:

```
git push -u origin week4-abstraction-interfaces
```

Open a Pull Request titled **Week 4: Abstract Classes & Interfaces**, fill in the template, and request a review from your lecturer.

Design first (10 minutes)

Update your UML (`docs/uml-week4.png`). Use `«interface»` boxes and dashed arrows for *implements*:



Question to answer in your PR: why is `Reservable` an interface instead of a method in `LibraryItem`?

Tasks

Task 1: `Loanable` and `Reservable`

```

public interface Loanable { void loan(Member member); void returnItem(); }
public interface Reservable { void reserve(Member m); void cancelReservation(); boolean isReserved(); }
  
```

Task 2: make `LibraryItem` abstract

- `public abstract class LibraryItem` implements `Loanable`.
- Move the loan logic in: `loan(Member)` refuses (prints) if already borrowed or the member is at `maxBorrowLimit`; otherwise sets `isBorrowed`, calls `member.incrementBorrowedCount()`, prints `<title>` loaned to `<name>`. `returnItem()` clears the flag and prints `<title>` returned.
- Change `calculateLateFee` to `public abstract double calculateLateFee(int daysLate);` (no body).

Task 3: subclasses

- `Book`: implements `Reservable` too; `reservedBy` field; fee `daysLate * 10.0`; reservation messages like Clean Code reserved for Mr. Kasun Silva.
- `DVD`: fee `daysLate * 25.0`; **not** reservable.
- **New** `Magazine` with `issueNumber` and fee `daysLate * 5.0`.

Task 4: `Member` implements `Comparable<Member>`

```

@Override public int compareTo(Member other) {
    return Integer.compare(other.borrowedCount, this.borrowedCount); // most active first
}
  
```

Task 5: `Library`

Keep the earlier methods and add:

- `lend(Loanable thing, Member member)` that calls `thing.loan(member)`.
- `reserveIfPossible(LibraryItem item, Member member)` using `instanceof Reservable r`; otherwise print `<title> cannot be reserved.`
- `printMembersByActivity()` that **copies** the member list, sorts with `Collections.sort`, prints `name (n borrowed)`.

Task 6: Main

Create a `Book`, a `DVD`, a `Magazine`; add them; add a `Student` and a `Staff`; print late fees for 3 days; lend the book to the student, then try to lend it to the staff member (refused); lend DVD and magazine to the staff member; reserve the book for staff, try to reserve the DVD; print the activity ranking.

Expected output

```
--- Late fees if 3 day(s) late ---
Clean Code: 30.0
Introduction to Algorithms: 75.0
Software Weekly: 15.0
--- Loans ---
Clean Code loaned to Nadeesha Perera
Clean Code is already on loan.
Introduction to Algorithms loaned to Mr. Kasun Silva
Software Weekly loaned to Mr. Kasun Silva
--- Reservations ---
Clean Code reserved for Mr. Kasun Silva
Introduction to Algorithms cannot be reserved.
--- Members ranked by activity ---
Mr. Kasun Silva (2 borrowed)
Nadeesha Perera (1 borrowed)
```

Checkpoints

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

- ☐ `new LibraryItem(...)` does not compile (abstract) and you captured the message
- ☐ `Magazine` cannot compile until it writes `calculateLateFee`
- ☐ `DVD` is not `Reservable`, yet `reserveIfPossible(dvd, ...)` works
- ☐ The ranking lists the most active member first
- ☐ `docs/uml-week4.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. Add `new LibraryItem("X", "Y")` to `Main`. Paste the error.
2. Delete `calculateLateFee` from `Magazine`. Paste the error.
3. Add `implements Reservable` to `DVD` and compile. Paste the error, then satisfy the compiler. What did the interface force you to write?
4. Change `compareTo` so the **least** active member comes first. Predict the new ranking, then run.

Stretch tasks (optional, no marks lost if skipped)

- Write `Poster` implements `Loanable` that is **not** a `LibraryItem`; pass it to `library.lend(...)`. Why does that compile?
- Add a default method to `Reservable`: `default String reservationLabel() { return isReserved() ? "Reserved" : "Free"; }`.
- Sort items by title using `Comparator.comparing(LibraryItem::getTitle)`.

Common pitfalls

Symptom	Fix
X is not abstract and does not override abstract method	Implement every method of the interface / abstract class, with the exact signature
attempting to assign weaker access privileges	Interface methods are <code>public</code> ; your implementation must be <code>public</code> too
Sorting does nothing	<code>Member</code> must implements <code>Comparable<Member></code> ; the sort needs the copy that you print
Ranking changes the real member list	Sort a copy (<code>new ArrayList<>(members)</code>)

What to submit

- ☐ PR **Week 4: Abstract Classes & Interfaces** from branch `week4-abstraction-interfaces`
- ☐ `docs/uml-week4.png`
- ☐ PR description: UML answer; the four compiler messages / results above
- ☐ Two review comments on a classmate's PR

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.