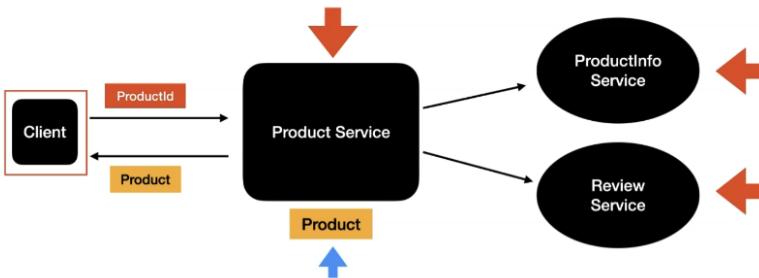


# Product Service



Code snippet from the Product Service:

```

22
23     public Product retrieveProductDetails(String productId) {
24         stopWatch.start();
25
26         CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
27             .supplyAsync(() -> productInfoService.retrieveProductInfo(productId));
28         CompletableFuture<Review> cfReview = CompletableFuture
29             .supplyAsync(() -> reviewService.retrieveReviews(productId));
30
31         Product product = cfProductInfo
32             .thenCombine(cfReview, (productInfo, review) -> new Product(productId, productInfo, review))
33             .join(); //block the thread
34
35         stopWatch.stop();
36         log("Total Time Taken : " + stopWatch.getTime());
37
38         return product;
39     }

```

Code snippet from the ProductServiceUsingCompletableFutureTest:

```

10
11     class ProductServiceUsingCompletableFutureTest {
12
13         private ProductInfoService pis = new ProductInfoService();
14         private ReviewService rs = new ReviewService();
15         ProductServiceUsingCompletableFuture pscf = new
16             ProductServiceUsingCompletableFuture(pis, rs);
17
18         @Test
19         void retrieveProductDetails() {
20             //given
21             String productId = "ABC123";
22
23             //when
24             Product product = pscf.retrieveProductDetails(productId);
25
26             //then
27             assertNotNull(product);
28             assertTrue(condition: product.getProductInfo().getProductOpti
29             assertNotNull(product.getReview());
30         }
31     }

```

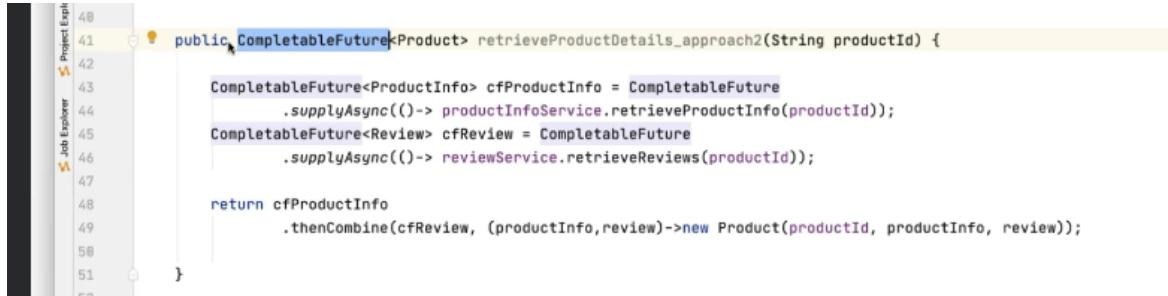
```

20
21
22
23         public Product retrieveProductDetails(String productId) {
24             stopWatch.start();
25
26             CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
27                 .supplyAsync(() -> productInfoService.retrieveProductInfo(productId));
28             CompletableFuture<Review> cfReview = CompletableFuture
29                 .supplyAsync(() -> reviewService.retrieveReviews(productId));
30
31             Product product = cfProductInfo
32                 .thenCombine(cfReview, (productInfo, review) -> new Product(productId, productInfo, review))
33                 .join(); //block the thread
34
35             stopWatch.stop();
36             log("Total Time Taken : " + stopWatch.getTime());
37
38             return product;
39
40     public static void main(String[] args) {
41
42         ProductInfoService productInfoService = new ProductInfoService();
43         ReviewService reviewService = new ReviewService();
44         ProductServiceUsingCompletableFuture productService = new ProductServiceUsingCompletableFuture(productInfoService, reviewService);
45         String productId = "ABC123";
46         Product product = productService.retrieveProductDetails(productId);

```

Retrieve product info has 1 sec delay similarly retrieve reviews. But output came in 1.1 second because of a completable Future.

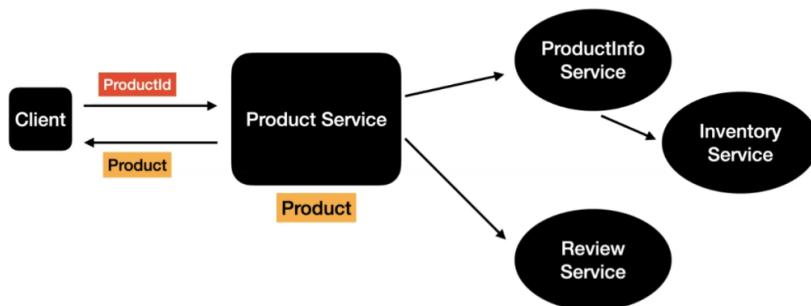
Convert return type to completableFuture



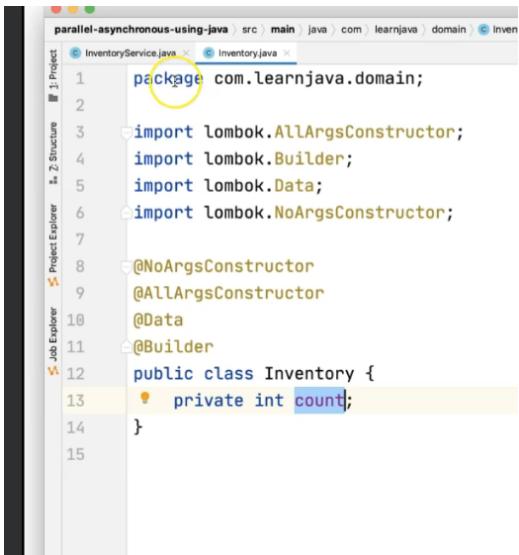
```
40
41     public CompletableFuture<Product> retrieveProductDetails_approach2(String productId) {
42
43         CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
44             .supplyAsync(() -> productService.retrieveProductInfo(productId));
45         CompletableFuture<Review> cfReview = CompletableFuture
46             .supplyAsync(() -> reviewService.retrieveReviews(productId));
47
48         return cfProductInfo
49             .thenCombine(cfReview, (productInfo, review) -> new Product(productId, productInfo, review));
50
51     }
52 }
```

