CompletableFuture – Methods- 2021-2022

**acceptEither (CompletionStage<? extends T> other, Consumer<? super T> action)**

🡺 CompletableFuture<Void>

Between two tasks, which completes first. This is used when you want either of the tasks which completes the earliest. **It does not provide result**.

**public static** String getAadharResponse() {  
 **try** {  
 System.***out***.println(**"Collecting Aadhar Information ..."**);  
 TimeUnit.***SECONDS***.sleep(7);  
 } **catch** (InterruptedException ie) {  
 ie.printStackTrace();  
 }  
 **return "Aadhar-No-1"**;  
}  
  
**public static** String getPanNoResponse() {  
 **try** {  
 System.***out***.println(**"Collecting PanNo Information ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (InterruptedException ie) {  
 ie.printStackTrace();  
 }  
 System.***out***.println(**"Got Pan No Information ..."**);  
 **return "Pan No-1"**;  
}

**public static void** test1() {  
 **CompletableFuture cf1 = CompletableFuture.*supplyAsync*(() -> *getAadharResponse*());  
 CompletableFuture cf2 = CompletableFuture.*supplyAsync*(() -> *getPanNoResponse*());**  
 **Consumer consumer = (value1) -> System.*out*.println("Actual Response : " + value1);**  
*//* ***CompletableFuture<Void> cf3 = cf1.acceptEither(cf2, consumer);  
// cf3.join();*** *//Or you can write like this***CompletableFuture<Void> cf33 = cf1.acceptEither(cf2, (responseValue) -> {  
 System.*out*.println("Who Provided the response first : " + responseValue);  
 });  
 cf33.join();**  
 }

If you want a result from the above, you have to use **applyToEither**()

# thenAcceptBoth (CompletionStage<? extends U> other, BiConsumer<? super T,? super U> action) 🡺 CompletableFuture<Void>

This method is useful when you perform two parallel tasks and take both the results for processing. It is of Void type, it does not return anything. It is only used for auditing purposes. This method is used to log both the results from two threads, but it does not return anything.

**public static** String updateAadhar(String adrs) {  
 **try** {  
 System.***out***.println(**"Updating Aadhar Information ..."**);  
 TimeUnit.***SECONDS***.sleep(9);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Aadhar-No-1 updated"**;  
}  
  
**public static** String updatePancard(String adrs) {  
 **try** {  
 System.***out***.println(**"Updatingg PanNo Information ..."**);  
 TimeUnit.***SECONDS***.sleep(7);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "PanNo-No-1 updated"**;  
}

**public static void** update() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *updateAadhar*(**"adrs"**));  
 CompletableFuture<String> cf2 = CompletableFuture.*supplyAsync*(() -> *updatePancard*(**"adrs"**));  
 **CompletableFuture<Void> cf3 = cf1.thenAcceptBoth(cf2, (result1, result2) -> {**  
 System.***out***.println(**"Aadhar Status : "** + result1);  
 System.***out***.println(**"PanNo Status: "** + result2);  
 });  
 cf3.join();  
 System.***out***.println(**"Adrs updated ..."**);  
}

**OUTPUT**

Updating Aadhar Information ...

Updatingg PanNo Information ...

Aadhar Status : Aadhar-No-1 updated

PanNo Status: PanNo-No-1 updated

Adrs updated ...

**applyToEither (CompletionStage<? extends T> other, Function<? super T,U> fn)**

🡺 **CompletableFuture<U>**

**public static void** test2() {  
 *//If you want to return the value* **CompletableFuture cf1 = CompletableFuture.*supplyAsync*(() -> *getAadharResponse*());  
 CompletableFuture cf2 = CompletableFuture.*supplyAsync*(() -> *getPanNoResponse*());**  
 **Function<String, String> returnFn = (value) -> "Who provided the response ? " + value;**  
*// cf1.applyToEither(cf2,(value) -> {  
// return "Who provided the response ? "+value;  
// });* **CompletableFuture<String> cf3 = cf1.applyToEither(cf2, returnFn);**  
 **try** {  
 String response = cf3.get();  
 System.***out***.println(response);  
 } **catch** (InterruptedException e) { e.printStackTrace();}

