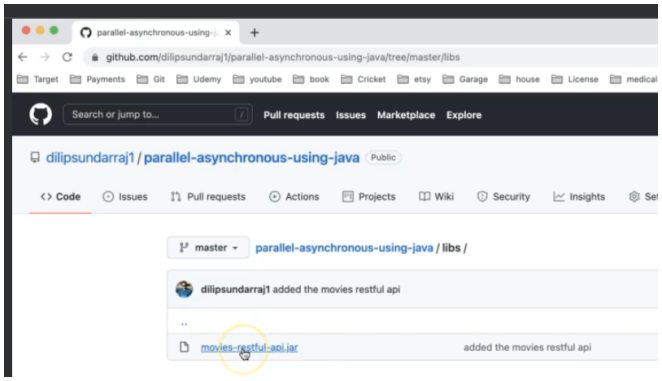
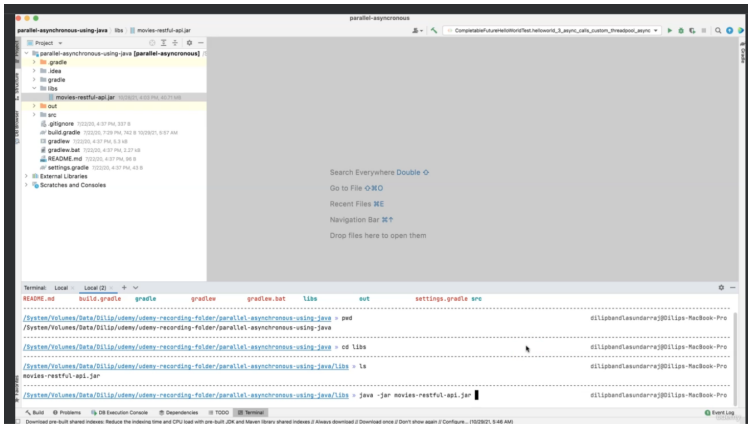
Build Restful API client using Spring WebClient and Completable Feature

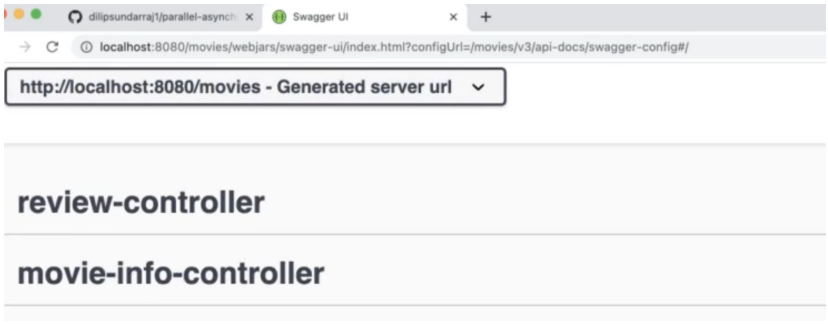
Why Spring WebClient?

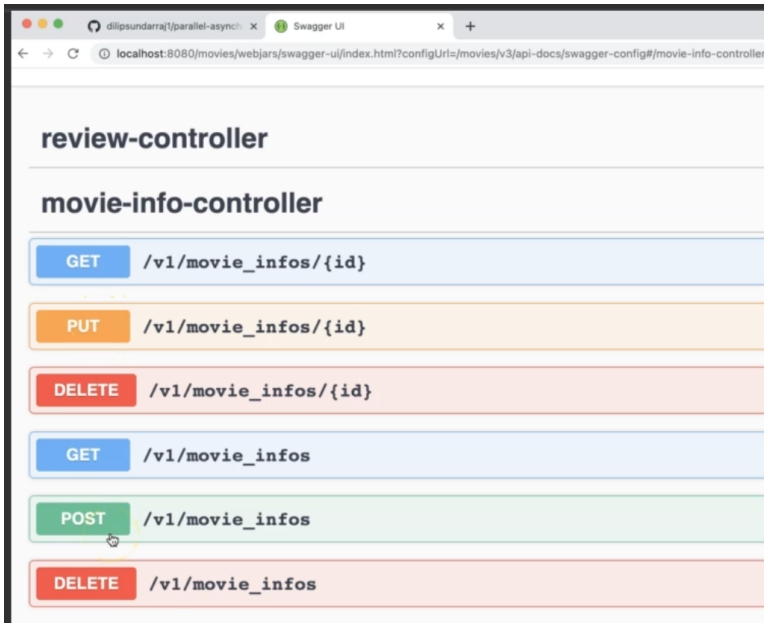
- Spring is one of the popular framework in the Java Community
- **Spring WebClient** is a rest client that got released as part of **Spring 5**
- **Spring WebClient** is a functional style RestClient
- **Spring WebClient** can be used as a blocking or non blocking Rest Client





```
2DBC repository interfaces.
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 r
unning 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
```





