Lab: Object-Oriented Design — Training Company App

Duration: 2 hours
Language: Java or C#

Learning Objectives

By the end of this lab, you will be able to:

1. Design and implement multiple related classes that represent real-world objects.
2. Store and manage groups of objects using arrays or lists.
3. Write methods that access and manipulate objects in arrays.
4. Understand how classes can work together through composition and method calls.

Scenario

You work for a **training company** that runs a variety of **courses** taught by different **trainers**.

Each **course**:

- Has a **title** (e.g., *Java Programming*),
- A **duration** in days,
- And a set of **skill**s required to teach it.

Each **trainer**:

- Has a **name**,
- And a set of **skill**s they can teach.

Your task is to build a simple OO system that models this company and allows you to:

- Store trainers and courses,
- Display all available trainers and courses,
- Find which trainers are qualified to run a given course.

Step 1: Identify the Classes

Think about which classes you need to represent this system.
At a minimum, you'll need:

- **Skill** – represents a single teaching skill.
- **Trainer** – represents a person who can deliver training.
- **Course** – represents a training course that requires certain skills.
- **TrainingCompany** – stores lists of trainers and courses and provides useful methods.

Task:

1. Write down what data (fields) each class should have.
2. Write down what each class should be able to *do* (methods).

Step 2: Design the Relationships

Decide how the classes interact:

- A Trainer *has* several Skill objects.
- A Course *requires* several Skill objects.

- A TrainingCompany *contains* an array or list of Trainer and Course objects.

Task:
Draw a small UML diagram (boxes and arrows) to show the relationships between classes.

Step 3: Create and Populate the Objects

Create a few example objects to test your design:

- 3–4 skills (e.g., Java, SQL, HTML, Python).
- 3 trainers, each with different skills.
- 3 courses, each requiring one or more skills.

Store them in arrays or lists inside the TrainingCompany class.

Task:
Plan what data you'll use for each object.

Step 4: Display Trainers and Courses

Write methods in your TrainingCompany class to:

1. List all trainers with their skills.
2. List all courses with their required skills.

Task:
Plan what each method should print to the console.

Step 5: Match Trainers to Courses

Add a method that:

- Takes a course title as input,
- Searches for that course,
- Then finds all trainers who have *all* the skills required to run it.

Hint:
You'll need nested loops — one to go through trainers and another to compare their skills with the course's required skills.

Discussion & Reflection

After testing your program:

- How do the classes relate to each other?
- Which OO principles have you used (encapsulation, composition, etc.)?
- What would change if you added new skills, trainers, or courses?
- How could you make the system easier to expand in the future?

Optional Extensions

1. Add a Booking class to link a course, trainer, and date.
2. Add methods to assign a trainer to a course automatically.
3. Allow the user to enter new courses or trainers during runtime.
4. Print a simple "training schedule" report.