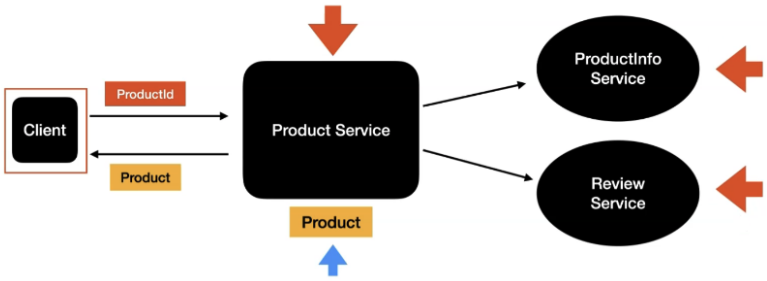
Product Service




```
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          return product;
38      }
```

```
10
11  class ProductServiceUsingCompletablaFutureTest {
12
13      private ProductInfoService pis = new ProductInfoService();
14      private ReviewService rs = new ReviewService();
15      ProductServiceUsingCompletablaFuture pscf = new
16              ProductServiceUsingCompletablaFuture(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      this.reviewService = reviewService;
21
22
23  ublic Product retrieveProductDetails(String productId) {
24      stopWatch.start();
25
26      CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
27              .supplyAsync(()-> productInfoService.retrieveProductInfo(produ
28      CompletableFuture<Review> cfReview = CompletableFuture
29              .supplyAsync(()-> reviewService.retrieveReviews(productId));
30
31      Product product = cfProductInfo
32              .thenCombine(cfReview, (productInfo,review)->new Product(produ
33              .join(); //block the thread
34
35      stopWatch.stop();
36      log("Total Time Taken : "+ stopWatch.getTime());
37      return product;
38
39
40  ublic static void main(String[] args) {
41
42      ProductInfoService productInfoService = new ProductInfoService();
43      ReviewService reviewService = new ReviewService();
44      ProductServiceUsingCompletablaFuture productService = new ProductServi
45      String productId = "ABC123";
46      Product product = productService.retrieveProductDetails(productId);
```

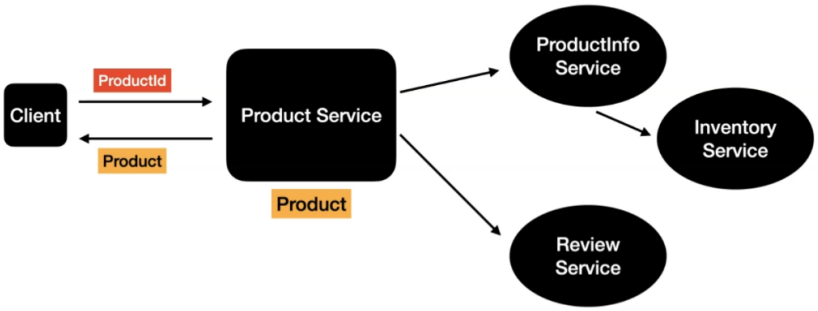
Retrieve product info has 1 sec delay similarly retrieve reviews. But output came in 1.1 second because of a completable Future.

Convert return type to completableFuture

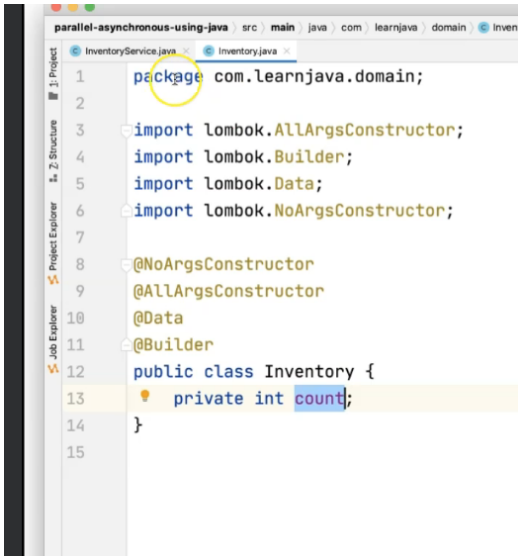
```
40
41  public CompletableFuture<Product> retrieveProductDetails_approach2(String productId) {
42
43      CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
44              .supplyAsync(()-> productInfoService.retrieveProductInfo(productId));
45      CompletableFuture<Review> cfReview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

Combining Streams & CompletableFuture

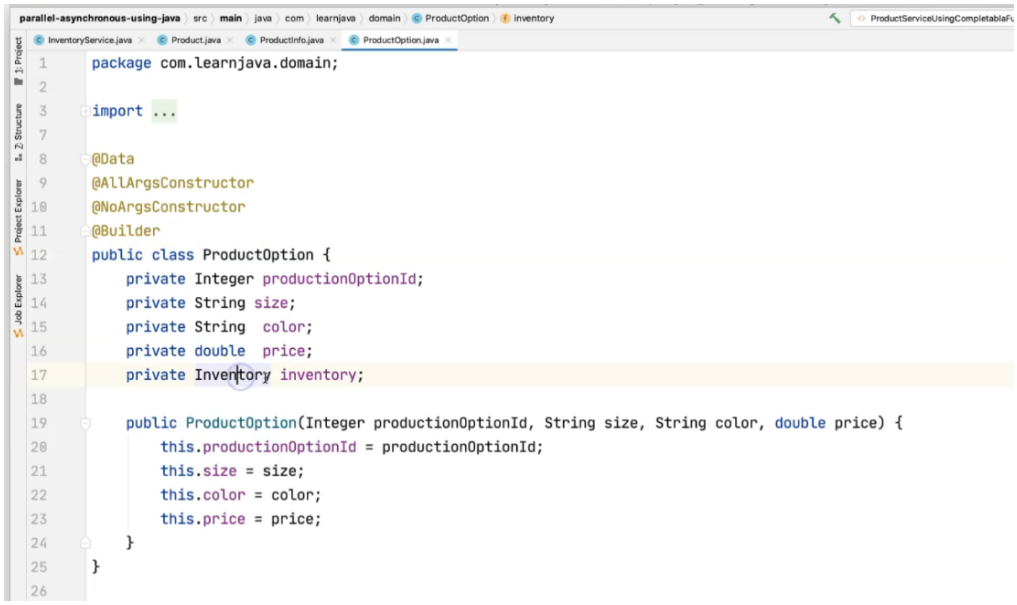
Product Service with Inventory



Inventory service is depends on productinfo service.


parallel-asynchronous-using-java › src › main › java › com › learnjava › domain › Inven
InventoryService.java
Inventory.java
1: Project
2: Structure
Project Explorer
Job Explorer
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

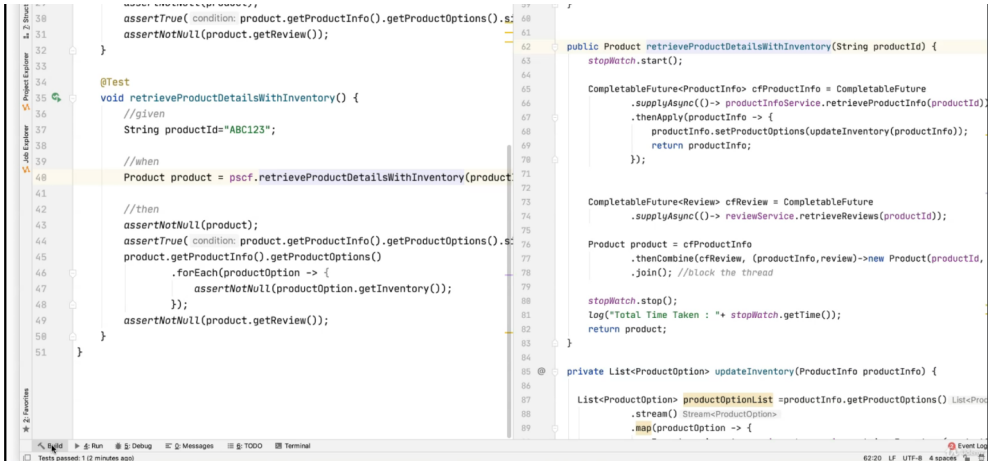

2: Structure
Project Explorer
Job Explorer
2
3 import ...
7
8 public class InventoryService {
9 public Inventory retrieveInventory(ProductOption productOption) {
10 delay(delayMilliSeconds: 500);
11 return Inventory.builder()
12 .count(2).build();
13
14 }
15 }
16


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

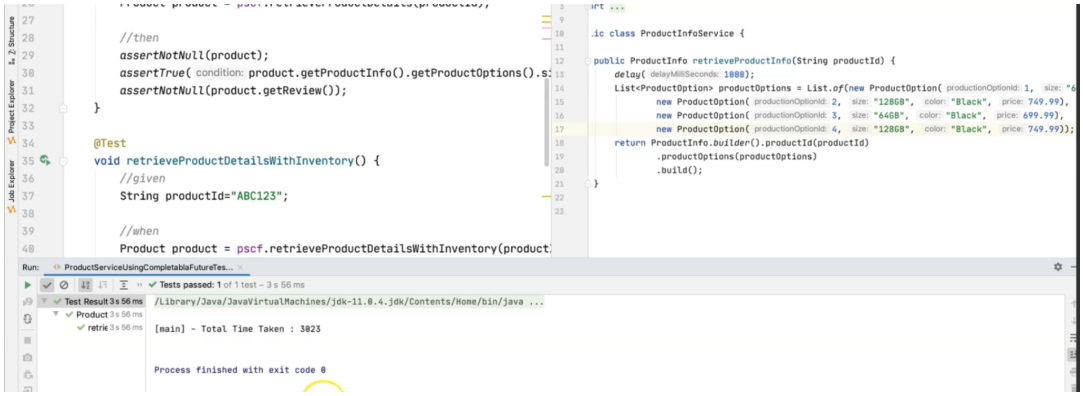
```java
public Product retrieveProductDetailsWithInventory(String productId) {
    stopWatch.start();

    CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
            .supplyAsync(()-> productInfoService.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;
}
```

```java
private List<ProductOption> updateInventory(ProductInfo productInfo) {

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

}
```

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

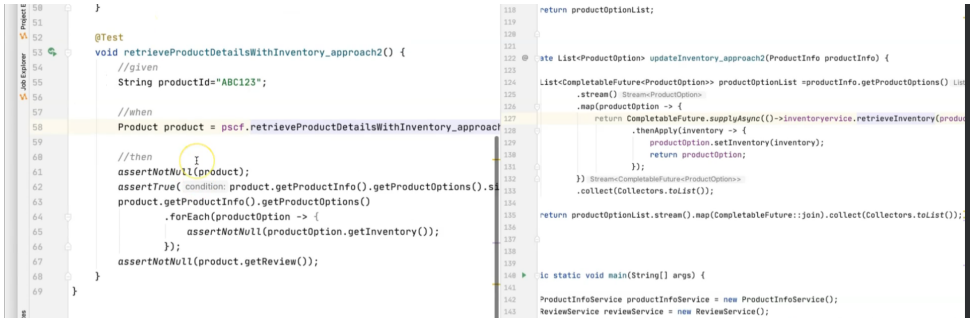


The more the number of productOption records - it takes more time.



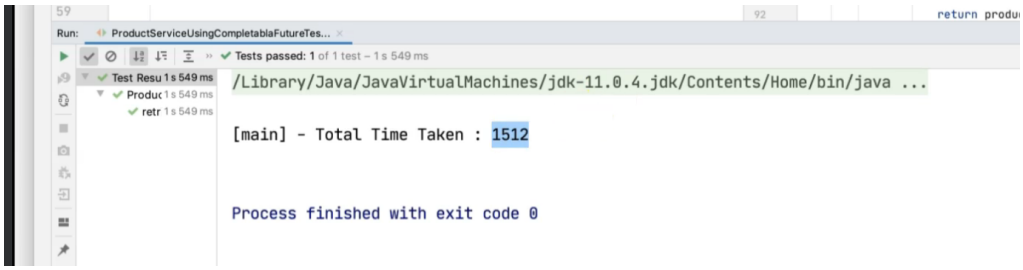
To improve performance covert update inventory to completable future as shown below.

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



```
}

@Test
void retrieveProductDetailsWithInventory_approach2() {
    //given
    String productId="ABC123";

    //when
    Product product = pscf.retrieveProductDetailsWithInventory_approach

    //then
    assertNotNull(product);
    assertTrue( condition: product.getProductInfo().getProductOptions().s
    product.getProductInfo().getProductOptions()
            .forEach(productOption -> {
                assertNotNull(productOption.getInventory());
            });
    assertNotNull(product.getReview());
}
}
```

```
return productOptionList;

ate List<ProductOption> updateInventory_approach2(ProductInfo productInfo) {

List<CompletableFuture<ProductOption>> productOptionList =productInfo.getProductOptions() List
        .stream() Stream<ProductOption>
        .map(productOption -> {
            return CompletableFuture.supplyAsync(()->inventoryervice.retrieveInventory(produc
                    .thenApply(inventory -> {
                        productOption.setInventory(inventory);
                        return productOption;
                    });
        }) Stream<CompletableFuture<ProductOption>>
        .collect(Collectors.toList());

return productOptionList.stream().map(CompletableFuture::join).collect(Collectors.toList());


ic static void main(String[] args) {

ProductInfoService productInfoService = new ProductInfoService();
ReviewService reviewService = new ReviewService();
```



```
Run:  ProductServiceUsingCompletablaFutureTes...