# Combining Streams & CompletableFuture

## Product Service with Inventory



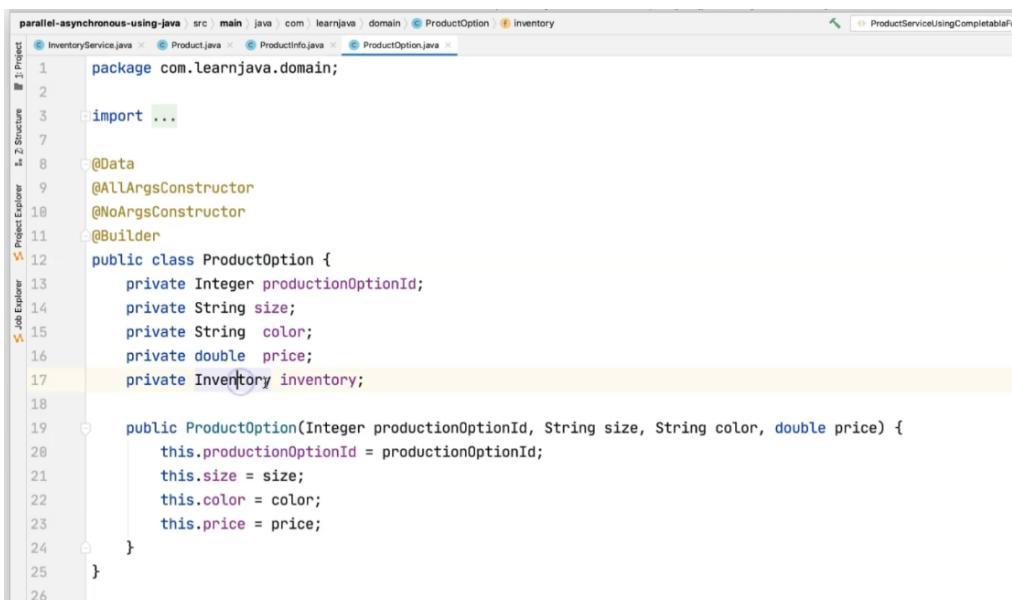
Inventory service depends on productinfo service.



```
parallel-asynchronous-using-java | src | main | java | com | learnjava | domain | Inventory.java
1 package com.learnjava.domain;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Builder;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 @NoArgsConstructor
9 @AllArgsConstructor
10 @Data
11 @Builder
12 public class Inventory {
13     private int count;
14 }
15
```



```
parallel-asynchronous-using-java | src | main | java | com | learnjava | domain | InventoryService.java
1 package com.learnjava.domain;
2
3 import ...
4
5 public class InventoryService {
6     public Inventory retrieveInventory(ProductOption productOption) {
7         delay(delayMilliseconds: 500);
8         return Inventory.builder()
9             .count(2).build();
10    }
11 }
12
13
14
15
16
```



```
parallel-asynchronous-using-java | src | main | java | com | learnjava | domain | ProductOption.java
1 package com.learnjava.domain;
2
3 import ...
4
5 @Data
6 @AllArgsConstructor
7 @NoArgsConstructor
8 @Builder
9 public class ProductOption {
10     private Integer productionOptionId;
11     private String size;
12     private String color;
13     private double price;
14     private Inventory inventory;
15
16     public ProductOption(Integer productionOptionId, String size, String color, double price) {
17         this.productionOptionId = productionOptionId;
18         this.size = size;
19         this.color = color;
20         this.price = price;
21     }
22 }
23
24
25
26
```

```

public Product retrieveProductDetailsWithInventory(String productId) {
    stopWatch.start();

    CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
        .supplyAsync(() -> productService.retrieveProductInfo(productId))
        .thenApply(productInfo -> {
            productInfo.setProductOptions(updateInventory(productInfo));
            return productInfo;
        });
}

CompletableFuture<Review> cfReview = CompletableFuture
    .supplyAsync(() -> reviewService.retrieveReviews(productId));

Product product = cfProductInfo
    .thenCombine(cfReview, (productInfo, review) -> new Product(productId, productInfo, review))
    .join(); //block the thread

stopWatch.stop();
log("Total Time Taken : " + stopWatch.getTime());
return product;
}

```

```

private List<ProductOption> updateInventory(ProductInfo productInfo) {
    List<ProductOption> productOptionList = productInfo.getProductOptions() List<ProductOption>
        .stream() Stream<ProductOption>
        .map(productOption -> {
            Inventory inventory = inventoryService.retrieveInventory(productOption);
            productOption.setInventory(inventory);
            return productOption;
        })
        .collect(Collectors.toList());
    return productOptionList;
}

```

Test case. The time taken depends on the number of product

```

@org.junit.Test
void retrieveProductDetailsWithInventory() {
    //given
    String productId="ABC123";

    //when
    Product product = pscf.retrieveProductDetailsWithInventory(productId);

    //then
    assertNotNull(product);
    assertTrue(condition: product.getProductInfo().getProductOptions().size() > 0);
    assertEquals("ABC123", product.getReview());
}

public Product retrieveProductDetailsWithInventory(String productId) {
    stopWatch.start();

    CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
        .supplyAsync(() -> productService.retrieveProductInfo(productId))
        .thenApply(productInfo -> {
            productInfo.setProductOptions(updateInventory(productInfo));
            return productInfo;
        });

    CompletableFuture<Review> cfReview = CompletableFuture
        .supplyAsync(() -> reviewService.retrieveReviews(productId));

    Product product = cfProductInfo
        .thenCombine(cfReview, (productInfo, review) -> new Product(productId, productInfo, review))
        .join(); //block the thread

    stopWatch.stop();
    log("Total Time Taken : " + stopWatch.getTime());
    return product;
}

private List<ProductOption> updateInventory(ProductInfo productInfo) {
    List<ProductOption> productOptionList = productInfo.getProductOptions() List<ProductOption>
        .stream() Stream<ProductOption>
        .map(productOption -> {

```

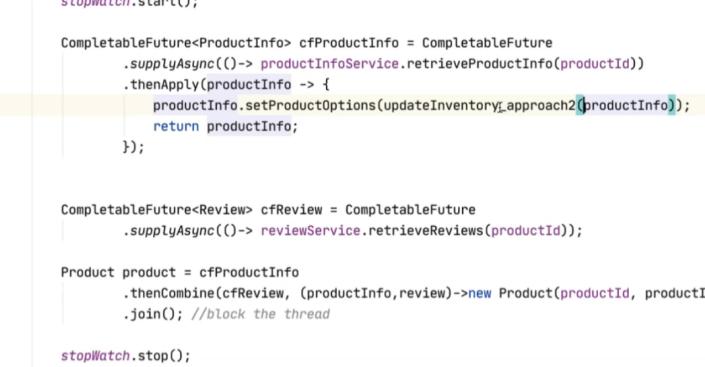
The more the number of productOption records - it takes more time.

```
27 //then
28 assertNotNull(product);
29 assertTrue(condition: product.getProductInfo().getProductOptions().size() > 0);
30 assertNotNull(product.getReview());
31 }
32
33
34 @Test
35 void retrieveProductDetailsWithInventory() {
36     //given
37     String productId="ABC123";
38
39     //when
40     Product product = pscf.retrieveProductDetailsWithInventory(product);
41 }
42
43
44 .ic class ProductInfoService {
45
46     public ProductInfo retrieveProductInfo(String productId) {
47         delay(1000);
48         List<ProductOption> productOptions = List.of(new ProductOption( productionOptionId: 1, size: "128GB", color: "Black", price: 749.99),
49                                         new ProductOption( productionOptionId: 2, size: "128GB", color: "Black", price: 749.99),
50                                         new ProductOption( productionOptionId: 3, size: "64GB", color: "Black", price: 699.99),
51                                         new ProductOption( productionOptionId: 4, size: "128GB", color: "Black", price: 749.99));
52
53         return ProductInfo.builder().productId(productId)
54             .productOptions(productOptions)
55             .build();
56     }
57 }
58
59
60 Run: ProductServiceUsingCompletableFutureTest...
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
```

To improve performance convert update inventory to completable future as shown below.

```
98
99 @
100
101     private List<ProductOption> updateInventory_approach2(ProductInfo productInfo) {
102
103         List<CompletableFuture<ProductOption>> productOptionList =productInfo.getProductOptions() List<ProductOption>
104             .stream() Stream<ProductOption>
105             .map(productOption -> {
106                 return CompletableFuture.supplyAsync(()->inventoryservice.retrieveInventory(productOption))
107                     .thenApply(inventory -> {
108                         productOption.setInventory(inventory);
109                         return productOption;
110                     });
111             });
112         Stream<CompletableFuture<ProductOption>>
113             .collect(Collectors.toList());
114
115
116         return productOptionList.stream().map(CompletableFuture::join).collect(Collectors.toList());
117     }
118 }
```

By the time the join line executes it collects all the information using multiple threads. Join consolidates that.



