Java Magazine

Java SE, Quiz

Quiz yourself: HashSet and TreeSet sources in Java streams


Simon Roberts


Mikalai Zaikin

October 11, 2021

Sometimes you must pay attention to the encounter order of elements in a stream.

Given the following code

```
List<String> wordList = Arrays.asList("how", "this", "may", "be", "correct");
Set<String> words1 = new HashSet<>(wordList);
Set<String> words2 = new TreeSet<>(wordList);
// CODE OMITTED HERE
System.out.print(words1.stream().findFirst().get());
System.out.println(words2.stream().findFirst().get());
```

Copy code snippet

Assume that the behavior of the code not shown at the `// CODE OMITTED HERE` line is to add more elements to both sets and then delete all the newly added elements. As a result, the number and values of the data items in the sets after this comment are identical to the contents before this line.

Which best describes the possible output? Choose one.

A. `bebe` — The answer is A.

B. `be`<any value from list> — The answer is B.

C. <any value from list>`be` — The answer is C.

D. <any value from list><any value from list> — The answer is D.

Answer. This question investigates the required behaviors of streams and `Set` objects, as well as the variations that they permit. The `findFirst` method has some flexibility that might cause unpleasant surprises if you're not familiar with the rules.

The name of the `findFirst` method appears to indicate that it specifically returns the first element from the stream, hinting that the output perhaps ought to be `bebe`, as in option A. However, there's a significant note in the documentation for `findFirst` that states the following and means that option A is incorrect:

> If the stream has no encounter order, then any element may be returned.

So, what does *encounter order* mean, and do these collections have it? The streams documentation says the following:

> Streams may or may not have a defined encounter order. Whether or not a stream has an encounter order depends on the source and the intermediate operations. Certain stream sources (such as `List` or arrays) are intrinsically ordered, whereas others (such as `HashSet`) are not. Some intermediate operations, such as `sorted()`, may impose an encounter order on an otherwise unordered stream, and others may render an ordered stream unordered, such as `BaseStream.unordered()`. Further, some terminal operations may ignore encounter order, such as `forEach()`.
>
> If a stream is ordered, most operations are constrained to operate on the elements in their encounter order; if the source of a stream is a `List` containing `[1, 2, 3]`, then the result of executing `map(x -> x*2)` must be `[2, 4, 6]`. However, if the source has no defined encounter order, then any permutation of the values `[2, 4, 6]` would be a valid result.
>
> For sequential streams, the presence or absence of an encounter order does not affect performance, only determinism. If a stream is ordered, repeated execution of identical stream pipelines on an identical source will produce an identical result; if it is not ordered, repeated execution might produce different results.

The code in this question draws its stream from two distinct implementations of the `Set` interface: first `HashSet` and then `TreeSet`. `TreeSet` is ordered, but as noted in the documentation excerpt above, `HashSet` is not.

Because of this, the stream extracted from the `HashSet` does *not* have a defined encounter order and, consequently, the first time the code runs `findFirst` it might return *any* item from the stream. Thus, any of the strings originally added to the set might be printed as the first output item.

The second item is printed from the `findFirst` operation executed on the stream drawn from the `TreeSet`. Because `TreeSet` is ordered, and nothing else prevents it, the stream runs with an encounter order. The first item in the `TreeSet` must be returned by the `findFirst` operation. Because the default ordering of a `TreeSet` is the so-called *natural order* of the items it contains, and the natural order of `String` objects is alphabetical, you can infer that the second output item will consistently be `be`.

Given these observations, options B and D are incorrect and the correct answer must be option C.

A side note on ordering is that the API documentation doesn't always reliably tell you if a data structure has an encounter order. It's usually a fairly safe guess that if ordering is an intrinsic part of the data structure, as with lists in general and with `HashSet`, encounter order will be defined. But for any given set, it's hard to tell (and from the codebase, it might be impossible to tell if you know only that you have "something that implements `Set`"). One trick worth knowing is that if you have a stream in a variable called `myStr`, the following expression will be true if the stream is ordered and false otherwise:

```
((myStr.spliterator().characteristics() & Spliterator.ORDERED) != 0)
```

Copy code snippet

Alternatively, if you simply want to test whether a particular data structure has an encounter order, you can request the spliterator directly from that structure and test it, without ever going to a stream. For a collection called `myColl`, you could use this code.

```
((myColl.spliterator().characteristics() & Spliterator.ORDERED) != 0)
```

Copy code snippet

Finally, it's worth mentioning that it's possible to encounter in a real exam question a comment that describes the behavior of a piece of code that's omitted from the body of a description, as with the `// CODE OMITTED HERE` line in this question. It's not something that happens often, and the complexity of the description in this case is almost certainly more than a real question would allow. However, if you do see a descriptive comment such as that in a quiz question, you should trust it.

Don't be afraid to take any such comments literally, and know that the hidden code is *not* an attempt to trick you. Just the opposite, in fact: The quiz writers are limiting the presented code to help you focus on what's important for understanding the question.

Conclusion. The correct answer is option C.

Related quizzes

Related quizzes

- Quiz yourself: TreeSets, LinkedLists, and ArrayDeques in Java
- Quiz yourself: Search stream data using the findFirst, findAny, and anyMatch methods
- Quiz yourself: Use Java streams to filter, transform, and process data

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.

Previous Post

Restock your tech library with these new Java books

Alan Zeichick | 19 min read