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

Test Resu 1 s 549 ms
Produc 1 s 549 ms
retr 1 s 549 ms

/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

[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_calls1() {
    try{
        CompletableFuture<String> hello = CompletableFuture.supplyAsync(() -> this.hws.hello());
        CompletableFuture<String> world = CompletableFuture.supplyAsync(() -> this.hws.world());
        CompletableFuture<String> hiCompletableFuture = CompletableFuture.supplyAsync(() -> {
            delay( delayMillSeconds: 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

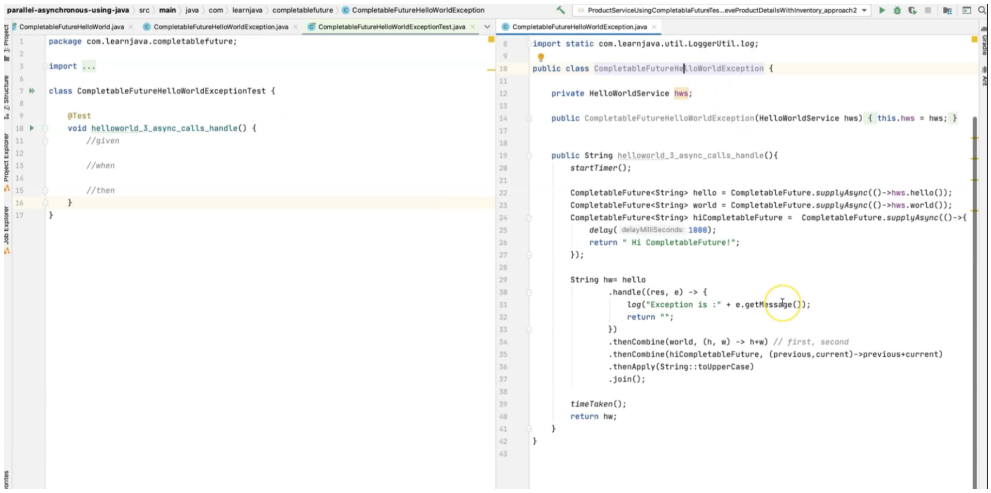
- CompletableFuture API has functional style of handling exceptions
- Three options available:
  - handle()
  - exceptionally()
  - whenComplete()

handle() / exceptionally() → Catch Exception and Recover

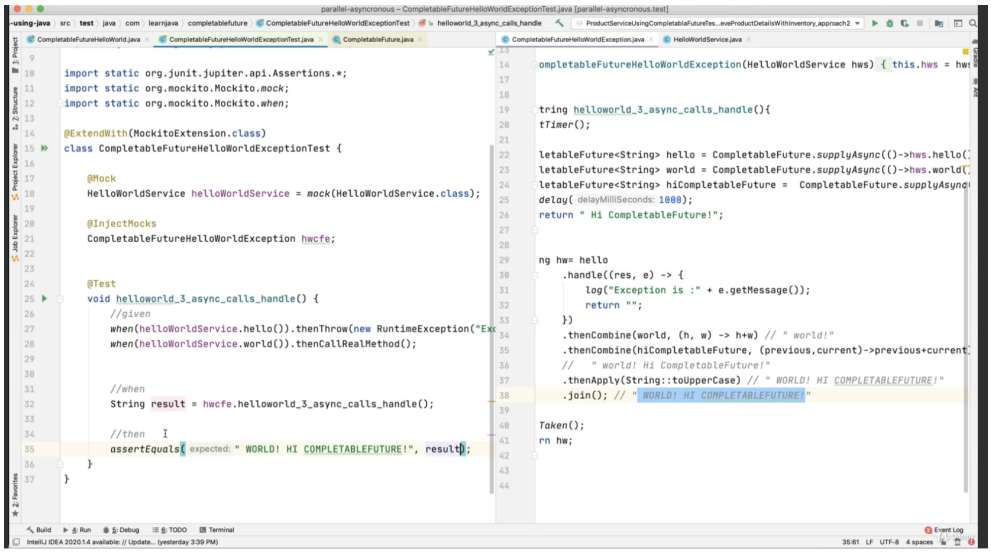
whenComplete() → Catch Exception and Does not Recover

Handle method takes BiFunction.
Exceptionally method takes Function.

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



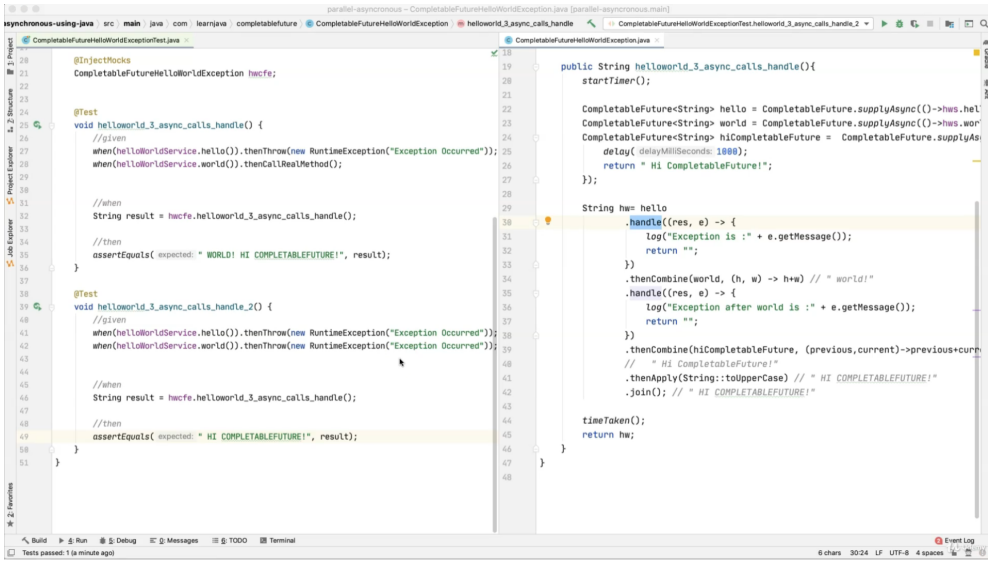
Unit test



```
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

@ExtendWith(MockitoExtension.class)
class CompletableFutureHelloWorldExceptionTest {

    @Mock
    HelloWorldService helloWorldService = mock(HelloWorldService.class);

    @InjectMocks
    CompletableFutureHelloWorldException hwcfe;


    @Test
    void helloworld_3_async_calls_handle() {
        //given
        when(helloWorldService.hello()).thenThrow(new RuntimeException("Ex
        when(helloWorldService.world()).thenCallRealMethod();


        //when
        String result = hwcfe.helloworld_3_async_calls_handle();

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

```
ompletableFutureHelloWorldException(HelloWorldService hws) { this.hws = hw

tring helloworld_3_async_calls_handle(){
tTimer();

