CompletableFuture Core Methods

Description	Example
Starts an asynchronous task that returns a result using `supplyAsync(Supplier<T> task)`.	`CompletableFuture.supplyAsync(() -> "Hello");`
Starts an asynchronous task without returning a result using `runAsync(Runnable task)`.	`CompletableFuture.runAsync(() -> System.out.println("Task"));`
Transforms the result of a computation using `thenApply(Function<T, R> fn)`.	`future.thenApply(str -> str.toUpperCase());`
Consumes the result without returning a new CompletableFuture using `thenAccept(Consumer<T> action)`.	`future.thenAccept(res -> System.out.println(res));`
Combines results from two tasks using `thenCombine(CompletableFuture<U> other, BiFunction<T, U, R> fn)`.	`future1.thenCombine(future2, (res1, res2) -> res1 + res2);`
Flattens and chains two asynchronous tasks using `thenCompose(Function<T, CompletableFuture<U>> fn)`.	`future.thenCompose(res -> CompletableFuture.supplyAsync(() -> res + " World"));`
Handles both successful results and exceptions using `handle(BiFunction<T, Throwable, R> fn)`.	`future.handle((res, ex) -> ex != null ? "Error" : res);`
Handles exceptions by providing a fallback result using `exceptionally(Function<Throwable, T> fn)`.	`future.exceptionally(ex -> "Fallback Result");`
Waits for all provided futures to complete using `allOf(CompletableFuture<?>... futures)`.	`CompletableFuture.allOf(future1, future2);`
Completes when any one of the provided futures completes using `anyOf(CompletableFuture<?>... futures)`.	`CompletableFuture.anyOf(future1, future2);`
Blocks and waits for the result of the computation using `join()`.	`String result = future.join();`
Completes exceptionally if not finished within the timeout using `orTimeout(long timeout, TimeUnit unit)`.	`future.orTimeout(5, TimeUnit.SECONDS);`
Completes with a fallback value if not finished within the timeout using `completeOnTimeout(T value, long timeout, TimeUnit unit)`.	`future.completeOnTimeout("Default", 5, TimeUnit.SECONDS);`