

# Homework 1: Basic Data Structures and Built-Ins

CS 150 with Dr. Sam Schwartz

Due: Sunday, February 8 at 11:59pm via Canvas Upload

## 1 Purpose

In this homework assignment you will gain experience with creating some of your own data structures, practicing with Java's built-in collections (including ArrayList, LinkedList, HashSet, HashMap), and applying recursion.

## 2 Tasks

### Part A – Build a Custom Linked Data Structure

Implement a singly-linked list of integers called `IntSinglyLinkedList`. You may not use `ArrayList`, `LinkedList`, or any built-in collection to store the elements internally.

1. Create a class `Node` with fields:

- `int value`
- `Node next`

2. In `IntSinglyLinkedList`, maintain:

- a reference to the head node
- an `int size`

3. Implement the following methods:

- `public void addFirst(int value)`
- `public void addLast(int value)`
- `public int get(int index)` (throw `IndexOutOfBoundsException`)
- `public int removeFirst()` (throw `NoSuchElementException`)
- `public boolean removeValue(int value)` (remove first match; return true if removed)
- `public int size()`
- `public boolean isEmpty()`
- `public String toString()` (format example: `[3, 8, 2]`)

Finally, write a short "demo" snippet in `main(String[] args)` which demonstrates the use of your `IntSinglyLinkedList` and instruments it with `System.out.println()` statements. Take a screenshot of it.

## Part B – Create a Small “Library” Using Java Collections

You will build a tiny library system that stores “books” and “borrowers.” You will use multiple built-in collections and practice choosing the right data structure for the job.

1. Create a class `Book` with:

- fields: `String isbn`, `String title`, `String author`
- a constructor that initializes all fields (validate that strings are not null nor blank)
- getters (no public setters)
- `toString`
- `equals` and `hashCode` based on `isbn` only

2. Create a class `Borrower` with:

- fields: `String id`, `String name`
- constructor + appropriate getters and setters
- `equals` and `hashCode` based on `id` only

3. Create a class `Library` that uses all of the following internally:

- `ArrayList<Book>` to store the full catalog in insertion order
- `HashMap<String, Book>` mapping `isbn` -> `Book` for fast lookup
- `HashSet<Book>` to track which books are currently available (in-library)
- `HashMap<Borrower, LinkedList<Book>>` mapping each borrower to a checkout queue/history

4. Implement the following `Library` methods:

- `public void addBook(Book book)` (no duplicates by ISBN)
- `public Book findByIsbn(String isbn)` (return null if not found)
- `public boolean isAvailable(String isbn)`
- `public void registerBorrower(Borrower b)` (id-based uniqueness)
- `public boolean checkout(String borrowerId, String isbn)`
- `public boolean checkin(String isbn)`
- `public LinkedList<Book> getBorrowerBooks(String borrowerId)` (return a copy)

5. Error handling requirements:

- If `checkout` is called with an unknown borrower or unknown ISBN, throw `IllegalArgumentException`.
- If `checkin` is called with an unknown ISBN, throw `IllegalArgumentException`.
- If `checkout` is attempted for a book that is not available, return `false`.

6. Write short comments (2–4 sentences) inside your code explaining why each collection choice makes sense (why `HashMap` here, why `HashSet` here, etc.).

Finally, write a short “demo” snippet in `main(String[] args)` which demonstrates the use of your library and instruments it with `System.out.println()` statements. Take a screenshot of it.

### **Part C — Submit to Canvas**

Upload the following five files to Canvas:

- `code.zip` which contains all of your code
- `screenshot-a.png`, or a similar filename, that demonstrates that the code you wrote in Task A is working as specified via a screenshot of your IDE/console.
- `screenshot-b.png`, or a similar filename, that demonstrates that the code you wrote in Task B is working as specified via a screenshot of your IDE/console.

## **3 Grading Criteria**

In general I am looking for the elements of validity, readability, and fluency in all code-based assignments. (See more below.)

I tend to dock 5ish points off for each error (although smaller or larger quantities like -1pt or -10pts exist based on the magnitude of the error), and will provide free-form feedback detailing why any points were missed in the comments on Canvas.

Please note that I do not get notifications about replies to my comments on Canvas, so if you have any questions please reach out to me directly.

### **Validity**

Student submitted files which implemented the tasks correctly and handled possible errors gracefully.

Moreover, the student submitted screenshot(s) displaying the code's output.

### **Readability**

The deliverable used professional English and typesetting throughout, and had all methods annotated with JavaDoc.

### **Fluency**

The deliverable was executed in such a way that an experienced practitioner would not find the deliverable "weird-in-a-bad-way" or unduly jarring.