**19. what is CompletableFuture ?**

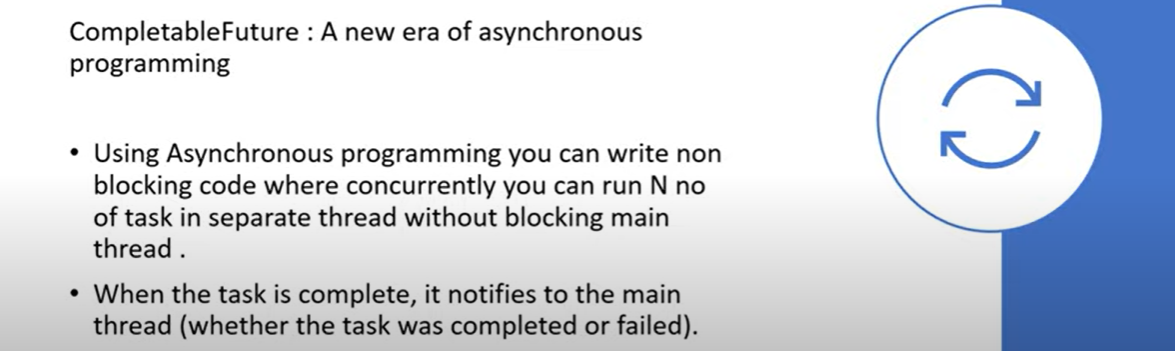
**CompletableFuture** is an enhance of a Future object which introduced in **jdk** **5**. CompletableFuture is used for asynchronous programming in java. Asynchronous programming is a means of writing non-blocking code by running a task on a separate thread than the main application thread and notifying the main thread about its progress, competition and failure.

**20. Why CompletableFuture why not Future ?**

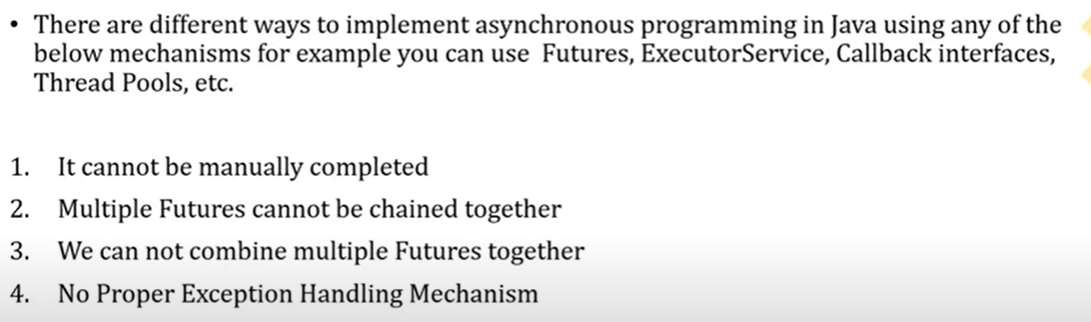
Future object as we discussed is introduced in java 5.

1. **It cannot be manually completed**. We don’t have any option to forcefully complete. Future object once you execute until and unless you will not call **.get()** method it will not complete the execution. It will keep blocking your thread, it cannot release the result until you call **.get()** method.
2. **Multiple Futures can not be chained together**. Let’s say I created 5 Future objects I can not chained future 1 that do something future 2 that do something like using the Future, but we can do using the CompletableFuture object.
3. **You cannot combine multiple Futures together**. Let’s say I have created 5 future objects. future 1, future 2…5 future objects. After getting all response I just want to consolidate a final response back to the API, I can’t do it using Future object. Because we don’t know which thread will execute first and which thread will take long time to give the result.
4. **No Exception Handling**. In Future implementation there is no proper exception handling.

**What is CompletableFuture ?**



**Why CompletableFuture ?**

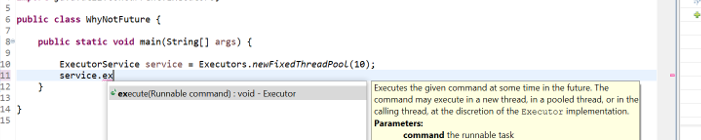


🡪**It Can not be manually completed?**

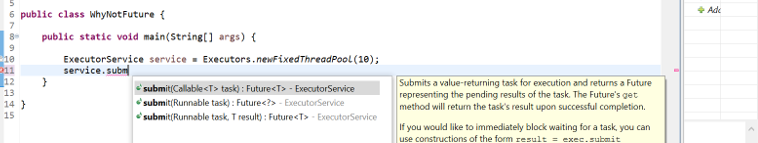
**What is the difference between submit() and execute() method of Executor Framework ?**

ExecutorService service = Executors.*newFixedThreadPool*(10);

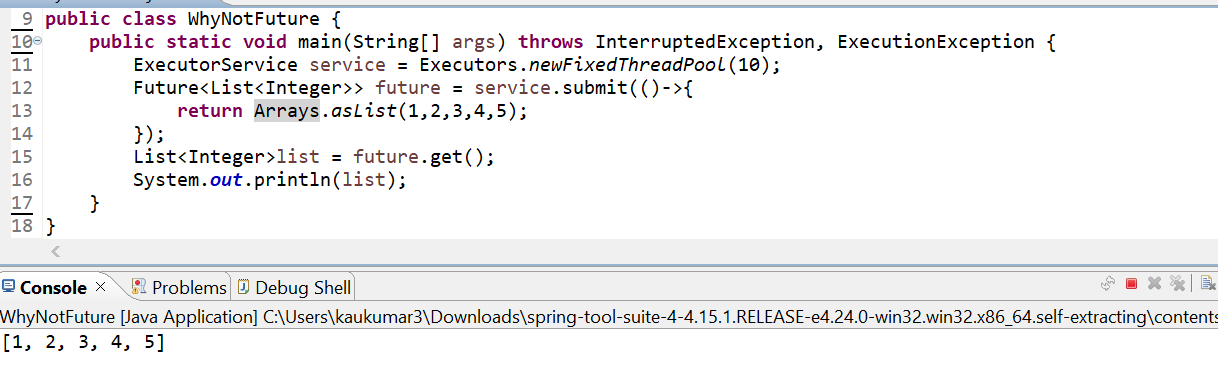
* + - * service.submit()
      * service.execute()



------------------------------------------------------------------------------



If u can see the method arguments **execute**() takes the argument as a **Runnable** where **submit**() takes the method argument as a **Callable**. Runnable and Callable both are a functional interface which contain abstract method like **void run()** and **V call(),** where Callable return some value so we will use in our **CompletableFuture**.

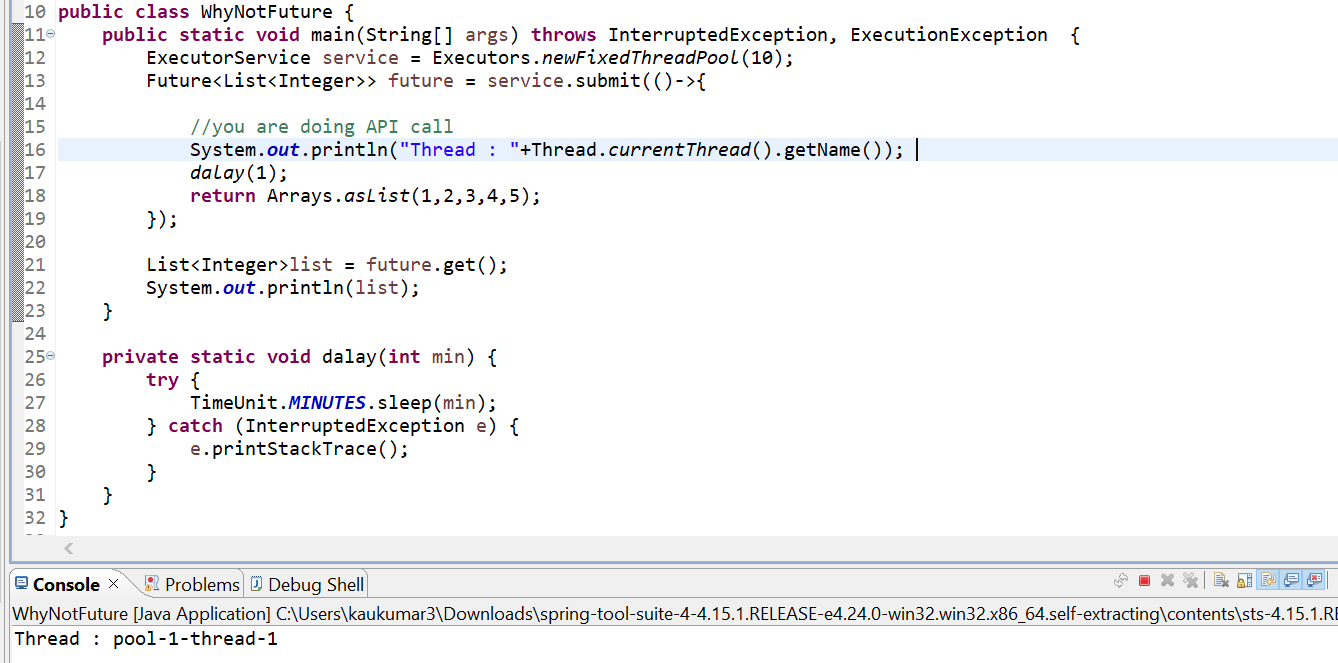


I am directly getting the response here.

Let’s assume from this I am doing some API call. Just assume API is just taking 1 minute time to give you response back.

Now in this case the main thread will block for 1 minute. I am just manually specifying here to block the thread to show you we can’t complete the future object manually.

Now in this scenario it will keep blocking your thread until and unless its execution is not completed.

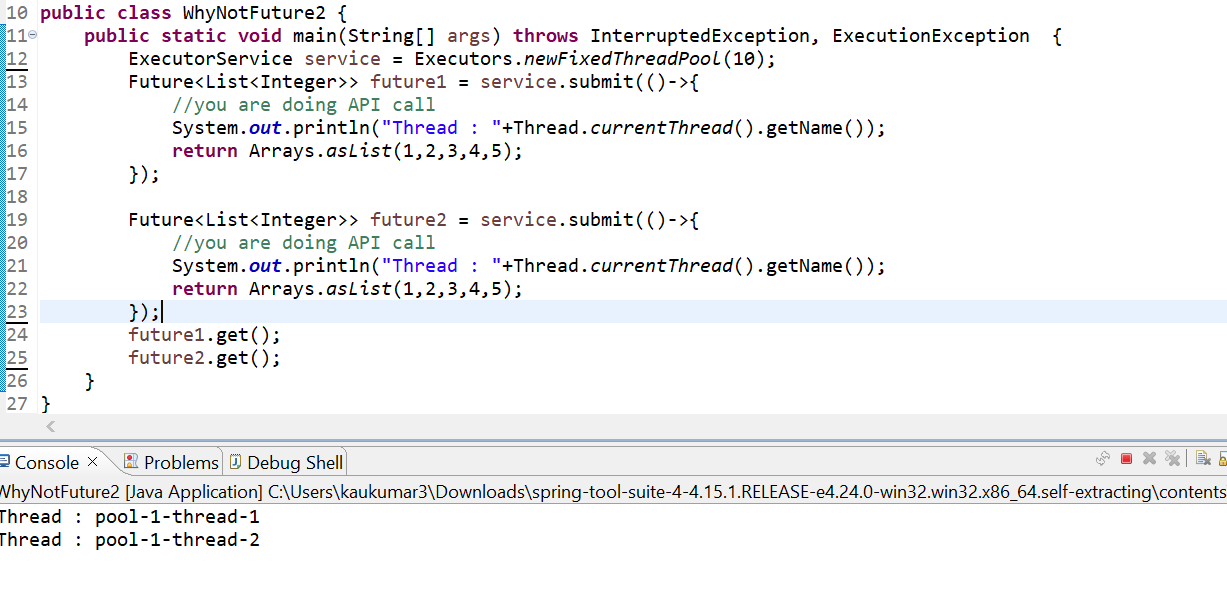


So, if you observed that after one minute, we are getting a response and Future will return the response back.

