**Future hoạt động (task → thread → Future → get)**

**1. Future là gì?**

* Future<T> là một **đối tượng đại diện cho kết quả của một tác vụ (task)** bất đồng bộ (asynchronous).
* Khi bạn nộp (submit) một công việc cho **ExecutorService**, nó trả về một Future.
* Với Future, bạn có thể:
  + Kiểm tra task đã xong chưa (isDone())
  + Hủy task (cancel())
  + Lấy kết quả (get())

👉 **Hình dung:** giống như bạn gửi một đơn hàng online → hệ thống cho bạn **mã đơn hàng (Future)** để tra cứu tình trạng hoặc nhận kết quả khi hoàn tất.

**3. Các phương thức chính của Future**

* boolean cancel(boolean mayInterruptIfRunning)  
  → hủy task nếu chưa xong.
* boolean isCancelled()  
  → kiểm tra task đã bị hủy chưa.
* boolean isDone()  
  → kiểm tra task đã hoàn thành chưa.
* T get()  
  → lấy kết quả, sẽ **block** (chặn) nếu chưa xong.
* T get(long timeout, TimeUnit unit)  
  → lấy kết quả, nhưng chờ tối đa timeout, nếu chưa xong thì ném TimeoutException.

✅ Quản lý công việc bất đồng bộ một cách rõ ràng.  
✅ Dễ kiểm soát kết quả hoặc trạng thái của task.  
✅ Hữu ích khi chạy nhiều task song song và gom kết quả.

**🔮 5. Nhược điểm & Điểm cần chú ý**

1. **get() bị blocking**
   * Nếu gọi future.get() → main thread sẽ bị chặn cho đến khi task xong.
   * 👉 Nếu muốn tránh block, dùng get(timeout, unit) hoặc isDone() để kiểm tra trước.
2. **Không có callback**
   * Future chỉ cho phép lấy kết quả sau khi xong, không có cách đăng ký "hàm callback" khi task hoàn thành.
   * 👉 Trong Java 8+, người ta hay dùng **CompletableFuture** để khắc phục.
3. **Cancel không phải lúc nào cũng hủy được**
   * Nếu task đang chạy và bạn cancel(false) → task vẫn tiếp tục.
   * Nếu cancel(true) → task có thể bị interrupt, nhưng chỉ nếu code trong task kiểm tra interrupt (Thread.interrupted()).
4. **Quản lý nhiều Future phức tạp**
   * Nếu bạn chạy hàng trăm task, phải giữ danh sách Future để check → code rối.
   * 👉 Có thể dùng ExecutorCompletionService hoặc CompletableFuture để tiện hơn.

**🔮 6. Trường hợp nên dùng Future**

* Khi bạn muốn **chạy 1 task ở thread khác** và sau đó **lấy kết quả**.
* Khi có nhiều công việc nặng (ví dụ: phân tích dữ liệu, tính toán) → chạy song song, rồi gom kết quả bằng Future.
* Khi cần kiểm soát trạng thái task (đang chạy, xong, hủy).

**🔮 7. Trường hợp không nên dùng Future**

* Khi bạn cần **xử lý kết quả ngay khi có** (callback, chaining).  
  → Nên dùng **CompletableFuture**.
* Khi số lượng task rất lớn và cần quản lý linh hoạt.  
  → Nên dùng **ForkJoinPool** hoặc **CompletableFuture**.
* Khi chỉ cần chạy task mà không quan tâm kết quả.  
  → Có thể chỉ dùng execute() thay vì submit().