The screenshot shows a Java code editor with the following code:

```
public Product retrieveProductDetailsWithInventory_approach2(String productId) {
    stopWatch.start();

    CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
        .supplyAsync(() -> productService.retrieveProductInfo(productId))
        .thenApply(productInfo -> {
            productInfo.setProductOptions(updateInventory_approach2(productInfo));
            return productInfo;
        });

    CompletableFuture<Review> cfReview = CompletableFuture
        .supplyAsync(() -> reviewService.retrieveReviews(productId));

    Product product = cfProductInfo
        .thenCombine(cfReview, (productInfo, review) -> new Product(productId, productInfo, review))
        .join(); //block the thread

    stopWatch.stop();
    log("Total Time Taken : " + stopWatch.getTime());
    return product;
}
```

```

50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }

118     return productOptionList;
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129     @Test
130     void retrieveProductDetailsWithInventory_approach2() {
131         //given
132         String productId="ABC123";
133
134         //when
135         Product product = pscf.retrieveProductDetailsWithInventory_approach2(productId);
136
137         //then
138         assertNotNull(product);
139         assertTrue( condition: product.getProductInfo().getProductOptions().size() > 0 );
140         product.getProductInfo().getProductOptions()
141             .forEach(productOption -> {
142                 assertNotNull(productOption.getInventory());
143             });
144         assertNotNull(product.getReview());
145     }
146 }

147     static void main(String[] args) {
148         ProductInfoService productService = new ProductInfoService();
149         ReviewService reviewService = new ReviewService();
150     }

```

Run: Tests passed: 1 of 1 test – 1s 549 ms

[main] - Total Time Taken : 1512

Process finished with exit code 0

## Exception Handling in Java

- Exception Handling in Java is available since the inception of Java

```

public void exceptionHandling() {
    try {
        // Code Statements
        // Code Statements
        // Code Statements
    } catch (Exception e) {
        // Handle the Exception ←
    }
}

```

## Exception Handling in CompletableFuture

- CompletableFuture is a functional style API

```

public String helloWorld_3.async_calls() {
    CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> this.hws.hello());
    CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> this.hws.world());
    CompletableFuture<String> hicompletableFuture = CompletableFuture.supplyAsync(() -> {
        delay( delayMilliseconds: 1000 );
        return " HI CompletableFuture!";
    });

    String hw = hello
        .thenCombine(world, (h, w) -> h + w) // (first,second)
        .thenCombine(hicompletableFuture, (previous, current) -> previous + current)
        .thenApply(String::toUpperCase)
        .join();
    return hw;
}

```

# Exception Handling in CompletableFuture

## try/catch

```
public String helloWorld_3_async_calls() {
    try{
        CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> this.hws.hello());
        CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> this.hws.world());
        CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(() -> {
            delay(1000);
            return " HI CompletableFuture!";
        });

        String hw = hello
            .thenCombine(world, (h, w) -> h + w) // (first,second)
            .thenCombine(hiCompletableFuture, (previous, current) -> previous + current)
            .thenApply(String::toUpperCase)
            .join();
        return hw;
    }catch (Exception e){
        log("Exception is " + e);
        throw e; ←
    }
}
```

# Exception Handling in CompletableFuture

- CompletableFuture API has functional style of handling exceptions
- Three options available:
  - handle()
  - exceptionally()
  - whenComplete()



Handle method takes BiFunction.

Exceptionally method takes Function.

Handling exceptions using the handle method. Handle method takes BiFunction.

The screenshot shows an IDE interface with several tabs open. The main tab contains Java code for handling exceptions using the handle method. The code defines a class `CompletableFutureHelloWorldExceptionTest` with a test method `void helloworld_3_async_calls_handle()`. This method uses the `handle` method to catch exceptions and log them. The code also demonstrates the use of `CompletableFuture` to perform asynchronous operations like `hello`, `world`, and `hiCompletableFuture`.

```
import java.util.concurrent.CompletableFuture;
import static com.learnjava.util.LoggerUtil.log;

public class CompletableFutureHelloWorldExceptionTest {

    @Test
    void helloworld_3_async_calls_handle() {
        //given
        //when
        //then
    }
}

class CompletableFutureHelloWorldExceptionTest {

    @Test
    void helloworld_3_async_calls_handle() {
        //given
        //when
        //then
    }
}

public class CompletableFutureHelloWorldException {
    private HelloWorldService hws;

    public CompletableFutureHelloWorldException(HelloWorldService hws) { this.hws = hws; }

    public String helloworld_3_async_calls_handle() {
        startTimer();

        CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> hws.hello());
        CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> hws.world());
        CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(() -> {
            delay(1000);
            return " HI CompletableFuture!";
        });

        String hw = hello
            .handle((res, e) -> {
                log("Exception is :" + e.getMessage());
                return res;
            })
            .thenCombine(world, (h, w) -> h + w) // first, second
            .thenCombine(hiCompletableFuture, (previous, current) -> previous + current)
            .thenApply(String::toUpperCase)
            .join();
        return hw;
    }
}
```

## Unit test

The screenshot shows an IntelliJ IDEA interface with the following details:

- Project Structure:** Shows a single module named "CompletableFutureHelloWorld".
- File:** "CompletableFutureHelloWorldExceptionTest.java" is open.
- Code:** The code is a JUnit test for a service that throws an exception. It uses Mockito for mocking and annotations like @ExtendWith and @Mock to set up the test environment.
- Annotations:** The code includes annotations such as @Test, @Mock, @InjectMocks, and @ExtendWith(MockitoExtension.class).
- Imports:** Standard Java imports for assertions and Mockito are present.
- Test Logic:** The test checks if an exception is thrown when a specific method is called on the mocked service. It then handles the exception by calling a handle method on a CompletableFuture object.
- IDE Features:** The code editor highlights syntax errors (red squiggly lines) and provides code completion and navigation features.

The above one is one completableFuture fail and below is multi completableFuture fails.

The screenshot shows an IDE interface with two tabs open: "parallel-asynchronous - CompletableFutureFutureHelloWorldException.java [parallel-asynchronous.java]" and "CompletableFutureHelloWorldException - CompletableFutureHelloWorldException.java [parallel-asynchronous.main]". The code is written in Java and demonstrates the use of `CompletableFuture` for parallel asynchronous operations.

```
parallel-asynchronous - CompletableFutureFutureHelloWorldException.java [parallel-asynchronous.java]
CompletableFutureHelloWorldException - CompletableFutureHelloWorldException.java [parallel-asynchronous.main]
```

```
parallel-asynchronous - CompletableFutureFutureHelloWorldException.java [parallel-asynchronous.java]
CompletableFutureHelloWorldException - CompletableFutureHelloWorldException.java [parallel-asynchronous.main]
```

```
1    package parallel-asynchronous;
2
3    import java.util.concurrent.CompletableFuture;
4    import java.util.concurrent.ExecutionException;
5    import java.util.concurrent.TimeUnit;
6    import java.util.concurrent.TimeoutException;
7
8    public class CompletableFutureFutureHelloWorldException {
9
10        @InjectMocks
11        CompletableFutureHelloWorldException hcwe;
12
13        @Test
14        void helloWorld_3_async_calls_Handle() {
15            //given
16            when(helloWorldService.hello()).thenThrow(new RuntimeException("Exception Occurred"));
17            when(helloWorldService.world()).thenCallRealMethod();
18
19            //when
20            String result = hcwe.helloworld_3_async_calls_Handle();
21
22            //then
23            assertEquals(" world! HI COMPLETABLEFUTURE!", result);
24        }
25
26        @Test
27        void helloWorld_3_async_calls_Handle_2() {
28            //given
29            when(helloWorldService.hello()).thenThrow(new RuntimeException("Exception Occurred"));
30            when(helloWorldService.world()).thenThrow(new RuntimeException("Exception Occurred"));
31
32            //when
33            String result = hcwe.helloworld_3_async_calls_Handle();
34
35            //then
36            assertEquals(" HI COMPLETABLEFUTURE!", result);
37        }
38
39        @Test
40        void helloWorld_3_async_calls_Handle_3() {
41            //given
42            when(helloWorldService.hello()).thenThrow(new RuntimeException("Exception Occurred"));
43            when(helloWorldService.world()).thenThrow(new RuntimeException("Exception Occurred"));
44
45            //when
46            String result = hcwe.helloworld_3_async_calls_Handle();
47
48            //then
49            assertEquals(" HI COMPLETABLEFUTURE!", result);
50        }
51    }
```

```
1    package parallel-asynchronous;
2
3    import java.util.concurrent.CompletableFuture;
4    import java.util.concurrent.ExecutionException;
5    import java.util.concurrent.TimeUnit;
6    import java.util.concurrent.TimeoutException;
7
8    public class CompletableFutureHelloWorldException {
9
10        @InjectMocks
11        CompletableFutureHelloWorldException hcwe;
12
13        @Test
14        void helloWorld_3_async_calls_Handle() {
15            startTimer();
16
17            CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> hws.hello());
18            CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> hws.world());
19            CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(() -> hws.hi());
20
21            delay(1000, TimeUnit.MILLISECONDS);
22            return " Hi CompletableFuture!";
23        }
24
25        String hws = hello
26            .handle((res, e) -> {
27                log("Exception is :" + e.getMessage());
28                return "";
29            })
30            .thenCombine(world, (h, w) -> h + w) // -> world!
31            .handle((res, e) -> {
32                log("Exception after world is :" + e.getMessage());
33                return "";
34            })
35            .thenCombine(hiCompletableFuture, (previous,current) -> previous + current)
36            .thenApply(String::toUpperCase) // -> " HI COMPLETABLEFUTURE!"
37            .join(); // -> " HI COMPLETABLEFUTURE!"
38
39        timeTaken();
40        return hcwe;
41    }
42}
```

