

Introduction to Object – Oriented Programming

Why OOP, classes and objects, the four pillars (preview), UML, first Java class

3-hour session

No graded lab this week

Companion slides: L01

Leads into Lab 1

Today's agenda (3 hours)

Time	Segment
0:00 – 0:15	Welcome and course overview
0:15 – 0:45	Why object-oriented programming?
0:45 – 1:20	Classes and objects: the core building blocks
1:20 – 1:35	Break
1:35 – 2:05	The four pillars of OOP (preview)
2:05 – 2:20	Reading UML class diagrams
2:20 – 2:55	Let's code: setup and your first Java class
2:55 – 3:00	Wrap-up and what's next

1. The course in one picture

You will build **one project all semester: a Library Management System (LMS)**. Each lab week adds one new idea to it, and you submit that week as a GitHub Pull Request. By Week 12 you have a working desktop application with a database.

Week	You learn	The LMS gains
1	Intro to OOP, Git	(setup)
2	Classes, objects, encapsulation	Book, Member, Library
3	Inheritance, polymorphism	LibraryItem → Book / DVD; Member → Student / Staff
4	Abstract classes, interfaces	Loanable, Reservable, Magazine
5	Exception handling	Clear errors instead of printed messages
6	Review (no lab)	Consolidate
7	Midterm (in class)	
8	GUI I: layouts and widgets	The main window
9	GUI II: events and MVC	Working buttons
10	Generics, collections, streams	Search and reports
11	JDBC	Data saved in a database
12	Multithreading, final integration	Smooth GUI, runnable .jar, demo

Check By the end of the course you can **describe** OOP principles, **explain** why OO helps, **design, build, test and debug** OO programs, **build GUIs**, and **connect programs to a database**. Every lab maps to one of these outcomes.

2. Why object-oriented programming?

The problem with "just functions and arrays"

```
String[] titles = {"Clean Code", "1984", "Dune"};
String[] authors = {"Martin", "Orwell", "Herbert"};
boolean[] borrowed = {false, false, false};

borrowBook(titles, authors, borrowed, 1); // every function needs ALL the arrays
```

This hurts as programs grow:

- Data lives far away from the code that uses it.
- The arrays must stay in step. Delete from one and forget another, and the program is silently wrong.
- Adding a field (say `isbn`) means editing every function.
- A book is not three arrays. The code does not look like the problem.

Real things have state and behaviour

Thing	State (what it knows)	Behaviour (what it can do)
Car	colour, speed	<code>drive()</code> , <code>brake()</code>
Phone	battery level	<code>call()</code> , <code>ring()</code>
Book	title, borrowed?	<code>borrow()</code> , <code>returnBook()</code>

OOP models software the same way: each thing becomes an **object** that keeps its state *and* its behaviour together.

Object-oriented vs procedural

Aspect	Object-oriented	Procedural
Focus	Data and behaviour bundled in one object	Functions acting on separate data
Reuse	High: extend and reuse classes	Low: logic gets copied
Modelling	Natural: an object <i>is</i> the thing	Awkward: arrays stand in for things
Changing later	Change stays inside one class	A change can ripple everywhere

Advantages of OOP (Learning Outcome 2)

- **Modularity**: independent classes you can understand one at a time.
- **Reusability**: inheritance lets you extend a class instead of rewriting it.
- **Maintainability**: fix or extend one class without breaking the rest.
- **Real-world modelling**: the code mirrors the problem.
- **Teamwork**: people build different classes in parallel. This is exactly how a Pull Request workflow works.

3. Classes and objects

A **class** is a template (blueprint). An **object** is one real thing built from it, holding its own data.

Tip A blueprint describes a house, but you cannot live in a blueprint. `Book` is the blueprint. `new Book("Clean Code", ...)` builds a house.

```
public class Book {
    private String title;           // field   (state)
    private boolean isBorrowed;

    public Book(String title) {     // constructor: runs once, when the object is created
        this.title = title;
        this.isBorrowed = false;
    }

    public void borrow() {          // method  (behaviour)
        isBorrowed = true;
    }
}
```

Many objects, one class: each object has **its own copy** of the fields.

```
Book b1 = new Book("Clean Code");
Book b2 = new Book("1984");
b1.borrow();           // only b1 changes. b2 is untouched.
```

4. The four pillars (preview)

Each pillar gets its own week. Today you only need the name and one picture.

Pillar	One-line idea	Analogy	Week
Encapsulation	Keep data private. Expose safe methods.	TV remote: buttons yes, circuit board no	2
Inheritance	A new class reuses and extends an old one.	A Sports Car <i>is a</i> Car	3
Polymorphism	Same request, each object answers in its own way.	Press "Play": song, podcast or audiobook	3
Abstraction	Show what matters, hide how it works.	Drive a car without understanding the engine	4

