Here’s a strong list of **Java Future interface interview questions**—including basic, practical, and comparison-based ones—to help you explain confidently in interviews:

**✅ BASIC QUESTIONS**

**1. What is a Future in Java?**

A Future is a Java interface used to represent the **result of an asynchronous computation**. It provides methods to check if the task is complete, to wait for completion, and to retrieve the result.

**2. How do you create a Future in Java?**

ExecutorService executor = Executors.newSingleThreadExecutor();

Future<Integer> future = executor.submit(() -> 10 + 20);

**3. What are the main methods of Future?**

* get(): Waits if necessary and returns the result.
* get(timeout, unit): Waits for the result up to the timeout.
* isDone(): Returns true if task is finished.
* isCancelled(): Returns true if task was cancelled.
* cancel(boolean mayInterruptIfRunning): Cancels the task.

**4. What happens if you call get() before the task is completed?**

The call to get() **blocks** the thread until the result is available.

**🔁 THREADING CONTROL**

**5. What is the purpose of cancel() method?**

It tries to cancel the execution of the task:

* If the task hasn't started, it won’t run.
* If already running, and mayInterruptIfRunning is true, the thread will be interrupted.

**6. What is the difference between isCancelled() and isDone()?**

* isDone() returns true when the task is completed **normally**, **through exception**, or **via cancellation**.
* isCancelled() returns true **only if the task was cancelled**.

**🔍 PRACTICAL QUESTIONS**

**7. What if the task throws an exception?**

When you call future.get(), the exception is wrapped inside an ExecutionException.

**8. Can multiple threads access the same Future instance?**

Yes, but Future itself is **not thread-safe for mutation**. You can call get() from multiple threads safely.

**9. What happens if timeout expires in future.get(timeout)?**

A TimeoutException is thrown.

**⚔️ COMPARISON-BASED QUESTIONS**

**10. Future vs Runnable**

| **Feature** | **Runnable** | **Future** |
| --- | --- | --- |
| Return result? | ❌ No | ✅ Yes |
| Used with? | Threads/Executor | ExecutorService |
| Check status? | ❌ No | ✅ Yes (isDone()) |

**11. Future vs Callable**

| **Feature** | **Callable** | **Future** |
| --- | --- | --- |
| Is it an object? | Functional interface | Interface to hold result |
| Purpose | Defines task | Holds task result |

**12. Future vs CompletableFuture**

| **Feature** | **Future** | **CompletableFuture** |
| --- | --- | --- |
| Asynchronous? | ✅ | ✅ |
| Chaining operations? | ❌ | ✅ (thenApply, etc.) |
| Exception Handling? | Manual | Built-in (exceptionally) |
| Multiple task combining? | ❌ | ✅ |

**13. When should you prefer CompletableFuture over Future?**

When you need:

* Better **control** over the flow
* **Non-blocking** chaining of tasks
* Better **error handling**
* **Combining multiple async tasks**

**👀 ADVANCED INTERVIEW QUESTION**

**14. Can you implement retry logic with Future?**

Yes, but you must implement it manually — Future itself doesn't support retries. Use a loop with try-catch and backoff strategy.

**15. How do you shut down an Executor after submitting Futures?**

executor.shutdown(); // no new tasks

executor.awaitTermination(5, TimeUnit.SECONDS); // wait to finish

Let me know if you'd like:

* Code examples for each
* A mock Q&A round
* Or scenario-based questions like “How would you use Future in a file processing system or API parallel calls?”

Would you like to move to ForkJoinPool or ScheduledExecutorService next?