Lesson 10: Collections in Java

Marcin Kurzyna

Lecture Goals

By the end of this laboratory session, students should be able to:

- Understand the purpose of the **Java Collections Framework**.
- Distinguish between main collection interfaces.
- Use `List`, `Set`, and `Map` implementations.
- Iterate over collections using different techniques.
- Choose appropriate collection types for given problems.

1 Introduction to Java Collections

The **Java Collections Framework (JCF)** provides a unified architecture for storing and manipulating groups of objects.

Benefits of collections:

- Dynamic size (unlike arrays),
- Built-in algorithms (sorting, searching),
- Improved code readability and reuse.

2 Main Collection Interfaces

The core interfaces in the Java Collections Framework are:

- **List** – ordered collection, allows duplicates,
- **Set** – unordered collection, no duplicates,
- **Map** – key-value pairs (not a subtype of Collection).

3 List Interface

A **List** maintains insertion order and allows duplicate elements.

Common implementations:

- `ArrayList`
- `LinkedList`

Example: Using ArrayList

```
import java.util.ArrayList;
import java.util.List;

List<String> names = new ArrayList<>();
names.add("Alice");
names.add("Bob");
names.add("Alice");

System.out.println(names);
```

4 Set Interface

A **Set** does not allow duplicate elements.
Common implementations:

- `HashSet`
- `TreeSet`

Example: Using HashSet

```
import java.util.HashSet;
import java.util.Set;

Set<String> names = new HashSet<>();
names.add("Alice");
names.add("Bob");
names.add("Alice");

System.out.println(names);
```

5 Map Interface

A **Map** stores data as key-value pairs.
Common implementations:

- `HashMap`
- `TreeMap`

Example: Using HashMap

```
import java.util.HashMap;
import java.util.Map;

Map<String, Integer> scores = new HashMap<>();
scores.put("Alice", 85);
```

```
scores.put("Bob", 90);

System.out.println(scores.get("Alice"));
```

6 Iterating Over Collections

There are multiple ways to iterate over collections:

- Enhanced `for` loop,
- `Iterator`,
- `forEach` with lambda expressions.

Example: forEach with Lambda

```
names.forEach(name -> System.out.println(name));
```

7 Collections Utility Class

The `Collections` class provides static utility methods such as:

- `sort()`
- `reverse()`
- `max()`, `min()`

Example: Sorting a List

```
import java.util.Collections;

Collections.sort(names);
System.out.println(names);
```

Summary

In this laboratory you learned:

- The structure of the Java Collections Framework.
- Differences between `List`, `Set`, and `Map`.
- How to add, retrieve, and iterate over collection elements.
- How to use utility methods from the `Collections` class.

8 Exercises

1. Create a `List` of integers and compute the average value.
2. Use a `Set` to remove duplicates from a list of strings.
3. Create a `Map` storing student names and grades.
4. Iterate over a map and print all keys and values.
5. (Challenge) Compare performance of `ArrayList` and `LinkedList`.