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


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

Taken();
rn hw;
```

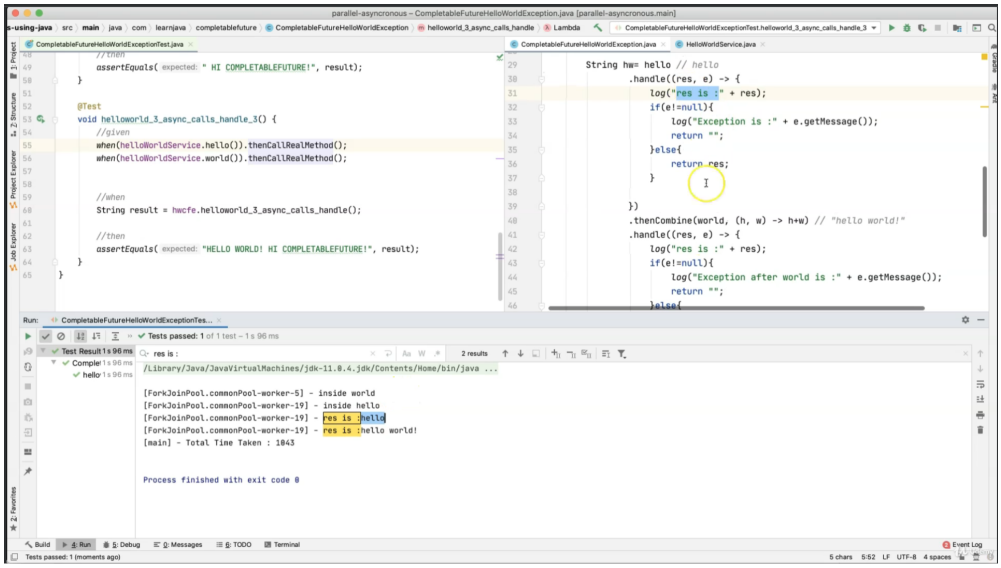
The above one is one completableFuture fail and below is multi completableFuture fails.



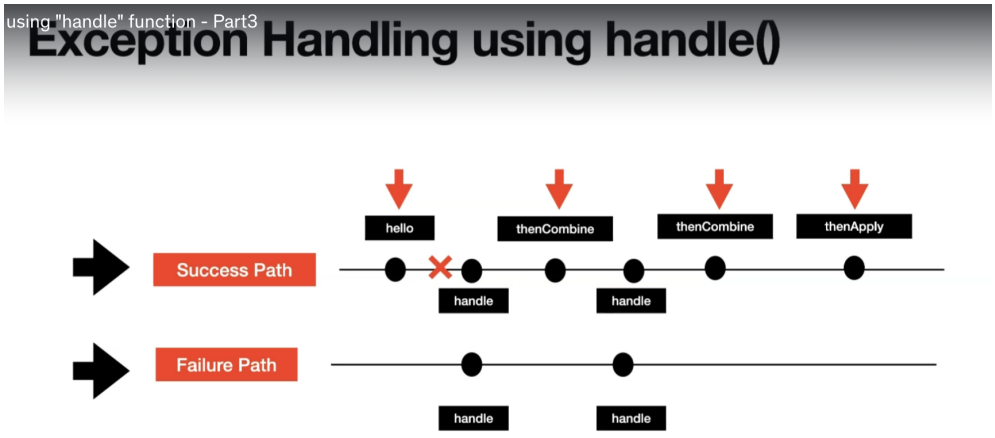
```
    @InjectMocks
    CompletableFutureHelloWorldException hwcfe;


    @Test
    void helloworld_3_async_calls_handle() {
        //given
        when(helloWorldService.hello()).thenThrow(new RuntimeException("Exception Occurred"));
        when(helloWorldService.world()).thenCallRealMethod();


        //when
        String result = hwcfe.helloworld_3_async_calls_handle();

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

    @Test
    void helloworld_3_async_calls_handle_2() {
        //given
        when(helloWorldService.hello()).thenThrow(new RuntimeException("Exception Occurred"));
        when(helloWorldService.world()).thenThrow(new RuntimeException("Exception Occurred"));


        //when
        String result = hwcfe.helloworld_3_async_calls_handle();

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

```
    public String helloworld_3_async_calls_handle(){
        startTimer();

        CompletableFuture<String> hello = CompletableFuture.supplyAsync(()->hws.hel
        CompletableFuture<String> world = CompletableFuture.supplyAsync(()->hws.wor
        CompletableFuture<String> hiCompletableFuture =  CompletableFuture.supplyAs
            delay( delayMilliSeconds: 1000);
            return " Hi CompletableFuture!";
        });

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

        timeTaken();
        return hw;
    }
}
```

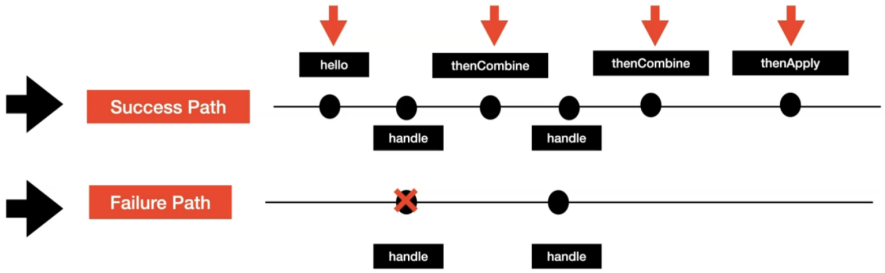
When there is no error the handle method still execute so we need to do null check and proceed as shown in the below screenshot.



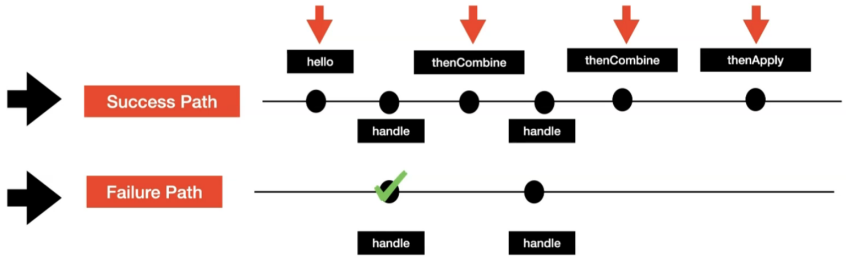
Summary how it wors



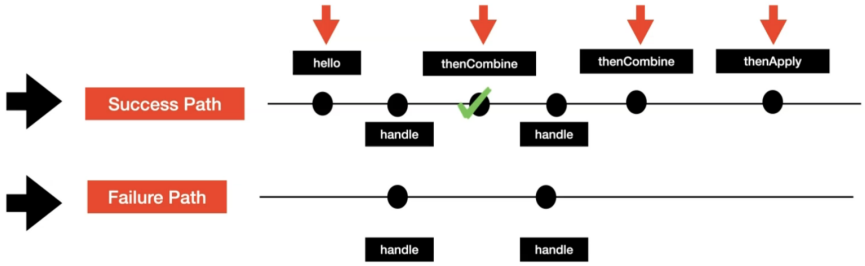
Ee got exception in the hello then it will go to failure path handle