When there is no error the handle method still execute so we need to do null check and proceed as shown in the below screenshot.

```

parallel-asynchronous ~ CompletableFutureHelloWorldException.java [parallel-asynchronous.main]
src/main/java/com/learnjava/completablefuture/CompletableFutureHelloWorldExceptionTest.java
src/main/java/com/learnjava/completablefuture/HelloWorldService.java

    //then
    assertEquals("HI COMPLETABLEFUTURE!", result);

}

@Test
void helloWorld_3_async_calls_handle_3() {
    //given
    when(helloWorldService.hello()).thenCallRealMethod();
    when(helloWorldService.world()).thenCallRealMethod();

    //when
    String result = hwcfe.helloworld_3_async_calls_handle_3();

    //then
    assertEquals("HELLO WORLD! HI COMPLETABLEFUTURE!", result);
}

}

String hwo hello // hello
    .handle((res, e) -> {
        log("res is :" + res);
        if(e!=null){
            log("Exception is :" + e.getMessage());
            return "";
        }else{
            return res;
        }
    })
    .thenCombine(world, (h, w) -> h+w) // "hello world!"
    .handle((res, e) -> {
        log("res is :" + res);
        if(e!=null){
            log("Exception after world is :" + e.getMessage());
            return "";
        }else{
    }

}

Test Result 1s 96 ms
Tests passed: 1 of 1 test - 1s 96 ms
Test Result is :
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...
[ForJoinPool.commonPool-worker-5] - inside world
[ForJoinPool.commonPool-worker-19] - inside hello
[ForJoinPool.commonPool-worker-19] - res is hello
[ForJoinPool.commonPool-worker-19] - res is hello world!
[main] - Total Time Taken : 1043

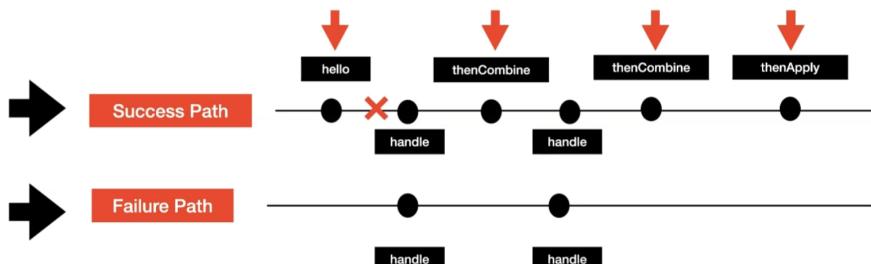
Process finished with exit code 0

```

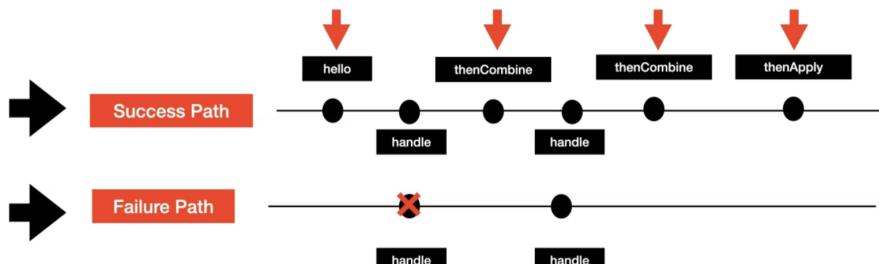
Summary how it works

using "handle" function - Part3

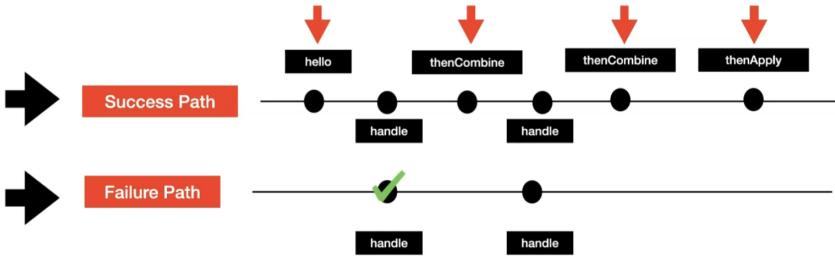
## Exception Handling using handle()



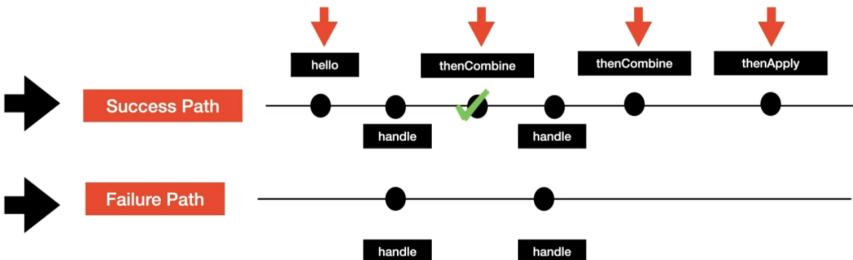
If we get exception in the hello then it will go to failure path handle



From there it recover



And go to the Success Path thenCombine.



Handle the same thing using exceptionally as shown below. For this no additional logic is required as we did in the handle method like null check. Since this exceptionally works only when there is an error there is no guarantee it will execute during pipeline execute.

The screenshot shows the IntelliJ IDEA interface with several tabs open. The main editor tab contains Java code for `CompletableFutureHelloWorldException.java`, which includes annotations like `@Test` and `@Test`. The code uses `CompletableFuture` to handle asynchronous calls. A yellow circle highlights the `I` icon in the gutter of the first test method. Below the editor, the tool bar includes icons for Build, Run, Debug, Messages, TODO, and Terminal. The status bar at the bottom indicates the file is 100% up-to-date and was last modified at 7:40 PM.

```
public class CompletableFutureHelloWorldException {
    @Test
    void testHelloWorld() {
        //when
        String result = hcfe.helloworld_3.async_calls.handle();
        //then
        assertEquals("expected = HI COMPLETABLEFUTURE!", result);
    }

    @Test
    void helloworld_3.async_calls.handle_3() {
        //given
        when(helloWorldService.hello()).thenReturnMethod();
        when(helloWorldService.world()).thenReturnMethod();

        //when
        String result = hcfe.helloworld_3.async_calls.handle();
        //then
        assertEquals("expected = HELLO WORLD! HI COMPLETABLEFUTURE!", result);
    }

    @Test
    void helloworld_3.async_calls.exceptionally() {
        //given
        when(helloWorldService.hello()).thenCallRealMethod();
        when(helloWorldService.world()).thenCallRealMethod();

        //when
        String result = hcfe.helloworld_3.async_calls.exceptionally();
        //then
        assertEquals("expected = HELLO WORLD! HI COMPLETABLEFUTURE!", result);
    }
}
```

```
public class CompletableFutureHelloWorldException {
    @Test
    void testHelloWorld() {
        //when
        String result = hcfe.helloworld_3.async_calls.handle();
        //then
        assertEquals("expected = HI COMPLETABLEFUTURE!", result);
    }

    @Test
    void helloworld_3.async_calls.handle_3() {
        //given
        when(helloWorldService.hello()).thenReturnMethod();
        when(helloWorldService.world()).thenReturnMethod();

        //when
        String result = hcfe.helloworld_3.async_calls.handle();
        //then
        assertEquals("expected = HELLO WORLD! HI COMPLETABLEFUTURE!", result);
    }

    @Test
    void helloworld_3.async_calls.exceptionally() {
        //given
        when(helloWorldService.hello()).thenCallRealMethod();
        when(helloWorldService.world()).thenCallRealMethod();

        //when
        String result = hcfe.helloworld_3.async_calls.exceptionally();
        //then
        assertEquals("expected = HELLO WORLD! HI COMPLETABLEFUTURE!", result);
    }
}
```

