Future vs CompletableFuture (Java)

1. What is Future?

Future was introduced in **Java 5** to represent the result of an asynchronous computation.
It allows a task to run in the background while the main thread continues.

Limitations of Future:

- get() is **blocking** (waits until task completes).
- Cannot **chain** tasks.
- Cannot **combine** multiple async results.
- Cannot attach **callbacks**.
- Cannot handle **errors** effectively.
- Cannot complete a task **manually**.

Future is simple, but not powerful enough for modern asynchronous programming needs.

2. What is CompletableFuture?

CompletableFuture was introduced in **Java 8** to solve the limitations of Future.
It provides a complete framework for **asynchronous, non-blocking, pipeline-based** programming.

Advantages of CompletableFuture:

- **Non-blocking calls**
- **Chaining tasks** (thenApply, thenCompose)
- **Combining tasks** (thenCombine, allOf, anyOf)
- **Callbacks** (whenComplete, thenRun)
- **Error handling** (exceptionally, handle)
- **Manual completion** (complete, completeExceptionally)
- Clean, modern async code
- Excellent for **parallel programming**, **microservices**, **API aggregation**, **database calls** etc.

3. Why was CompletableFuture introduced if Future already existed?

Because **Future was too limited**.

Modern applications require:

- Non-blocking pipelines
- Parallel API calls
- Combining multiple responses
- Microservice interactions

- Better error handling
- Efficient async processing

Future did not support any of these.
CompletableFuture was designed to bring a powerful, flexible, and non-blocking async model.

4. Future vs CompletableFuture (Comparison Table)

Feature	Future (Java 5)	CompletableFuture (Java 8)
Nature	Represents async result	Full async programming framework
Blocking?	Yes (get() blocks)	No (async callbacks)
Chaining Tasks	Not supported	Yes (thenApply, thenCompose)
Combining Tasks	Not supported	Yes (thenCombine, allOf)
Callbacks	No	Yes (thenRun, whenComplete)
Error Handling	Very limited	Rich: exceptionally(), handle()
Manual Completion	Not allowed	complete(), completeExceptionally()
Real-world Use	Simple background jobs	Microservices, APIs, DB calls, parallel tasks