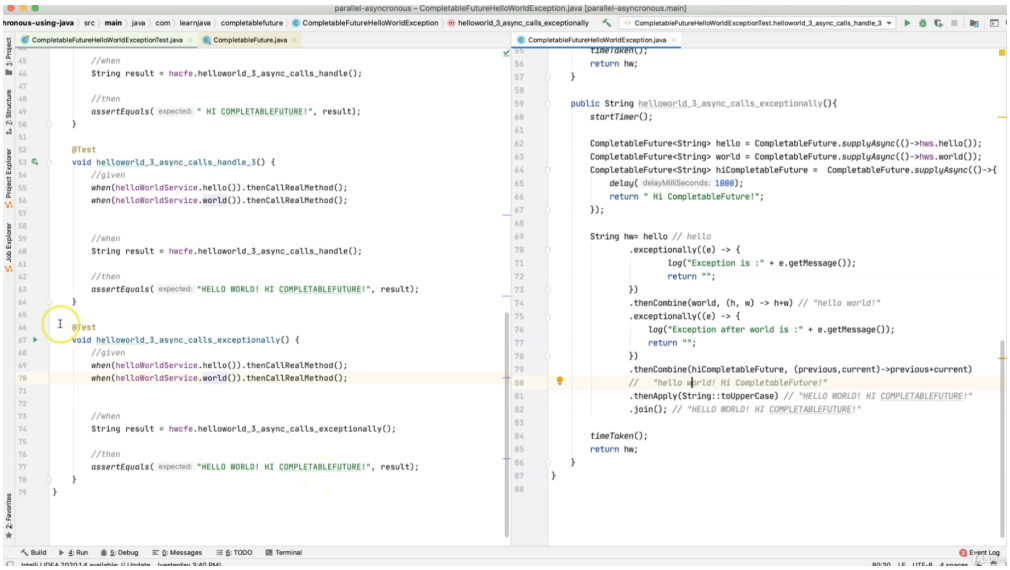
From there it recover



And go to the Success Path thenCombine.



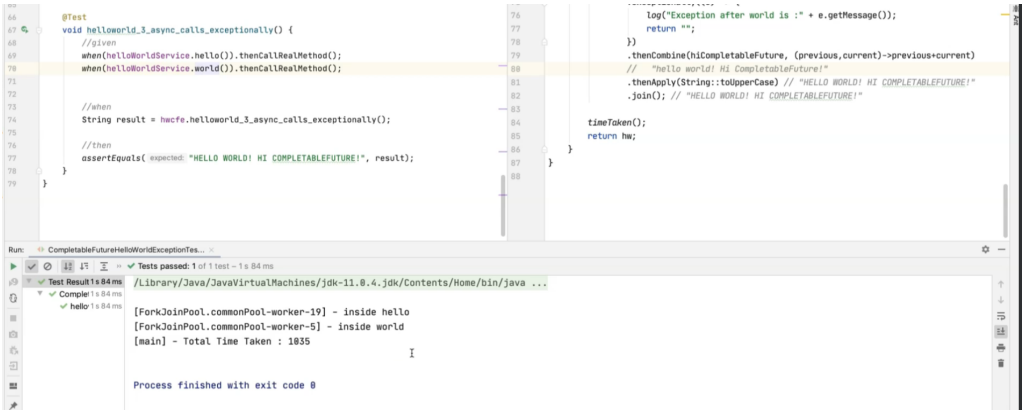
Handle the same thing using exceptionally as shown below. For this no additional logic is required as we did in the handle method like null check. Since this exceptionally works only when there is an error there is no guarantee it will execute during pipeline execute.



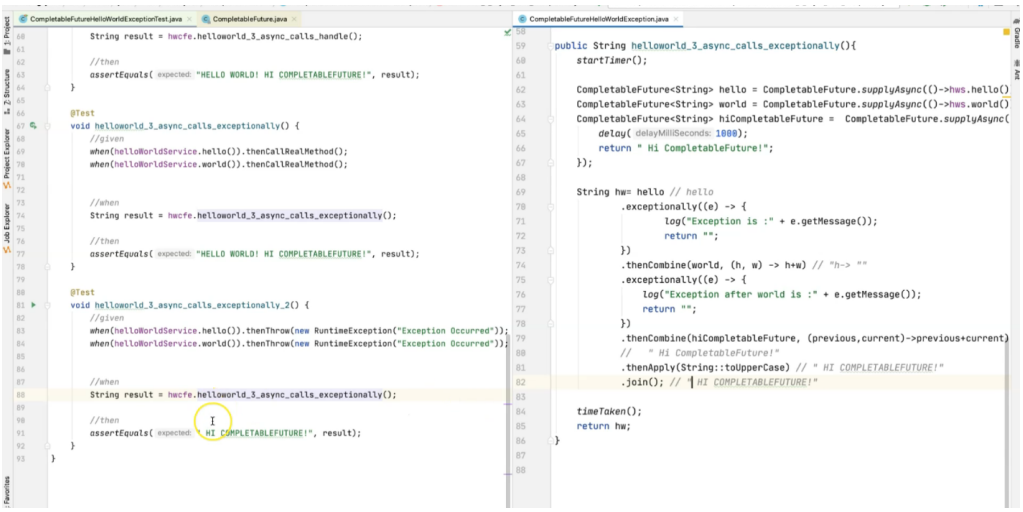
```
//when
String result = hwcfe.helloworld_3_async_calls_handle();

//then
assertEquals( expected: " 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();

//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 = hwcfe.helloworld_3_async_calls_exceptionally();

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

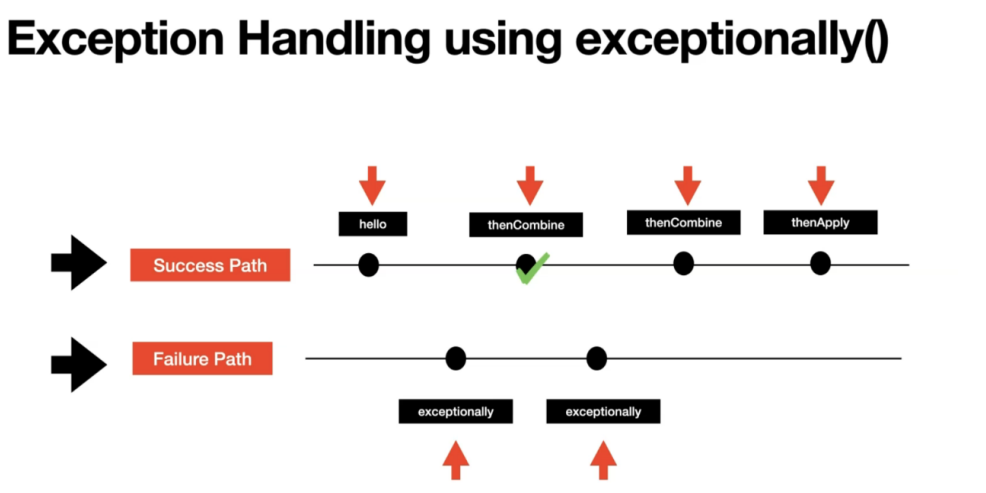
```
timeTaken();
return hw;
}