**catch** (ExecutionException e) { e.printStackTrace(); }  
 }

**whenComplete (BiConsumer<? super T,? super Throwable> action)**

🡺 **CompletableFuture<T>**

**This method is useful when you want to perform certain intermediate operation before returning the value.** For example querying a certain result, update the table for audit entry. This is a useful function where you want to log both good result or bad result like exception. It is of type BiConsumer. It is useful when the function returns the result without any exception.

**public static** String getPassportDetails(String someDetails) {  
 **try** {  
 System.***out***.println(**"Collecting Passport Information ..."**);  
 TimeUnit.***SECONDS***.sleep(5);  
 } **catch** (InterruptedException ie) {  
 ie.printStackTrace();  
 }  
 System.***out***.println(**"Got Passport Information ..."**);  
 **if** (someDetails == **null**) **throw new** IllegalArgumentException(**"Invalid Information"**);  
 **return "Passport No-1"**;  
}

**public static void** check1() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *getPassportDetails*(**""**));  
*// CompletableFuture<String> cf2 = cf1.whenComplete((result, error) -> {  
// System.out.println("------ Doing other/Performing intermediate operation -------");  
// System.out.println("Good Result : " + result);  
// System.out.println("Error Details : " + error.toString());  
// });  
  
 // Or, you can write* **BiConsumer<String, Throwable> biConsumer** = (result, error) -> {  
 System.***out***.println(**"Good Result : "** + result);  
 System.***out***.println(**"Error Details : "** + error);  
 };  
 CompletableFuture<String> cf2 = cf1.whenComplete(biConsumer);  
  
 String response = cf2.join();  
 System.***out***.println(**"Actual Response : "** + response);  
 }

**Output**

Collecting Passport Information ...

Got Passport Information ...

Good Result : Passport No-1

Error Details : null

Actual Response : Passport No-1

In case of Exception, it will be like this.

**public static void** check1() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *getPassportDetails*(**null**));  
 CompletableFuture<String> cf2 = cf1.whenComplete((result, error) -> {  
 System.***out***.println(**"------ Doing other/Performing intermediate operation -------"**);  
 System.***out***.println(**"Good Result : "** + result);  
 System.***out***.println(**"Error Details : "** + error.toString());  
 });  
 *// Or, you can write* String response = cf2.join();  
 System.***out***.println(**"Actual Response : "** + response);  
 }

**Output**

Collecting Passport Information ...

Got Passport Information ...

------ Doing other/Performing intermediate operation -------

Good Result : null

Error Details : java.util.concurrent.CompletionException: java.lang.IllegalArgumentException: Invalid Information

Exception in thread "main" java.util.concurrent.CompletionException: java.lang.IllegalArgumentException: Invalid Information

at java.base/java.util.concurrent.CompletableFuture.encodeThrowable(CompletableFuture.java:314)

# runAfterBoth (CompletionStage<?> other, Runnable action)🡺 CompletableFuture<Void>

This method is useful when you want to perform a third task after completion two parallel tasks **without any responses**.

**public class** RunAfterBoth1 {

**private static void** sleep(**int** seconds) {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(seconds);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **public static** String task1() {  
 System.***out***.println(**"Started executing task-1"**);  
 *sleep*(3);  
 System.***out***.println(**"task-1 completed"**);  
 **return "task-1"**;  
 }  
  
 **public static** String task2() {  
 System.***out***.println(**"Started executing task-2"**);  
 *sleep*(5);  
 System.***out***.println(**"task-2 completed"**);  
 **return "task-2"**;  
 }  
  
