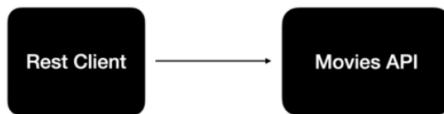
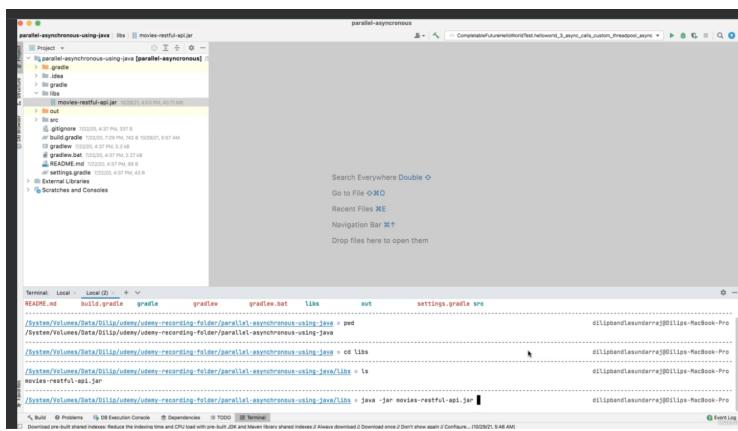
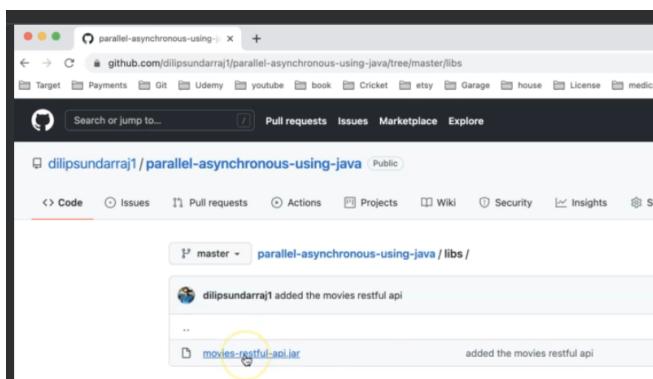


Build Restful API client using Spring WebClient and Completable Feature



Why Spring WebClient?

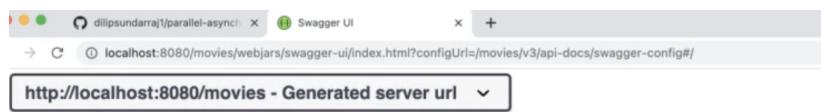
- Spring is one of the popular framework in the Java Community
- **Spring WebClient** is a rest client that got released as part of **Spring 5**
- **Spring WebClient** is a functional style RestClient
- **Spring WebClient** can be used as a blocking or non blocking Rest Client



```

2021-10-31 05:01:57.506 INFO 23932 --- [           main] com.movies.config.DBConfig          : Inside the DB initializer
2021-10-31 05:01:59.073 INFO 23932 --- [           main] o.s.b.web.embedded.netty.NettyWebServer : Netty started on port 8080
2021-10-31 05:01:59.083 INFO 23932 --- [           main] c.movies.MoviesRestfulApiApplicationKt : Started MoviesRestfulApiApplicationKt in 4.745 seconds (JVM running for 5.436)
2021-10-31 05:01:59.254 INFO 23932 --- [           main] com.movies.initialize.DataInitializer : Persisted Movie Ids are : [1, 2, 3, 4, 5, 6, 7]
2021-10-31 05:01:59.258 INFO 23932 --- [           main] com.movies.initialize.DataInitializer : No of persisted Movies are : 7
2021-10-31 05:03:23.564 INFO 23932 --- [ctor-http-nio-3] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 347 ms

```



review-controller

movie-info-controller

Movie Endpoint

```

Curl
curl -X GET "http://localhost:8080/movies/v1/movie_infos" -H "accept: */*"

Request URL
http://localhost:8080/movies/v1/movie_infos

Server response
Code Details

200 Response body
{
  "cast": [
    "Christian Bale",
    "Michael Caine"
  ],
  "release_date": "2005-06-15"
},
{
  "movieInfoId": 2,
  "name": "The Dark Knight",
  "year": 2008,
  "cast": [
    "Christian Bale",
    "Heath Ledger"
  ],
  "release_date": "2008-07-18"
},
{
  "movieInfoId": 3,
  "name": "Dark Knight Rises",
  "year": 2012
}