public String helloworld_3_async_calls_exceptionally(){
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
        .exceptionally((e) -> {
            log("Exception is :" + e.getMessage());
            return "";
        })
        .thenCombine(world, (h, w) -> h+w) // "hello world!"
        .exceptionally((e) -> {
            log("Exception after world is :" + e.getMessage());
            return "";
        })
        .thenCombine(hiCompletableFuture, (previous,current)->previous+current)
        //  "hello world! Hi CompletableFuture!"
        .thenApply(String::toUpperCase) // "HELLO WORLD! HI COMPLETABLEFUTURE!"
        .join(); // "HELLO WORLD! HI COMPLETABLEFUTURE!"

timeTaken();
return hw;
}
}
```

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



Same code when we have exception in unit testing





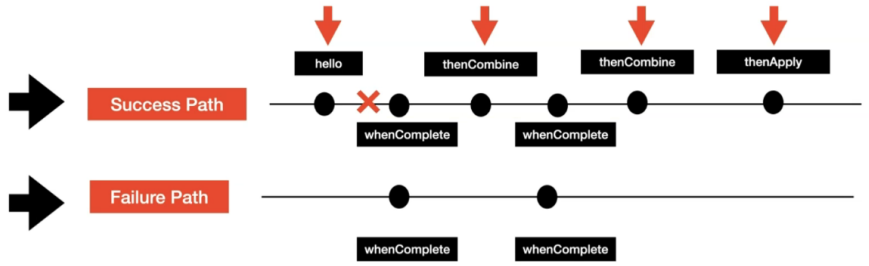
exceptionally is not there in success path and if we have failure then only exceptionally will work from failure path - just like try catch block.

whenComplete()

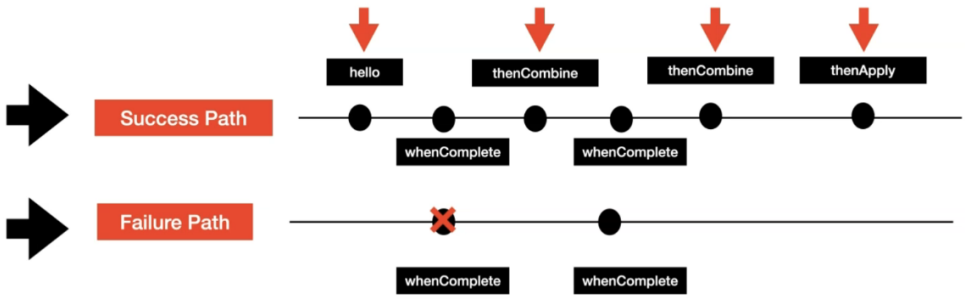
- Exception handler in CompletableFuture API
- Catches the Exception but does not recover from the exception

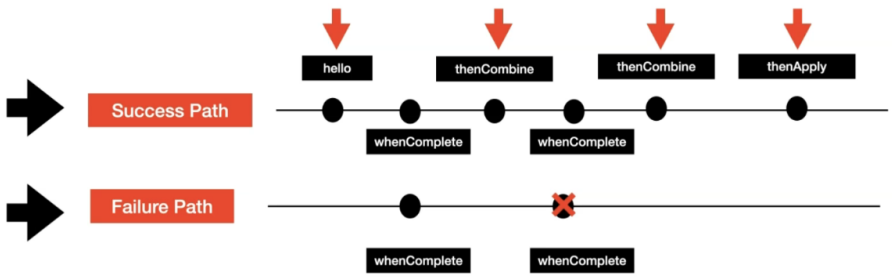
When complete is part of both success and failure path

Exception Handling using whenComplete()



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



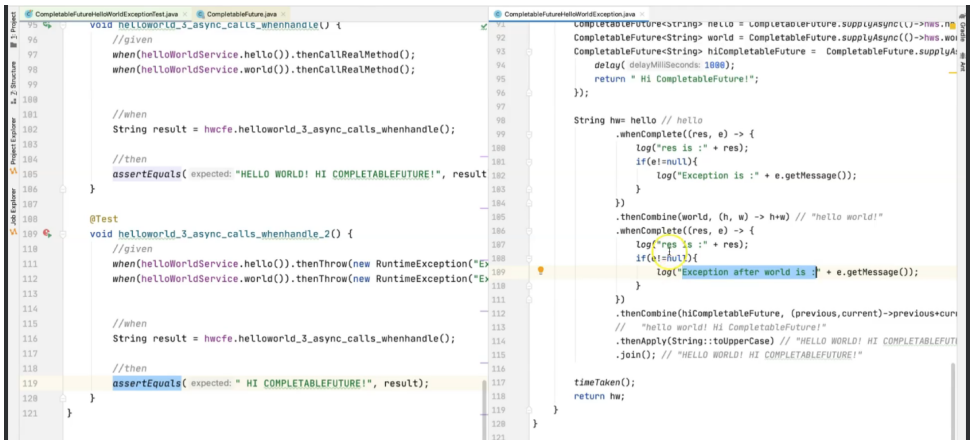


We won’t use this much. Good knowledge and apply when necessary.

WhenComplete will take Biconsumer as an argument. So you won’t see any return statement

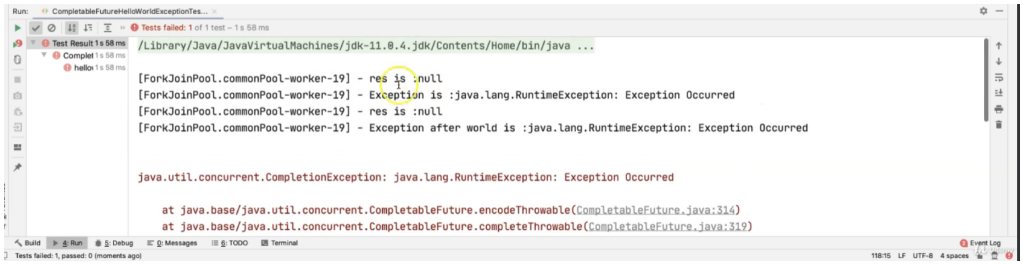
```
public String helloworld_3_async_calls_whenhandle(){
    startTimer();

