JAVA SE

Quiz yourself: TreeSets, LinkedLists, and ArrayDeques in Java

These three classes are powerful—but they can be tricky when combined.

by Simon Roberts and Mikalai Zaikin

April 26, 2021

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

Today's quiz works with different types of lists. Given this code

```
Set<String> ss   = new TreeSet<>();
Queue<String> q  = new LinkedList<>();
Deque<String> dq = new ArrayDeque<>(1);

ss.add("John");
ss.add("Alan");

q.offer("Bert");
q.offer("Kathy");

dq.add("Jane");
dq.addFirst("Alex"); // line n1

ss.addAll(q);
dq.addAll(ss); // line n2

System.out.print(dq);
```

What is the result? Choose one.

A. Runtime exception at line n1 — The answer is A.

B. Runtime exception at line n2 — The answer is B.

C. `[Jane, Alan, John, Bert, Kathy]` — The answer is C.

D. `[Jane, Alan, Bert, John, Kathy]` — The answer is D.

E. `[Alex, Jane, Alan, Bert, John, Kathy]` — The answer is E.

F. `[Alex, Jane, Alan, John, Bert, Kathy]` — The answer is F.

Answer. This code creates three data structures: a `TreeSet`, a `LinkedList`, and an `ArrayDeque`. The variables that refer to them are of type `Set`, `Queue`, and `Deque`, respectively. Of course, the effect of the reference variables is (potentially) to limit the methods that are accessible; they do not change the behavior of those methods.

For each of these structures, the code adds some data items, and then those items are added from one structure to another and then to the third structure to complete the result. Let's look at the sequence of operations to determine the outcome.

The first structure created is a `TreeSet`. A `Set` rejects duplicates, but no duplicates are added, so what matters in this case is that a `TreeSet` is a sorted collection. That is, the items it contains are stored in order. In this example, the `String` items will be sorted in their natural order, which is essentially alphabetical. Therefore, after adding `John` and `Alan`, the items will be in the order `[Alan, John]`.

The next structure created is an instance of `java.util.LinkedList`, which is assigned to a variable of type `Queue`. Although `LinkedList` implements `Deque`, since its methods are called via the `Queue`-type variable, only a queue-related subset of methods is available.

A queue has a head and a tail. If you add elements to the tail and get elements from the head, the head is also the left-hand end. Items on this side will be the first to be encountered in an iteration. A queue is represented as follows:

```
Head (take from here) ⇐ [1] [2] [3] [4] [5] ⇐
```

The code calls the `offer()` method twice. These calls add the items to the tail, exactly as would be the case if you used `add` on a `List`. After these two calls, the queue's contents are `[Bert, Kathy]`.

Next, review the double-ended queue (deque) steps. The `add()` method adds `Jane` to the tail. Of course, it would make no difference which end `Jane` is added to, since that is the first item

to be added. Then, `addFirst()` adds `Alex` to the head, which yields `[Alex, Jane]`.

Option A suggests that an exception will be thrown by this `addFirst` call, but that will not happen; the `ArrayDeque` automatically resizes, just like an `ArrayList`. From this, you know option A is incorrect.

The next step is to copy the elements of the `Queue` into the `Set`. Because the `Set` imposes its own ordering, the original order in the `Queue` is irrelevant. At this point, the elements in the set will be in their natural order, resulting in `[Alan, Bert, John, Kathy]`.

The last step before printing the result is to add all the elements from the sorted set to the deque. This operation is declared to throw some exceptions, but none will arise in this case, so you can determine that option B is incorrect.

As with a `List`, the `Deque`'s `addAll` method adds items to the end (tail) of the queue, so the items from the set will be to the right of the existing two elements `Alex` and `Jane`.

So, the final contents of the `Deque` are `[Alex, Jane, Alan, Bert, John, Kathy]`. These will print in the order in which they're stored, and that means option E is correct and all other options are incorrect.

Conclusion. The correct answer is option E.

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified*

Programmer for Java study guides by Kathy Sierra and Bert Bates.

Share this Page