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 Cane"
    ],
    "release_date": "2005-06-15"
  },
  {
    "movieInfoId": 2,
    "name": "The Dark Knight",
    "year": 2008,
    "cast": [
      "Christian Bale",
      "HeathLedger"
    ],
    "release_date": "2008-07-18"
  },
  {
    "movieInfoId": 3,
    "name": "Dark Knight Rises",
    "year": 2008,
```


Review Endpoint

Request URL

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

Server response

Code | Details

200

Response body

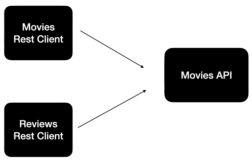

```
[
  {
    "reviewId": 1,
    "movieInfoId": 1,
    "rating": 8.2,
    "comment": "Nolan is the real superhero"
  }
]
```


Response headers

```
content-type: application/json
transfer-encoding: chunked
```

Responses

Rest Clients



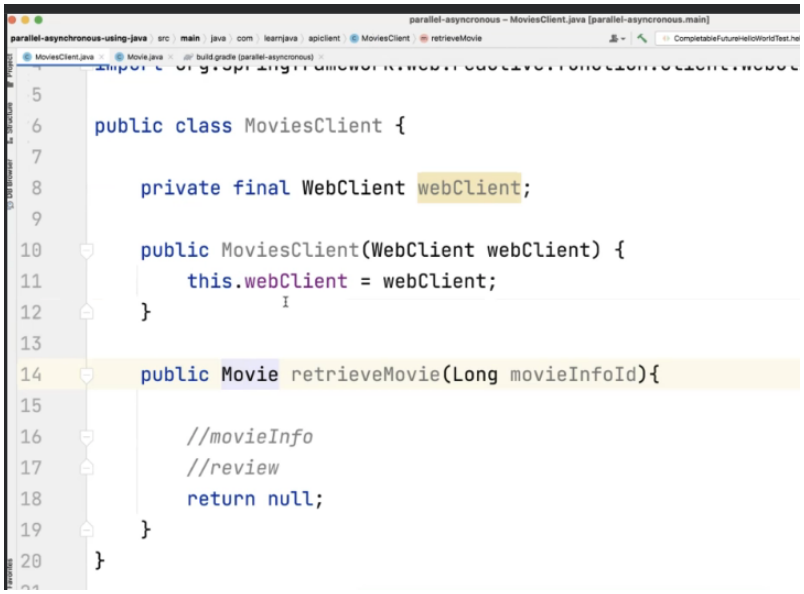
```java
import ...

@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.