 **public static** String task3() {  
 System.***out***.println(**"Started executing task-3"**);  
 *sleep*(5);  
 System.***out***.println(**"task-3 completed"**);  
 **return "task-3"**;  
 }

**public static void** main(String[] args) {  
 **CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
 CompletableFuture<String> cf2 = CompletableFuture.*supplyAsync*(() -> *task2*());** *// Task-3 will be started after the completion of task1 and task2***CompletableFuture<Void> cf3 = cf1.runAfterBoth(cf2, () -> *task3*());** cf3.join();  
 String value = cf1.join();  
 System.***out***.println(**"value = "** + value);  
 }  
}

OUTPUT

Started executing task-1

Started executing task-2

task-1 completed

task-2 completed

Started executing task-3

task-3 completed

value = task-1

How to perform 4th tasks after performing 3 tasks parallelly?

**runAfterEither(CompletionStage<?> other, Runnable action) 🡺 CompletableFuture<Void>**

Which completes first and then run another function to perform.

**public static** String task2(String input) {  
 **try** {  
 System.***out***.println(**"Executing Task 2 ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **if**(input == **null**) **throw new** IllegalArgumentException(**"Invalid value"**);  
 System.***out***.println(**"Task-2 completed"**);  
 **return "Task-2"**;  
}

**public static** String task1() {  
 **try** {  
 System.***out***.println(**"Executing Task 1 ..."**);  
 TimeUnit.***SECONDS***.sleep(5);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(**"Task-1 completed"**);  
 **return "Task-1-Result"**;  
}

**public static** String task5(String input) {  
 **try** {  
 System.***out***.println(**"Executing Task 5 ..."**);  
 TimeUnit.***SECONDS***.sleep(7);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "value-5"**;  
}

**public static void** check() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task2*(**"some"**));  
 CompletableFuture<String> cf2 = CompletableFuture.*supplyAsync*( () -> *task1*());  
 CompletableFuture<Void> cf = cf1.runAfterEither(cf2, () -> *task5*(**"Some Value"**));  
  
 cf.join();  
}

**OUTPUT**

Executing Task 1 ...

Executing Task 2 ...

Task-2 completed

Executing Task 5 ...

Task-1 completed

**thenCombine (CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn) 🡺 <U,V> CompletableFuture<V>**

**public static** String getPanNoResponse() {  
 **try** {  
 System.***out***.println(**"Collecting PanNo Information ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (InterruptedException ie) {  
 ie.printStackTrace();  
 }  
 System.***out***.println(**"Got Pan No Information ..."**);  
 **return "Pan No-1"**;  
}

**public static** String getAadharResponse() {  
 **try** {  
 System.***out***.println(**"Collecting Aadhar Information ..."**);  
 TimeUnit.***SECONDS***.sleep(7);  
 } **catch** (InterruptedException ie) {  
 ie.printStackTrace();  
 }  
 **return "Aadhar-No-1"**;  
}

**public static** String[] performThridTask(String s1, String s2) {  
 System.***out***.println(**"Running Thrid Operation .."**);  
 **return new** String[]{s1,s2};  
}

**public static void** check1() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*( () -> *getPanNoResponse*());  
 CompletableFuture<String> cf2 = CompletableFuture.*supplyAsync*( () -> *getAadharResponse*());  
 **CompletableFuture<String[]> cf3 = cf1.thenCombine(cf2,(result1,result2) -> {  
 return *performThridTask*(result1, result2);  
 });**

**// OR**

BiFunction<String, String, String[]> bfn =  
 (result1, result2) -> *performThridTask*(result1, result2);

CompletableFuture<String[]> cf3 = cf1.thenCombine(cf2,bfn);

String[] value = cf3.join();  
 System.***out***.println(value[0]+**" "**+value[1]);  
}

**OUTPUT**

Collecting PanNo Information ...

Collecting Aadhar Information ...

Got Pan No Information ...

Running Thrid Operation ..