So, in this case tis thread-1 will completely blocked for one minute. That is what even if we want, we can not manually complete it forcefully. This is what the one disadvantages of future object.

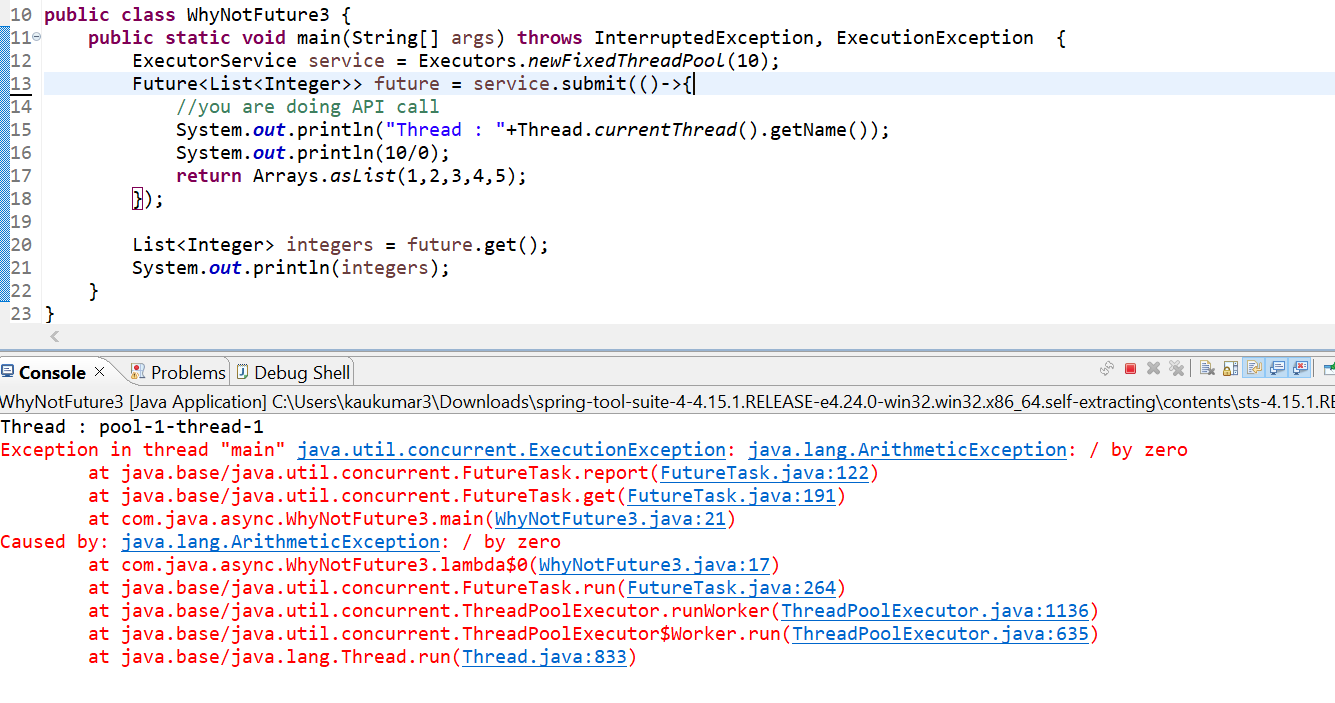
**-> We cannot combine multiple futures together**

Now we have a 3 future objects. Now I can run it separately **future1.get()**, **future2.get()…** I can’t join these 3 like **future1+future2+**…. or **future1.future2…**… bcz there are no methods available for that. I mean I just want to combine all the 3-thread execution that is not possible.



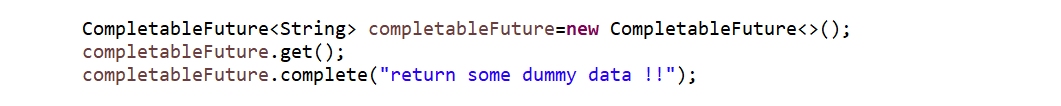
So, using future not possible to run multiple future thread simultaneously. So, in this case if I want to execute future 1 so I will just directly go to **future1.get().** Similarly,if I want to execute future 2 so I will directly go to **future2.get().**

**-> No Exception Handling mechanism provided in Future Object**



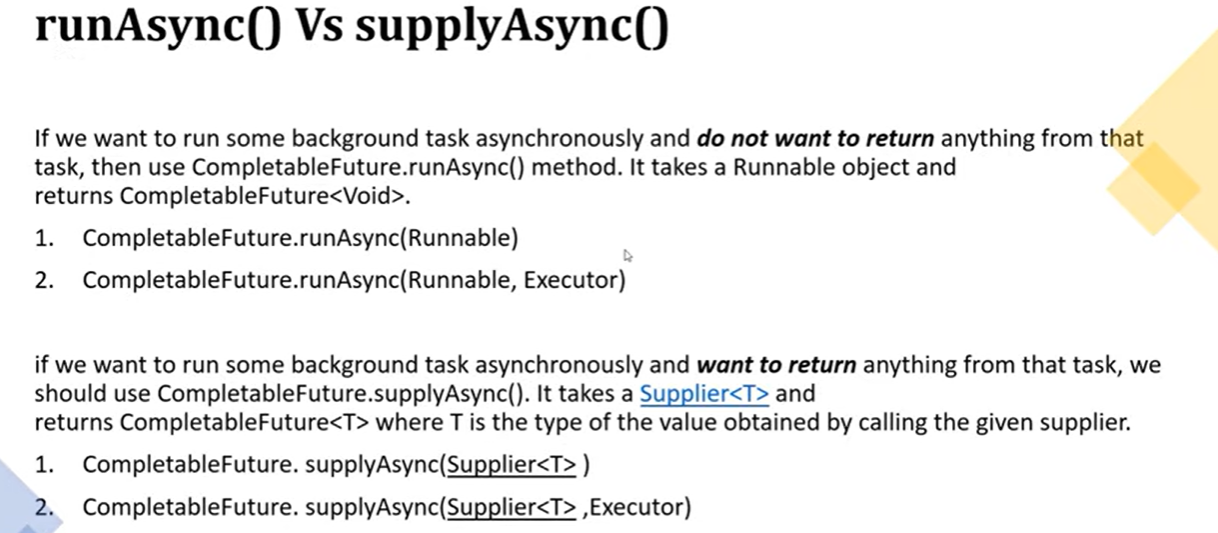
We don’t have any option to handle the future object.

**How to create CompletableFuture object ?**



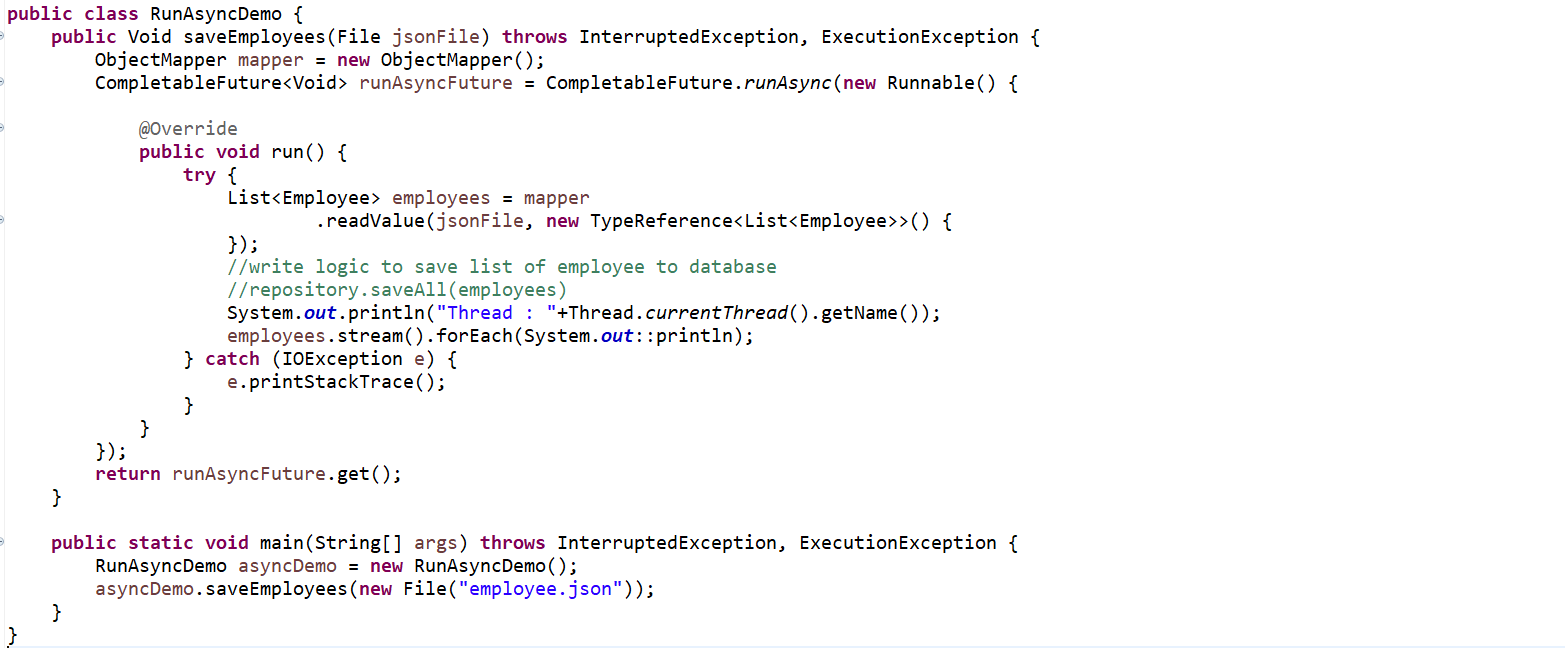
In this completable future we can call the **get**() method. Now once we call the **get**() method again it will block the main thread until and unless this completable future does not succeed. It will still block my thread to avoid that there is a feature called **complete**() method. If thread is taking a long time, then forcefully you can complete it and return some dummy data.

