Chapter

23 Java

23.1 Core language

Programs are written in Java 1.7. Most programs are compatible with earlier versions—some Java 1.7 constructs that we use are the `Objects` utility class, which eases writing hash functions and comparators, the diamond operator (<>), which reduces the verbosity of declaring and instantiating variables, and binary literals and underscores, which make integral constants easier to read, e.g., `0b001_00001`.

Usually we declare helper classes as static inner classes of the top-level class that provides the solution. You should be comfortable with the syntax used to instantiate an anonymous class, e.g., the comparator object in Solution 22.17 on Page 426.

Our test code lives within main method of the class. We also have a Junit test suite, which wraps the main methods.

23.2 Libraries and constructs

We have elected not to use popular third-party libraries, like Apache Commons, or Guava—every program uses the JDK, and nothing else. We have a very small number of utility functions that we wrote, largely for test code, e.g., fill and print methods.

The JDK container classes and interfaces that we use, in order of decreasing frequency are given in Table 23.1. You are strongly encouraged to review their Javadoc. The table does not include the most basic library types and methods, e.g., `java.util.Math` and `java.io`, which we also use.

Random	List	ArrayList	Arrays	Collections
LinkedList	Set	Map	HashSet	HashMap
Deque	Objects	Comparator	PriorityQueue	TreeSet
Queue	Iterator	SortedSet	TreeMap	LinkedHashMap
BigInteger	NavigableSet	NavigableMap	BitSet	

Table 23.1: JDK classes and interfaces used in EPI.

ListNode Stack	DoublyListNode Queue	BinaryTreeNode BSTNode	PostingListNode

Table 23.2: EPI container classes.

Wherever possible, you should use the standard library for basic containers. Some problems explicitly require you to write your own container classes. The container classes we implemented are described in Table 23.2. (Note that Queue appears in both the JDK and EPI lists—this is because in some cases we are required to implement a queue, in other cases we can use an implementation of the Queue interface in JDK.)

23.3 Best Practices for Interview Code

First we describe practices we use in EPI that are not suitable for production code. They are necessitated by the finite time constraints of an interview. See Section 2 on Page 13 for more insights.

- We make fields public, rather than use getters and setters.
- We do not protect against invalid inputs, e.g., null references, negative entries in an array that's supposed to be all nonnegative, input streams that contain objects that are not of the expected type, etc.
- We occasionally use static fields to pass values—this reduces the number of classes we need to write, at the cost of losing thread safety.
- We use IO-exceptions to detect the end of a stream, rather than encoding the number of objects at the start of the stream, or a sentinel object.
- Since we want programs to be standalone for ease of readability, we have duplicate code in our codebase. (This is not a hard rule—we use the program for computing the kth smallest element in an array 12.9 on Page 194 in the program for computing the elements closest to the median 22.12 on Page 416. We also reuse custom container classes.)

There are some practices we follow in EPI which are industry-standard, but we would not recommend for an interview.

- We use long identifiers for pedagogy, e.g., `queueOfMaximalUsers`. In an actual interview, you should use shorter, less descriptive names than we have in our programs—writing `queueOfMaximalUsers` repeatedly is very time-consuming compared to writing `q`.
- We follow the Google Java style guide, which you should review before diving into EPI. The guide is fairly straightforward—it mostly addresses naming and spacing conventions, which should not be a high priority when presenting your solution.
- We use the appropriate exception type, e.g., `NoSuchElementException` when dequeing an empty queue. This is not needed in an interview, where throwing `RunTimeException` suffices.
- When specifying types, we use the weakest type that provides the functionality we use, e.g., the argument to a function for computing the kth smallest element in an array is a `List`, rather than `ArrayList`. This is generally considered a best

practice—it enables code reuse via polymorphism—but is not essential to use in an interview. (You should spend your time making sure the program works, rather than, e.g., if you should be using `List` or `RandomAccess` as the argument type.)

A best practice that we do use in EPI, which is industry-standard, that we recommend you use in an interview is explicitly creating classes for data clumps, i.e., groups of values that do not have any methods on them. Many programmers would use a generic Pair or Tuple class, but we have found that this leads to confusing and hence buggy programs.

23.4 Books

Our favorite Java reference is Peter Sestoft's "Java Precisely", which does a great job of covering the language constructs with examples. The definitive book for Java is "Java: The Complete Reference" by Oracle Press—this is the go-to resource for nuances.

Joshua Bloch's "Effective Java" (second edition) is the best all-round programming book we have come across, addressing everything from the pitfalls of inheritance, to the Executor framework.

For design patterns, we like "Head First Design Patterns" by Freeman {em et al. It's primary drawback is its size. Tony Bevis' "Java Design Pattern Essentials" conveys the same content in a more succint though less entertaining fashion.

Note that our programs are written at too small a scale to take advantage of design patterns.

"Java Concurrency in Practice", by Goetz *et al.* does a wonderful job of explaining pitfalls and practices around multithreaded Java programs.