

Generics, Collections and Streams

One Repository for everything, Maps, Optional, and streams for search and reports

Lecture + Lab 7

Branch: week10-generics-streams

Learning outcome 3

What you will be able to do after today

- Write and use a **generic class** `Repository<T>`.
- Choose between `List`, `Set` and `Map`.
- Use `Optional` instead of `null`.
- Filter, sort and group with **streams** and **lambdas**.

1. The problem: two lists, same code

The LMS keeps items **and** members. Both need *add*, *find by id*, *list all*. Writing `ItemList` and `MemberList` would be copy-paste. And a plain `List` of `Object` forces casts, and casts crash at run time.

A **generic class** is written once with a **type parameter** `T` ("some type, chosen by the user of the class").

```
public class Repository<T> {
    private final Map<String, T> store = new LinkedHashMap<>();
    private final Function<T, String> idOf; // how to get an id out of a T

    public Repository(Function<T, String> idOf) { this.idOf = idOf; }

    public void add(T entity) {
        String id = idOf.apply(entity);
        if (store.containsKey(id)) throw new IllegalArgumentException("An entry with id '" + id + "'
already exists");
        store.put(id, entity);
    }

    public Optional<T> findById(String id) { return Optional.ofNullable(store.get(id)); }
    public boolean exists(String id) { return store.containsKey(id); }
    public List<T> findAll() { return new ArrayList<>(store.values()); }
    public List<T> search(Predicate<T> condition) {
        return store.values().stream().filter(condition).toList();
    }
}
```

Using it:

```
Repository<LibraryItem> items = new Repository<>(LibraryItem::getId);
Repository<Member> members = new Repository<>(Member::getMemberId);

items.add(new Book("B001", "Clean Code", "Robert C. Martin"));
// items.add(new Student(...)); // compile error: wrong type. Caught BEFORE running.
```

Tip Generics move errors from **run time** to **compile time**, which is always cheaper.

2. Choosing a collection

Collection	Keeps	Duplicates	Fast at	LMS use
List (ArrayList)	Order, by position	Yes	Looping, <code>get(i)</code>	Results to show in the table
Set (HashSet)	No order guarantee	No	"Is it in there?"	Unique tags or ids
Map (HashMap, LinkedHashMap)	Key → value	Keys unique	Lookup by key	Items and members by id

A `Map` finds an item by id without scanning the whole list, and `LinkedHashMap` keeps insertion order so the table looks the same every time.

3. Optional: "maybe nothing"

```
public Member findMember(String memberId) {
    return members.findById(memberId)
        .orElseThrow(() -> new MemberNotFoundException(memberId));
}
```

`Optional` forces the caller to decide what to do when nothing is found, instead of crashing later with `NullPointerException`.

4. Streams: say *what*, not *how*

```
public List<LibraryItem> search(String text) {
    String q = text.trim().toLowerCase();
    return items.search(i -> i.getTitle().toLowerCase().contains(q)
        || i.getId().toLowerCase().contains(q)
        || (i instanceof Book b && b.getAuthor().toLowerCase().contains(q)));
}

public Map<String, Long> loansPerMember() {
    return items.findAll().stream()
        .filter(LibraryItem::isBorrowed)
        .collect(Collectors.groupingBy(i -> i.getBorrower().getName(), TreeMap::new,
            Collectors.counting()));
}

public List<LibraryItem> availableItems() {
    return items.findAll().stream()
        .filter(i -> !i.isBorrowed())
        .sorted(Comparator.comparing(LibraryItem::getTitle))
        .toList();
}
```

Read a stream like a sentence: *"take the items, keep the borrowed ones, group by borrower's name, count each group."*

Stream step	Meaning
<code>filter(cond)</code>	keep matching elements
<code>map(fn)</code>	transform each element
<code>sorted(cmp)</code>	order them
<code>collect(...)</code> / <code>toList()</code>	gather results
<code>count()</code> , <code>anyMatch(...)</code>	summarise

A **lambda** `i -> !i.isBorrowed()` is a tiny anonymous function. `LibraryItem::getTitle` (a *method reference*) is shorthand for `i -> i.getTitle()`.

5. Result in the GUI

Library Management System

Search:

New item

Type:

ID:

Title:

Detail:

Add Item

ID	Title	Type	Detail	Status
B001	Clean Code	Book	Robert C. Martin	On loan: Nadeesha Perera
B002	Refactoring	Book	Martin Fowler	Available
D001	Introduction to Algorithms	DVD	120 min	On loan: Mr. Kasun Silva
G001		e	Issue 42	Available

i
Items currently on loan:
Mr. Kasun Silva: 1
Nadeesha Perera: 1

Available: 2

OK

D001 borrowed by M002

Member:

Library Management System

Search:

New item

Type:

ID:

Title:

Detail:

Add Item

ID	Title	Type	Detail	Status
B001	Clean Code	Book	Robert C. Martin	On loan: Nadeesha Perera
B002	Refactoring	Book	Martin Fowler	Available

D001 borrowed by M002

Member:

6. Lab 7 preview (Week 10)

Create `Repository<T>`; make `Library` use it for items and members; add `search`, `loansPerMember` and `availableItems`; add a search box and a "Loan Report" button. Write one extra stream query of your own. Branch `week10-generics-streams`.

7. Self-check

1. What does `Repository<Member>` forbid that `Repository<Object>` would allow?
2. When is a `Map` better than a `List`?
3. Rewrite as a stream: *"titles of all borrowed items, sorted"*.
4. What would `Collectors.counting()` count in `loansPerMember()`?