In the unit test we are calling real code. We are not throwing any exceptions.

```

66     @Test
67     void helloworld_3_async_calls_exceptionally() {
68         //given
69         when(helloworldService.hello()).thenCallRealMethod();
70         when(helloworldService.world()).thenCallRealMethod();
71
72         //when
73         String result = hcfe.helloworld_3_async_calls_exceptionally();
74
75         //then
76         assertEquals(expected: "HELLO WORLD! HI COMPLETABLEFUTURE!", result);
77     }
78 }

76     log("Exception after world is :" + e.getMessage());
77     return "";
78 }
79 .thenCombine(hicCompletableFuture, (previous,current)->previous+current)
80 // "hello world! HI CompletableFuture!"
81 .thenApply(String::toUpperCase) // "HELLO WORLD! HI COMPLETABLEFUTURE!"
82 .join(); // "HELLO WORLD! HI COMPLETABLEFUTURE!"

83 timeTaken();
84 return hw;
85 }
86
87 }
88

```

Run: CompletetableFutureHelloWorldExceptionTest...  
 Tests passed: 1 of 1 test - 1s 84ms  
 /Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...  
 [ForkJoinPool.commonPool-worker-19] - inside hello  
 [ForkJoinPool.commonPool-worker-5] - inside world  
 [main] - Total Time Taken : 1035

Process finished with exit code 0

Same code when we have exception in unit testing

```

66     String result = hcfe.helloworld_3_async_calls_handle();
67
68     //then
69     assertEquals(expected: "HELLO WORLD! HI COMPLETABLEFUTURE!", result);
70 }

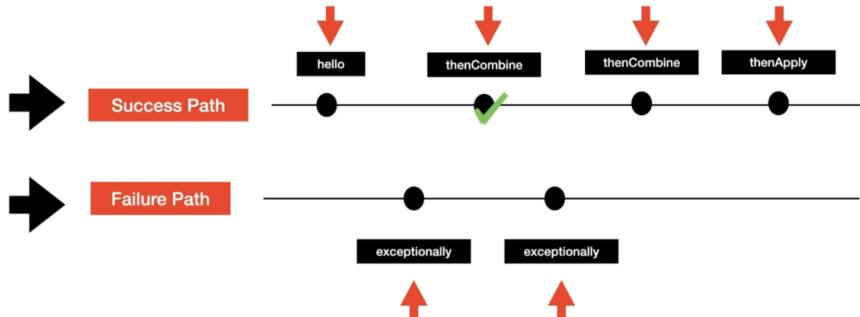
71
72 @Test
73 void helloworld_3.async_calls_exceptionally() {
74     //given
75     when(helloworldService.hello()).thenCallRealMethod();
76     when(helloworldService.world()).thenCallRealMethod();
77
78     //when
79     String result = hcfe.helloworld_3.async_calls_exceptionally();
80
81     //then
82     assertEquals(expected: "HELLO WORLD! HI COMPLETABLEFUTURE!", result);
83 }
84
85 @Test
86 void helloworld_3.async_calls_exceptionally_2() {
87     //given
88     when(helloworldService.hello()).thenThrow(new RuntimeException("Exception Occurred"));
89     when(helloworldService.world()).thenThrow(new RuntimeException("Exception Occurred"));
90
91     //when
92     String result = hcfe.helloworld_3.async_calls_exceptionally();
93
94     //then
95     assertEquals(expected: "HI COMPLETABLEFUTURE!", result);
96 }
97 }

98 public String helloworld_3.async_calls_exceptionally(){
99     startTimer();
100
101     CompletableFuture<String> hello = CompletableFuture.supplyAsync(()->hws.hello())
102     CompletableFuture<String> world = CompletableFuture.supplyAsync(()->hws.world())
103     CompletableFuture<String> hicCompletableFuture = CompletableFuture.supplyAsync(
104         delay( delayMilliseconds: 1000);
105         return " Hi CompletableFuture!";
106     });
107
108     String hw= hello // hello
109     .exceptionally(e) -> {
110         log("Exception is :" + e.getMessage());
111         return "";
112     }
113     .thenCombine(world, (h, w) -> h+w) // "h-> "
114     .exceptionally(e) -> {
115         log("Exception after world is :" + e.getMessage());
116         return "";
117     }
118     .thenCombine(hicCompletableFuture, (previous,current)->previous+current)
119 // " Hi CompletableFuture!"
120     .thenApply(String::toUpperCase) // " HI COMPLETABLEFUTURE!"
121     .join(); // " HI COMPLETABLEFUTURE!"

122     timeTaken();
123     return hw;
124 }
125
126

```

## Exception Handling using exceptionally()



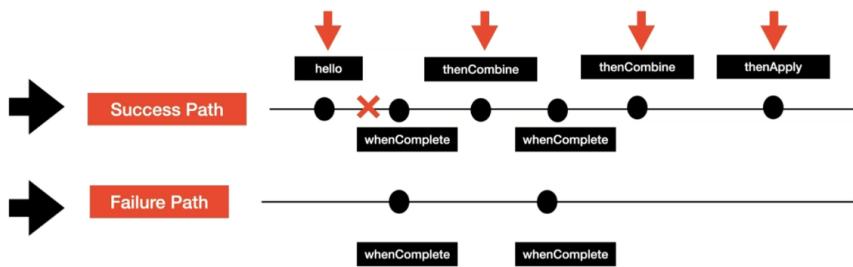
exceptionally is not there in success path and if we have failure then only exceptionally will work from failure path - just like try catch block.

## whenComplete()

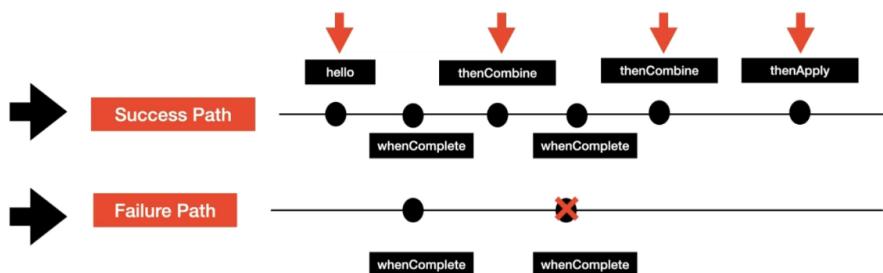
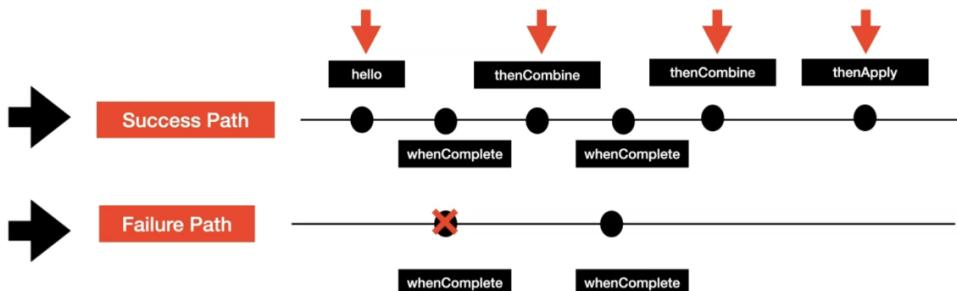
- Exception handler in CompletableFuture API
- Catches the Exception but does not recover from the exception

