**✅ Future and CompletableFuture in Java**

Both Future and CompletableFuture are used for **asynchronous programming** in Java, but CompletableFuture provides more flexibility and advanced features.

**1️⃣ Future in Java**

* Introduced in **Java 5** (java.util.concurrent.Future)
* Represents the **result of an asynchronous computation**
* Cannot be manually completed
* Provides methods like:
  + get() (blocks until the result is available)
  + isDone() (checks if the task is complete)
  + cancel() (attempts to cancel the task)

**📌 Example of Future**

import java.util.concurrent.\*;

public class FutureExample {

public static void main(String[] args) throws ExecutionException, InterruptedException {

ExecutorService executor = Executors.newFixedThreadPool(2);

// Submitting a task that returns a result

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

Thread.sleep(2000); // Simulate delay

return 42;

});

System.out.println("Task submitted. Doing other work...");

// Blocking call (waits for result)

Integer result = future.get();

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

executor.shutdown();

}

}

**🔴 Limitations of Future**

1. **Cannot handle dependent tasks** (No chaining of tasks)
2. **Blocking get() method** (Waits for completion)
3. **No manual completion** (Once submitted, we can't modify it)

**2️⃣ CompletableFuture in Java**

* Introduced in **Java 8** (java.util.concurrent.CompletableFuture)
* **Non-blocking** and supports **functional programming**
* Allows **manual completion**
* Provides methods like:
  + thenApply(), thenAccept(), thenRun() (for chaining)
  + supplyAsync() and runAsync() (for asynchronous execution)
  + whenComplete(), exceptionally() (for error handling)
  + combine(), compose() (for dependent computations)

**📌 Example: CompletableFuture (Non-Blocking)**

import java.util.concurrent.CompletableFuture;

public class CompletableFutureExample {

public static void main(String[] args) {

// Run an async task

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

try { Thread.sleep(2000); } catch (InterruptedException e) {}

return 42;

});

// Continue without waiting

System.out.println("Task submitted. Doing other work...");

// Register a callback (non-blocking)

future.thenAccept(result -> System.out.println("Result: " + result));

try { Thread.sleep(3000); } catch (InterruptedException e) {} // Wait for task to finish

}

}

**✅ Key Differences Between Future and CompletableFuture**

| **Feature** | **Future** | **CompletableFuture** |
| --- | --- | --- |
| **Introduced In** | Java 5 | Java 8 |
| **Chaining Tasks** | ❌ No | ✅ Yes (thenApply, thenAccept) |
| **Non-blocking Calls** | ❌ No (get() blocks) | ✅ Yes (thenApply(), thenAccept()) |
| **Manual Completion** | ❌ No | ✅ Yes (complete()) |
| **Exception Handling** | ❌ No | ✅ Yes (exceptionally(), handle()) |
| **Combining Futures** | ❌ No | ✅ Yes (thenCombine(), thenCompose()) |

**3️⃣ Advanced CompletableFuture Features**

**📌 Chaining Asynchronous Tasks**

CompletableFuture.supplyAsync(() -> "Hello")

.thenApply(result -> result + " World")

.thenAccept(System.out::println);

**📌 Combining Two Futures**

CompletableFuture<Integer> future1 = CompletableFuture.supplyAsync(() -> 10);

CompletableFuture<Integer> future2 = CompletableFuture.supplyAsync(() -> 20);

CompletableFuture<Integer> result = future1.thenCombine(future2, (x, y) -> x + y);

System.out.println(result.join()); // 30

**📌 Handling Errors Gracefully**

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

if (Math.random() > 0.5) throw new RuntimeException("Something went wrong");

return 42;

}).exceptionally(ex -> {

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

return -1;

});

System.out.println(future.join()); // Either 42 or -1 (in case of exception)

**4️⃣ Common Interview Questions**

1. **What is the difference between Future and CompletableFuture?**
2. **How do you handle exceptions in CompletableFuture?**
3. **How do you chain multiple asynchronous operations?**
4. **What are thenApply(), thenAccept(), and thenRun()?**
5. **How do you combine multiple futures in Java?**

**✅ Conclusion**

* Use **Future** when you only need **basic async execution**.
* Use **CompletableFuture** for **non-blocking**, **chaining**, and **error handling**.

Would you like examples of ForkJoinPool or real-world applications? 🚀