Object-Oriented Programming in Java
Summary Lab: Classes, Library, Inheritance, Polymorphism

Zbigniew Dendzik
Institute of Physics, University of Silesia

2026

This lab summarises and consolidates the material from the first four labs: basic classes and objects, mathematical examples, using Java library classes and documentation, inheritance, interfaces and polymorphism. The exercises below are new but analogous in difficulty and structure to the original ones.

Part A: Classes and Objects – Geometry and Physics

Ex. S.1 – Triangle and Point3D

Write implementations of the classes `Point3D` and `Triangle3D`.

- Class `Point3D` should contain:
  - fields: `double x, y, z;`
  - constructor: `Point3D(double x, double y, double z);`
  - method: `public String toString()` returning coordinates.
- Class `Triangle3D` should contain:
  - fields: three vertices `Point3D a, b, c;`
  - constructor taking three vertices,
  - method `double perimeter()` computing the perimeter in 3D,
  - method `double area()` computing the area using Heron's formula in 3D (lengths of sides computed from distances between points).
- Write a program that creates several different triangles and prints their perimeters and areas.

Ex. S.2 – Moving Bodies in 2D

Write a class `Vector2D` and a class `RigidBody2D`.

- `Vector2D`:
  - fields: `double x, y;`
  - constructors, `toString()`, methods `add(Vector2D other)`, `scale(double k)`.
- `RigidBody2D`:
  - fields: `Vector2D position;` and `Vector2D velocity;`,
  - constructor taking initial position and velocity,
  - method `void move(double dt)` that updates position according to simple formula position = position + velocity $\cdot\, dt$,
  - method `public String toString()`.
- In the program simulate the motion of one body for several time steps (e.g. `dt = 0.5`) and print successive positions.

Ex. S.3 – Polygon and Contains Test

Extend your geometric model with a class `Polygon2D`.

- Store vertices in an array or list: `Point[] vertices;` or `List<Point2D>`.
- Implement a method `double perimeter()` summing distances between consecutive vertices (including last to first).
- Implement a method `boolean contains(Point2D p)` using any known algorithm for point-in-polygon test in 2D (ray casting, winding number or simplified version).
- Test your implementation on a simple convex polygon (e.g. regular pentagon) and several points inside and outside.

Part B: Algebraic Classes – Fractions, Complex Numbers, Vectors

Ex. S.4 – RationalFunction: fraction of polynomials

Define a class `QuadraticPolynomial` representing a polynomial $ax^2 + bx + c$ with:

- fields: `double a, b, c;`
- methods: `double valueAt(double x)`, `QuadraticPolynomial add(QuadraticPolynomial other)`, `QuadraticPolynomial multiply(QuadraticPolynomial other)`.

Then define a class `RationalFunction` representing a fraction of two quadratic polynomials:

```
class RationalFunction {
    private QuadraticPolynomial numerator;
    private QuadraticPolynomial denominator;

    // constructors and methods...
}
```

- Implement a constructor taking numerator and denominator.
- Implement a method `double valueAt(double x)` computing $\frac{P(x)}{Q(x)}$ (remember about division by zero).
- Implement a method `String toString()` returning a readable representation of the fraction.
- Write a program that creates two rational functions and prints their values for several x.

Ex. S.5 – ComplexMatrix 2x2

Write the definition of a `ComplexNumber` class (real and imaginary parts, addition, multiplication). Then define a class `ComplexMatrix2x2` representing 2x2 matrices of complex numbers.

- Store entries as four `ComplexNumber` fields or as a 2x2 array.
- Implement methods:
  - `ComplexMatrix2x2 add(ComplexMatrix2x2 other)`,

  - `ComplexMatrix2x2 multiply(ComplexMatrix2x2 other)`,
  - `ComplexNumber determinant()`.
- Write a test program creating two complex matrices, printing their sum, product and determinants.

Ex. S.6 – Vector3D Operations and Projections

Write a class `Vector3D` with fields `double x, y, z;` and implementations of:

- vector addition, scalar multiplication,
- dot product, cross product,
- method `double length()`,
- method `Vector3D projectionOn(Vector3D other)` returning the projection of one vector on another.