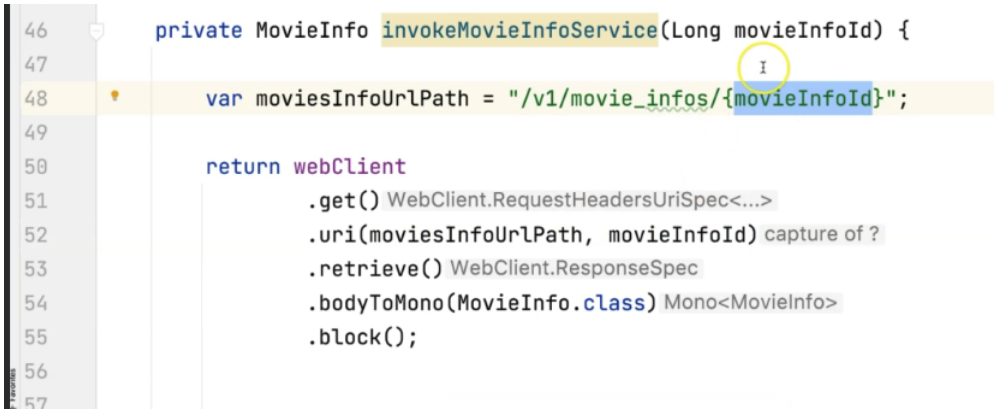
Copy below dependencies

raw.githubusercontent.com/dilipsundarraj1/parallel-asynchronous-using-java/final/build.gradle

```
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'
}
```



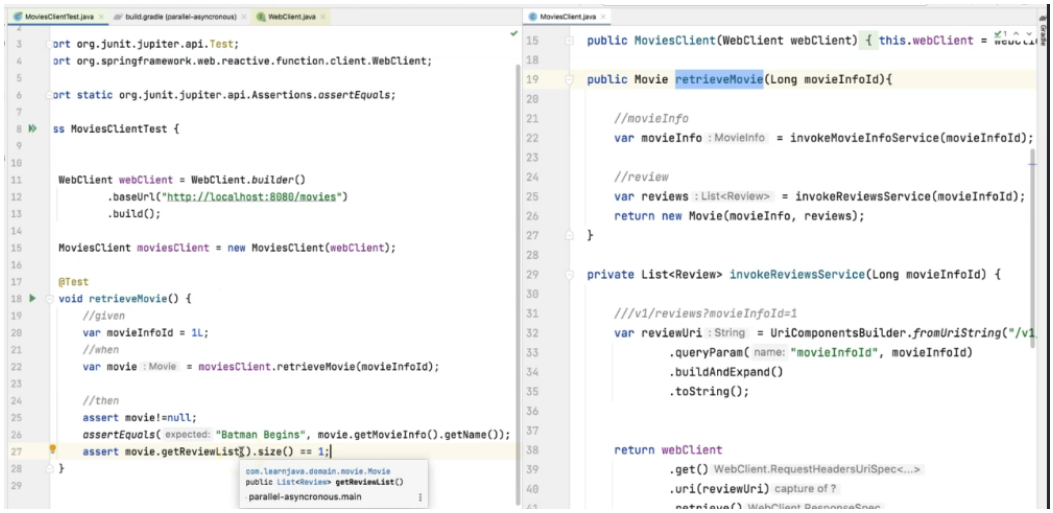
```
public class MoviesClient {

    private final WebClient webClient;

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

    public Movie retrieveMovie(Long movieInfoId){

        //movieInfo
        //review
        return null;
    }
}
```



```
    private MovieInfo invokeMovieInfoService(Long movieInfoId) {

        var moviesInfoUrlPath = "/v1/movie_infos/{movieInfoId}";

        return webClient
                .get() WebClient.RequestHeadersUriSpec<...>
                .uri(moviesInfoUrlPath, movieInfoId) capture of ?
                .retrieve() WebClient.ResponseSpec
                .bodyToMono(MovieInfo.class) Mono<MovieInfo>
                .block();
```

```
private List<Review> invokeReviewsService(Long movieInfoId) {

    ///v1/reviews?movieInfoId=1
    var reviewUri : String = UriComponentsBuilder.fromUriString("/v1/reviews")
            .queryParam( name: "movieInfoId", movieInfoId)
            .buildAndExpand()
            .toString();


    return webClient
            .get() WebClient.RequestHeadersUriSpec<...>
            .uri(reviewUri) capture of ?
            .retrieve() WebClient.ResponseSpec
            .bodyToFlux(Review.class) Flux<Review>
            .collectList() Mono<List<Review>>
            .block();
```

```
public Movie retrieveMovie(Long movieInfoId){

    //movieInfo
    var movieInfo : MovieInfo = invokeMovieInfoService(movieInfoId);

    //review
    var reviews : List<Review> = invokeReviewsService(movieInfoId);
    return new Movie(movieInfo, reviews);
}
```



