Future vs. CompletableFuture

CompletableFuture is used for asynchronous programming in Java. Asynchronous programming is a means of writing non-blocking code by running a task on a separate thread than the main application thread and notifying the main thread about its progress, completion, or failure.

1. Manual Completion

future provides an isDone() method to check whether the computation is done or not, and a get() method to retrieve the result of the computation when it is done. But if there comes a scenario where you need to complete it manually, Future does not provide any means to do so.

But in Java 8’s completableFuture, CompletableFuture.complete() method helps us to manually complete a Future.

To fetch the result, you can use get() method.

String result = completableFuture.get();

Since we know that get() method blocks until the future is complete, the above call will block forever since the future is never completed. Therefore, we can use complete() method in order to manually complete the result.

2. Attaching a callable method

While using Future, we do not get notified when it is complete neither does it provides us a callable method which will automatically be called when the result is available but CompletableFuture provides us with a lot of callable methods which can be used as per our use case.

* runAsync()

It is used for running some background tasks asynchronously but not returning anything by using a Runnable instance. It takes a Runnable object and returns CompletableFuture.

* supplyAsync()

This method is used when you want to return some value from the background task running asynchronously. It takes a Supplier and returns CompletableFuture.

* thenApply()

It takes a Function as an argument. A function is a functional interface that represents a function that takes argument of type T and returns the argument of type R.

* thenAccept()

It takes a Consumer and returns CompletableFuture. It has access to the result of the CompletableFuture on which it is attached.

* thenRun()

It also takes a Consumer and returns CompletableFuture. If you neither need the value of the computation nor want to return some value at the end of the chain, then you can pass a Runnable lambda to the thenRun() method. Hence, in this case, we do not have access to future’s result.