When complete is part of both success and failure path

### Exception Handling using whenComplete()

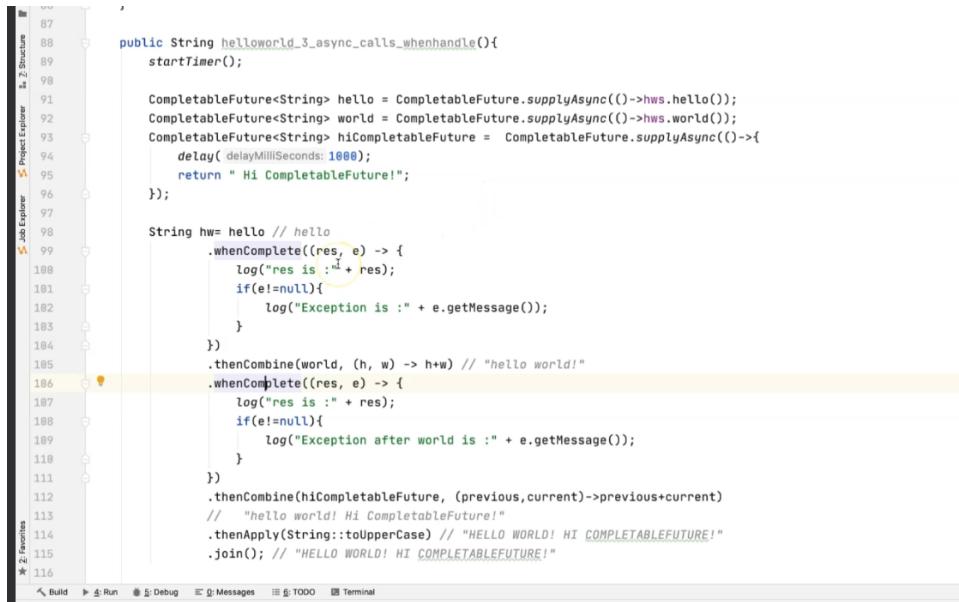


In case of any error it will not come to success path all.



We won't use this much. Good knowledge and apply when necessary.

WhenComplete will take BiConsumer as an argument. So you won't see any return statement



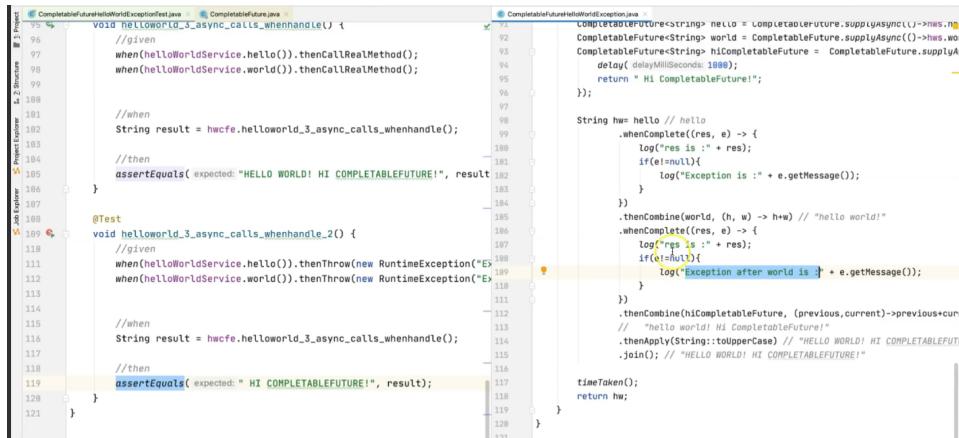
```
public String helloworld_3_async_calls_whenhandle(){
    startTimer();

    CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> hws.hello());
    CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> hws.world());
    CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(() ->{
        delay(delayMilliseconds: 1000);
        return " Hi CompletableFuture!";
    });

    String hw= hello // hello
        .whenComplete((res, e) -> {
            log("res is :" + res);
            if(e!=null){
                log("Exception is :" + e.getMessage());
            }
        })
        .thenCombine(world, (h, w) -> h+w) // "hello world!"
        .whenComplete((res, e) -> {
            log("res is :" + res);
            if(e!=null){
                log("Exception after world is :" + e.getMessage());
            }
        })
        .thenCombine(hiCompletableFuture, (previous,current)->previous+current)
        // "hello world! Hi CompletableFuture!"
        .thenApply(String::toUpperCase) // "HELLO WORLD! HI COMPLETABLEFUTURE!"
        .join(); // "HELLO WORLD! HI COMPLETABLEFUTURE!"

    return hw;
}
```

## Positive and negative test cases



```
void helloworld_3_async_calls_whenhandle() {
    //given
    when(helloWorldService.hello()).thenCallRealMethod();
    when(helloWorldService.world()).thenCallRealMethod();

    //when
    String result = hwcfe.helloworld_3_async_calls_whenhandle();
    //then
    assertEquals(expected: "HELLO WORLD! HI COMPLETABLEFUTURE!", result);
}

@Test
void helloworld_3_async_calls_whenhandle_2() {
    //given
    when(helloWorldService.hello()).thenThrow(new RuntimeException("Ex"));
    when(helloWorldService.world()).thenThrow(new RuntimeException("Ex"));

    //when
    String result = hwcfe.helloworld_3_async_calls_whenhandle();
    //then
    assertEquals(expected: " HI COMPLETABLEFUTURE!", result);
}
```

In negative test case it is very important to know if whenComplete execute at once it will not recover and will execute only whenComplete in the pipeline.



Run:  CompletableFutureHelloWorldExceptionTest...  
Test Result: 1 failed / 1 passed  
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...  
[ForkJoinPool.commonPool-worker-19] - res is :null  
[ForkJoinPool.commonPool-worker-19] - Exception is :java.lang.RuntimeException: Exception Occurred  
[ForkJoinPool.commonPool-worker-19] - res is :null  
[ForkJoinPool.commonPool-worker-19] - Exception after world is :java.lang.RuntimeException: Exception Occurred  
  
java.util.concurrent.CompletionException: java.lang.RuntimeException: Exception Occurred  
at java.base/java.util.concurrent.CompletableFuture.encodeThrowable(CompletableFuture.java:314)  
at java.base/java.util.concurrent.CompletableFuture.completeThrowable(CompletableFuture.java:319)

Handle this using exceptionally as shown below

The screenshot shows an IDE interface with two tabs open. The left tab displays a Java file named `CompletableFutureHelloWorldTest.java`, which contains a test for a `HelloWorldService`. The right tab shows the generated `CompletableFutureHelloWorldTest$CompletableFutureHelloWorldTest$1` class, which implements the logic for the test methods.

```
CompletableFutureHelloWorldTest.java
```

```
    //given
    when(helloWorldService.hello()).thenCallRealMethod();
    when(helloWorldService.world()).thenCallRealMethod();

    //when
    String result = hwcfe.helloworld_3_async_calls_whenhandle();

    //then
    assertEquals(expected: "HELLO WORLD! HI COMPLETABLEFUTURE!", result);

    @Test
    void helloworld_3_async_calls_whenhandle_2() {
        //given
        when(helloWorldService.hello()).thenThrow(new RuntimeException("E"));
        when(helloWorldService.world()).thenThrow(new RuntimeException("E"));

        //when
        String result = hwcfe.helloworld_3_async_calls_whenhandle();

        //then
        assertEquals(expected: " HI COMPLETABLEFUTURE!", result);
    }
}

CompletableFutureHelloWorldTest$CompletableFutureHelloWorldTest$1
```

```
    CompletableFuture<String> world = CompletableFuture.supplyAsync(()->ws.getWorld());
    CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(()->{
        delay(1000);
        return " HI CompletableFuture!";
    });

    String hw = hello();
    .whenComplete((res, e) -> {
        log("res is :" + res);
        if(e!=null){
            log("Exception is :" + e.getMessage());
        }
    })
    .thenCombine(world, (h, w) -> h+w) // "hello world"
    .whenComplete((res, e) -> {
        log("res is :" + res);
        if(e!=null){
            log("Exception after world is :" + e.getMessage());
        }
    })
    .exceptionally(e) -> {
        log("Exception after thenCombine is :" + e.getMessage());
        return "";
    }
    .thenCombine(hiCompletableFuture, (previous,current)->previous+current
    // " HI CompletableFuture!"
    .thenApply(String::toUpperCase) // " HI COMPLETABLEFUTURE!"
    .join(); // | HI COMPLETABLEFUTURE|"
```

```
    timeToken();
    return hw;
}
```

