# MODULE: Java Programming

## ASSIGNMENT 8 – Java 8 Streams, Collectors & Functional Processing

## Learning Objectives

By completing this assignment, learners will:

- Understand the fundamentals of Java Streams
- Apply Stream operations such as **filter**, **map**, **sorted**, **distinct**, **collect**, **reduce**
- Work with `Collectors` for grouping, averaging, counting and mapping
- Apply stream pipelines on complex objects (Book → Author → Country)
- Use Stream APIs to perform analytics-style transformations
- Understand lazy evaluation and pipeline design

## General Instructions

1. Use the **exact Book, Author, and Subject classes provided** in the sample code.
2. Add all required:
   - constructors
   - getters and setters
   - toString() for Book and Author
3. Use **Streams API only** for the tasks (no loops unless explicitly required).
4. Do not change the data loaded in `loadAllBooks()`.
5. Each requirement must be implemented using a separate Stream pipeline.
6. Print clear, readable outputs for each task.

## Estimated Time & Difficulty

| Task Range | Time | Difficulty |
|---|---|---|
| Basic filtering & mapping | 20–30 min | Beginner |
| Sorting, slicing, reducing | 30–45 min | Intermediate |
| Grouping, advanced collectors | 45–60 min | Intermediate → Advanced |

## Evaluation Rubric

| Criteria | Weight |
|---|---|
| Correct Stream usage | 40% |
| Proper use of Collectors | 25% |
| Code readability | 15% |
| Output correctness | 10% |
| Completion of all tasks | 10% |

## Starter Code (Given)

You must keep this structure exactly the same:

```
public class CopyOfDemoBookCaseStudyProblem {

        public static void main(String[] args) {

                List<Book> allBooks = loadAllBooks();

                // 1. Find books with more than 400 pages

                // 2. Find all books that are Java books and more than 400 pages

                // 3. We need the top three longest books

                // 4. We need from the fourth to the last longest books

                // 5. We need to get all the publishing years

                // 6. We need all the authors' names who have written a book

                // 7. We need all the origin countries of the authors

                // 8. We want the most recent published book.

                // 9. We want to know if all the books are written by more than one
author

                // 10. We want one of the books written by more than one author.
(findAny)

                // 11. We want the total number of pages published.

                // 12. We want to know how many pages the longest book has.

                // 13. We want the average number of pages of the books

                // 14. We want all the titles of the books

                // 15. We want the book with the highest number of authors

                // 16. We want a Map of <year, list of books>

                // 17. We want to count how many books are published per year.
        }

        private static List<Book> loadAllBooks() {
                List<Book> books = new ArrayList<Book>();
                List<Author> authors1 = Arrays.asList(new Author("raj", "gupta", "in"),
                                new Author("ekta", "gupta", "in"));

                List<Author> authors2 = Arrays.asList(new Author("raj", "gupta", "in"));
                List<Author> authors3 = Arrays.asList(new Author("gunika", "gupta",
"us"),
                                new Author("keshav", "gupta", "us"));

                books.add(new Book("java", authors1, 400, Subject.JAVA, 2000, "1213"));
                books.add(new Book("python", authors2, 479, Subject.JAVA, 2007, "1218"));
                books.add(new Book("Mgt", authors3, 600, Subject.DOT_NET, 2000, "1293"));

                return books;
        }
}

class Author {
        private String name;
        private String lastname;
        private String country;
}

class Book {
        private String title;
        private List<Author> authors;
        private int pages;
        private Subject subject;
        private int year;
        private String isbn;
}

enum Subject {
        JAVA, DOT_NET, ORACLE;
}
```

**NOTE:**

You MUST add **constructor, getters, setters, and toString()** for both `Book` and `Author`.

## LAB TASKS — Implement Each Stream Operation Below

### 1. Find all books with more than 400 pages

`filter() + collect()`

### 2. Find all books that are Java books *and* more than 400 pages

`filter(b → b.getSubject() == Subject.JAVA && b.getPages() > 400)`

### 3. Get the top 3 longest books (by pages)

`sorted(reverse order by pages).limit(3)`

### 4. Get books ranked from 4th position to last (skip first 3)

`skip(3).collect(...)`

### 5. Extract all publishing years (List<Integer>)

`map(Book::getYear)`

### 6. Get the names of ALL authors who have written any book

Hint:
`flatMap(book → book.getAuthors().stream()).map(Author::getName)`

Use `distinct()`.

### 7. Get all origin countries of authors (unique list)

`flatMap → map → distinct()`

---

### 8. Find the most recently published book

Use:

- `max(Comparator.comparing(Book::getYear))`
  OR
- `sorted(reverse).findFirst()`

### 9. Check if all books are written by more than one author

`allMatch(book → book.getAuthors().size() > 1)`

### 10. Get ANY one book written by more than one author

`filter(condition).findAny()`

### 11. Compute the total number of pages across all books

`map(Book::getPages).reduce(0, Integer::sum)`

### 12. Find how many pages the longest book has

`map(Book::getPages).max(...)`

### 13. Compute the average number of pages

`Collectors.averagingInt(Book::getPages)`

### 14. Get all titles of all books

`map(Book::getTitle).collect(toList())`

### 15. Find the book with the highest number of authors

`max(Comparator.comparing(b → b.getAuthors().size()))`

### 16. Create a Map of `<year, List<Book>>`

`Collectors.groupingBy(Book::getYear)`

### 17. Count how many books are published per year

`Collectors.groupingBy(Book::getYear, Collectors.counting())`

## BONUS CHALLENGES (Optional)

### ⭐ Bonus 1 – Sort authors alphabetically across all books

`flatMap → sorted by lastname then firstname`

### ⭐ Bonus 2 – Create a Map: `<country, List<Author>>`

Group authors by origin country.

### ⭐ Bonus 3 – Find total authors involved in all books (distinct unique people)

### ⭐ Bonus 4 – Create a Map: `<Subject, total pages>`

`groupingBy(subject, summingInt(pages))`

## Reflection Questions

1. How does `flatMap()` help when dealing with nested lists (Book → Author)?
2. Why is `distinct()` important when collecting author information?
3. Compare `reduce()` vs Stream mathematical collectors (sum, avg).
4. Why must sorting come before `limit()` or `skip()`?
5. How did using Stream API improve readability over loops?