Pan No-1 Aadhar-No-1

Multiple combine

CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
CompletableFuture<String> cf2 = CompletableFuture.*supplyAsync*(() -> *task2*(**"input1"**));  
  
**CompletableFuture<Void> cf01 = cf1.thenCombine(cf2, (r1, r2) -> *task3*(r1));  
CompletableFuture<Integer> cf02 = cf01.thenCombine(cf01, (r1, r2) -> *task4*(1));**  
Integer val = cf02.join();  
System.***out***.println(**"Value : "**+val);

**thenCompose()(Function<? super T,? extends CompletionStage<U>> fn)**

**🡺 CompletableFuture<U>**

**public static** String task1() {  
 **try** {  
 System.***out***.println(**"Executing Task 1 ..."**);  
 TimeUnit.***SECONDS***.sleep(5);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Task-1-Result"**;  
}  
  
**public static** String task2(String input) {  
 **try** {  
 System.***out***.println(**"Input : "**+input);  
 System.***out***.println(**"Executing Task 2 ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Task-2"**;  
}

**public static void** check() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
 CompletableFuture<String> cf = cf1.thenCompose((resultFromCf1) -> CompletableFuture.*supplyAsync*(() -> *task2*(resultFromCf1)));  
 String response = cf.join();  
 System.***out***.println(**"Response : "** + response);  
}

**Difference Between thenApply() and thenCompose()**

We can use **thenApply()** method to work with a result of the previous call. However, a key point to remember is that the return type will be combined of all calls. So this method is useful when we want to transform the result of a CompletableFuture call.

The thenCompose() method is similar to thenApply() in that both return a new Completion Stage. However, **thenCompose() uses the previous stage as the argument**. It will flatten and return a Future with the result directly, rather than a nested future as we observed in thenApply()

**thenApply** is used if you have a synchronous mapping function.

CompletableFuture<Integer> future =

CompletableFuture.supplyAsync(() -> 1)

.thenApply(x -> x+1);

**thenCompose** is used if you have an asynchronous mapping function (i.e. one that returns a CompletableFuture). It will then return a future with the result directly, rather than a nested future.

CompletableFuture<Integer> future =

CompletableFuture.supplyAsync(() -> 1)

.thenCompose(x -> CompletableFuture.supplyAsync(() -> x+1));

**thenAccept (Consumer<? super T> action) 🡺 CompletableFuture<Void>**

**It does not return any result**, simply executes a function. Get the response from a method and pass to another method which does not provide result. An example will be get Aadhar No by providing some input and then execute another method by passing Aadhar No and Address. The second method does not provide result.

**public static** String task1() {  
 **try** {  
 System.***out***.println(**"Executing Task 1 ..."**);  
 TimeUnit.***SECONDS***.sleep(5);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Task-1-Result"**;  
}  
  
**public static void** task2(String input) {  
 **try** {  
 System.***out***.println(**"Executing Task 2 ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
}  
  
**public static void** update() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
 CompletableFuture<Void> cf2 = cf1.thenAccept((result) -> *task2*(result));  
 cf2.join();  
}

OUTPUT

Task1 started running ...

Task1 completed

result = task1

Task2 started running ...

Task2 completed

response = task1

# thenRun(Runnable action) 🡺 CompletableFuture<Void>

This method is useful when you perform an another task asynchronously after completion of main task.

Always remember, **.thenRun() does not return any value**. It is of type void. It will be useful if you do an audit operation when you want to store the values in database in each .thenRun().

**CompletableFuture<String> cf1 = CompletableFuture.supplyAsync(() -> task1());**

**CompletableFuture<Void> cfVoid = cf1.thenRun(() -> task2());**

//If you do not write the following line ie. cfVoid.join(),

//task2() will not be executed completely, only one line will be executed.

**cfVoid.join();**

**String result = cf1.join();**

**System.out.println("Final Result : " + result);**

Example is given below.