## Exception handle for review service

```
private InventoryService isMock;

@Mock
ProductServiceUsingCompletableFuture pscf;

@Test
void retrieveProductDetailsWithInventory_approach2() {
    //given
    String productId = "ABC123";
    when(isMock.retrieveProductInfo(any())).thenCallRealMethod();
    when(rshock.retrieveReviews(any())).thenThrow(new RuntimeException());
    when(ihock.retrieveInventory(any())).thenCallRealMethod();

    //when
    Product product = pscf.retrieveProductDetailsWithInventory_approach2(productId);

    //then
    assertNotNull(product);
    assertEquals(true, condition.product.getProductInfo().getProductOptions().size());
    product.getProductInfo().getProductOptions()
        .forEach(productOption -> {
            assertNotNull(productOption.getInventory());
        });
    assertNotNull(product.getReview());
    assertEquals(expected, product.getReview().getNoOfReviews());
}

public Product retrieveProductDetailsWithInventory_approach2(String productId) {
    stopWatch.start();

    CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
        .supplyAsync(() -> productService.retrieveProductInfo(productId))
        .thenApply(productInfo -> {
            productInfo.setProductOptions(updateInventory_approach2(productId));
            return productInfo;
        });

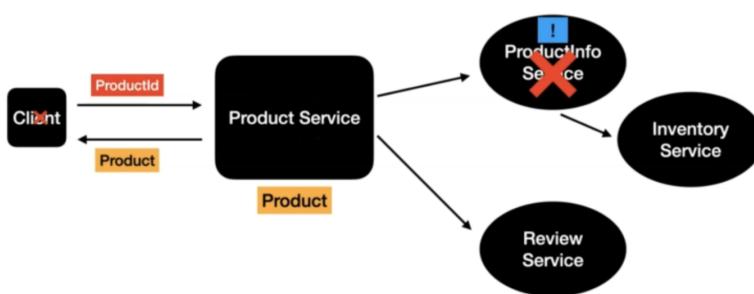
    CompletableFuture<Review> cfReview = CompletableFuture
        .supplyAsync(() -> reviewService.retrieveReviews(productId))
        .exceptionally(e -> {
            log("Handled the Exception in reviewService : " + e.getMessage());
            return Review.builder()
                .noOfReviews(1).overallRating(0.0)
                .build();
        });

    Product product = cfProductInfo
        .thenCombine(cfReview, (productInfo, review) -> new ProductBuilder()
            .join()); //block the thread

    stopWatch.stop();
    log("Total Time Taken : " + stopWatch.getTime());
    return product;
}
```

If product service fails entire calls fails

## **Product Service with Inventory**



The above exception scenario can be handle with `whenComplete()`

The screenshot shows the IntelliJ IDEA interface with a Java code editor. The code uses `CompletableFuture` to retrieve product details. It includes a `stopWatch` for timing, `productInfoService` for product info, and `reviewService` for reviews. The code constructs a `Product` object by combining `ProductInfo` and `Review` objects.

```
public Product retrieveProductDetailsWithInventory_approach2(String productId) {
    stopWatch.start();

    CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
        .supplyAsync(() -> productService.retrieveProductInfo(productId))
        .thenApply(productInfo -> {
            productInfo.setProductOptions(updateInventory_approach2(productInfo));
            return productInfo;
        });

    CompletableFuture<Review> cfReview = CompletableFuture
        .supplyAsync(() -> reviewService.retrieveReviews(productId))
        .exceptionally((e) -> {
            log("Handled the Exception in reviewService : " + e.getMessage());
            return Review.builder()
                .noOfReviews(0).overallRating(0.0)
                .build();
        });

    Product product = cfProductInfo
        .thenCombine(cfReview, (productInfo, review) -> new Product(productId, productInfo,
            review))
        .whenComplete(((product1, ex) -> {
            log("Inside WhenComplete : " + product1 + " and the exception is " + ex);
        }))
        .join(); //block the thread

    stopWatch.stop();
    log("Total Time Taken : " + stopWatch.getTime());
    return product;
}
```

e content Overview Q&A Notes Announcements Reviews Learning tools

The screenshot shows the IntelliJ IDEA interface with a Java test class `ProductServiceUsingCompletableFutureExceptionTest`. The test method `retrieveProductDetailsWithInventory_productInfoServiceError` asserts that an `AssertionError` is thrown when the `productInfoService` throws a `RuntimeException`.

```
assertNotNull(product.getReview());
assertEquals( expected: 0,product.getReview().getNoOfReviews());

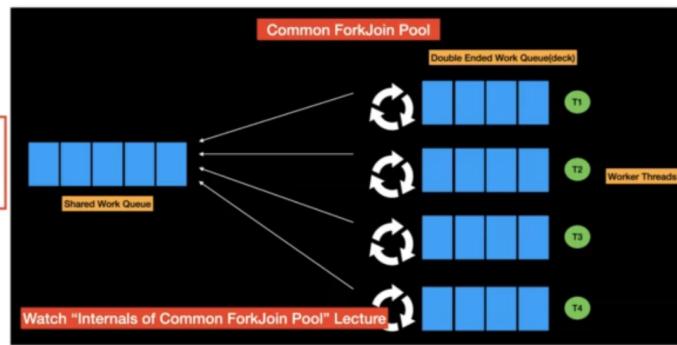
@Test
void retrieveProductDetailsWithInventory_productInfoServiceError() {
    //given
    String productId = "ABC123";
    when(pisMock.retrieveProductInfo(any())).thenThrow(new RuntimeException("Exception Occurred"));
    when(rsMock.retrieveReviews(any())).thenCallRealMethod();

    //then
    Assertions.assertThrows(RuntimeException.class, ()->pscf.retrieveProductDetailsWithInventory(productId));
}
```

# CompletableFuture - ThreadPool

- By default, CompletableFuture uses the **Common ForkJoinPool**

## Common ForkJoin Pool



```
66
67
68 public String helloworld_3_async_calls_log(){
69     startTimer();
70
71     CompletableFuture<String> hello = CompletableFuture.supplyAsync(()->hws.he
72     CompletableFuture<String> world = CompletableFuture.supplyAsync(()->hws.worl
73     CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyA
74         delay( delayMillisSeconds: 1000);
75         return " Hi CompletableFuture!";
76    });
77
78    String hw= hello
79        .thenCombine(world, (h, w) -> {
80            log("thenCombine h/w");
81            return h+w;
82        }) // first, second
83        .thenCombine(hiCompletableFuture, (previous,current)->{
84            log("thenCombine previous/current");
85            return previous+current;
86        })
87        .thenApply(s -> {
88            log("thenApply");
89            return s.toUpperCase();
90        })
91        .join();
92
93    timeTaken();
94    return hw;
95 }
```

The screenshot shows an IDE interface with two panes. The left pane displays Java code for a test class named `CompletableFutureFutureHelloWorldTest`. It contains two test methods: `helloWorld_3_async_calls_log()` and `helloWorld_thenCompose()`. The right pane shows the execution results of the tests. The output window displays the command used to run the test and the log output. The log output includes several entries from the `log` method, such as "inside world", "inside hello", and "thenCombine h/w". It also shows the execution of `thenCombine previous/current` and `thenApply` methods. The total time taken for the test is 1829 ms.

```
    @Test
    void helloWorld_3_async_calls_log() {
        //given
        //when
        String helloWorld = cfhw.helloWorld_3_async_calls_log();
        //then
        assertEquals("expected \"HELLO WORLD! HI COMPLETABLEFUTURE!\", helloWorld");
    }

    @Test
    void helloWorld_thenCompose() {
        //given
        startTimer();
        //when
        CompletableFuture<String> completableFuture = cfhw.helloWorld_thenCompose();
        //then
    }
}

Run: Completed: CompletableFutureFutureHelloWorldTest.helloworld_3_async_calls_log()
Run: Completed: CompletableFutureFutureHelloWorldTest.helloworld_thenCompose()

Test Result: 1/2 Tests passed: 1 of 2 test - 1s 53 ms
  ✓ Compat : 1/1 ms
  ✓ hello : 1/1 ms

/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...
[ForkJoinPool.commonPool-worker-5] - inside world
[ForkJoinPool.commonPool-worker-19] - inside hello
[ForkJoinPool.commonPool-worker-19] - thenCombine h/w
[ForkJoinPool.commonPool-worker-19] - thenCombine previous/current
[ForkJoinPool.commonPool-worker-19] - thenApply
[main] - Total Time Taken : 1829
```