```

Review Endpoint

```

Request URL
http://localhost:8080/movies/v1/reviews?movieInfoId=1

Server response
Code Details

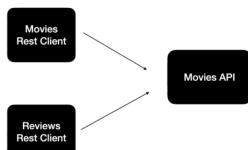
200 Response body
[
  {
    "reviewId": 1,
    "movieInfoId": 1,
    "rating": 8.2,
    "comment": "Nolan is the real superhero"
  }
]

Response headers
content-type: application/json
transfer-encoding: chunked

Responses

```

Rest Clients

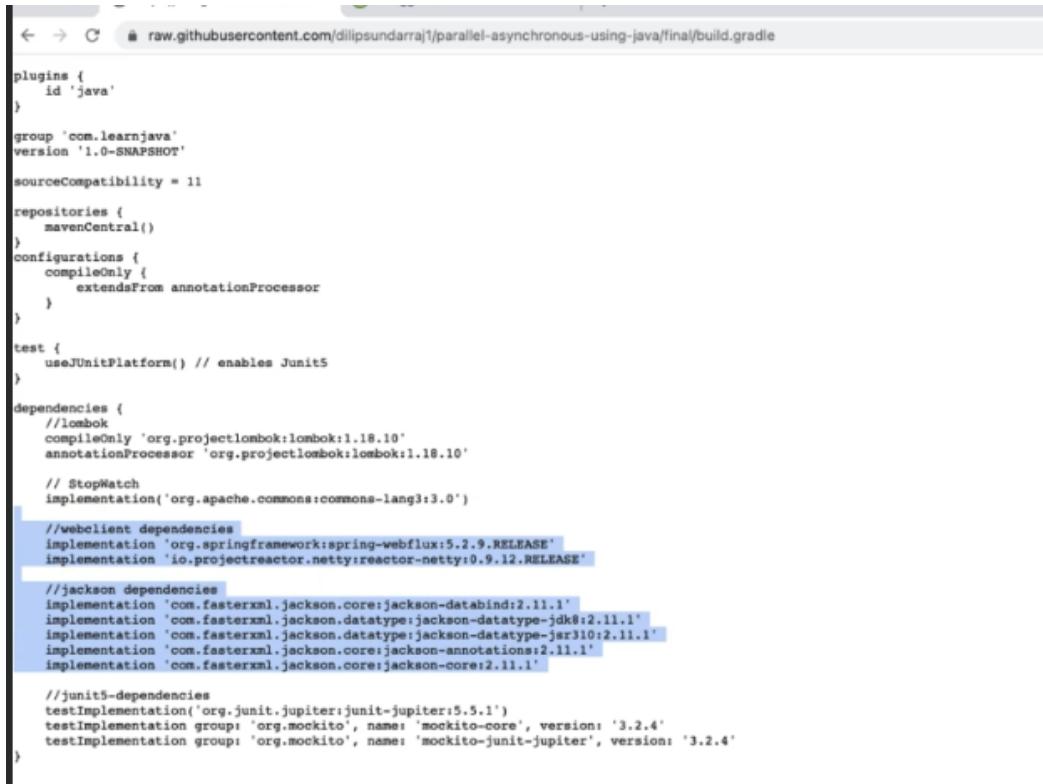




```
import ...
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Movie {
    private MovieInfo movieInfo;
    private List<Review> reviewList;
}
```

In real time the movie api and review api are two different micro services.

