Yes! Let's look at **CompletableFuture** and how it improves **exception handling** over Future. 🚀

**📌 Handling Exceptions in CompletableFuture**

Unlike Future, CompletableFuture provides **non-blocking exception handling** using .exceptionally() or .handle().

**✅ Example: Handling Exceptions in CompletableFuture**

import java.util.concurrent.\*;

public class CompletableFutureExceptionHandling {

public static void main(String[] args) {

CompletableFuture<Integer> future = CompletableFuture.supplyAsync(() -> {

System.out.println(Thread.currentThread().getName() + " is running...");

if (true) throw new RuntimeException("Something went wrong!");

return 100; // Will never execute

}).exceptionally(ex -> {

System.out.println("Exception occurred: " + ex.getMessage());

return -1; // Provide default value in case of failure

});

System.out.println("Result: " + future.join()); // No need for try-catch!

}

}

**🔴 Output**

ForkJoinPool.commonPool-worker-1 is running...

Exception occurred: Something went wrong!

Result: -1

**🔹 Key Differences Between Future and CompletableFuture**

| **Feature** | **Future** | **CompletableFuture** |
| --- | --- | --- |
| Exception Handling | try-catch with future.get() | .exceptionally() or .handle() (Non-blocking) |
| Blocking Nature | Blocks on future.get() | Non-blocking with async callbacks |
| Multiple Callbacks | Not possible | Supports .thenApply(), .thenAccept(), .thenRun() |
| Chaining Tasks | Not possible | Supports chaining multiple async operations |

**✨ Benefits of CompletableFuture**

1. **Non-blocking execution** – No need to wait with .get().
2. **Built-in exception handling** – Use .exceptionally() or .handle().
3. **Better performance** – No need to block threads while waiting for results.
4. **Supports asynchronous operations** – Can chain multiple tasks together.

**🔥 Alternative: Using .handle() for More Control**

CompletableFuture<Integer> future = CompletableFuture.supplyAsync(() -> {

throw new RuntimeException("Task failed!");

}).handle((result, ex) -> {

if (ex != null) {

System.out.println("Handled Exception: " + ex.getMessage());

return -1;

}

return result;

});

System.out.println("Final Result: " + future.join());

☝ Here, .handle() gives **both the result and exception**, so you can decide how to handle errors.

**❓ Do you need an example of chaining multiple tasks in CompletableFuture? 🚀**