**import** java.util.concurrent.CompletableFuture;  
**import** java.util.concurrent.TimeUnit;  
**public class** ThenRun1 {  
 **private static void** sleep(**int** seconds) {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(seconds);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **public static** String task1() {  
 System.***out***.println(**"Started executing task-1"**);  
 *sleep*(3);  
 **return "task-1"**;  
 }  
  
 **public static** String task2() {  
 System.***out***.println(**"Started executing task-2"**);  
 *sleep*(5);  
 **return "task-2"**;  
 }  
  
 **public static void** main(String[] args) {  
 **CompletableFuture<Void> cf = CompletableFuture.*runAsync*(() -> *task1*())  
 .thenRun(() -> *task2*());  
 Void blank = cf.join();**  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
 String value = cf1.join();  
 System.***out***.println(**"value = "** + value);  
 }  
}

**OUTPUT**

Started executing task-1

Started executing task-2

Started executing task-1

value = task-1

Another Example is given below.

**import** java.util.concurrent.CompletableFuture;  
**import** java.util.concurrent.TimeUnit;  
**public class** ThenRunTest {  
  
 **private static void** sleep() {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **public static** String task1() {  
 System.***out***.println(**"Task1 started running ..."**);  
 *sleep*();  
 System.***out***.println(**"Task1 completed"**);  
 **return "task1"**;  
 }  
  
 **public static** String task2() {  
 System.***out***.println(**"Task2 started running ..."**);  
 *sleep*();  
 System.***out***.println(**"Task2 completed"**);  
 **return "task2"**;  
 }  
  
 **public static void** main(String[] args) {  
 **CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
 CompletableFuture<Void> cfVoid = cf1.thenRun(() -> *task2*());  
 cfVoid.join();** *// If you do not write it, task2() will not be executed completely, only one line  
 // will be executed.* String result = cf1.join();  
 System.***out***.println(**"Final Result : "** + result);  
 }  
}

OUTPUT

Task1 started running ...

Task1 completed

Task2 started running ...

Task2 completed

Final Result : task1

# thenApply (Function<? super T,? extends U> fn) 🡺 CompletableFuture<U>

**In case of .thenApply() method, result is derived from the previous thread execution and added/manipulated result with manipulation will be returned**. The example is given below.

Always remember, it returns a value. Complete example is given below.  
**public static** String getAadhar(String input) {  
 **try** {  
 System.***out***.println(**"Executing Aadhar Information ..."**);  
 TimeUnit.***SECONDS***.sleep(9);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Aadhar-No-1"**;  
}  
  
**public static** String getPancard(String input) {  
 **try** {  
 System.***out***.println(**"Executing PanNo Information ..."**);  
 TimeUnit.***SECONDS***.sleep(7);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "PanNo-No-1"**;  
}  
  
**public static** String getPassport(String input) {  
 **try** {  
 System.***out***.println(**"Executing Passport Information ..."**);  
 TimeUnit.***SECONDS***.sleep(9);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Passport-No-1"**;  
}

**public static void** update() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *getAadhar*(**"input"**));  
 **CompletableFuture<String> cf2 = cf1  
 .thenApply((aadharResult) -> *getPancard*(aadharResult))  
 .thenApply((panResult) -> *getPassport*(panResult));**  
 String result = **null**;  
 **try** {  
 **result = cf2.get();** System.***out***.println(**"Result : "** + result);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 } **catch** (ExecutionException e) {  
 e.printStackTrace();  
 }  
}

**allOf(CompletableFuture<?>... cfs)** 🡺**CompletableFuture<Void>**

Executes all the tasks parallelly without result.