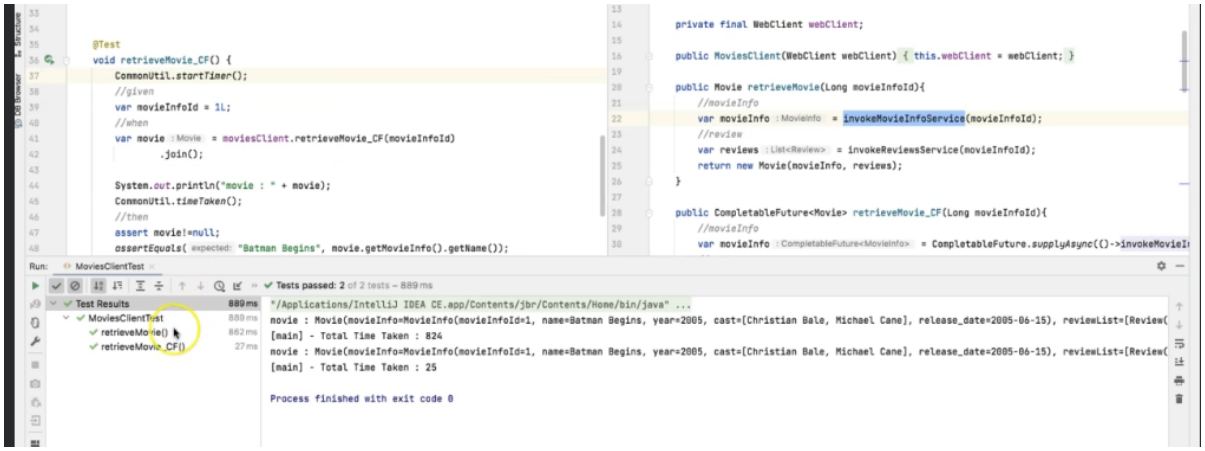
```
// MoviesClientTest.java
import org.junit.jupiter.api.Test;
import org.springframework.web.reactive.function.client.WebClient;

import static org.junit.jupiter.api.Assertions.assertEquals;

class MoviesClientTest {

    WebClient webClient = WebClient.builder()
            .baseUrl("http://localhost:8080/movies")
            .build();

    MoviesClient moviesClient = new MoviesClient(webClient);

    @Test
    void retrieveMovie() {
        //given
        var movieInfoId = 1L;
        //when
        var movie : Movie = moviesClient.retrieveMovie(movieInfoId);

        //then
        assert movie!=null;
        assertEquals( expected: "Batman Begins", movie.getMovieInfo().getName());
        assert movie.getReviewList().size() == 1;
    }
```

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

public Movie retrieveMovie(Long movieInfoId){

    //movieInfo
    var movieInfo : MovieInfo = invokeMovieInfoService(movieInfoId);