**What is runAsync() & supplyAsync() method? How to use it?**

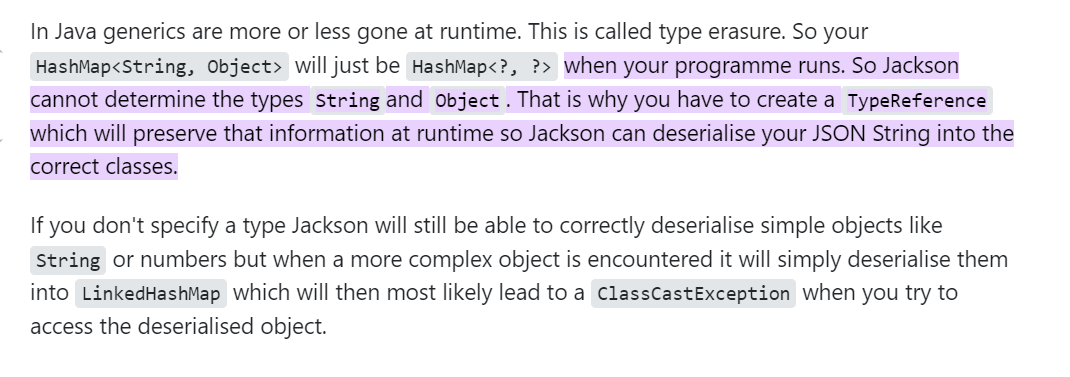


* If you want to execute your thread and if you are not expecting any return type after your thread execution, then you can go with **runAsync**() method. The return type of **runAsync**() method is **CompletableFuture**<Void>, it will not return anything after the thread execution. But if you are expecting some return type after your thread execution then you should go for **supplyAsync**() method.
* If you see the method signature of **runAsync()** it takes an argument as **Runnable** and another one is the overloaded method of **CompletableFuture,** and overloaded method takes another argument as **Executor**. Similarly, if you observed the signature of **supplyAsync()** methodit takes an argument of **Supplier** because you are expecting some return type after thread execution and another one is the overloaded method takes another argument as **Executor**.
* If you use **runAsync**(Runnable) if u are not giving any executor, then we need to understand how this **CompletableFuture** will get the thread to execute my code. So, in that case **CompletableFuture** will get the thread **f**rom Fork Join Global pool. It will get the thread **f**rom Fork Join pool, and it will execute the code.
* If you are giving executor mean your custom executor, then **CompletableFuture** will get the thread **f**rom custom executor not from the fork join pool that’s the reason they provided 2 overloaded methods. In case you are not giving your executor object then it will get the thread from the fork join pool and it will start executing your code.

We need to use this runAsync() method if u are not expecting any return type after your thread execution. Let’s say I have some file and I want to extract that file and I want to process that object to my database, and after that I can directly verify the data into the database rather than giving some return statement from thread.in that case you can go for runAsync() method().

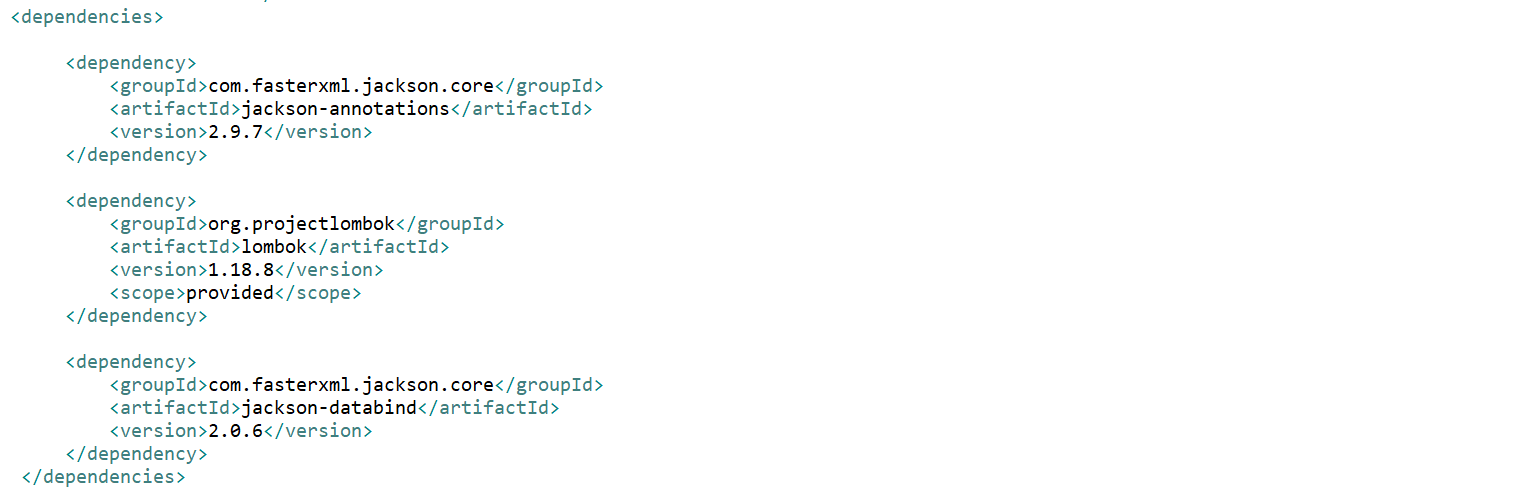


When we will get only a single json we can use **Entity**.**class** instead of **TypeReference** otw for multiple json better to use TypeReference.

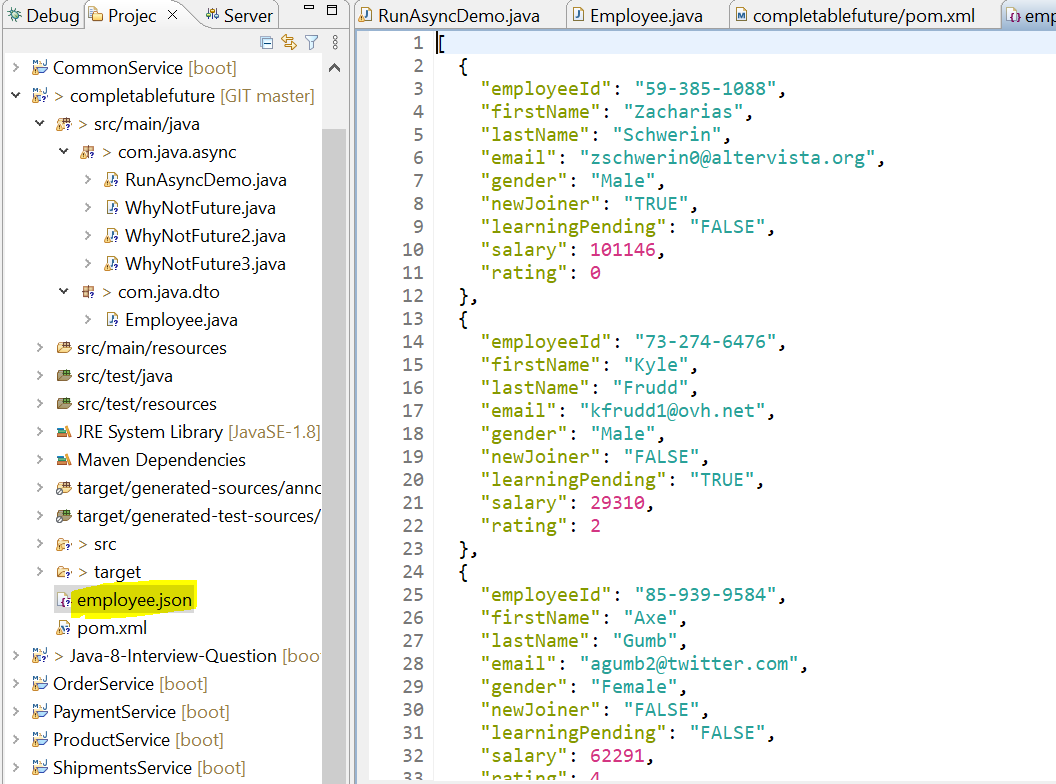




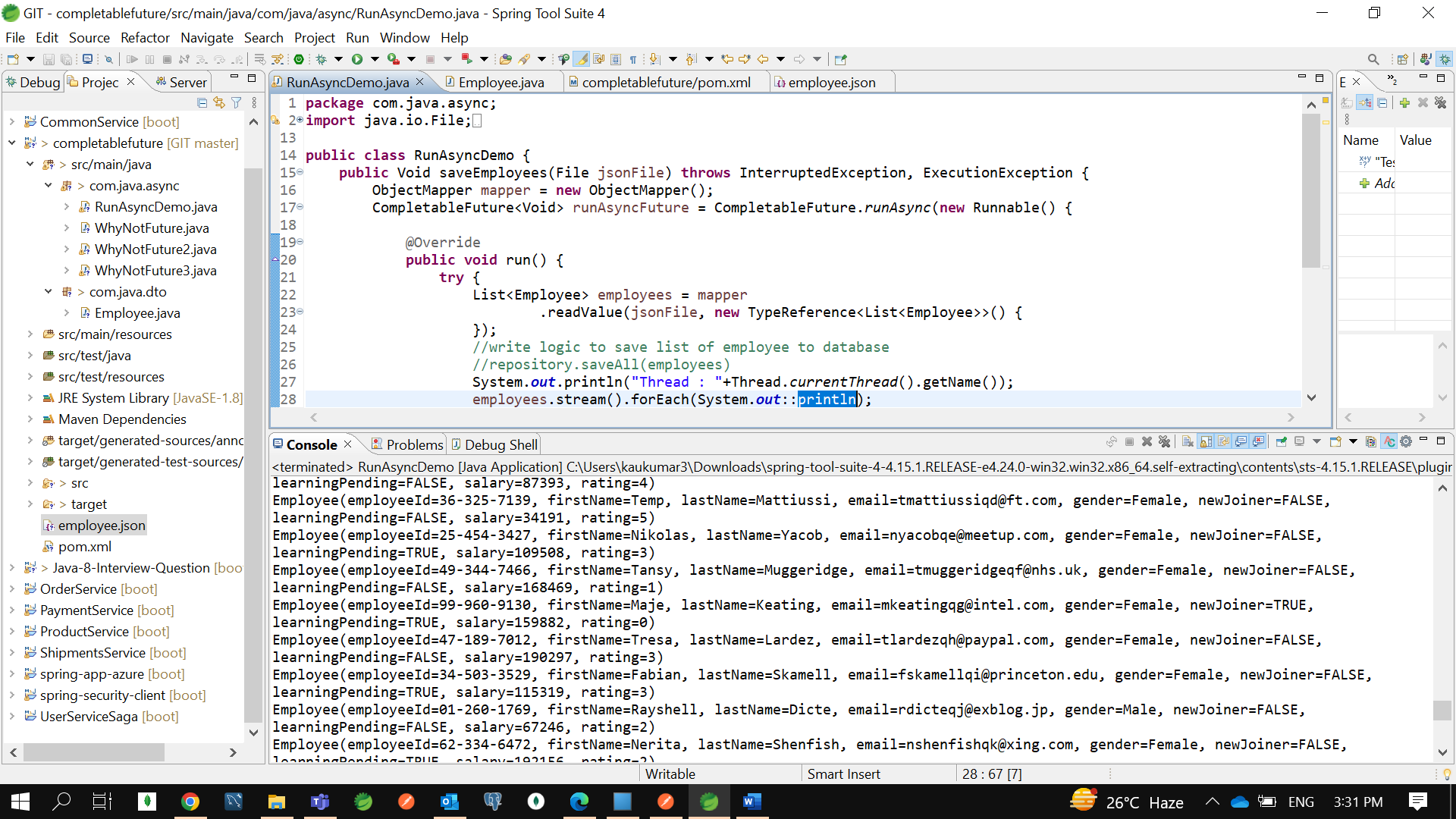
To read json file and convert json to java object we need to add dependencies Jackson-core and Jackson-databinding.



