Achieving Concurrency in Java using Executor Service and Execution Completion Service:

In order to achieve concurrent execution of tasks in Java we use the `ExecutorService`.
In ExecutorService we provide a pool of threads (Cached thread pool, Fixed thread pool etc) and submit tasks to it, the execution of this task in one of the threads of the thread pool and putting it in a response list when execution is done is all taken care of by executor service itself.

There are two types of execution service broadly:

1) Executor Service.
2) Executor Completion Service.

Executor Service : In here a task is submitted to the executor service and a placeholder value is returned at the moment.
Whenever the execution of the task will be finished in the Executor Service it will be stored in that placeholder.
We can then do future.get() call in order to get the result of the executed task.
If execution of the task is done in Executor Service, the value of future will already be set and future.get() will immediately return the result value, but if not then the thread that made the future.get() call will be in a blocked state until it gets the value of the result. (Waste of time!!)

In order to resolve the waste of time issue we have the execution completion service.

Executor Completion Service : Here, say we have submitted 7 tasks to the service, the result of the 6th task is ready but not of the other tasks. We can make a call to the Execution Completion Service and instead of telling it to return the value of a particular future that has the result value, we can tell it to return any future that has already the result value in it.

EXAMPLE :

Variable name	Location in Memory (Memory block address)
Future f1 = executionService.submit(Task1);	X1
Future f2 = executionService.submit(Task2);	X2
Future f3 = executionService.submit(Task3);	X3

executionService.submit will execute Task 1, it returns a future variable whose address is X1.
Once it is done executing the task……it will store the result at address location X1.

Say Task1 and Task3 are executed completely and results are stored in their corresponding locations. Task2 is still executing.

Then the user will do, f1.get() to get the value of future1 which is the result of Task1.
This will return some value.

f2.get(), when this is called, since the value of future is not set yet as the task is still being executed, thus the thread will be blocked and in a waiting state until the value is set at the address X2 and read. Time wasted in waiting

The problem here is that, even though there was a future f3 which was ready with the result, we could not make use of it as we were blocked on f2.get().

The solution to this is Executor Completion Service.

Variable name	Location in Memory (Memory block address)
Future f1 = executionCompletionService.submit(Task1);	X1
Future f2 = executionCompletionService.submit(Task2);	X2
Future f3 = executionCompletionService.submit(Task3);	X3

Instead of using future.get which might result in blocking of thread some time, we can ask the ECS to return any of the futures whose result value is set.

Assume Task1 and Task3 have finished execution and value is set up in the corresponding address locations.

When the execution of the task is finished in ECS, it will add its future to a completionQueue.

Completion Queue :

f1	f3

Thus in order to get a completed future, we can pop from this queue.

ExecutionCompletionService.take() => This will pop up the f1 from the queue.
ExecutionCompletionService.take() => This will pop up the f3 from the queue.

Thus the problem where we called f2 before f3 even though execution of f2 was still on is solved in ECS.

ExecutionCompletionService.take() => Nothing in the queue. Will be in blocked state until any future is added to the queue (when a task is completed).

ECS has 3 separate ways of extracting value from the Completion Queue :

- .poll() method returns the head of the blocking queue if exists, otherwise returns null immediately.

- .poll(timeout) method returns the head of the blocking queue if it exists, if any entry arrives to the queue within the time out it will be returned, otherwise will return null.

- .take() method on a blocking queue will return the head of the queue, if the queue is empty, it will block the thread until the queue has something to return (Thread will be put to sleep and notified when there is an entry added in the queue). This method will never return null and keep waiting unexpectedly.

NOTE : This much info should be enough for the understanding of the possible solutions document.

Oracle Official Document Links :

- Execution Service : https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html

- Execution Completion Service : https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorCompletionService.html