    //review
    var reviews : List<Review> = invokeReviewsService(movieInfoId);
    return new Movie(movieInfo, reviews);
}

private List<Review> invokeReviewsService(Long movieInfoId) {

    ///v1/reviews?movieInfoId=1
    var reviewUri : String = UriComponentsBuilder.fromUriString("/v1
            .queryParam( name: "movieInfoId", movieInfoId)
            .buildAndExpand()
            .toString();


    return webClient
            .get() WebClient.RequestHeadersUriSpec<...>
            .uri(reviewUri) capture of ?
            .retrieve() WebClient.ResponseSpec
```

Same thing using CompletableFeature

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

Difference between normal and Completable Feature

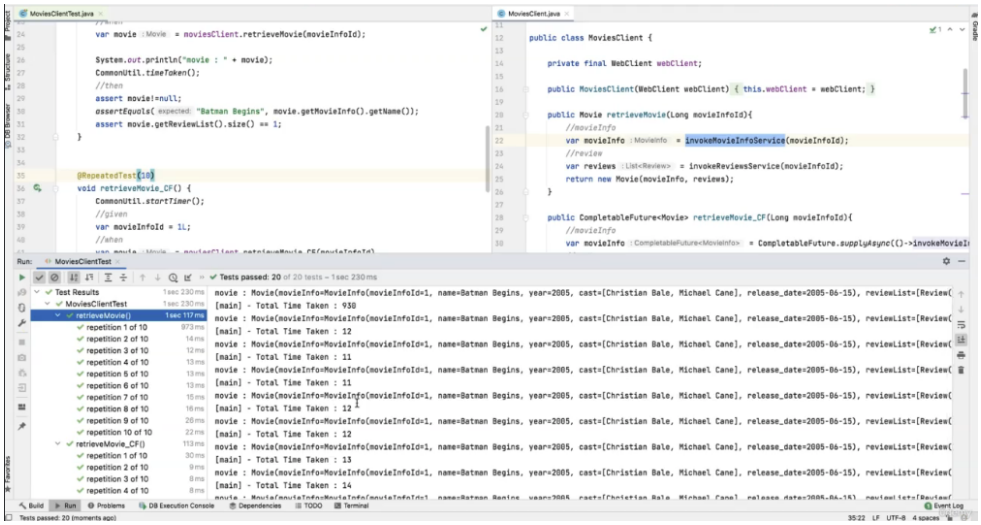


```
@Test
void retrieveMovie_CF() {
    CommonUtil.startTimer();
    //given
    var movieInfoId = 1L;
    //when
    var movie : Movie = moviesClient.retrieveMovie_CF(movieInfoId)
            .join();

    System.out.println("movie : " + movie);
    CommonUtil.timeTaken();
    //then
    assert movie!=null;
    assertEquals( expected: "Batman Begins", movie.getMovieInfo().getName());
```

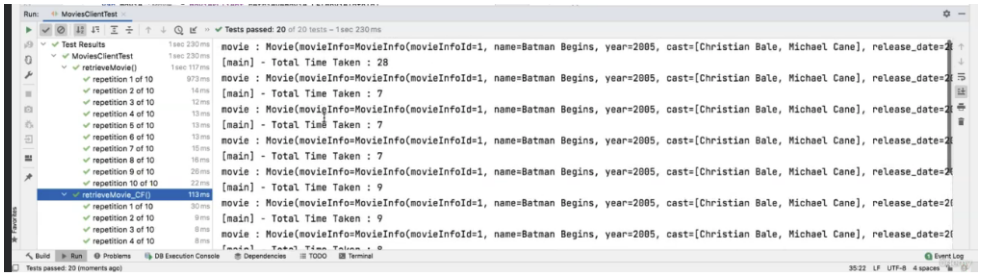
```
private final WebClient webClient;

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

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

public CompletableFuture<Movie> retrieveMovie_CF(Long movieInfoId){
    //movieInfo
    var movieInfo : CompletableFuture<MovieInfo> = CompletableFuture.supplyAsync(()->invokeMovieI
```

```
Run: MoviesClientTest
Tests passed: 2 of 2 tests – 889 ms
Test Results 889 ms
MoviesClientTest 889 ms
retrieveMovie() 862 ms
retrieveMovie_CF() 27 ms
"/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 Cane], 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 Cane], release_date=2005-06-15), reviewList=[Review(
[main] - Total Time Taken : 25