    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!"
```

Positive and negative test cases



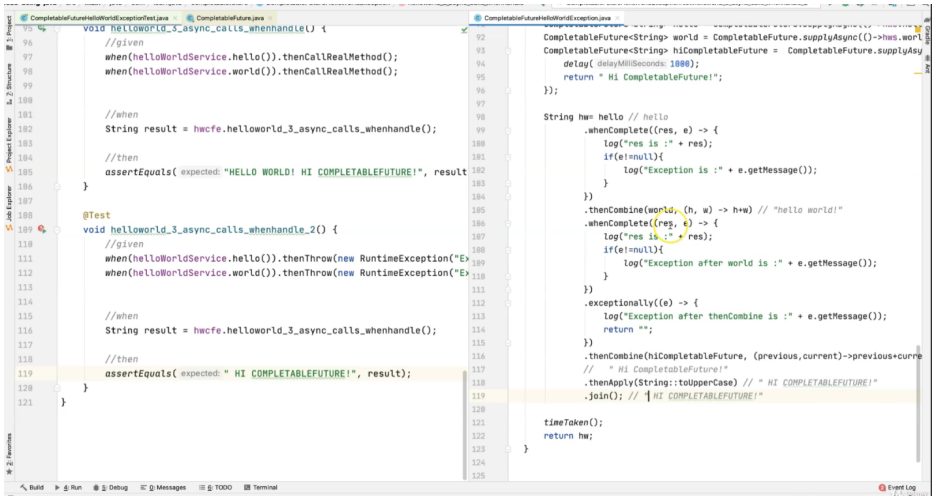
```
void helloworld_3_async_calls_whenhandle() {
    //given
    when(helloWorldService.hello()).thenCallRealMethod();
    when(helloWorldService.world()).thenCallRealMethod();


    //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("E
    when(helloWorldService.world()).thenThrow(new RuntimeException("E


    //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.

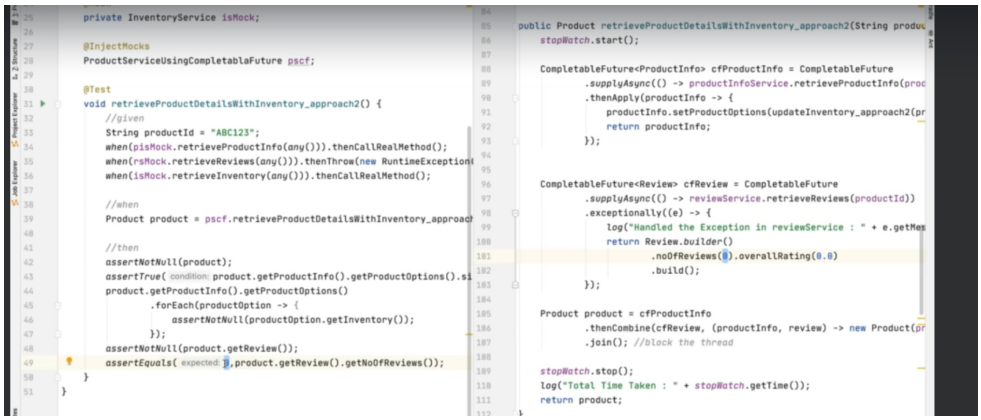


```
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

[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)
```

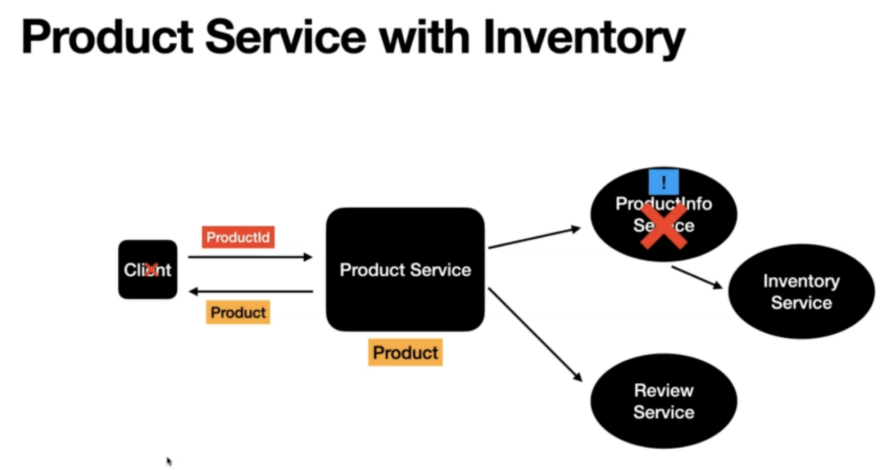
Handle this using exceptionally as shown below



Exception handle for review service



If product service fails entire calls fails



The above exception scenario can be handle with whenComplete()

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

    CompletableFuture<ProductInfo> cfProductInfo = CompletableFuture
            .supplyAsync(() -> productInfoService.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, product
            .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;
```

Event Log

```java
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
```

Build ▶ 4: Run 5: Debug 0: Messages 6: TODO Terminal

IntelliJ IDEA 2020.1.4 available: // Update... (yesterday 3:58 PM) 18 chars 89:54 LF UTF-8 4 spa

e content **Overview** Q&A Notes Announcements Reviews Learning tools

java com learnjava completablefuture ProductServiceUsingCompletablaFutureExceptionTest retrieveProductDetailsWithInventory_produ

ProductServiceUsingCompletablaFuture.java ProductServiceUsingCompletablaFutureExceptionTest.java Assertions.java

```java
        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 Occ
        when(rsMock.retrieveReviews(any())).thenCallRealMethod();