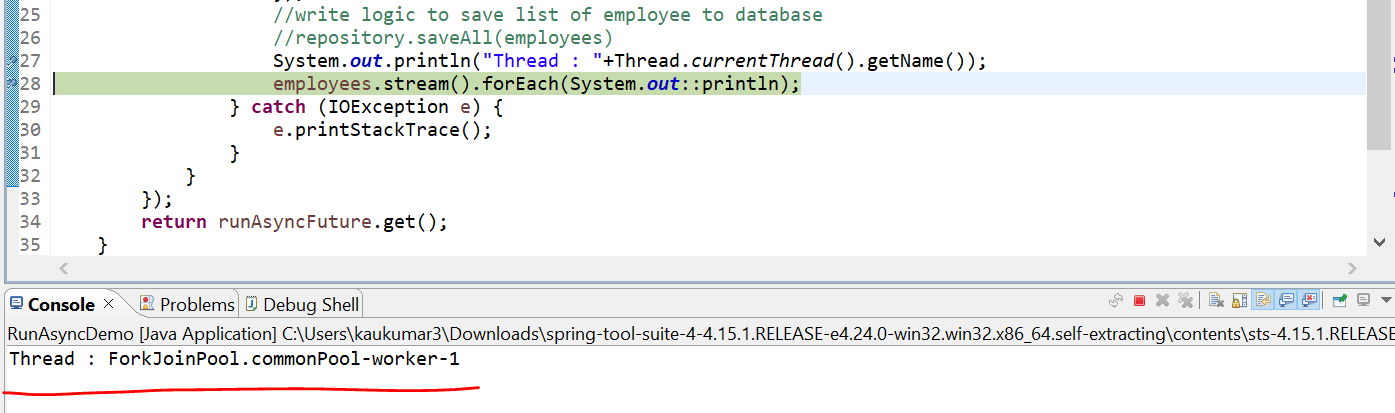
I have an **employee**.**json** object which contains 1000 of employee objects.



So, we can see we are getting the output here.



Now we are getting 1000 employee objects. If u can see it is getting the Thread from the fork join pool- common pool. That is what there is one global fork join pool where it will get a thread and it will start executing your code.



So, worker-1 thread is getting from the common thread pool by CompletableFuture. If you don’t want that you need to provide your own Executor framework. That’s the reason there is 2 overloaded methods.

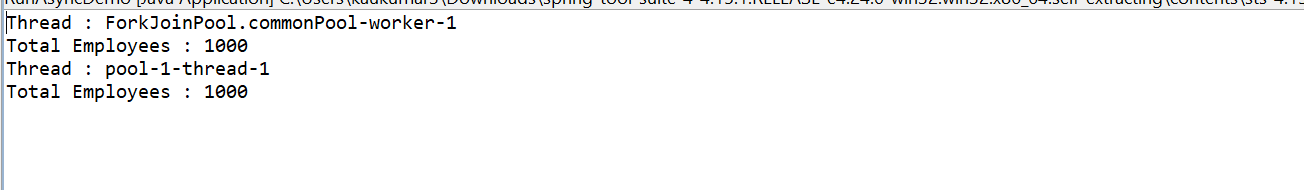
Now since we already know how to write lambda, so instead of this complex logic we can write lambda expression.

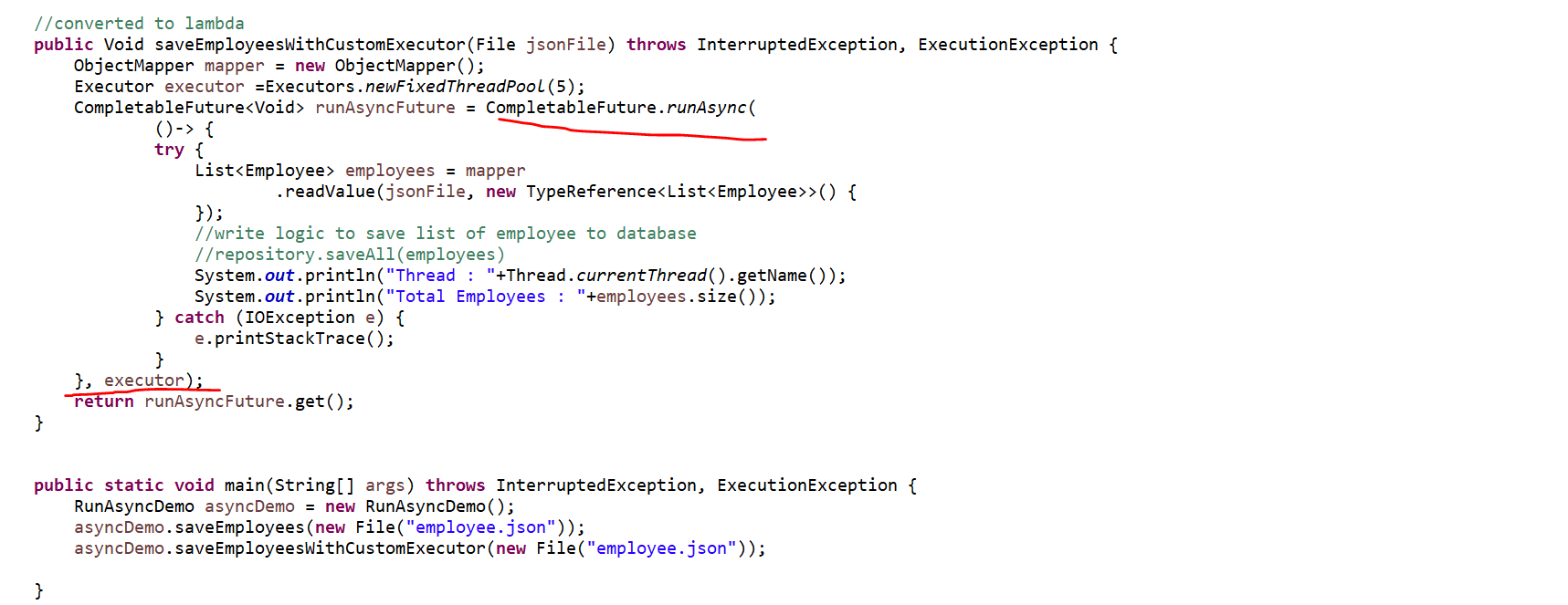
So, let’s write another method here.

Earlier one was Anonymous implementation of Runnable and let’s do using lambda expression.

So, if you observed there is 2 overloaded methods **runAsync(Runnable)** with runnable and **runAsync(Runnable, Executor)** runnable as well as executor framework.

In first method we didn’t provide any custom executor that the reason it got the thread from the fork-join pool that is global pool. Let’s use the overloaded method I mean second method. So, second method will get the thread from my custom executor, and it will start executing. Rather than looking into the fork join global pool.

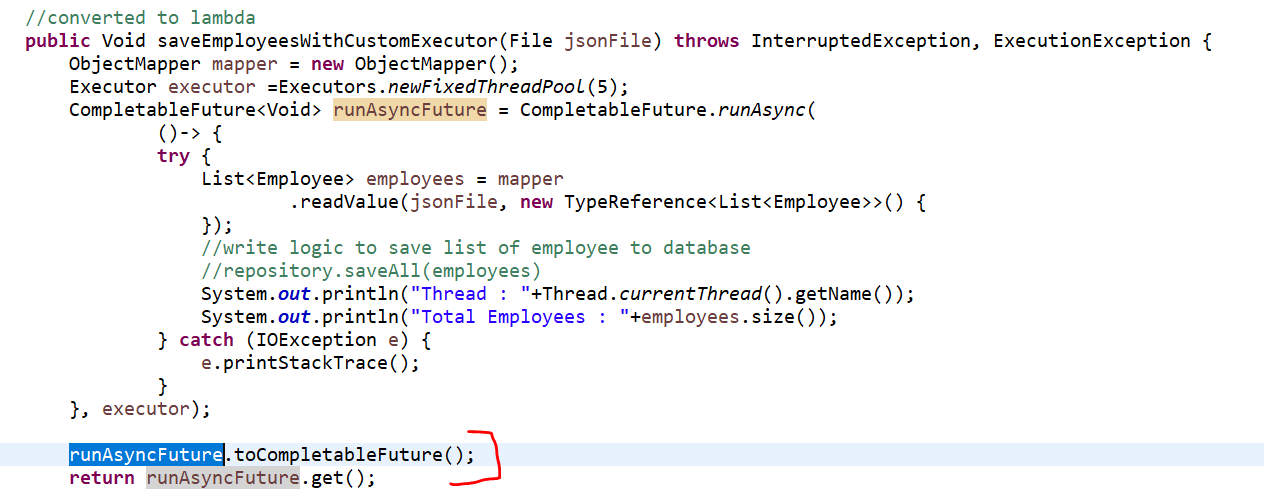




So, if u provide your own executor, then it will get the thread from there if not then it will get thread from the global pool and start executing.