**public static** String task1() {  
 **try** {  
 System.***out***.println(**"Executing Task 1 ..."**);  
 TimeUnit.***SECONDS***.sleep(5);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Task-1-Result"**;  
}  
  
**public static** String task2(String input) {  
 **try** {  
 System.***out***.println(**"Executing Task 2 ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "Task-2"**;  
}  
  
**public static** Void task3(String input) { *// Capital V, Void type* **try** {  
 System.***out***.println(**"Executing Task 3 ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return null**;  
}  
  
**public static** Integer task4(**int** input) {  
 **try** {  
 System.***out***.println(**"Executing Task 4 ..."**);  
 TimeUnit.***SECONDS***.sleep(7);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** 1;  
}

**public static void** check() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
 CompletableFuture<String> cf2 = CompletableFuture.*supplyAsync*(() -> *task2*(**"input1"**));  
 CompletableFuture<Void> cf3 = CompletableFuture.*supplyAsync*(() -> *task3*(**"input2"**));  
 CompletableFuture<Integer> cf4 = CompletableFuture.*supplyAsync*(() -> *task4*(4));  
  
 **CompletableFuture<Void> cf = CompletableFuture.*allOf*(cf1, cf2, cf3, cf4);  
 cf.join();**  
  
 String result1 = cf1.join();  
 String result2 = cf2.join();  
 Integer result4 = cf4.join();  
 System.***out***.println(result1 + **" : "** + result2 + **" : "** + result4);}

**anyOf(CompletableFuture<?>... cfs)** 🡺 [CompletableFuture](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)<[Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)>

Which task completes first.

CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task1*());  
CompletableFuture<String> cf2 = CompletableFuture.*supplyAsync*(() -> *task2*(**"input1"**));  
CompletableFuture<Void> cf3 = CompletableFuture.*supplyAsync*(() -> *task3*(**"input2"**));  
CompletableFuture<Integer> cf4 = CompletableFuture.*supplyAsync*(() -> *task4*(4));  
  
CompletableFuture<Object> cf = CompletableFuture.*anyOf*(cf1, cf2, cf3, cf4);  
Object response = cf.join();  
System.***out***.println(**"Who completed first : "**+response);

# exceptionally() and handle(): exceptionally() is used to get the fallback where as handle() is used to take some necessary action. Both return the values.

**handle (BiFunction<? super T,Throwable,? extends U> fn) 🡺 CompletableFuture<U>**

**public static** String task2(String input) {  
 **try** {  
 System.***out***.println(**"Executing Task 2 ..."**);  
 TimeUnit.***SECONDS***.sleep(3);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **if**(input == **null**) **throw new** IllegalArgumentException(**"Invalid value"**);  
 **return "Task-2"**;  
}

**public static void** check() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task2*(**null**));  
 CompletableFuture<String> cf = cf1.handle((result, error) -> {  
 System.***out***.println(**"Result : "**+result);  
 System.***out***.println(**"Error : "**+error);  
 **return "some business result"**;  
 });  
 String result = cf.join();  
 System.***out***.println(**"Result : "**+result);  
}

**OUTPUT**

Executing Task 2 ...

Result : null

Error : java.util.concurrent.CompletionException: java.lang.IllegalArgumentException: Invalid value

Result : some business result

**exceptionally(Function<Throwable,? extends T> fn) 🡺 CompletableFuture<T>**

**public static void** check() {  
 CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> *task2*(**null**));  
 CompletableFuture<String> cf = **cf1.exceptionally( ex -> {  
 System.*out*.println("Exception Details: "+ex.getMessage());  
 return "Good Result";  
 });** String result = cf.join();  
 System.***out***.println(**"Result : "**+result);  
}

OUTPUT

Task2 started running ...

finalResult = Some exception, so some Default Value

Task2 started running ...

finalResult = some default value as there is an exception

**Difference between supplyAsync() and runAsync()**

[runAsync](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#runAsync-java.lang.Runnable-) takes Runnable as input parameter and returns CompletableFuture<Void>, which means it does not return any result.

CompletableFuture<Void> run = CompletableFuture.runAsync(()-> System.out.println("hello"));

But [suppyAsync](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html" \l "supplyAsync-java.util.function.Supplier-) takes Supplier as argument and returns the CompletableFuture<U> with result value, which means it does not take any input parameters but it returns result as output.