Copy below dependencies



```
plugins {
    id 'java'
}

group 'com.learnjava'
version '1.0-SNAPSHOT'

sourceCompatibility = 11

repositories {
    mavenCentral()
}
configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}
test {
    useJUnitPlatform() // enables Junit5
}

dependencies {
    //lombok
    compileOnly 'org.projectlombok:lombok:1.18.10'
    annotationProcessor 'org.projectlombok:lombok:1.18.10'

    // Stopwatch
    implementation('org.apache.commons:commons-lang3:3.0')

    //webclient dependencies
    implementation 'org.springframework:spring-webflux:5.2.9.RELEASE'
    implementation 'io.projectreactor.netty:reactor-netty:0.9.12.RELEASE'

    //jackson dependencies
    implementation 'com.fasterxml.jackson.core:jackson-databind:2.11.1'
    implementation 'com.fasterxml.jackson.datatype:jackson-datatype-jdk8:2.11.1'
    implementation 'com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.11.1'
    implementation 'com.fasterxml.jackson.core:jackson-annotations:2.11.1'
    implementation 'com.fasterxml.jackson.core:jackson-core:2.11.1'

    //junit5-dependencies
    testImplementation('org.junit.jupiter:junit-jupiter:5.5.1')
    testImplementation group: 'org.mockito', name: 'mockito-core', version: '3.2.4'
    testImplementation group: 'org.mockito', name: 'mockito-junit-jupiter', version: '3.2.4'
}
```

```
parallel-asynchronous – MoviesClient.java [parallel-asynchronous.main]
parallel-asynchronous-using-java | src | main | java | com | learjava | apclient | MoviesClient | retrieveMovie
MoviesClient.java | Movie.java | build.gradle (parallel-asynchronous) | CompletableFutureHelloWorldTest.h
import org.springframework.web.reactive.function.client.WebClient
5
6 public class MoviesClient {
7
8     private final WebClient webClient;
9
10    public MoviesClient(WebClient webClient) {
11        this.webClient = webClient;
12    }
13
14    public Movie retrieveMovie(Long movieInfoId){
15
16        //movieInfo
17        //review
18        return null;
19    }
20}
```

```
46    private MovieInfo invokeMovieInfoService(Long movieInfoId) {
47
48        var moviesInfoUrlPath = "/v1/movie_infos/{movieInfoId}";
49
50        return webClient
51            .get() WebClient.RequestHeadersUriSpec<...>
52            .uri(moviesInfoUrlPath, movieInfoId) capture of ?
53            .retrieve() WebClient.ResponseSpec
54            .bodyToMono(MovieInfo.class) Mono<MovieInfo>
55            .block();
56
57
```

```
27    private List<Review> invokeReviewsService(Long movieInfoId) {
28
29        ///v1/reviews?movieInfoId=1
30        var reviewUri: String = UriComponentsBuilder.fromUriString("/v1/reviews")
31            .queryParam(name: "movieInfoId", movieInfoId)
32            .buildAndExpand()
33            .toString();
34
35
36        return webClient
37            .get() WebClient.RequestHeadersUriSpec<...>
38            .uri(reviewUri) capture of ?
39            .retrieve() WebClient.ResponseSpec
40            .bodyToFlux(Review.class) Flux<Review>
41            .collectList() Mono<List<Review>>
42            .block();
43
```

```
parallel-asynchronous-using-java src main java com learnjava apiclient MoviesClient retrieveMovie  
MoviesClient.java Movie.java build.gradle (parallel-asynchronous)  
19 public Movie retrieveMovie(Long movieInfoId){  
20  
21     //movieInfo  
22     var movieInfo : MovieInfo = invokeMovieInfoService(movieInfoId);  
23  
24     //review  
25     var reviews : List<Review> = invokeReviewsService(movieInfoId);  
26     return new Movie(movieInfo, reviews);  
27 }
```

Same thing using CompletableFeature

```
27
28     public CompletableFuture<Movie> retrieveMovie_CF(Long movieInfoId){
29         //movieInfo
30         var movieInfo :CompletableFuture<MovieInfo> = CompletableFuture.supplyAsync(()->invokeMovieInfoService(movieInfoId));
31         //review
32         var reviews :CompletableFuture<List<Review>> =CompletableFuture.supplyAsync(()-> invokeReviewsService(movieInfoId));
33         return movieInfo
34             .thenCombine(reviews, Movie::new);
35     }
36 }
```

Difference between normal and Completable Feature

The screenshot shows the IntelliJ IDEA interface with two panes. The left pane displays a Java test class named `MoviesClientTest` containing a single test method `retrieveMovie_CF()`. The right pane shows the execution results of this test, indicating it passed with a duration of 889ms. The output window shows the printed movie details and the total time taken.

```
private final WebClient webClient;

public MoviesClient(WebClient webClient) { this.webClient = webClient; }

public Movie retrieveMovie(Long movieInfoId){
    //movieInfo
    var movieInfo : MovieInfo = invokeMovieInfoService(movieInfoId);
    //review
    var reviews : List<Reviews> = invokeReviewsService(movieInfoId);
    return new Movie(movieInfo, reviews);
}

public CompletableFuture<Movie> retrieveMovie_CF(Long movieInfoId){
    //movieInfo
    var movieInfo : CompletableFuture<MovieInfo> = CompletableFuture.supplyAsync(() -> invokeMovieInfoService(movieInfoId));
    //review
    var reviews : CompletableFuture<List<Reviews>> = CompletableFuture.supplyAsync(() -> invokeReviewsService(movieInfoId));
    return movieInfo.thenCompose(movieInfo1 -> reviews.thenApply(reviews1 -> new Movie(movieInfo1, reviews1)));
}
```