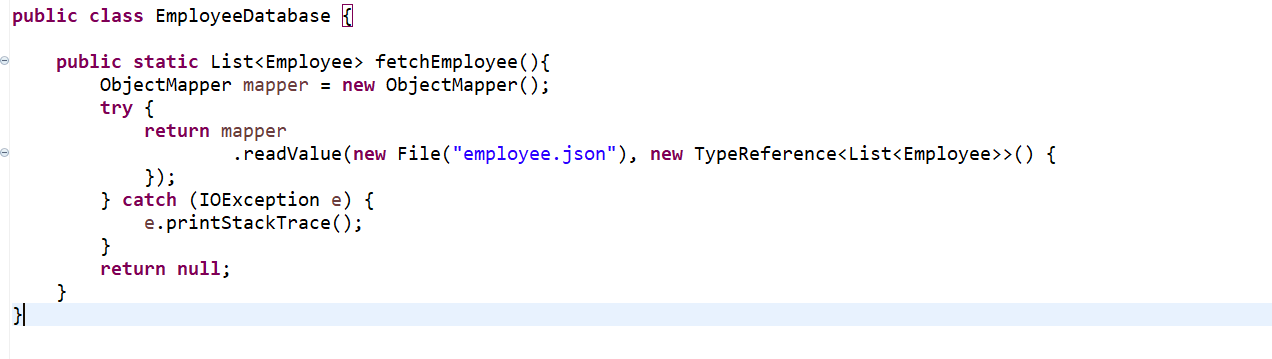
If you don’t want to return anything after your thread execution, then happily go with **runAsync**() method because its return type is **CompletableFuture**<Void>.

If you want, you can complete your thread manually. Let’s say this is blocking your code then you can use **runAsync.toCompletableFuture()** to complete the execution if its blocking. So to complete manually you can use **runAsync.toCompletableFuture().**



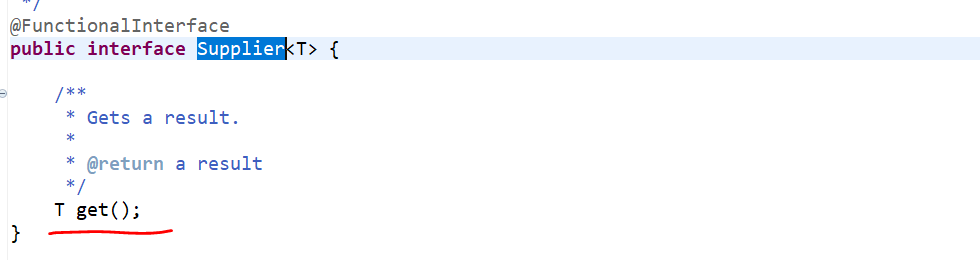
**How we can use supplyAsync() method ?**

Let’s understand how we can use supplyAsync() method where my thread after executing it will return some values. So that going forward we will use that value, or we can pass it to other thread, or we will do some data transformation on that return type or on those results.



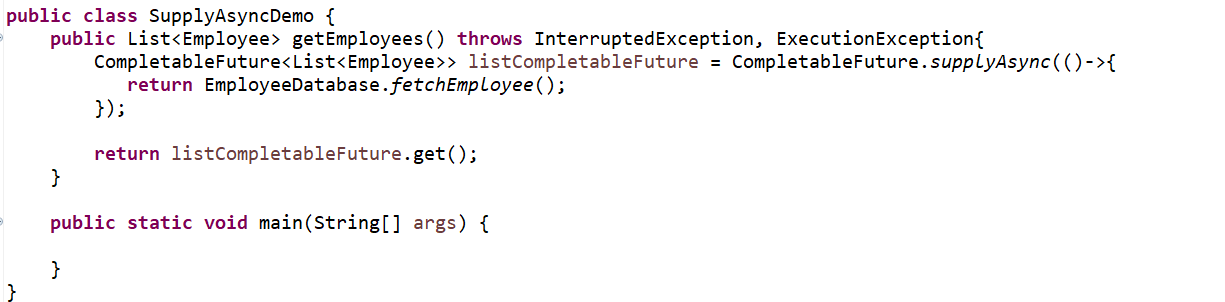
Let’s assume I have a database where we will fetch the list of employee object and my thread will connect to the database and return the list of employee objects. Inside **EmployeeDatabase** we will write same logic what we did in **runAsync**() method.

**supplyAsync**() expecting the argument as **Supplier**



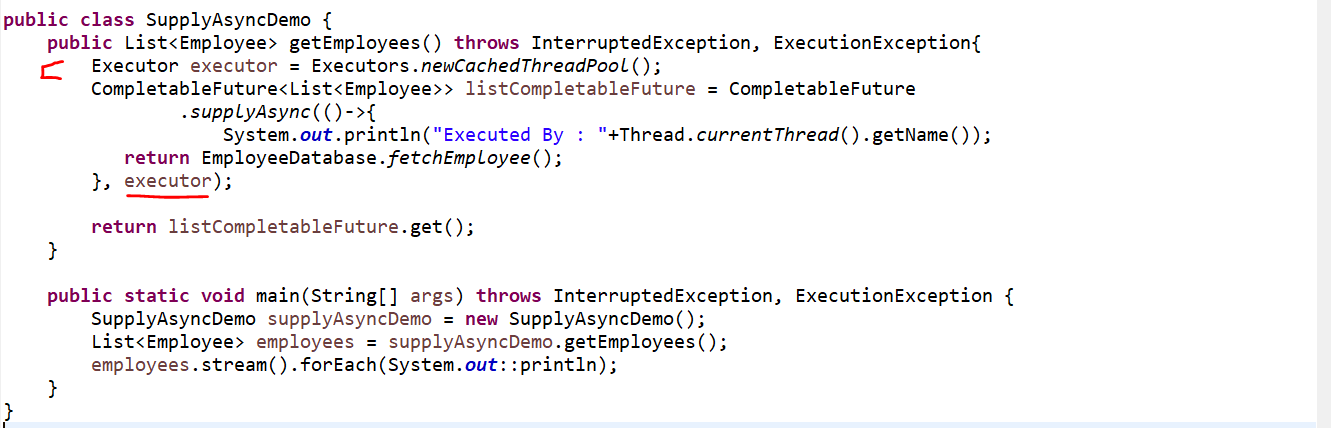
When we will go to supplier, we can see it won’t take any input, but it is expecting something to return

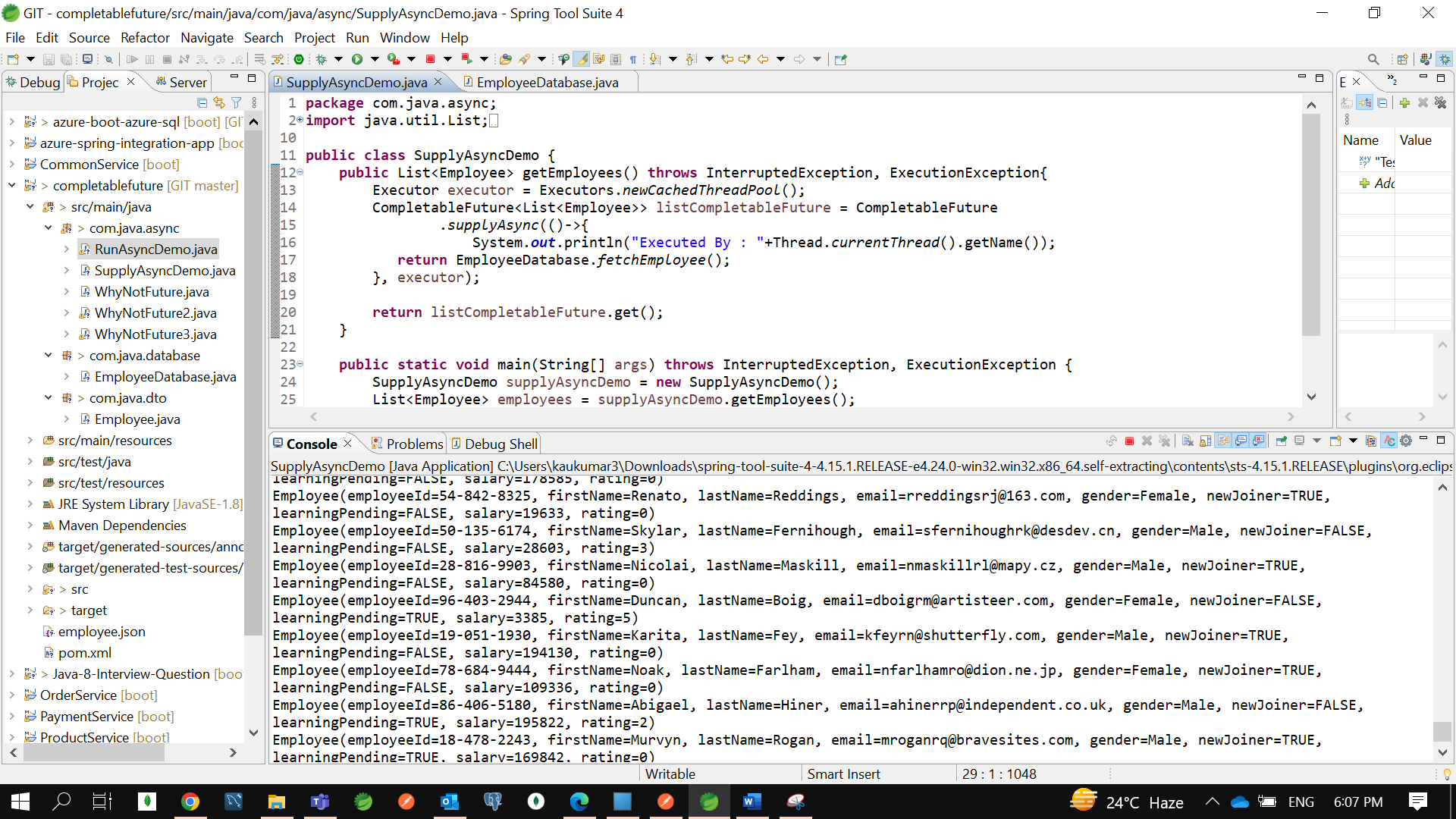
The purpose of using this **supplyAsync**() , after executing the thread I am expecting the result something which is List of Employee. I am just getting this value from the database. if you want to sort that value or if u want to do some data manipulation. That transformation logic you can perform using other methods provided in **CompletableFuture** i.e **thenApply**(), **thenAccept**() and **thenRun**().

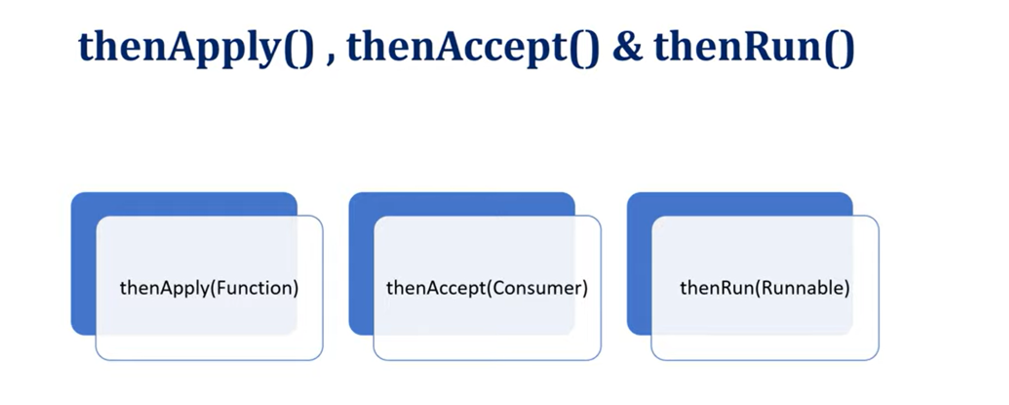


**If you can see this supplyAsync() is also an overloaded method. First is with Supplier and second is the Supplier with Executor.**

**You can use your own executor also.**







In this part we will understand how to chain multiple future together. I mean after getting result from thread execution how we can transform that value or how we can pass that value to another thread for execution.