CompletableFuture<String> supply = CompletableFuture.supplyAsync(() -> {

System.out.println("Hello");

return "result";

});

System.out.println(supply.get()); //result

**Conclusion :** So if you want the result to be returned, then choose supplyAsync or if you just want to run an async action, then choose runAsync.

# What is the difference between get() and join() in CompletableFutre ?

The only difference is how methods throw exceptions. get() is declared in Future interface as

V get() throws InterruptedException, ExecutionException.

The join() method doesn't throw checked exceptions.

public T join() Instead it throws unchecked CompletionException. So you do not need a try-catch block and instead you can fully harness exceptionally() method

Summary

**acceptEither()**

CompletableFuture<Void> cf = **cf1.acceptEither(cf2, (result) -> thirdOps(result));**  
cf.join();

**applyEither()**

CompletableFuture<String> cf = **cf1.applyToEither(cf2, (result) -> thirdOps(result));**  
cf.join();

**runAfterEither()**

CompletableFuture<Void> cf = **cf1.runAfterEither(cf2, () -> thirdOps("result"));**  
cf.join();

**runAfterBoth()**

CompletableFuture<Void> cf = **cf1.runAfterBoth(cf2, () -> thirdOps("result"));**  
cf.join();

**whenComplete()**

CompletableFuture<String> cf = **cf1.whenComplete((result, error)** -> {  
 System.***out***.println(**"Result : "**+result);  
 System.***out***.println(**"Error : "**+error);  
 thirdOps(**"some input"**);  
});  
String response = cf.join();  
System.***out***.println(response); // Response from cf1

**thenAcceptBoth()**

CompletableFuture<Void> cf = **cf1.thenAcceptBoth(cf2, (rCf1,rCf2) -> fourthOps(rCf1,rCf2));**  
cf.join();

**thenCombine()**

CompletableFuture<String> cf = **cf1.thenCombine(cf2, (rCf1,rCf2) -> fourthOps(rCf1, rCf2));**  
String response = cf.join();  
System.***out***.println(response); // response from fourthOps()

**thenCompose()**

CompletableFuture<String> cf =  
 **cf1.thenCompose((result) -> CompletableFuture.*supplyAsync*( () -> thirdOps(result)));**  
String response = cf.join();  
System.***out***.println(response); // Result from thirdOps()

**thenAccept()**

CompletableFuture<Void> cf = **cf2.thenAccept((result) -> thirdOps(result));**  
cf.join();

**thenRun()**

CompletableFuture<Void> cf = **cf2.thenRun(() -> thirdOps("someInput"));**cf.join();

**thenApply()**

CompletableFuture<String> cf = **cf2.thenApply((result) -> thirdOps(result));**String value = cf.join();  
System.***out***.println(value); // response from thirdOps()

**handle()**

CompletableFuture<String> cf = **cf2.handle((result, error) -> thirdOps(result));**  
String value = cf.join();  
System.***out***.println(value); // response from thirdOps()

**exceptionally()**

CompletableFuture<String> cf1 = CompletableFuture.*supplyAsync*(() -> throwErr(**null**));  
CompletableFuture<String> cf = **cf1.exceptionally((error) -> thirdOps("result"));**  
String value = cf.join();  
System.***out***.println(value);

**anyOf()**

CompletableFuture<Object> cf = **CompletableFuture.*anyOf*(cf1, cf2, cf3, cf4);**  
Object response = cf.join();  
System.***out***.println(**"Who completed first : "**+response);

**allOf()**

**CompletableFuture<Void> cf = CompletableFuture.*allOf*(cf1, cf2, cf3, cf4);  
cf.join();**  
  
String result1 = cf1.join();  
String result2 = cf2.join();  
Integer result4 = cf4.join();  
System.***out***.println(result1 + **" : "** + result2 + **" : "** + result4);