Run: MoviesClientTest

Tests Results

Tests passed: 2 of 2 tests – 889ms

"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/Home/bin/java" ...

movie : Movie(movieInfo=MovieInfo(movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Caine], release_date=2005-06-15), reviewList=[Review([main] - Total Time Taken : 824

movie : Movie(movieInfo=MovieInfo(movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Caine], release_date=2005-06-15), reviewList=[Review([main] - Total Time Taken : 25

Process finished with exit code 0

Repeated Test with blocking call

Repeated Test with unblocking blocking call - CompletableFuture

From the above statistics, CompletableFuture is better than blocking calls.

```
36 @RepeatedTest(10)
37 void retrieveMovies() {
38     CommonUtil.startTimer();
39     //given
40     var movieInfoIds : List<Long> = List.of(1L,2L, 3L,4L,5L,6L,7L);
41     //when
42     var movies : List<Movie> = moviesClient.retrieveMovies(movieInfoIds);
43
44     System.out.println("movie : " + movies);
45     CommonUtil.timeToken();
46     //then
47     assert movies!=null;
48     assert movies.size()==7;
49 }
50
51 @RepeatedTest(10)
52 void retrieveMovieList_CF() {
53     CommonUtil.startTimer();
54     //given
55     var movieInfoIds : List<Long> = List.of(1L,2L, 3L,4L,5L,6L,7L);
56     //when
57     var movies : List<Movie> = moviesClient.retrieveMovieList_CF(movieInfoIds);
58
59     System.out.println("movie : " + movies);
60     CommonUtil.timeToken();
61     //then
62     assert movies!=null;
63     assert movies.size()==7;
64 }
65
```

```
57 public MoviesClient(WebClient webClient) { this.webClient = webClient; }
58
59 public Movie retrieveMovie(Long movieInfoId){
60     //movieInfo
61     var movieInfo : MovieInfo = invokeMovieInfoService(movieInfoId);
62     //review
63     var review : List<Review> = invokeReviewsService(movieInfoId);
64     return new Movie(movieInfo, reviews);
65 }
66
67 public List<Movie> retrieveMovies(List<Long> movieInfoIds){
68
69     return movieInfoIds
70         .stream()
71         .map(this::retrieveMovie)
72         .collect(Collectors.toList());
73 }
74
75 public List<Movie> retrieveMovieList_CF(List<Long> movieInfoIds){
76
77     var movieFutures : List<CompletableFuture<Movie>> = movieInfoIds
78         .stream()
79         .map(this::retrieveMovie_CF)
80         .collect(Collectors.toList());
81
82     return movieFutures
83         .stream()
84         .map(CompletableFuture::join)
85         .collect(Collectors.toList());
86 }
87
88 public CompletableFuture<Movie> retrieveMovie_CF(Long movieInfoId){
89     var movieInfo : CompletableFuture<MovieInfo> = CompletableFuture.supplyAsync(()> invokeMovieInfo
90     //review
91     var reviews : CompletableFuture<List<Review>> = CompletableFuture.supplyAsync(()> invokeReviews
92     return movieInfo
93         .thenCombine(reviews, Movie::new);
94 }
```

