**🔹 Basic-Level Questions**

**1. What is the Java Stream API and why is it used?**

Java Stream API, introduced in Java 8, provides a functional approach to process sequences of elements (like collections). It allows for efficient data processing using operations like filter, map, reduce, etc.

**2. What is the difference between a Collection and a Stream in Java?**

* **Collection** stores data and is a data structure.
* **Stream** is for computation; it processes data from collections or arrays in a functional style.

**3. How do you create a Stream in Java?**

java

CopyEdit

List<String> list = Arrays.asList("a", "b", "c");

Stream<String> stream = list.stream();

**4. What are the types of Streams in Java?**

* **Sequential Stream**: Processes elements in a single thread.
* **Parallel Stream**: Splits data and processes in multiple threads.

**5. What is the difference between stream() and parallelStream()?**

* stream() is sequential.
* parallelStream() uses multiple threads and can be faster for large data.

**6. What is the difference between intermediate and terminal operations in Streams?**

* **Intermediate**: Lazy operations that return a Stream (e.g., filter(), map()).
* **Terminal**: Triggers processing and ends the stream (e.g., collect(), forEach()).

**7. Explain the difference between map() and flatMap().**

* map() transforms each element individually.
* flatMap() flattens nested structures (like a Stream of Lists) into a single stream.

**8. What does the filter() method do in a stream?**

It filters elements based on a predicate.

java

CopyEdit

list.stream().filter(s -> s.startsWith("A")).collect(Collectors.toList());

**9. What does the collect() method do?**

It transforms the elements of a stream into a collection or another form:

java

CopyEdit

List<String> result = stream.collect(Collectors.toList());

**10. How can you convert a List of Strings to uppercase using Stream API?**

java

CopyEdit

list.stream().map(String::toUpperCase).collect(Collectors.toList());

**🔹 Intermediate-Level Questions**

**11. How does lazy evaluation work in Streams?**

Streams don't process data until a terminal operation is invoked. This improves performance by avoiding unnecessary computations.

**12. Can a Stream be reused? Why or why not?**

No. Once consumed, a stream cannot be reused. A new stream must be created.

**13. What are some terminal operations in Stream API?**

* collect()
* forEach()
* count()
* anyMatch()
* reduce()

**14. How would you remove duplicates from a collection using Stream API?**

java

CopyEdit

List<String> unique = list.stream().distinct().collect(Collectors.toList());

**15. What is the role of the Collectors class?**

It provides utility methods for collecting stream elements into collections, strings, maps, etc.

**16. How do you group elements in a stream by a certain field?**

java

CopyEdit

Map<String, List<Employee>> byDept = employees.stream()

.collect(Collectors.groupingBy(Employee::getDepartment));

**17. How do you sort a collection using streams?**

java

CopyEdit

list.stream().sorted().collect(Collectors.toList());

Or with custom comparator:

java

CopyEdit

list.stream().sorted(Comparator.comparing(Person::getAge)).collect(Collectors.toList());

**18. How to handle null values in a Stream?**

java

CopyEdit

list.stream().filter(Objects::nonNull).collect(Collectors.toList());

**19. What’s the difference between reduce() and collect()?**

* reduce() performs reduction (e.g., sum, concat).
* collect() gathers elements into a container like List or Map.

**20. How to find the maximum or minimum element in a stream?**

java

CopyEdit

Optional<Integer> max = list.stream().max(Integer::compareTo);

**🔹 Advanced-Level Questions**

**21. How does short-circuiting work in Stream operations?**

Some terminal operations (like findFirst(), anyMatch()) can stop processing as soon as a result is found, improving performance.

**22. Explain the concept of stateless and stateful operations in Streams.**

* **Stateless**: Each element is processed independently (filter(), map()).
* **Stateful**: Depends on previous elements (sorted(), distinct()).

**23. How does parallel stream processing work internally?**

Parallel streams split the source into substreams using Fork/Join framework, and process them in parallel threads.

**24. What are some performance considerations when using parallel streams?**

* Overhead of thread management.
* Suitability for CPU-bound vs I/O-bound tasks.
* Side effects and shared mutable state can lead to bugs.

**25. What is the effect of side effects in stream operations?**

Side effects (like modifying external variables) break functional purity and can cause bugs, especially in parallel streams.

**26. How can you implement a custom collector?**

java

CopyEdit

Collector<String, StringBuilder, String> customCollector =

Collector.of(StringBuilder::new, StringBuilder::append, StringBuilder::append, StringBuilder::toString);

**27. Can you write a stream pipeline that returns the second highest number in a list?**

java

CopyEdit

Optional<Integer> secondMax = list.stream()

.distinct()

.sorted(Comparator.reverseOrder())

.skip(1)

.findFirst();

**28. How do you join strings from a list using streams?**

java

CopyEdit

String result = list.stream().collect(Collectors.joining(", "));

**29. How to limit the number of results in a stream?**

java

CopyEdit

list.stream().limit(5).collect(Collectors.toList());

**30. Can you explain the internal working of collect(Collectors.groupingBy())?**

It uses a classifier function to group elements into a Map. Internally, it uses a mutable accumulator (like Map<K, List<T>>) and merges results as needed.