Write a program that:

- creates two non-orthogonal vectors,
- prints their dot product and cross product,
- prints the length of the projection of one vector on the other.

Part C: Using Java Library Classes and I/O

Ex. S.7 – java.time.LocalDate and Period

Using Java library classes from the `java.time` package write a program that simulates a simple "subscription" management system.

- Define a class `Subscription` with fields:
  - `String userName;`
  - `java.time.LocalDate startDate;`
  - `java.time.Period duration;`
- Implement a method `LocalDate getEndDate()` returning the end date.
- Implement a method `boolean isActiveOn(LocalDate date)`.
- Write a program that creates several subscriptions (different durations) and checks which are active on today's date.

Ex. S.8 – Text File: Simple Log of Measurements

Write a class `Measurement` representing a physical measurement (e.g. temperature).

- Fields: `LocalDateTime timestamp;` (from `java.time`), `double value;`
- Method `String toCSV()` returning a line in format: `2026-03-01T12:30;23.5`

Write a program that:

- reads a few measurement values from the keyboard (using `BufferedReader`),
- saves them to a text file as CSV (one line per measurement),
- then reads the file back and prints all measurements to the console.

Ex. S.9 – Using an External Library (or Math Library)

Write a small program demonstrating the use of classes from an external or advanced library.

- Option A: Use `java.security.MessageDigest` to compute and print the SHA-256 hash of a text typed from the keyboard.
- Option B: Use any external library (e.g. Apache Commons Math) to solve a small numerical problem (root finding, interpolation, random distributions, etc.).
- Describe in comments:
  - which classes from the library you used,
  - what problem they solve,
  - how to add the library to the project (short note).

Part D: Inheritance, Interfaces and Polymorphism

Ex. S.10 – Vehicle Hierarchy

Design a simple class hierarchy for different types of vehicles.

- Abstract base class `Vehicle`:
  - fields: `String brand;` and `double maxSpeed;`
  - abstract method `double range()` returning estimated travel range in km,
  - method `void info()` printing basic information about the vehicle.
- Derived classes:
  - `Car` with additional field `double fuelConsumption;` (litres/100km),
  - `ElectricScooter` with field `double batteryCapacity;` (kWh) and `double consumptionPerKm;` (kWh/km),
  - each overrides `range()` using a simple formula.
- Write a program that puts several different `Vehicle` objects in an array and prints their info and ranges using polymorphism.

Ex. S.11 – Interface Filterable and File Search

Define an interface `Filterable` with one method:

```
interface Filterable {
    boolean matches(String pattern);
}
```

Then design the following classes:

- `AudioFile` with fields: `String title, artist;` and `int durationSeconds;`,
- `VideoFile` with fields: `String title, director;` and `int lengthMinutes;`,
- both implement `Filterable` and consider a match if the pattern appears (case-insensitive) in the title or creator field.

- Write a class `MediaLibrary` holding an array of `Filterable` and a method:
  - `void search(String pattern)` printing all matching objects.
- Test the search using different patterns (artist name, director name, part of the title).

Ex. S.12 – Serializable Contacts and Persistence

Create a small "contacts" application using Java serialization.

- Define a class `Contact` implementing `java.io.Serializable` with fields:
  - `String name, email;`
  - `String phone;`
- Define a class `ContactBook` with:
  - an internal `List<Contact> contacts;`
  - methods to add a contact, list all contacts,
  - methods `void saveToFile(String fileName)` and `static ContactBook loadFromFile(String fileName)` using object streams.
- Write a program that:
  - creates a `ContactBook`, adds a few contacts (read from keyboard),
  - saves them to a file,
  - then loads them from the file and prints all contacts.
  - Handle appropriately exceptions when file cannot be opened.

Optional project:

- Design your own small application combining:
  - at least one non-trivial class hierarchy with inheritance and polymorphism,
  - at least one interface with multiple implementations,
  - at least one use of Java library classes (I/O, time, math, collections).
- Briefly document class relationships (e.g. in comments) and main responsibilities.