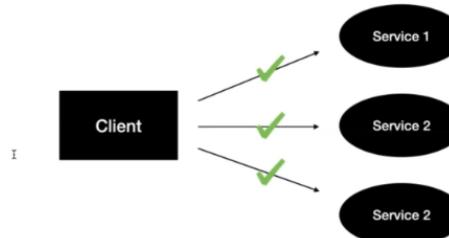
Retrieve movie List block vs unblock

From the above and below statistics, CompletableFuture is better than blocking calls.

```
Run: MoviesClientTest
  ▶ MoviesClientTest
    ▶ Test Results
      2 sec 215 ms
        Tests passed: 40 of 40 tests - 2 sec 215 ms
        movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
        [main] - Total Time Taken : 35 ms
      movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
      [main] - Total Time Taken : 26 ms
      movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
      [main] - Total Time Taken : 22 ms
      movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
      [main] - Total Time Taken : 23 ms
      movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
      [main] - Total Time Taken : 23 ms
      movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
      [main] - Total Time Taken : 34 ms
      movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
      [main] - Total Time Taken : 47 ms
      movie : [Movie{movieInfo=MovieInfo{movieInfoId=1, name=Batman Begins, year=2005, cast=[Christian Bale, Michael Cane], release_date=2005-06-15}, reviewList=[Review{reviewerName=John Smith, rating=5, reviewText=Great movie!}, Review{reviewerName=Jane Doe, rating=4, reviewText=Good movie!}, Review{reviewerName=Bob Johnson, rating=3, reviewText=OK movie!}, Review{reviewerName=Sarah Lee, rating=2, reviewText=Not bad movie!}, Review{reviewerName=David Wilson, rating=1, reviewText=Not good movie!}]}]
      [main] - Total Time Taken : 24 ms
    ▶ Test Suite Summary
      2 sec 215 ms
      Tests passed: 40 (4 minute ago)
```

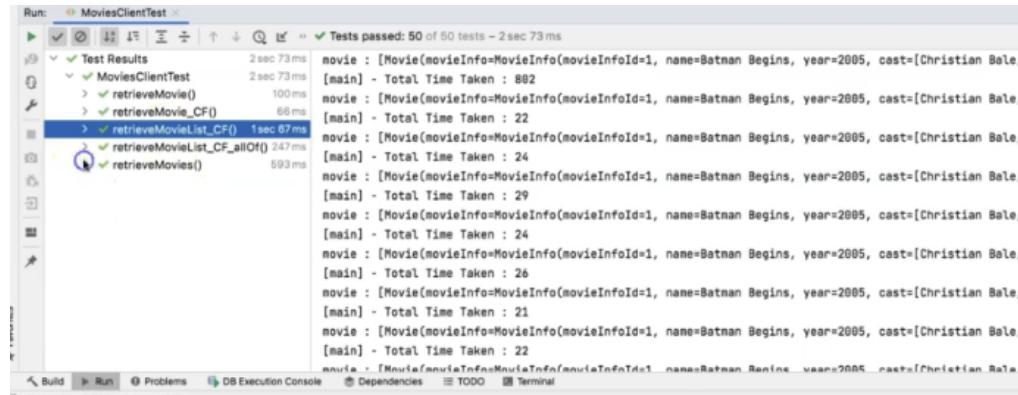
allOf() - Dealing with Multiple CompletableFutures

- static method that's part of CompletableFuture API
 - Use allOf() when you are dealing with Multiple CompletableFuture

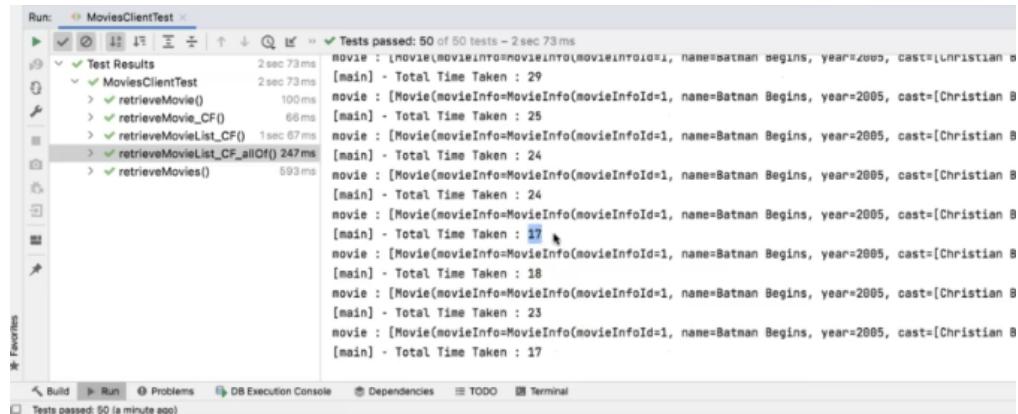


```
49
50     @
51     public List<Movie> retrieveMovieList_CF_allOf(List<Long> movieInfoIds){
52
53         var movieFutures :List<CompletableFuture<Movie>> = movieInfoIds
54             .stream()
55             .map(this::retrieveMovie_CF)
56             .collect(Collectors.toList());
57
58         var cfAllOf :CompletableFuture<Void> = CompletableFuture.allOf(movieFutures.toArray(new CompletableFuture[movieFutures.size()]));
59
60         return cfAllOf.
61             thenApply(v -> movieFutures
62                     .stream()
63                     .map(CompletableFuture::join)
64                     .collect(Collectors.toList()))
65             .join();
66     }
67 }
```

```
65
66     @RepeatedTest(10)
67     void retrieveMovieList_CF_allOff() {
68         CommonUtil.startTimer();
69         //given
70         var movieInfoIds :List<Long> = List.of(1L,2L, 3L,4L,5L,6L,7L);
71         //when
72         var movies :List<Movie> = moviesClient.retrieveMovieList_CF_allOff(movieInfoIds);
73
74         System.out.println("movie : " + movies);
75         CommonUtil.timeTaken();
76         //then
77         assert movies!=null;
78         assert movies.size()==7;
79     }
80
81 }
```

