JAVA SE

Quiz yourself: Manipulating Java lists—and views of lists

Is a list unmodifiable? Is it immutable? What about the views of the list?

by Mikalai Zaikin and Simon Roberts

February 15, 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.

Given the following code fragment:

```
String[] arr = new String[] {"A1", "A2"};

List<String> ls = new ArrayList<>();
ls.add("L1");ls.add("L2");

List<String> la = Arrays.asList(arr);
List<String> lf = List.of(arr);
List<String> lc = List.copyOf(ls);
List<String> lu = Collections.unmodifiableLis

arr[1]="A3";
ls.set(1, "L3");

// line n1

System.out.println("la=" + la);
System.out.println("lf=" + lf);
System.out.println("lc=" + lc);
System.out.println("lu=" + lu);
```

Which statements are correct? Choose three.

A. The output contains `la=[A1, A2]`.

The answer is A.

B. The output contains `lf=[A1, A2]`.

The answer is B.

C. The output contains `lc=[L1, L3]`.

The answer is C.

D. The output contains `lu=[L1, L3]`.

The answer is D.

E. If you add the following at line n1, the output will contain `lf=[A1, A3]: lf.set(1, "A3");`.

The answer is E.

F. If you add the following at line n1, the output will contain `la=[A1, A3] la.set(1, "A3");`.

The answer is F.

Answer. This quiz question explores different approaches to make lists that reflect data provided in arrays (known as *bridging* arrays and lists), as well as making detached copies of lists and views of lists.

The code in the question creates an array containing the values `A1` and `A2` and then an `ArrayList` containing the values `L1` and `L2`. Those are then used to initialize four more lists.

Next, the code creates a `List` initialized from the array using the `static` factory method `Arrays.asList(array)`. This method creates a `List` that acts as a view into the array. Being a view means that the methods of the `List` interact directly with the data stored in the array. The `get` operations return values from the array, and `set` operations change values in the array. Furthermore, if the array values are changed directly in the array, the data seen through subsequent `get` operations on the view will reflect those changes.

Of course, the length of a Java array cannot be changed after creation, and as a result the length of the `List` created by `Arrays.asList` cannot be changed either. You cannot add new elements, nor can you remove elements.

Given that `List la` is a view on the array `arr`, the assignment `arr[1]="A3";` will also change the contents of `List la`. So the result of the line that prints `la` will be `la=[A1, A3]`.

Therefore, you know that option A is incorrect.

Provided you do not attempt to change the length of a `List` that is a view on an array, you *can* change the elements using the `List.set` methods. Such changes will be visible in both the view and the underlying array. Option F suggests adding the code `la.set(1, "A3");` at line n1. This code compiles and executes without any problems, although it doesn't change any values. Given that the output already contains the text `la=[A1, A3]`, option F is correct.

The second `List` is created from the array using the `List.of(array)` factory method. Each of the dozen overloaded `List.of` factories creates *unmodifiable lists*. The term *unmodifiable* means that the *structure* of the `List` itself cannot be altered. This differs from a truly *immutable* `List` in that if the elements in the `List` are mutable (which is very common in Java), you can change the contents of the elements themselves. You cannot, however, alter the `List` so that it refers to any different elements, nor can you add or remove any elements.

A consequence of this is that if the elements in the `List` are themselves immutable (which the strings in this case are), you actually do have a fully immutable `List`.

The `List.of()` factory copies the array elements into a new structure for the `List` it creates. This ensures that the new `List` is independent of the array from which it was initialized. Based on this, you can conclude that modification of the source array `arr[1]="A3";` will not affect the `List` state. The line that prints the contents of `List lf` will produce `lf=[A1, A2]`. Therefore, you know that option B is correct.

As mentioned above, products of the `List.of` factory methods are unmodifiable. Such lists do not simply ignore attempts to change them. Instead, if there's an attempt to modify them, they throw an exception. Option E suggests adding a `set` method call on the `List`. Such an action will cause the code to throw an exception rather than cause the output suggested in option E. From this, it's clear that option E is incorrect.

The third `List`, `lc`, is created using the `List.copyOf(collection)` factory method. This method also returns an unmodifiable list that reflects the state of the initializing collection at the time the factory is invoked. This tells you that the code `ls.set(1, "L3")`, which is executed after the initialization of `lc`, will not affect `lc List`. Therefore, option C is incorrect.

By the way, the `List.copyOf` method was added in Java 10, and its behavior depends on whether the argument is already an unmodifiable list. If it is, the argument list is promptly returned. However, if the argument `List` might be mutable, an unmodifiable copy is created. This allows code to create unmodifiable copies when necessary without needlessly creating duplicates if the original was already modifiable.

The fourth `List` is created using the factory `Collections.unmodifiableList(list)`. This method creates an *unmodifiable view*. An unmodifiable view is a type of view, so `get` operations on it return the data in the underlying `List` (that is, the `List` that's the argument to the `unmodifiableList` factory method). If the underlying `List` is changed, the results reported by calling `get` on the view will change. However, any attempt to change the data by calling methods on the unmodifiable view itself will fail. Specifically, all such operations on the unmodifiable view (such as `set`, `add`, `remove`, or `clear`) throw an `UnsupportedOperationException`. In this question, the underlying `List` has been changed to contain `[L1, L3]`, and so the output will contain `lu=[L1, L3]`. This means option D is correct.

Conclusion: The correct answers are options B, D, and F.

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.

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.

Share this Page