There is few methods provided by **CompletableFuture** :

* **thenApply**(**Function**)
* **thenAccept**(**Consumer**)
* **thenRun**(**Runnable**)

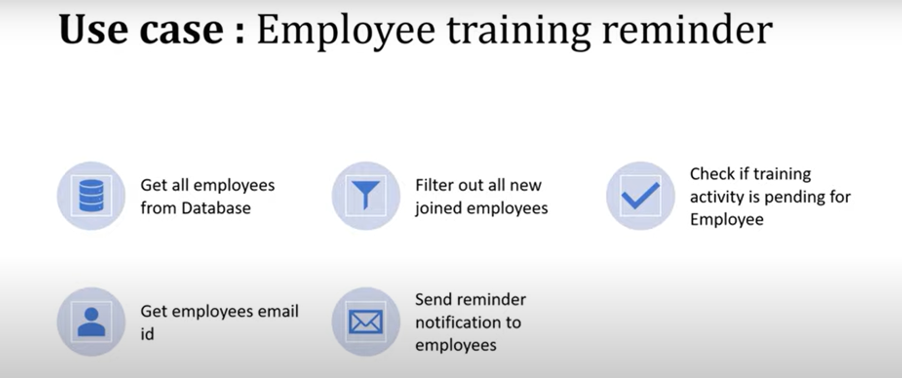
if you see the arguments of each method :

**thenApply()** takes the argument as Function which means it will take the input from you or after thread execution it will get the input and it will do data manipulation then it will return something.

**thenAccept ()** takes the argument as Consumer, and as you know Consumer is a Functional Interface. It will always take input, but it won’t return anything.

**thenRun()** if you see thenAccept() and thenRun() both are a same kind of methods. But the argument is only different. In case of thenRun() it will take input as a runnable. In both the methods it won’t return anything.

So, you can use anyone either thenAccept() or thenRun() based on your need.

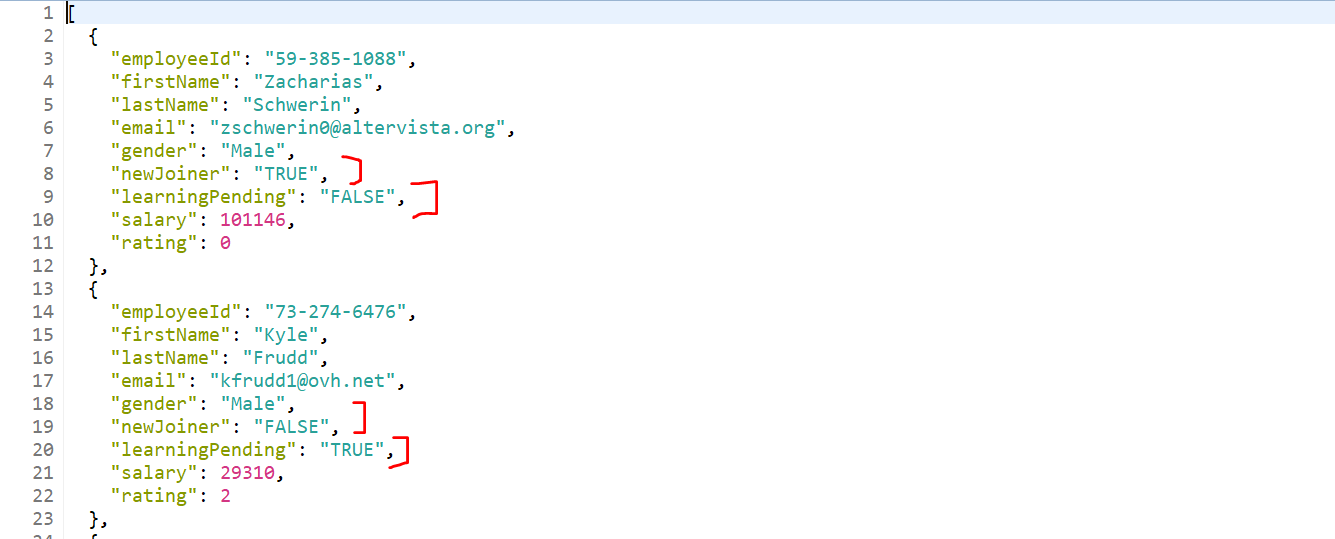


**If you can see here, there is total of 5 tasks. Each task I want to execute in a separate thread.**

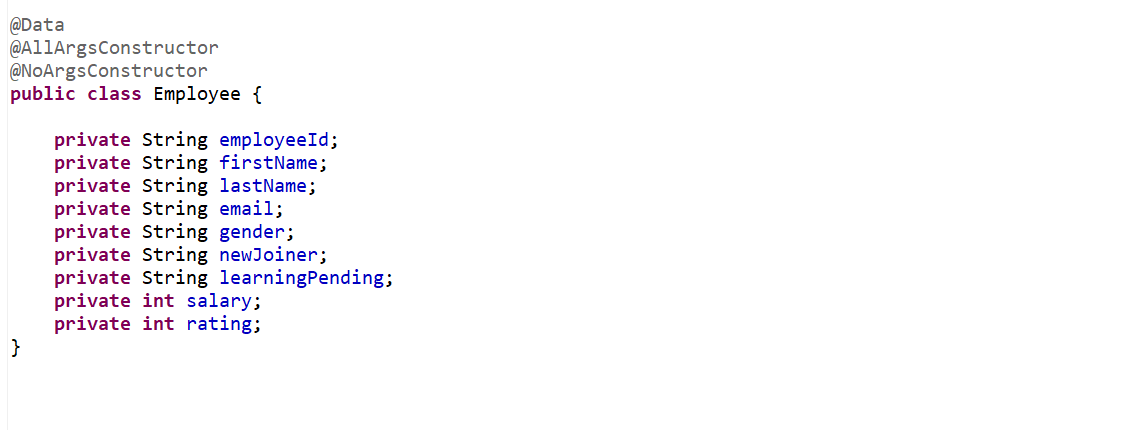
* First Thread will get the all-employee data from database, and it will pass the list of employees to the second thread.
* Second thread will filter out all the new joined employees and again list of new joined employees will pass to the next thread.
* Third thread will check if the training activity is pending for new joined employees, it will filter all the list of employees who didn’t complete the training activity.
* Fourth thread will again extract the email id from the list of employee id or list of employees who didn’t complete the training activity.
* Finally, Fifth thread will send the notification to the employees.

First thread will execute, and it will get the result and return to the second thread, then second thread will execute and get result and returned that result to the third thread …this is how I will just attach callback to the next CompletableFuture. That is why I can achieve using **thenAccept**(), **thenApply**() and **thenRun**() method.

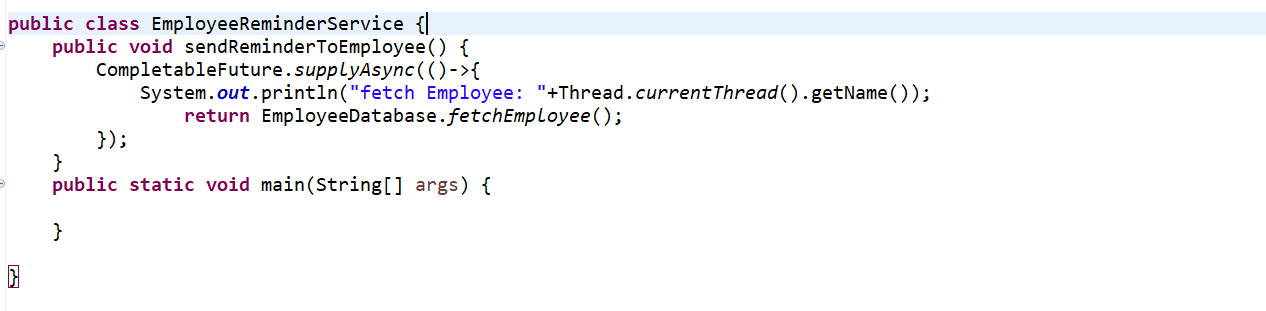
I am going to use the same code base which we have used already for **supplyAsync**() and **runAsync**() and the same json employee.json **raw** data I am going to use. I can filter out either the employee is a new joiner or not. Or either that employee learning activity is pending or not. That is what the 2 fields I have used in our json directly we can use it.



If you go to Employee class all the fields, we have mapped here.



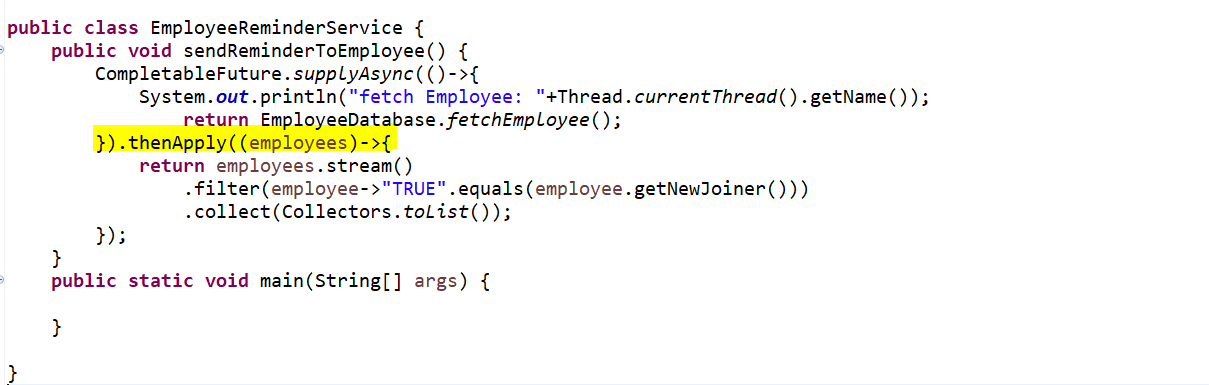
Let’s create a class **EmployeeReminderService.** Now first thread will fetch all the employees from database.so, I can directly use the **supplyAsync().** So, it will take argument as Supplier. It won’t take input but return some output.



This will return the list of employees as per the ppt the next thread I want to filter out who are the new joined.

