TOP 20 INTERVIEW QUESTIONS ON JAVA 8 STREAMS

1. **What are Java 8 Streams? How do they differ from collections?**
    - **Answer:** Streams are a sequence of elements supporting sequential and parallel aggregate operations. Unlike collections, streams do not store elements; they process data in a declarative way, focusing on "what" rather than "how."

---

2. **Explain the difference between `Stream` and `ParallelStream`. When would you use each?**
    - **Answer:** `Stream` processes elements sequentially, while `ParallelStream` divides the data into multiple parts and processes them in parallel. Use `ParallelStream` when you need faster processing on large datasets.

---

3. **What is the difference between intermediate and terminal operations in a stream?**
    - **Answer:** Intermediate operations (e.g., `filter`, `map`) return a stream and are lazy, meaning they don't process until a terminal operation is invoked. Terminal operations (e.g., `collect`, `forEach`) produce a result or side effect and mark the end of the stream.

---

4. **How does the `filter()` method work in Java Streams? Can you provide an example?**
    - **Answer:** `filter()` is an intermediate operation that returns a stream containing elements that match a given predicate. Example: `stream.filter(x -> x > 10)` filters out numbers less than or equal to 10.

---

5. **What is the purpose of the `map()` function in streams? How does it differ from `flatMap()`?**
    - **Answer:** `map()` transforms each element of the stream, while `flatMap()` flattens nested streams or collections into a single stream. Example: `map()` is used for element transformation, whereas `flatMap()` is used to handle nested lists.

---

6. **Can you explain the working of the `reduce()` method in streams with an example?**
    - **Answer:** `reduce()` combines elements of a stream into a single result. Example: `stream.reduce(0, Integer::sum)` adds all elements in the stream.

---

7. **What are collectors in Java Streams? Explain `Collectors.toList()` and `Collectors.toMap()`.**
    - **Answer:** Collectors are utilities for gathering elements of a stream into a collection or another form. `Collectors.toList()` collects elements into a `List`, while `Collectors.toMap()` collects elements into a `Map`.

---

8. **How does `Stream.sorted()` work? Can it sort in descending order?**
    - **Answer:** `Stream.sorted()` sorts elements in natural order or with a custom comparator. To sort in descending order, use `stream.sorted(Comparator.reverseOrder())`.

---

9. **What is the difference between `findFirst()` and `findAny()`?**
    - **Answer:** `findFirst()` returns the first element in the stream, whereas `findAny()` returns any element, optimized for parallel streams.

---

10. **Explain the `forEach()` method in streams. How is it different from a traditional loop?**
    - **Answer:** `forEach()` is a terminal operation that processes each element in the stream. Unlike traditional loops, it works on streams and doesn't guarantee order in parallel streams.

---

11. **What is lazy evaluation in streams? How does it affect performance?**
    - **Answer:** Lazy evaluation means intermediate operations are not executed until a terminal operation is called, improving performance by processing only necessary data.

---

12. **How do you handle exceptions in streams?**
    - **Answer:** Handling exceptions can be done using wrapper methods or custom utility functions that catch and rethrow exceptions as unchecked ones.

---

13. **What is the purpose of `peek()` in streams? When should it be used?**
    - **Answer:** `peek()` is mainly used for debugging purposes, allowing you to see the elements as they pass through the pipeline. It's an intermediate operation.

---

14. **Can you convert a `Stream` back to a `Collection`? How?**
    - **Answer:** Yes, by using `collect(Collectors.toList())`, `collect(Collectors.toSet())`, or similar methods.

---

15. **What is the difference between `anyMatch()`, `allMatch()`, and `noneMatch()` in streams?**
    - **Answer:** `anyMatch()` checks if any element matches a condition, `allMatch()` checks if all elements match, and `noneMatch()` checks if none of the elements match.

---

16. **Explain `limit()` and `skip()` methods in streams. How are they useful?**
    - **Answer:** `limit()` restricts the number of elements in a stream, and `skip()` skips a given number of elements. Useful for pagination and sampling.

---

17. **How can you concatenate two streams in Java 8?**
    - **Answer:** Use `Stream.concat(stream1, stream2)` to merge two streams into one.

---

18. **What are parallel streams? How do you create one?**
    - **Answer:** Parallel streams allow parallel processing of data. Create one using `parallelStream()` or `stream.parallel()`.

---

19. **Explain short-circuiting operations in streams. Give an example.**
    - **Answer:** Short-circuiting operations (e.g., `findFirst()`, `anyMatch()`) stop processing as soon as a condition is met. Example: `findFirst()` halts after finding the first element.

---

20. **What is the role of the `distinct()` method in streams?**
    - **Answer:** `distinct()` filters out duplicate elements from the stream, ensuring all elements are unique.