# CompletableFuture & User Defined ThreadPool using ExecutorService

## Why use a different ThreadPool ?

- Common ForkJoinPool is shared by
    - ParallelStreams
    - CompletableFuture
  - Its common for applications to use **ParallelStreams** and **CompletableFuture** together
    - The following issues may occur:
      - Thread being blocked by a time consuming task
      - Thread not available

## Creating a User-Defined Thread Pool

```
Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
```

```
99
100 public String helloworld_3_async_calls_custom_threadpool(){
101     startTimer();
102
103     ExecutorService executorService = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
104     CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> hws.hello(), executorService);
105     CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> hws.world(), executorService);
106     CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(() ->{
107         delay(1000, TimeUnit.MILLISECONDS);
108         return " Hi CompletableFuture!";
109     }, executorService);
110
111     String hw= hello
112         .thenCombine(world, (h, w) -> {
113             log("thenCombine h/w");
114             return h+w;
115         }) // first, second
116         .thenCombine(hiCompletableFuture, (previous,current)->{
117             log("thenCombine previous/current");
118             return previous+current;
119         })
120         .thenApply(s -> {
121             log("thenApply");
122             return s.toUpperCase();
123         })
124         .join();
125
126     timeTaken();
127
128     return hw;
129 }
```

The screenshot shows the IntelliJ IDEA interface with two panes. The left pane displays Java code for testing a `helloWorld` method using `CompletableFuture`. The right pane shows the corresponding test results in the 'Run' tool window, indicating 1 test passed in 50 ms. The terminal below shows the command used to run the test.

```
    @Test
    void helloWorld_3_async_calls_custom_threadpool() {
        //given
        when
        String helloWorld = cfhw.helloworld_3.async_calls_custom_threadpool();
        //then
        assertEquals(expected: "HELLO WORLD! HI COMPLETABLEFUTURE!", helloWorld);
    }

    @Test
    void helloWorld_thenCompose() {
        //given
        startTimer();
        //when
        CompletableFuture<String> completableFuture = cfhw.helloWorld_thenCompose();
        //then
    }
}

Run: CompletableFutureHelloWorldTest.helloworld...
✓ Test Result 1 of 50 ms
✓ Compat 1 of 50 ms
✓ hello 1 of 50 ms

[pool-1-thread-2] - inside world
[pool-1-thread-1] - inside hello
[pool-1-thread-1] - thenCombine h/w
[pool-1-thread-1] - thenCombine previous/current
[pool-1-thread-1] - thenApply
[main] - Total Time Taken : 1827

Process finished with exit code 0
```

# Threads In CompletableFuture

In the above first one alone took different threads. rest are taken by a single thread. This is as per design. This is applicable for both common forkjoin pool and ExecutorService.

### Threads In CompletableFuture

There is overload methods as shown below for every method in next two slides.

## Async Overloaded Functions

- thenAccept()

```
public CompletableFuture<Void> thenAccept(Consumer<? super T> action) {  
    return uniAcceptStage(Objects.requireNonNull(action));  
}  
  
public CompletableFuture<Void> thenAcceptAsync(Consumer<? super T> action) {  
    return uniAcceptStage(defaultExecutor(), action);  
}  
  
public CompletableFuture<Void> thenAcceptAsync(Consumer<? super T> action,  
    Executor executor) {  
    return uniAcceptStage(screenExecutor(executor), action);  
}
```

### Async() Overloaded Functions

Regular Functions	Async() overloaded Functions
• thenCombine()	• thenCombineAsync()
• thenApply()	• thenApplyAsync()
• thenCompose()	• thenComposeAsync()
• thenAccept()	• thenAcceptAsync()

### Async() Overloaded Functions

- Using async() functions allows you to change the thread of execution
- Use this when you have blocking operations in your CompletableFuture pipeline

```

public String helloworld_3_async_calls_log_async(){
    startTimer();

    CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> hws.hello());
    CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> hws.world());
    CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(() -> {
        delay(1000);
        return " Hi CompletableFuture!";
    });

    String hw = hello
        .thenCombineAsync(world, (h, w) -> {
            log("thenCombine h/w");
            return h+w;
        }) // first, second
        .thenCombineAsync(hiCompletableFuture, (previous,current)->{
            log("thenCombine previous/current");
            return previous+current;
        })
        .thenApply(s -> {
            log("thenApply");
            return s.toUpperCase();
        })
        .join();

    timeTaken();
    return hw;
}

public String helloworld_3_async_calls_custom_threadpool(){
    startTimer();
}

```

At Least we see the difference in sequence.

```

@Test
void helloworld_3_async_calls_log_async() {
    //given
    //when
    String helloworld = cfhw.helloworld_3_async_calls_log_async();
}

Run
[ForJoinPool.commonPool-worker-19] - inside helloworld
[ForJoinPool.commonPool-worker-5] - inside world
[ForJoinPool.commonPool-worker-19] - thenCombine h/w
[ForJoinPool.commonPool-worker-19] - thenCombine previous/current
[ForJoinPool.commonPool-worker-19] - thenApply
[main] - Total Time Taken : 1838

```

When we apply ExecutorService

```

public String helloworld_3_async_calls_custom_threadpool() {
    startTimer();
    return hw;
}

private String helloworld() {
    String helloworld = cfhw.helloworld_3_async_calls_log_async();
    //then
    assertEquals("HELLO WORLD! HI COMPLETABLEFUTURE!", helloworld);
}

@Test
void helloworld_3_async_calls_custom_threadpool() {
    //given
    //when
    String helloworld = cfhw.helloworld_3_async_calls_custom_threadpool();
    //then
    assertEquals("HELLO WORLD! HI COMPLETABLEFUTURE!", helloworld);
}

@Test
void helloworld_3_async_calls_custom_threadpool_async() {
    //given
    //when
    String helloworld = cfhw.helloworld_3_async_calls_custom_threadpool_async();
    //then
    assertEquals("HELLO WORLD! HI COMPLETABLEFUTURE!", helloworld);
}

@Test
void helloworld_thenCompose() {
    //given
    startTimer();
    //when
    CompletableFuture<String> completableFuture = helloworld.thenCompose(h -> {
        return h + " thenCompose";
    });
    //then
    assertEquals("HELLO WORLD! HI COMPLETABLEFUTURE! thenCompose", completableFuture.get());
}

Run
[pool-1-thread-1] - inside helloworld
[pool-1-thread-2] - inside world
[pool-1-thread-4] - thenCombine h/w
[pool-1-thread-5] - thenCombine previous/current
[pool-1-thread-6] - thenApply
[main] - Total Time Taken : 1831

```

Lot of threads being used

```

public void helloworld_3_async_calls_custom_threadpool() {
    //given
    //when
    String helloworld = cfhw.helloworld_3_async_calls_custom_threadpool();
    //then
    assertEquals("HELLO WORLD! HI COMPLETABLEFUTURE!", helloworld);
}

@Test
void helloworld_thenCompose() {
    //given
    startTimer();
    //when
    CompletableFuture<String> completableFuture = helloworld.thenCompose(h -> {
        return h + " thenCompose";
    });
    //then
    assertEquals("HELLO WORLD! HI COMPLETABLEFUTURE! thenCompose", completableFuture.get());
}

Run
[pool-1-thread-1] - inside helloworld
[pool-1-thread-2] - inside world
[pool-1-thread-4] - thenCombine h/w
[pool-1-thread-5] - thenCombine previous/current
[pool-1-thread-6] - thenApply
[main] - Total Time Taken : 1831

```