Java Part 5: Arrays, Lists and Maps

Real programs handle many values: all the products, all the customers, all the orders of a day. Separate variables do not scale, so Java offers structures that hold many values together. Four steps cover the everyday choices: array, list, map and choosing the right one. In this part, Omar, an inventory manager for a chain of electronics stores, uses them to track the stock of every product, and each step answers one practical question.

1. Array

An array holds a fixed number of values of the same type. We fix its size when we create it, and the size cannot change afterwards. Each value is reached by its index, and the first index is zero, not one. The length of an array is available as length, without parentheses. Asking for an index outside the array stops the program with an ArrayIndexOutOfBoundsException, one of the most common beginner errors. The question for this step is: do we know exactly how many values there will be?

Example: A clinic keeps its opening hours for the seven days of the week in an array of seven values. The first slot has index zero, so Monday sits at zero and Sunday at six. The week never gains an eighth day, so a fixed size fits perfectly. Asking for index seven would stop the program, because the array ends at index six.

At work: Omar stores the quantity of one product in each of the five regional warehouses in an array of five numbers. The loop visits index zero to index four, and a loop that went one step further would crash with the index error. The array is a good fit here because the five warehouses never change. The following month a sixth warehouse opens, and he sees the limit of a fixed size.

2. List

A list is a resizable sequence that keeps the order in which values were added. In everyday Java the common implementation is ArrayList, found in the java util package, which we bring into our file with an import line. We declare the type of its elements in angle brackets, called generics, for example a list of String, so the compiler rejects a wrong type. Useful methods are add, get, size, remove and contains, and the for each loop walks through all the elements. A list holds objects, so a list of whole numbers uses the Integer wrapper, which Java converts automatically. We usually declare the variable with the List interface and create an ArrayList, which keeps our code flexible. The question for this step is: does the number of values change, and does the order matter?

Example: A help desk keeps a list of open tickets in the order they arrive. New tickets are added at the end with the add method, and solved tickets are removed with the remove method. The list grows on a busy morning when many tickets arrive, and shrinks again when the team clears the backlog. The code stays exactly the same in both cases, because a list needs no fixed size.

At work: Omar replaces the array with a list of quantities, because warehouses open and close. He adds the sixth quantity with one call to the add method, and a for each loop totals the stock. When a warehouse closes, a call to remove takes it out of the list. The compiler rejects an attempt to add text to a list of numbers, which saves him from a silent mistake.

3. Map

A map stores pairs: a key and the value that belongs to it. Looking up a value by its key is direct, so it stays fast even when the map is large. The common implementation is HashMap. Its main methods are put to

This document is a property of Asaseeyat Academy - © Copy Right Protected.

store a pair, get to read a value, containsKey to ask whether a key exists, and remove. Keys are unique, so putting a pair with an existing key replaces the old value, and get returns null when the key is missing. A HashMap does not promise any order, so when order matters we choose LinkedHashMap or TreeMap. The question for this step is: do we find each value by a name or a code rather than by position?

Example: A university stores each student's name under the student code. One call to the get method with the code returns the name immediately, with no search through the other students. If the code is not in the map, the call returns null, so the program must check before using it. Calling a method on null stops the program with an error.

At work: Omar's products are identified by a product code, so he needs the stock of any code in a moment. He declares the map with `Map<String, Integer> stock = new HashMap<>();`, with the code as the key and the quantity as the value. Receiving goods becomes a read of the old quantity followed by a put with the new total. Before reading an unknown code he asks containsKey, so a missing product never causes a surprise.

4. Choose

The right structure follows the question we ask. If the number of values is fixed and known, an array is simple and compact. If the number changes and the order matters, a list is the natural choice. If we look values up by a key, a map is the tool, and if we only need to know whether something is present, without duplicates, a set such as HashSet does the job. Mixing them is normal: a map from product code to a list of movements is a common shape. The question for this step is: how will we ask for the data most often?

Example: A travel agency needs to check quickly whether a passport number was already processed. A set answers the question directly with one call to the contains method, and never keeps duplicates. A list would need a search through every entry, and a map would ask for a value that nobody wants to store. The set is the right tool because the only question is presence.

At work: Omar's final design uses a map from product code to stock quantity. A second map links each product code to a list of its stock movements, in the order they happened. A set holds the codes of discontinued products, so he can check them in one step. Each choice follows the way he asks his questions.

Conclusion

To recap, arrays hold a fixed number of values reached by an index that starts at zero. Lists grow and shrink and keep the order, and maps find a value by its key. We choose by the question we ask most often, and a set answers whether something is present without duplicates. Omar's stock program now handles any number of warehouses and finds a product by its code. Next time, we will describe products and employees with classes and objects. A good habit is to declare the variable with the interface type, such as List or Map, and to decide the implementation in only one place.

Sources

- Oracle: The Java Tutorials, Arrays – https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html
- Oracle: The Java Tutorials, Collections Trail (Lists, Sets, Maps) – https://docs.oracle.com/javase/tutorial/collections/index.html
- Oracle: The Java Tutorials, Generics – https://docs.oracle.com/javase/tutorial/java/generics/index.html
- dev.java: Learn Java – https://dev.java/learn/

- Java is a trademark of Oracle and/or its affiliates. This video is independent training material and is not affiliated with Oracle.

This document is a property of Asaseeyat Academy - © Copy Right Protected.