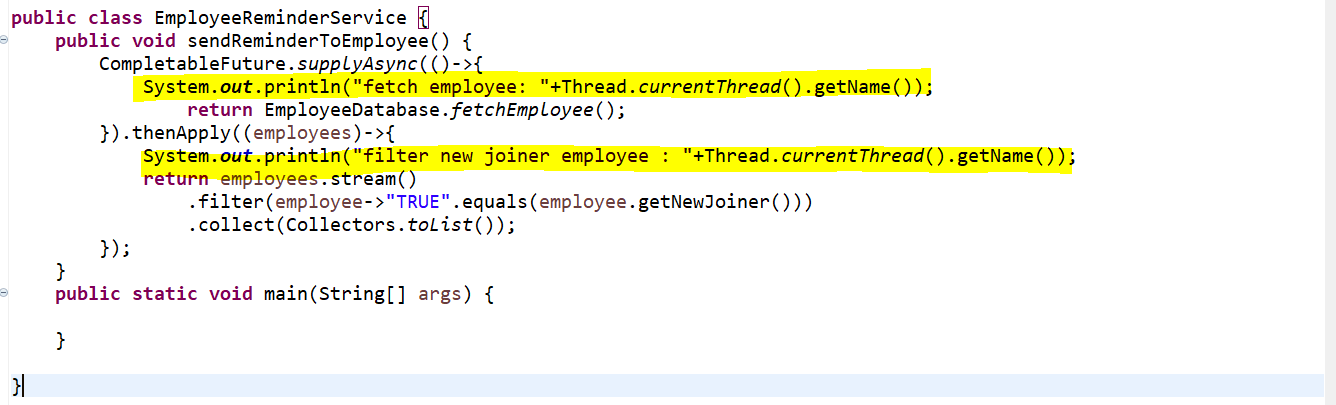
Here I just want to pass the list of employees to another thread.

So, I can directly use **thenApply**() , if u see the argument of thenApply() is **Function**. So, Function will take one input and return output.

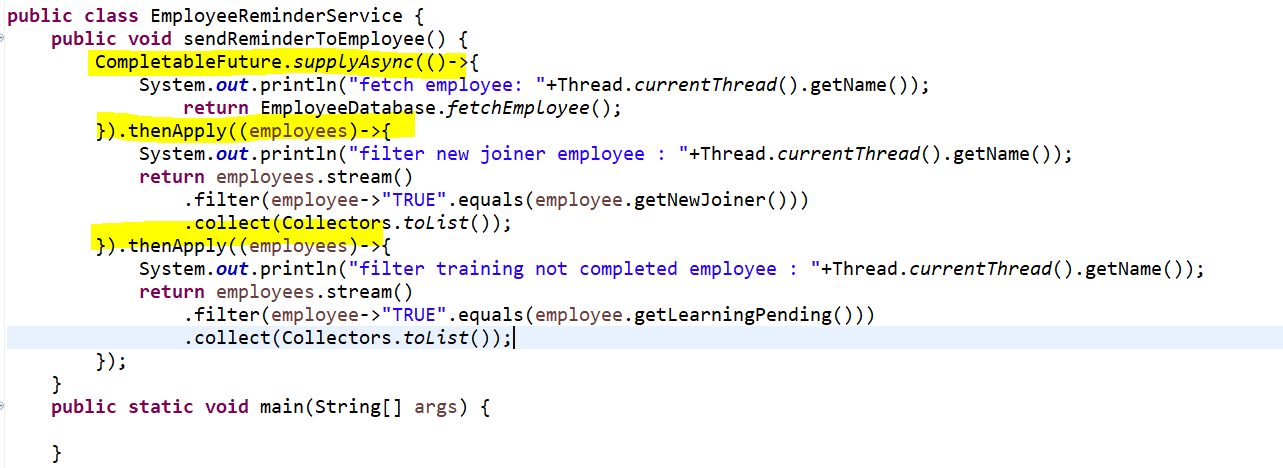


Now first thread will return the list of employee’s objects. Second thread will take a result as an input and just filter out all the employees who is new joined.

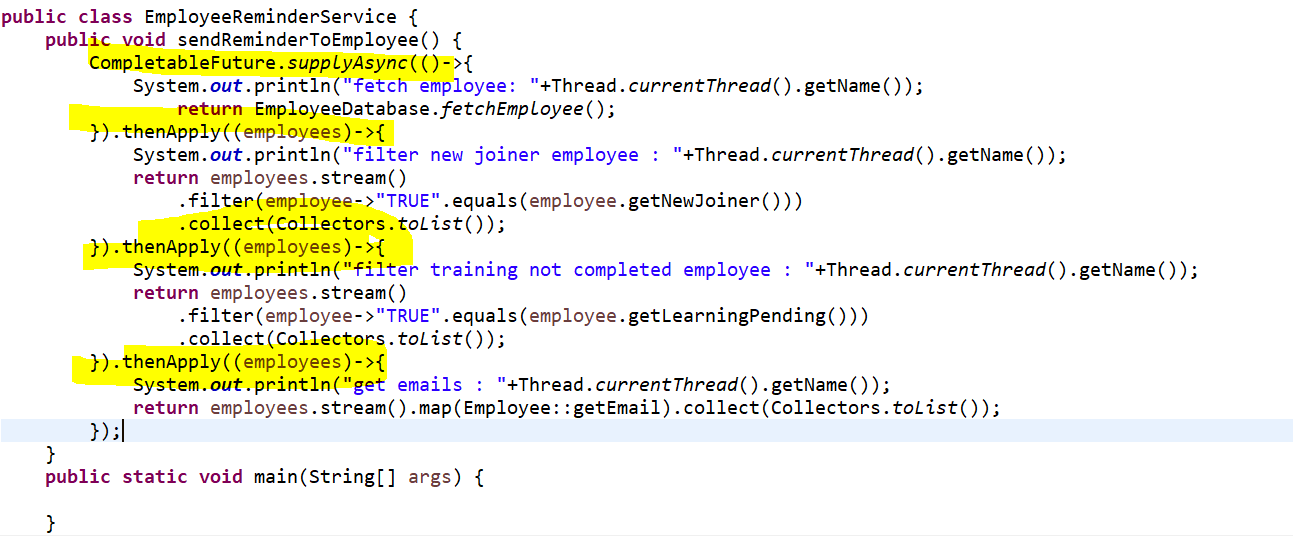
Here also let’s add the thread name so I can show you each logic is execute in a different thread.



Now the next thread will take input as a new joined and filter out who all didn’t complete training activity.

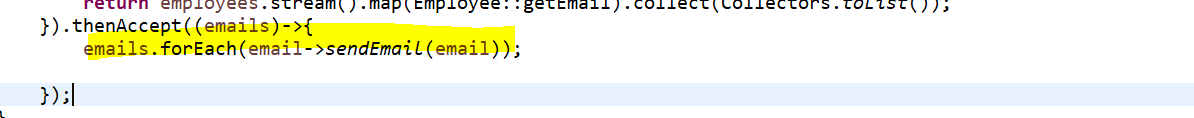


Now the fourth thread will fetch list of employee email from all list of employee.

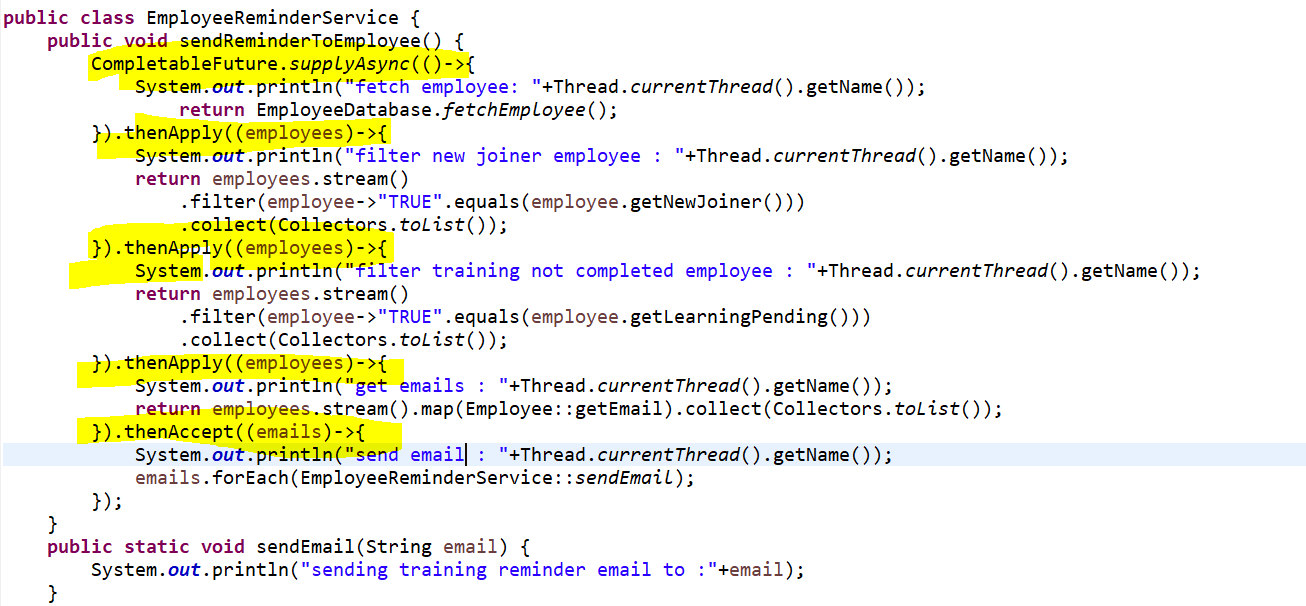


If u observed, I have created 4 threads **thenApply**() 3 times. I used thenApply() because after getting the result from these thread execution I just want to take that as an input, and I just want to do a data manipulation and again I just want to return employee to the next thread call or a next **CompletableFuture**.

Now once I get an email, I just want to trigger email mechanism in my code. So, in this case while sending a mail I don’t want to return anything.so, here either I can go for **thenAccept**() or **thenRun**(). If u don’t want to return anything after execution u can go for anyone. Now I am using here thenAccept() which take input as a Consumer. So, consumer will take a input but no output.



