Great question, Gyanenjay! 👏

By default, using Future.get() is **blocking**, which means the thread calling get() is **paused** until the result is ready. In real-world scenarios (like UI apps or high-throughput systems), **blocking is bad** because it wastes threads. Let’s go over **how to handle or avoid blocking with Future**.

**✅ Problem with Blocking:**

Future<String> future = executor.submit(() -> {

Thread.sleep(5000); // long task

return "done";

});

String result = future.get(); // ❌ BLOCKS here for 5 seconds

**🔄 Solutions to Handle or Avoid Blocking in Future**

**1. Use get(timeout, TimeUnit)**

Avoid infinite blocking. This way, you can **timeout gracefully**.

try {

String result = future.get(2, TimeUnit.SECONDS);

} catch (TimeoutException e) {

System.out.println("Task took too long, exiting...");

}

✅ Useful in APIs where **delay shouldn't freeze the flow**.

**2. Poll using isDone()**

You can keep checking whether the task is done instead of blocking directly.

while (!future.isDone()) {

System.out.println("Still processing...");

Thread.sleep(500); // Do something or yield

}

String result = future.get(); // Now safe to call

✅ Good if you want to do **other work in the meantime**.

**3. Use CompletableFuture (Best Approach)**

Avoid blocking altogether. Handle results asynchronously using callbacks.

CompletableFuture.supplyAsync(() -> {

try {

Thread.sleep(3000);

} catch (InterruptedException e) {}

return "Async result";

}).thenAccept(result -> {

System.out.println("Received: " + result);

});

✅ Doesn’t block the calling thread.

**4. Use ExecutorService with a Timeout Wrapper**

ExecutorService executor = Executors.newSingleThreadExecutor();

Future<String> future = executor.submit(() -> {

// some long task

return "result";

});

if (future.isDone()) {

System.out.println(future.get());

} else {

System.out.println("Still waiting...");

}

**🔐 Best Practice Suggestion**

* For **low-latency systems**, avoid blocking using CompletableFuture.
* For **safety**, always use get(timeout) instead of plain get().
* Combine Future with ScheduledExecutorService to create your own **retry/timeout logic**.

Would you like to convert a Future into a CompletableFuture?  
Or see how to use these in a **microservices call** (e.g., calling 3 APIs in parallel without blocking)?