Process finished with exit code 0
```

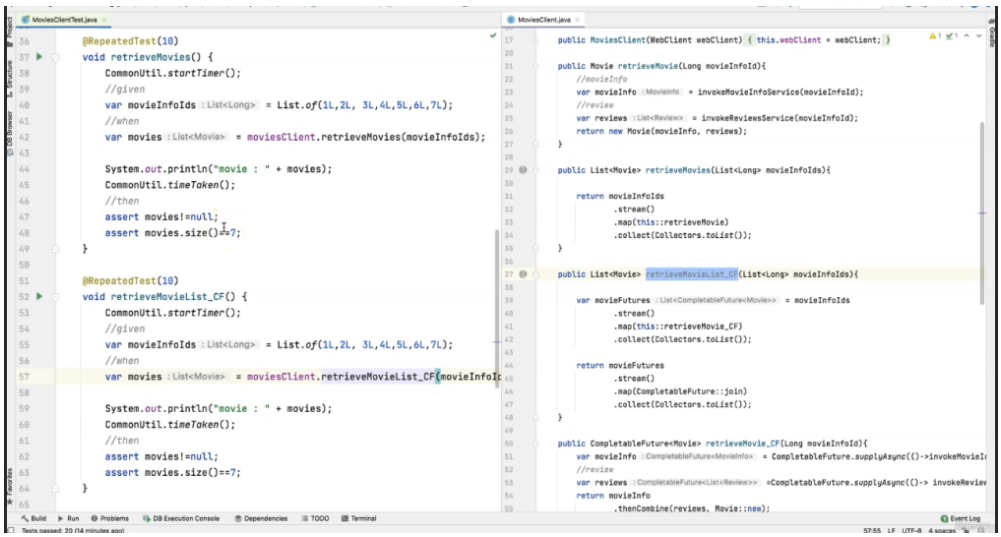
Repeated Test with blocking call



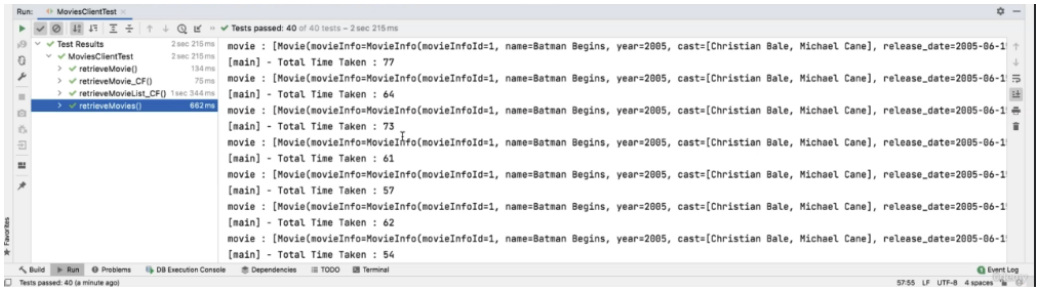
Repeated Test with unblocking blocking call - CompletableFuture



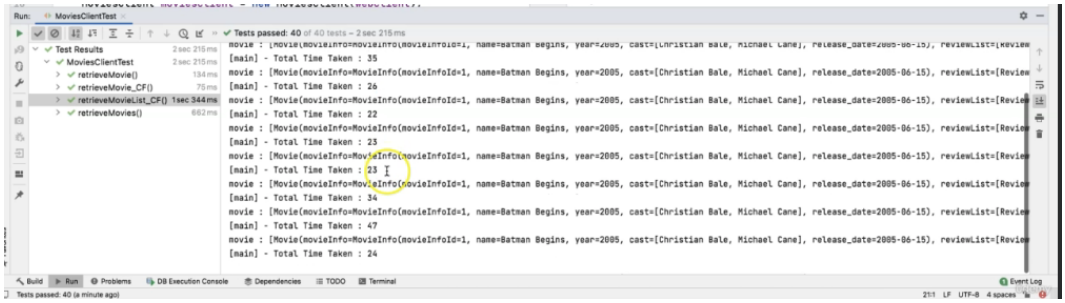
From the above statistics, CompletableFuture is better than blocking calls.



Retrieve movie List block vs unblock