5. Reading UML class diagrams

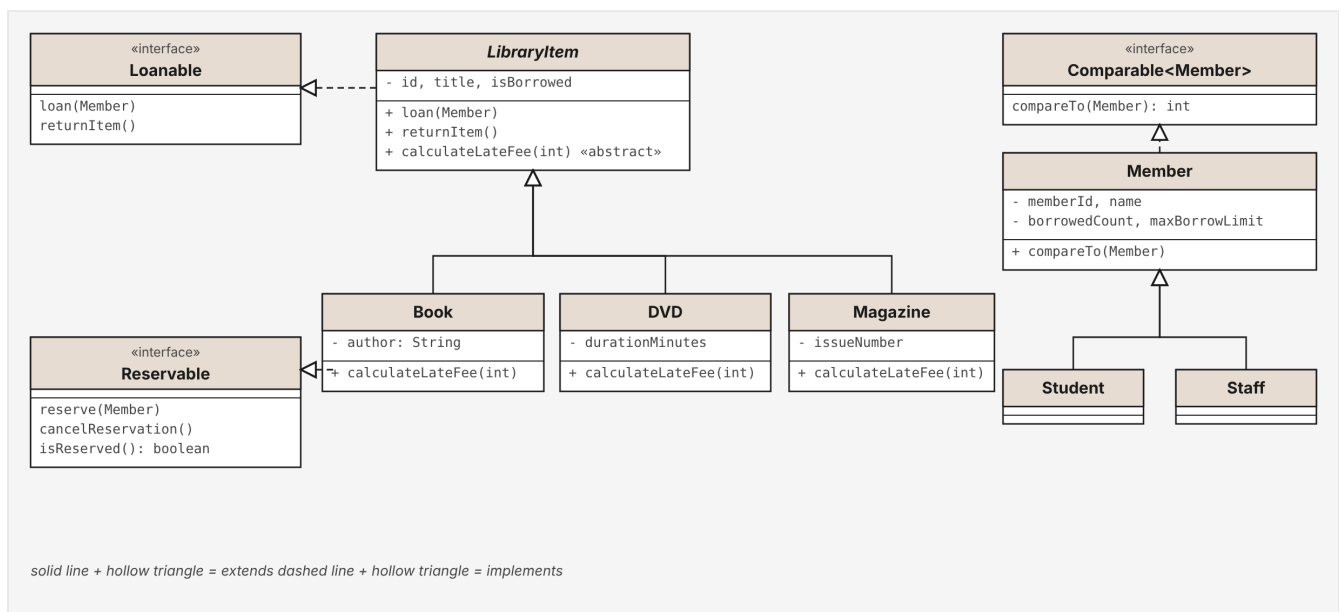
A class box has three parts: **name**, **fields**, **methods**. `-` means private, `+` means public.

```

+-----+
|           Book           |
+-----+
| - title : String         |
| - author : String        |
| - isBorrowed : boolean  |
+-----+
| + borrow() : void        |
| + returnBook() : void    |
+-----+

```

You will sketch a small UML diagram **at the start of every lab, before writing Java**. It takes two minutes and saves long redesigns. By Week 4 the LMS looks like this:



6. Set up your tools (before Lab 1)

- JDK 21** (Adoptium Temurin). Check with `java -version` and `javac -version`.
- An IDE**. Pick one and keep it all semester: IntelliJ IDEA Community, Eclipse, or VS Code with the Extension Pack for Java.

3. **Git.** Check with `git --version`. Then read the **Git Support Guide** and complete its "Before Lab 1" checklist.

7. Your first Java class: live coding

Create `Book.java`:

```
public class Book {
    private String title;
    private String author;
    private boolean isBorrowed;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isBorrowed = false;
    }

    public void borrow() {
        if (!isBorrowed) {
            isBorrowed = true;
            System.out.println(title + " has been borrowed.");
        } else {
            System.out.println(title + " is already borrowed.");
        }
    }

    public void returnBook() {
        isBorrowed = false;
        System.out.println(title + " has been returned.");
    }

    public void display() {
        System.out.println(title + " by " + author + (isBorrowed ? " [borrowed]" : " [available]"));
    }
}
```

Create `Main.java` in the same folder:

```
public class Main {
    public static void main(String[] args) {
        Book b1 = new Book("Clean Code", "Robert C. Martin");
        Book b2 = new Book("Effective Java", "Joshua Bloch");

        b1.display();
        b1.borrow();
        b1.borrow(); // second time: already borrowed
        b1.display();
        b2.display(); // b2 is a different object: still available
    }
}
```

Compile and run:

```
javac Book.java Main.java
java Main
```

```
Clean Code by Robert C. Martin [available]
Clean Code has been borrowed.
Clean Code is already borrowed.
Clean Code by Robert C. Martin [borrowed]
Effective Java by Joshua Bloch [available]
```

Check Can you explain why `b2` stayed *available*? (Each object has its own `isBorrowed`.)

Try it yourself (10 minutes)

1. Add a `isbn` field. Which lines must change? (Only `Book` — the point of OOP.)
2. Add `isAvailable()` that returns a `boolean`.
3. Create a third book and borrow it. Predict the output before you run.
4. Break something on purpose: remove `this.` in the constructor. What happens to `title`?

8. Glossary

Term	Meaning
Class	Template that defines fields and methods
Object	One instance built from a class
Field / attribute	A variable stored inside an object (state)
Method	A function stored inside a class (behaviour)
Constructor	Special method that creates an object
<code>this</code>	The object you are currently inside
UML	A drawing language for software design

9. Homework before Lab 1

- ☐ Install and verify JDK, IDE and Git
- ☐ Create a GitHub account and a **private** repository `lms-<index>`; add your lecturer and TA
- ☐ Push `.gitignore` and `README.md` to `main`
- ☐ Practise the weekly routine once on a throw-away branch
- ☐ Re-type the `Book` and `Main` classes from memory. Do not copy-paste.