        //then
        Assertions.assertThrows(RuntimeException.class, ()->pscf.retrieveProductDetailsWithInv
    }
}
```

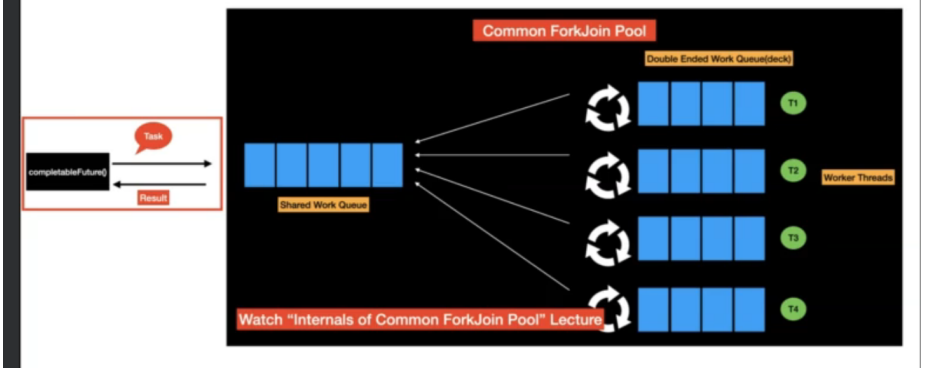
Run: ProductServiceUsingCompletablaFutureExc...

Tests passed: 1 of 1 test – 1 s 474 ms

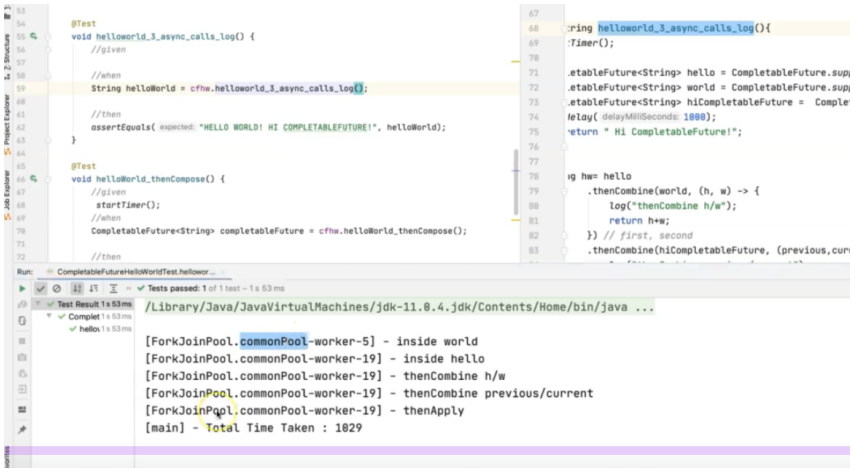
CompletableFuture - ThreadPool

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

Common ForkJoin Pool




```
}

public String helloworld_3_async_calls_log(){
    startTimer();

    CompletableFuture<String> hello = CompletableFuture.supplyAsync(()->hws.he
    CompletableFuture<String> world = CompletableFuture.supplyAsync(()->hws.wo
    CompletableFuture<String> hiCompletableFuture =  CompletableFuture.supplyA
        delay( delayMilliSeconds: 1000);
        return " Hi CompletableFuture!";
    });

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

    timeTaken();
    return hw;
}
```



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 ThreadPool

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

```
public String helloworld_3_async_calls_custom_threadpool(){
    startTimer();

    ExecutorService executorService = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
    CompletableFuture<String> hello = CompletableFuture.supplyAsync(()->hws.hello(),executorService);
    CompletableFuture<String> world = CompletableFuture.supplyAsync(()->hws.world(),executorService);
    CompletableFuture<String> hiCompletableFuture =  CompletableFuture.supplyAsync(()->{
        delay( delayMilliSeconds: 1000);
        return " Hi CompletableFuture!";
    },executorService);

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

    timeTaken();
    return hw;
}
```

```
@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
```

```
public String helloworld_3_async_calls_custom_thre
    startTimer();

    ExecutorService executorService = Executors.ne
    CompletableFuture<String> hello = CompletableF
    CompletableFuture<String> world = CompletableF
    CompletableFuture<String> hiCompletableFuture
        delay( delayMilliSeconds: 1000);
        return " Hi CompletableFuture!";
    },executorService);

    String hw= hello
            .thenCombine(world, (h, w) -> {
                log("thenCombine h/w");
                return h+w;
            }) // first, second
            .thenCombine(hiCompletableFuture, (pre
                log("thenCombine previous/current"
```

```
Run:  CompletableFutureHelloWorldTest.hellowor...
Tests passed: 1 of 1 test – 1 s 50 ms
Test Result 1 s 50 ms
Complet 1 s 50 ms
hello 1 s 50 ms
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

[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 : 1027


Process finished with exit code 0
```

Threads
In
CompletableFuture

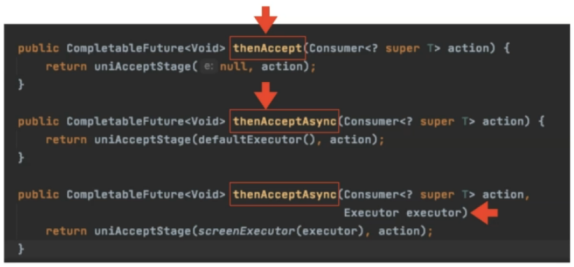
In the above first one alone took different threads. rest are taken by a single thread. This is as per design. This is applicable for both common forkjoin pool and ExecutorService.

Threads
In
CompletableFuture

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

Async Overloaded Functions

- thenAccept()



Async() Overloaded Functions

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

Async() Overloaded Functions

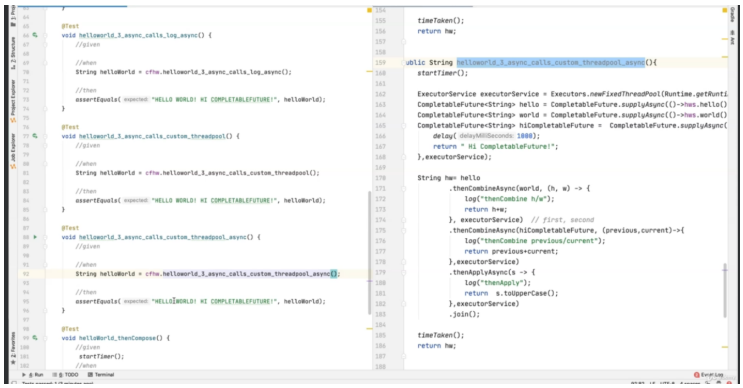
- Using async() functions allows you to change the thread of execution
- Use this when you have blocking operations in your Completablefuture pipeline

```java
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( delayMilliSeconds: 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;
            })
            .thenApplyAsync(s -> {
                log("thenApply");
                return  s.toUpperCase();
            })
            .join();

    timeTaken();
    return hw;
}

public String helloworld_3_async_calls_custom_threadpool(){
    startTimer();
```

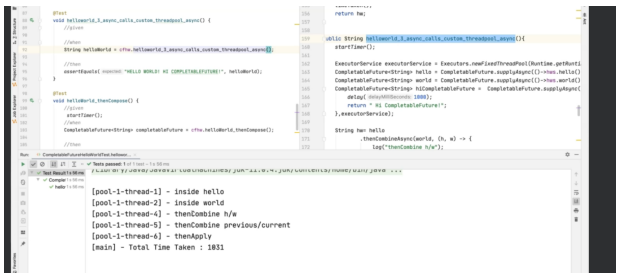
At Least we see the difference in sequence.

```
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

[ForkJoinPool.commonPool-worker-19] - inside hello
[ForkJoinPool.commonPool-worker-5] - inside world
[ForkJoinPool.commonPool-worker-19] - thenCombine h/w
[ForkJoinPool.commonPool-worker-19] - thenCombine previous/current
[ForkJoinPool.commonPool-worker-19] - thenApply
[main] - Total Time Taken : 1030
```

When we apply ExecutorService



Lot of threads being used



```
[pool-1-thread-1] - inside hello
[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 : 1031
```