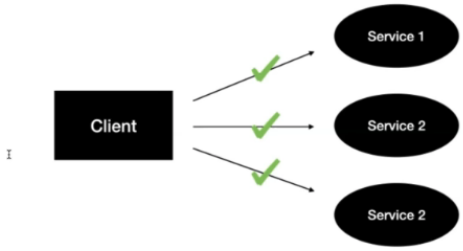


From the above and below statistics, CompletableFuture is better than blocking calls.

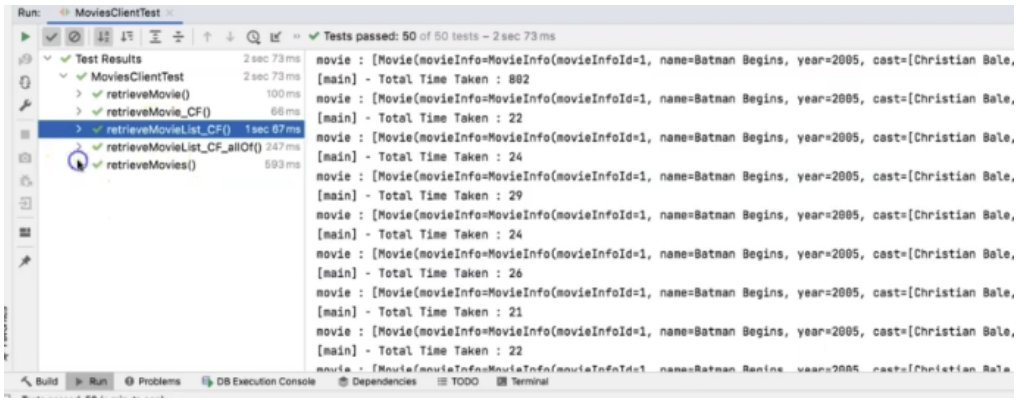


allOf() - Dealing with Multiple CompletableFutures

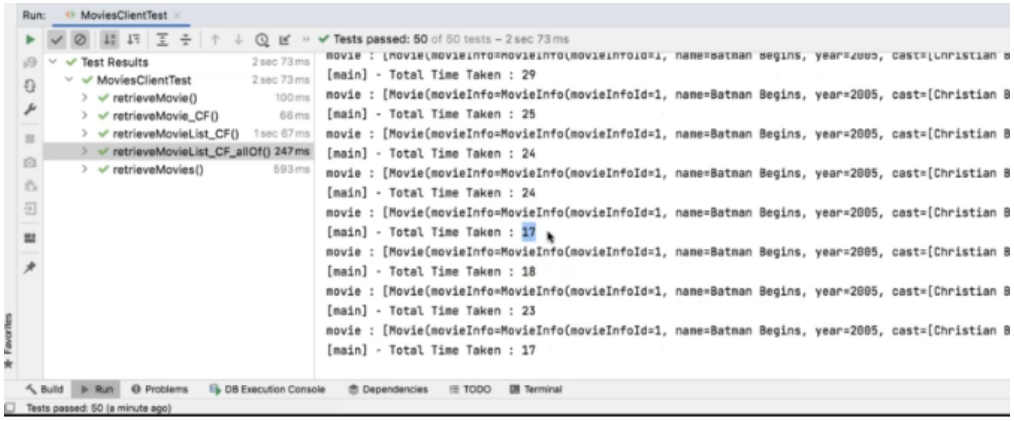
- static method that's part of CompletableFuture API
- Use allOf() when you are dealing with Multiple CompletableFuture