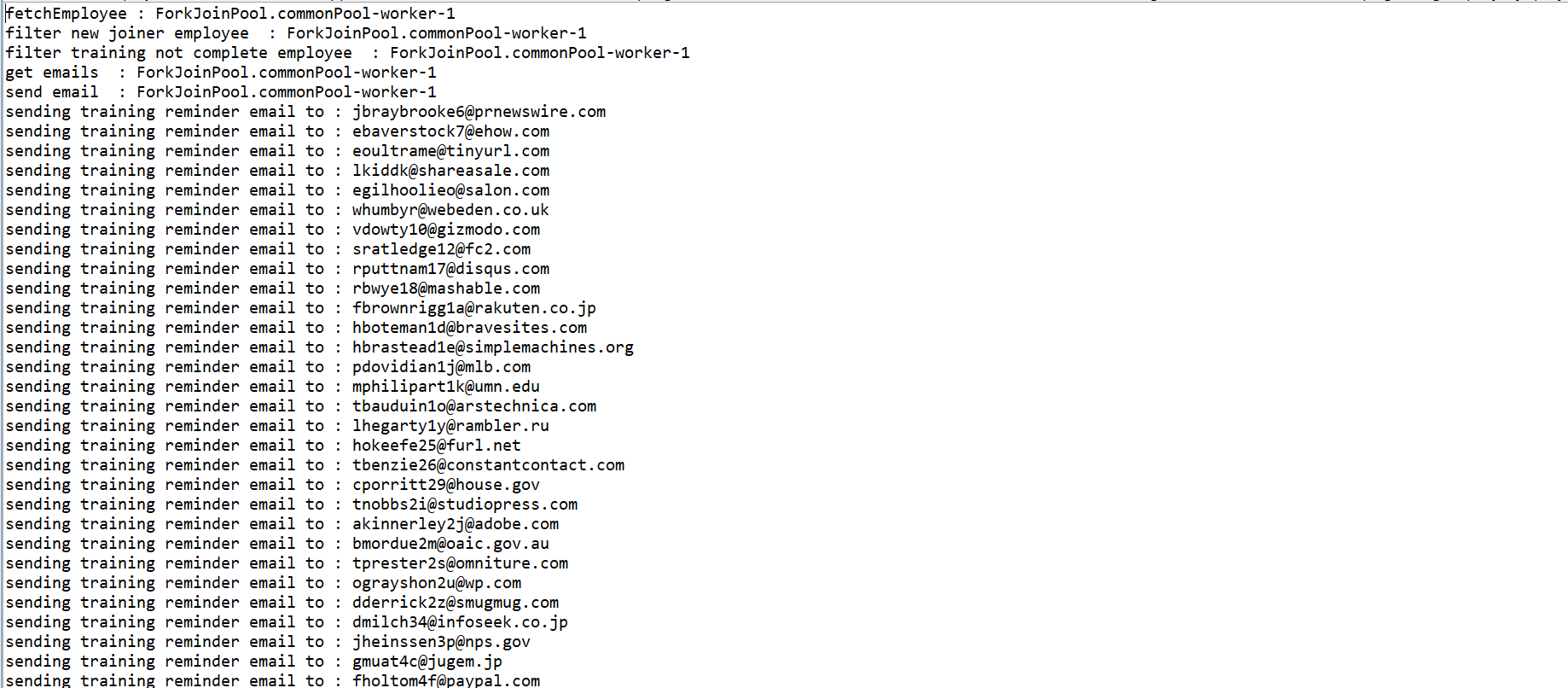
I can directly converted into the Method Reference bcz I have created a static method. forEach take input as a Consumer.



Now we have done all 5 tasks in a 5 separate threads.

Now let me use local variable **CompletableFuture<Void>** which our task will return.

Now let’s run.

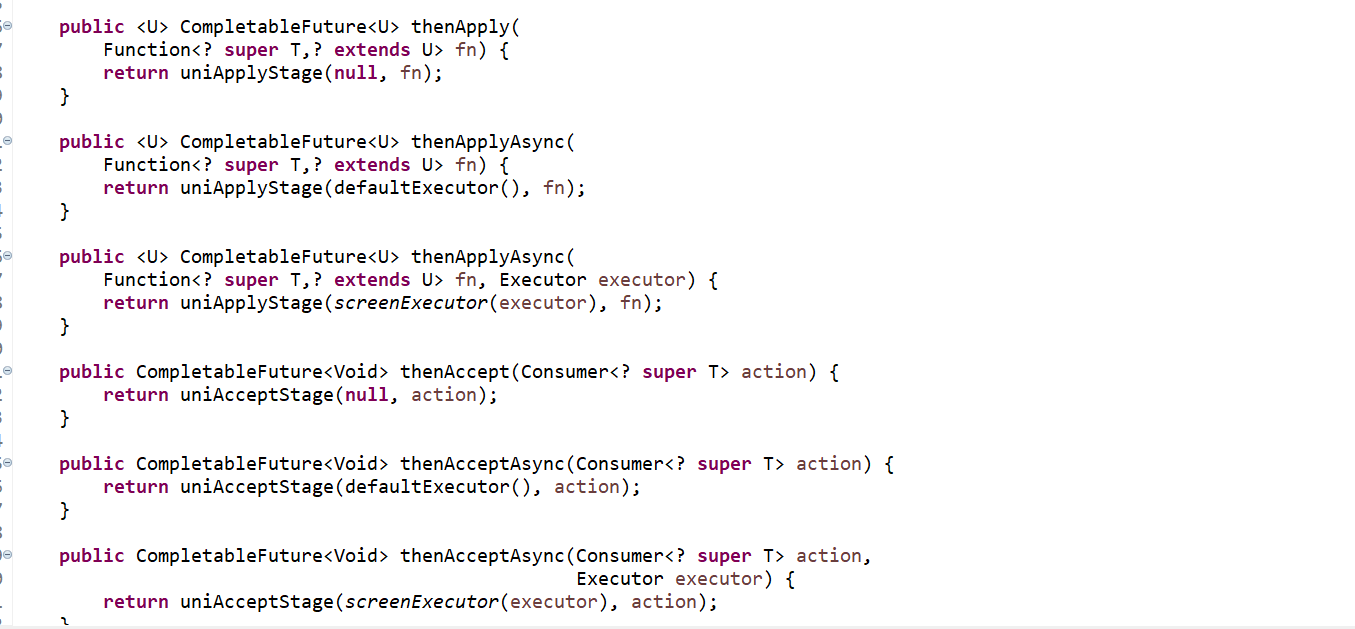




If u see fetch Employee executed by **ForkJoinPool-commonPool-worker-1…** so all are executing in a single thread. Because even though we are executing concurrently, and we are not specifying our own executor, so it is getting it from fork join common thread pool. If u observed fetch employee, filter new joiner …everything running using a single thread. Even though we have one thousand records.

Now what is the multithreading concept here because everything is running in a single thread there is no use of using these thenApply() or thenAccept() method.to show you that there is a 2 methods **thenApply**() and **thenApplyAsync**(). thenAccept() and **thenAcceptAsync**() method.

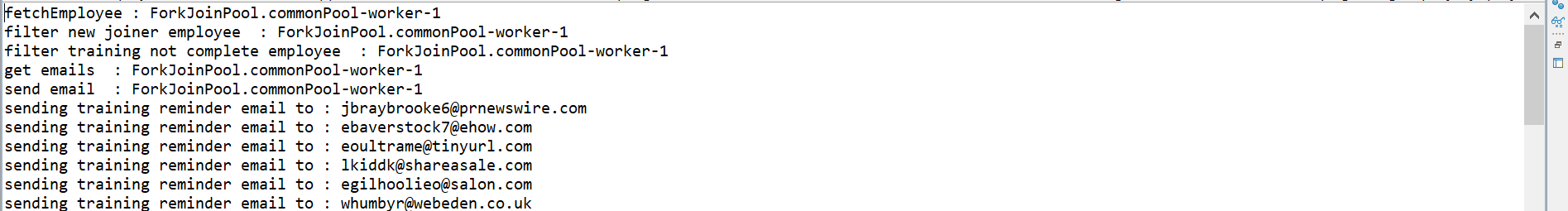
Now if we will go to the source code… u can see this is a overloaded methods.



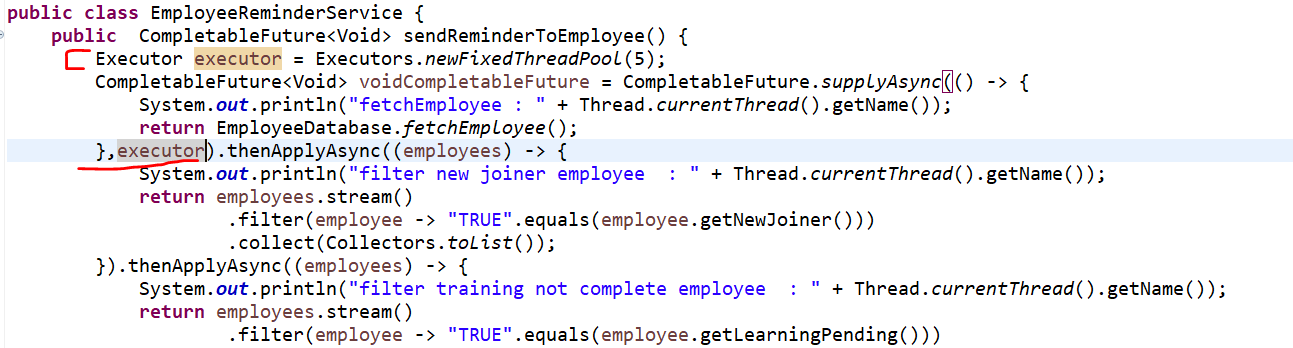
If you want to execute in a separate thread concurrently then you can go for **thenApplyAsync**() if u don’t specify then it will use **thenApply**() and it execute in a single flow. There will be no concurrent flow. Then go to code and change it to **thenApplyAsync**(). If u want to perform asynchronous communication or in a concurrent way u need to go for **Async.**

Since we have not provided own executor so its taking thread from a fork join pool only.





We just need to provide our own executor. Now I just want to provide this executor to my **supplyAsync**() method.



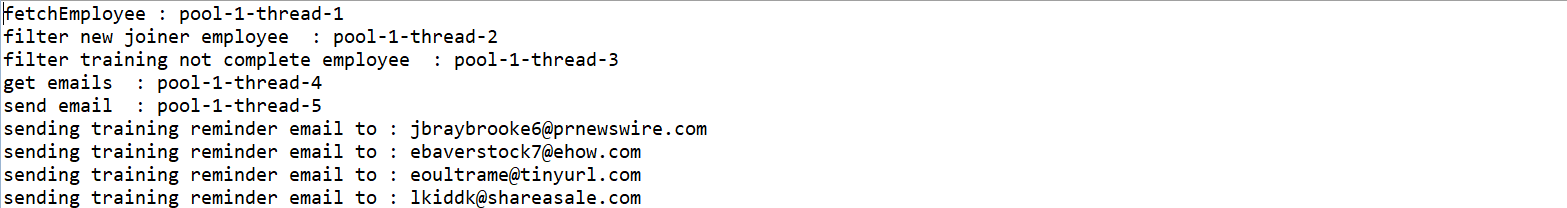
Now this particular this **supplyAsync** logic up to this executor will execute with my own executor or it will take my own custom thread pool. But the rest other method like thenApplyAsync()….will take threads from the fork join global pool because I am not specifying executor there.

Now let me run this piece of code and see.



Fetch employee run from the pool-1-thread-1 this is my own pool and others are executing from the fork join pool global pool.

**Now I will pass this executor to all async method. So, all tasks I am giving this executor. If I will run this every task will execute in a different thread.**



If u can observe here, we have just chained multiple future. This is what 5 futures with chained together.

This is what the disadvantages of Future object which we can achieve using **CompletableFuture**.

Now I just want to provide Separate Executor to each thread. That also u can do that.

I am not going to do that its not recommended its based on Ur CPU core.