JAVA SE

Quiz yourself: The Java MapUtility class

Null keys? No null keys? Let's see what the quizmasters say.

by Mikalai Zaikin and Simon Roberts

June 14, 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.

In this scenario, a coworker is writing a framework that enhances work with Java collections. In particular, the following `MapUtility` class allows a client to receive the `java.util.NoSuchElementException` exception if the key is missing in the map or the value found is `null`:

```
import java.util.Map;
import java.util.NoSuchElementException;

public class MapUtility<K,V> {
  private Map<K,V> map;
  public MapUtility(Map<K,V> map) {
    this.map = map;
  }
  public V get(K key) throws NoSuchElementExc
    return map.computeIfAbsent(key,
      k -> { throw new NoSuchElementException
  }
}
```

However, after sharing the framework within your company, the code's author starts getting bug reports from colleagues.

What is the problem with this class? Choose one.

A. The `java.util.Map` interface does not allow `null` keys, so passing in a `null` in the `get` method code throws `NullPointerException`.

The answer is A.

B. The `java.util.Map` interface does not allow `null` values, so this code never throws `NoSuchElementException`.

The answer is B.

C. The class may throw `UnsupportedOperationException`.

The answer is C.

D. None of the above.

The answer is D.

Answer. The `java.util.Map` interface does *not* prohibit `null` keys, so because option A states the contrary, you can conclude that option A is incorrect. Of course, at most one key can be `null`, because the keys of a map form a set. It's possible that the map passed to the constructor of this utility might not support `null` keys, but no such generalization can be made. In particular, a `java.util.HashMap` works perfectly well with a `null` key.

Option B is incorrect for the same essential reason as option A, because option B asserts that the `Map` interface prohibits `null` as a value. However, the interface makes no such restriction. In addition, the `HashMap` class permits not only `null` as a key but also as a value.

In this situation, the method will work entirely as expected, effectively translating a `null` value into a `java.util.NoSuchElementException`. Certainly, some `Map` implementations might show a different behavior, but the core assertion regarding nulls being used as values is an invalid generalization.

The code shown below works correctly, throwing the expected `NoSuchElementException` when getting the `null` that is paired with the key `K1`.

```
Map<String, String> map = new HashMap<>();
map.put("K1", null);
MapUtility<String, String> mu = new MapUtilit
System.out.println(mu.get("K1"));
// java.util.NoSuchElementException: Missing
```

Unfortunately, the code's author either assumed that the `computeIfAbsent` method always works identically for any map or assumed that if no actual modification to the map contents is made, the method always works. However some Java maps are unmodifiable and they simply throw `java.lang.UnsupportedOperationException` on *any* potentially mutating operation. In general, the overriding `computeIfAbsent` methods in these unmodifiable maps throw this exception upon being called, and they do not defer the exception to a point when an actual change is about to be made. Thus, if the `MapUtility` is wrapped around an unmodifiable map, it will fail anytime the `get` method is called. From this it's clear that option C is correct.

By the way, there are several situations that can give rise to an unmodifiable map in the Java APIs. One situation is when you use the `Collections.unmodifiableMap` feature, which creates a wrapper (more strictly, it's a Gang of Four "Proxy" pattern) that presents a read-only view of an existing map (which might itself be modifiable or not). Another is when you use the `Map.of(...)` factory methods.

In this example that uses the unmodifiable wrapper, the caller will receive the `UnsupportedOperationException` instead of the `NoSuchElementException`. If the caller tried to handle the situation with a `try-catch` construction, this exception will probably not get caught by the intended handler.

```
Map<String, String> map = new HashMap<>();
map.put("USA", "North America");
map.put("Belarus", "Europe");
map = Collections.unmodifiableMap(map);
MapUtility<String, String> mu = new MapUtilit
System.out.println(mu.get("Germany"));
// java.lang.UnsupportedOperationException
```

Option D asserts that none of the other options is correct and, therefore, option D must be incorrect.

How could you refactor the `MapUtility` class to avoid these mutation operations and work with unmodifiable maps? One of the possible ways is to use the `Optional` class.

```
public V get(K key) throws NoSuchElementExcep
  return Optional.ofNullable(map.get(key)).
    orElseThrow(() ->
      new NoSuchElementException("Missing key
}
```

With this modification, the attempts to `get` using the key `Germany` works as expected and throws the `NoSuchElementException` with the `Missing key:Germany` message.

[Read about the `Optional` class in "12 recipes for using the Optional class as it's meant to be used" and "The Java Optional class: 11 more recipes for preventing null pointer exceptions." — *Ed.*]

Conclusion. The correct answer is option C.

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