```
public List<Movie> retrieveMovieList_CF_allOf(List<Long> movieInfoIds){

    var movieFutures : List<CompletableFuture<Movie>> = movieInfoIds
            .stream()
            .map(this::retrieveMovie_CF)
            .collect(Collectors.toList());

    var cfAllOf : CompletableFuture<Void> = CompletableFuture.allOf(movieFutures.toArray(new CompletableFuture[movieFutures.size()]));

    return cfAllOf.
            thenApply(v -> movieFutures
                    .stream()
                    .map(CompletableFuture::join)
                    .collect(Collectors.toList()))
            .join();

}
```

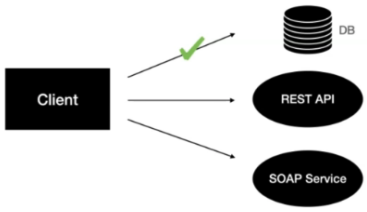


Still we are seeing some reduction in time when we use all of



anyOf() - Dealing with Multiple CompletableFutures

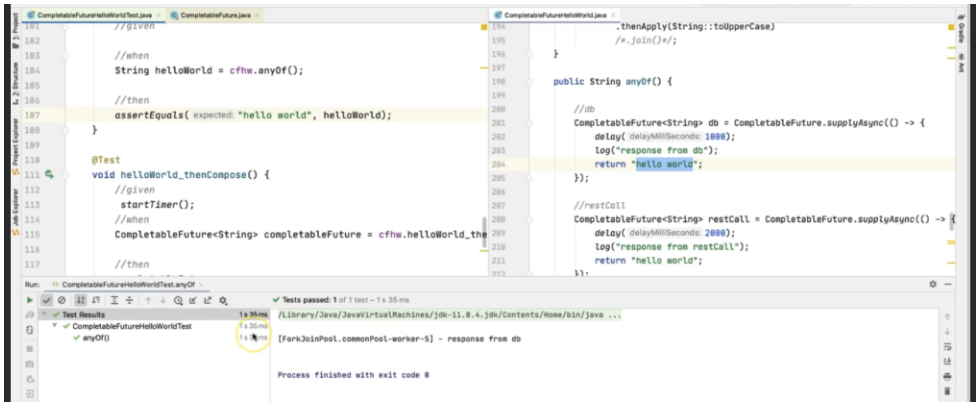
- static method that's part of CompletableFuture API
- Use anyOf() when you are dealing with retrieving data from multiple Data Sources



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);
```

```
221         List<CompletableFuture<String>> cfList = List.of(db, restCall, soapCall);
222
223         CompletableFuture<Object> cfAnyOf = CompletableFuture.anyOf(cfList.toArray(new CompletableFuture[cfList.size()]));
224
225         String result = (String) cfAnyOf.thenApply(v -> {
226
227             if (v instanceof String) {
228                 return v;
229             }
230             return null;
231
232         })
233                 .join();
234
235         return result;
236
237     }
238
```


CompletableFutureHelloWorldTest.java
CompletableFuture.java
CompletableFutureHelloWorld.java
//given
//when
String helloWorld = cfhw.anyOf();
//then
assertEquals(expected: "hello world", helloWorld);
@Test
void helloWorld_thenCompose() {
//given
startTimer();
//when
CompletableFuture<String> completableFuture = cfhw.helloWorld_the
//then
.thenApply(String::toUpperCase)
/*.join()*/;
public String anyOf() {
//db
CompletableFuture<String> db = CompletableFuture.supplyAsync(() -> {
delay(delayMilliSeconds: 1000);
log("response from db");
return "hello world";
});
//restCall
CompletableFuture<String> restCall = CompletableFuture.supplyAsync(() -> {
delay(delayMilliSeconds: 2000);
log("response from restCall");
return "hello world";
Run: CompletableFutureHelloWorldTest.anyOf
Tests passed: 1 of 1 test – 1 s 35 ms
Test Results 1 s 35 ms
CompletableFutureHelloWorldTest 1 s 35 ms
anyOf() 1 s 35 ms
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...
[ForkJoinPool.commonPool-worker-5] - response from db
Process finished with exit code 0