Interview Questions: Collections

Eight questions on arrays vs lists, List/Set/Map, HashMap order, and contains() -- the answer that names the mechanism (buckets, equals, iteration order) rather than just naming the class.

Automation Foundations · ashishautomationlabs.store

Q15 · ●●● -- What is the difference between an array and an `ArrayList`?

Weak answer: "An array has a fixed size and an `ArrayList` can grow."

Strong answer: "An array is fixed at creation -- you must know the size up front, and it cannot change. An `ArrayList` grows and shrinks as you add and remove, and it gives you methods like `add`, `remove`, `contains`, and `size`. Arrays hold primitives directly; an `ArrayList` holds objects. In practice I use an `ArrayList` for almost everything in test code, because you rarely know in advance how many results or rows you will collect. Arrays still appear in `main(String[] args)` and wherever a fixed set is genuinely fixed."

Follow-up: *"How would you get the number of items in each?"*

Where in this book: Chapter 5.

Q16 · ●●● -- What is the difference between a `List`, a `Set`, and a `Map`?

Weak answer: "A `List` is ordered, a `Set` has no duplicates, and a `Map` has keys and values."

Strong answer: "A `List` keeps insertion order and allows duplicates, so it fits a record of everything that happened, such as every test result in order. A `Set` holds each item once and has no positions, so it fits questions like 'have I already seen this?' -- distinct failure reasons, or pages a crawler has visited. A `Map` stores key-value pairs and looks a value up by its key, which fits configuration and lookups: environment name to URL, test name to result. The choice comes from the question you need to answer, not from preference."

Follow-up: *"Which would you use to count how many times each status appears?"*

Where in this book: Chapters 5 and 10.

Q17 · ●●● -- How do you get a value out of a `Map`, and what happens if the key is not there?

Weak answer: "You use `get()`, and it returns `null` if the key is missing."

Strong answer: "`get(key)` returns the value, and a missing key returns `null` rather than throwing -- which is exactly why it causes trouble. The `null` travels on and produces a

`NullPointerException` somewhere else entirely. So I either check with `containsKey` first, or use `getOrDefault(key, fallback)`, which returns my default when the key is absent. In a config loader that is the difference between a clear message and a mystery failure three classes away."

Follow-up: *"What happens if you put a key that already exists?"*

Where in this book: Chapter 10.

Q18 · ●●■ -- Does a `HashMap` keep the order you inserted things in?

Weak answer: "No, it is unordered."

Strong answer: "No, and it also does not sort. The order it returns is an internal detail that you must never rely on -- I have seen it come back Search, Login, Cart when inserted Login, Search, Cart. For a tester that matters because an assertion depending on `HashMap` order will pass for months and then fail for no visible reason. If order is part of the requirement, I say so in the type: `LinkedHashMap` for insertion order, `TreeMap` for sorted keys. Then the guarantee is in the code rather than in hope."

Follow-up: *"How would you write that assertion so order does not matter?"*

Where in this book: Chapter 10.

Q19 · ●●■ -- How do you loop through a `Map`?

Weak answer: "With a for-each loop over the keys."

Strong answer: "Three views are available depending on what I need. `keySet()` for the keys, `values()` for the values, and `entrySet()` when I need both -- which is the common case in reporting. The entry loop gives me `getKey()` and `getValue()` on each pair, so building a summary line per test is one loop. If I only need to count something, `values()` is enough."

Follow-up: *"Show me the entry loop."*

Where in this book: Chapter 10.

Q20 · ●●■ -- How do you check whether a list contains a particular value?

Weak answer: "Use `contains()`."

Strong answer: "`contains(value)` returns `true` or `false`, and it uses `equals` under the hood, so it is case-sensitive for Strings. That is a real trap in test data: a list containing `'PASS'` will report `false` for `'pass'`, and the test quietly takes the wrong branch. So I normalize case when the data is human-entered. For my own classes, `contains` only works correctly if I have overridden `equals`, because otherwise it compares object identity."

Follow-up: *"What if the list holds your own* `TestResult` *objects?"*

Where in this book: Chapter 5, with the equality rule in Chapters 6 and 7.

Q21 · ●●■ -- What is the difference between `ArrayList` and `LinkedList`?

Weak answer: "`ArrayList` uses an array and `LinkedList` uses nodes."

Strong answer: "`ArrayList` stores items in a resizable array, so reading by index is fast and it is the right default. `LinkedList` chains nodes together, which makes inserting or removing in the middle cheaper but reading by index slower. The honest senior answer is that `ArrayList` wins in almost all real code, including test frameworks, because we mostly add to the end and read through. The more useful point is that both are a `List`, so I declare `List` on the left and can swap the implementation without touching any other line."

Follow-up: *"So why declare* `List` *rather than* `ArrayList`*?"*

Where in this book: Chapters 5 and 12.

Q22 · ●●■ -- What does `Set.add()` return, and why is that useful?

Weak answer: "It adds an item to the set."

Strong answer: "It returns a boolean: `true` if the item was actually added, `false` if it was already present. That is genuinely useful, because it answers 'is this new?' and 'record it' in one call. A crawler tracking visited pages, or a report collecting distinct failure messages, needs exactly that. For my own classes it depends on `equals` and `hashCode` being written correctly, otherwise the set will happily store two objects I consider identical."

Follow-up: *"What does the set size come to if you add the same value three times?"*

Where in this book: Chapter 10.