

Abstract Classes and Interfaces

Abstract methods, interfaces as contracts, Comparable, composition vs inheritance

Lecture + Lab 3

Branch: week4-abstraction-interfaces

Learning outcomes 1 and 3

What you will be able to do after today

- Explain **abstraction** and write an **abstract class** with an abstract method.
- Write an **interface** and make a class **implement** it.
- Choose between an abstract class and an interface.
- Make objects sortable with `Comparable`.

1. A class that should never be created

Does the library own "a library item"? No. It owns books, DVDs and magazines. Creating a plain `LibraryItem` makes no sense, and what would its late fee be? An **abstract class** says: *"I am an incomplete idea. Subclasses must finish me."*

```
public abstract class LibraryItem implements Loanable {
    // fields and shared code as before ...

    public abstract double calculateLateFee(int daysLate); // NO body: every subclass MUST provide
one
}
```

Try this	The compiler says
<code>new LibraryItem(...)</code>	<code>LibraryItem is abstract; cannot be instantiated</code>
A subclass that forgets the method	<code>Magazine is not abstract and does not override abstract method calculateLateFee(int) in LibraryItem</code>

Tip The compiler enforcing the rule is the whole point: you cannot *forget* to define a late fee for a new kind of item.

2. Interfaces: a contract

An **interface** lists *what* a class can do, with no data and (normally) no code. Any class can promise to do it by saying `implements`.

```

public interface Loanable {
    void loan(Member member);
    void returnItem();
}

public interface Reservable {
    void reserve(Member m);
    void cancelReservation();
    boolean isReserved();
}

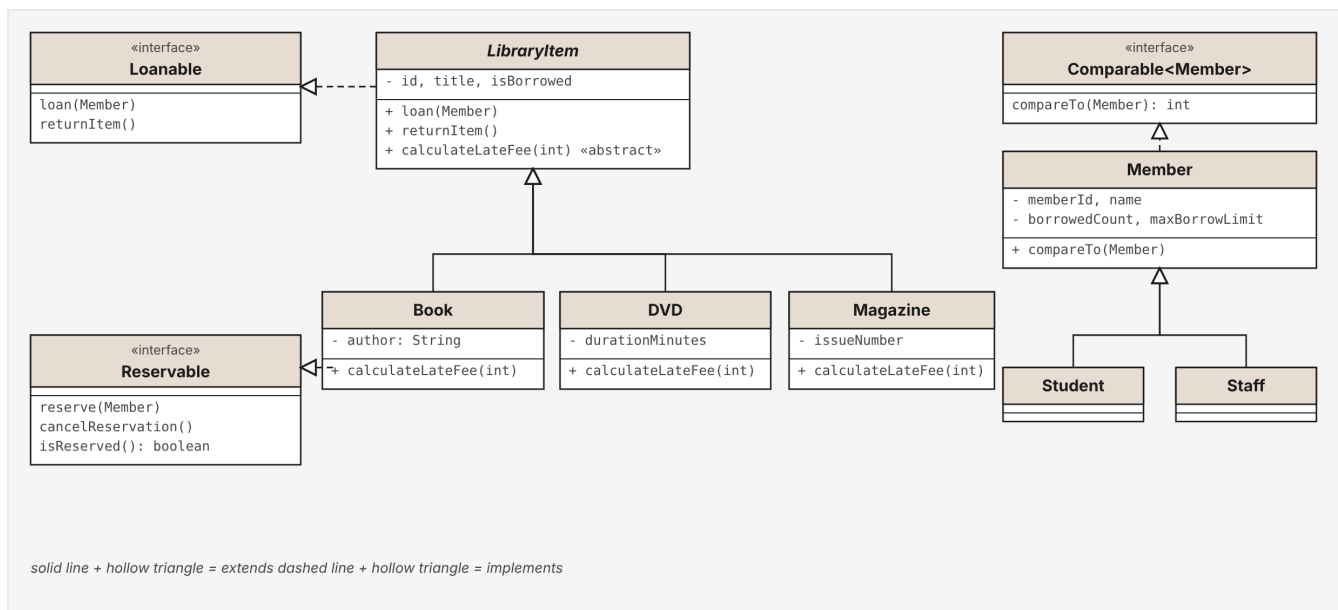
```

Book is both a LibraryItem and Reservable. A class can extend only one class but implement many interfaces.

```

public class Book extends LibraryItem implements Reservable { ... }
public class DVD extends LibraryItem { ... } // loanable, but NOT reservable

```



Now code can depend on the **ability** and not the concrete class:

```

public void lend(Loanable thing, Member member) { thing.loan(member); }

public void reserveIfPossible(LibraryItem item, Member member) {
    if (item instanceof Reservable r) r.reserve(member);
    else System.out.println(item.getTitle() + " cannot be reserved.");
}

```

If next year someone adds AudioBook implements Reservable, reserveIfPossible works **unchanged**.

3. Abstract class or interface?

	Abstract class	Interface
Relationship	is-a (a Book is a LibraryItem)	can-do (a Book can be reserved)
Fields and shared code	Yes	No (constants only)
How many	A class extends one	A class implements many
Use when	Related classes share state and code	Unrelated classes share a capability

4. Comparable: sorting your own objects

`Collections.sort(list)` can sort any `Comparable` objects. Make `Member` comparable so the most active members come first:

```
public class Member implements Comparable<Member> {
    @Override
    public int compareTo(Member other) {
        return Integer.compare(other.borrowedCount, this.borrowedCount); // bigger count first
    }
}
```

`compareTo` returns negative (this goes first), zero (equal) or positive (this goes after).

5. The Week 4 LMS in action

```
library.lend(book, nadeesha);
library.lend(book, kasun);           // already on loan
library.lend(dvd, kasun);
library.lend(mag, kasun);
library.reserveIfPossible(book, kasun);
library.reserveIfPossible(dvd, nadeesha);
library.printMembersByActivity();
```

```
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
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)
```

Warning The messages "is already on loan" are *printed*, so the caller cannot react to them. In Week 5 they become **exceptions**.

6. Lab 3 preview (Week 4)

Make `LibraryItem` abstract; add `Loanable` and `Reservable`; add `Magazine`; make `Member` comparable. Capture the compiler error for a missing abstract method. Branch `week4-abstraction-interfaces`.

7. Self-check

1. Why can you not write `new LibraryItem("X", "Y")`?
2. `DVD` is not `Reservable`. What do you change to make it reservable, and what must it now provide?
3. When would you choose an interface over an abstract class?
4. Write `compareTo` so the *least* active member sorts first.