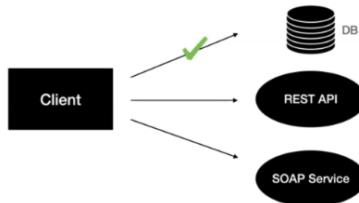


Still we are seeing some reduction in time when we use all of



anyOf() - Dealing with Multiple CompletableFuture

- static method that's part of CompletableFuture API
- Use anyOf() when you are dealing with retrieving data from multiple Data Sources



All sources give same value - anyOf will pick whichever give fast value.

```
197
198 public String anyOf() {
199
200     //db
201     CompletableFuture<String> db = CompletableFuture.supplyAsync(() -> {
202         delay( delayMilliseconds: 1000 );
203         log("response from db");
204         return "hello world";
205     });
206
207     //restCall
208     CompletableFuture<String> restCall = CompletableFuture.supplyAsync(() -> {
209         delay( delayMilliseconds: 2000 );
210         log("response from restCall");
211         return "hello world";
212     });
213
214     //soapCall
215     CompletableFuture<String> soapCall = CompletableFuture.supplyAsync(() -> {
216         delay( delayMilliseconds: 3000 );
217         log("response from soapCall");
218         return "hello world";
219     });
220
221     List<CompletableFuture<String>> cfList = List.of(db, restCall, soapCall);
222
223
224     List<CompletableFuture<String>> cfList = List.of(db, restCall, soapCall);
225     CompletableFuture<Object> cfAnyOf = CompletableFuture.anyOf(cfList.toArray(new CompletableFuture[cfList.size()]));
226
227     String result = (String) cfAnyOf.thenApply(v -> {
228
229         if (v instanceof String) {
230             return v;
231         }
232         return null;
233     })
234         .join();    I
235
236     return result;
237 }
```

The screenshot shows an IDE interface with two code editors and a run/test results panel.

Code Editors:

- CompletableFutureHelloWorldTest.java:**

```
181 // Given
182
183 //when
184 String helloWorld = cfhw.anyOf();
185
186 //then
187 assertEquals(expected: "hello world", helloWorld);
188 }
189
190 @Test
191 void helloWorld_thenCompose() {
192     //given
193     startTimer();
194     //when
195     CompletableFuture<String> completableFuture = cfhw.helloWorld_th
196     //then
197 }
```
- CompletableFutureHelloWorld.java:**

```
198     .thenApply(String::toUpperCase)
199     .join();
200 }
201
202 public String anyOf() {
203     //db
204     CompletableFuture<String> db = CompletableFuture.supplyAsync(() -> {
205         delay( delayMilliseconds: 1000 );
206         log("response from db");
207         return "hello world";
208     });
209
210     //restCall
211     CompletableFuture<String> restCall = CompletableFuture.supplyAsync(() -> {
212         delay( delayMilliseconds: 2000 );
213         log("response from restcall");
214         return "hello world";
215     });
216 }
```

Run/Test Results:

- Test results:** 1 of 1 test - 1s 35ms
- CompletableFutureHelloWorldTest:**
 - anyOf()
- Output:** [ForkJoinPool.commonPool-worker-5] - response from db
- Summary:** Process finished with exit code 0