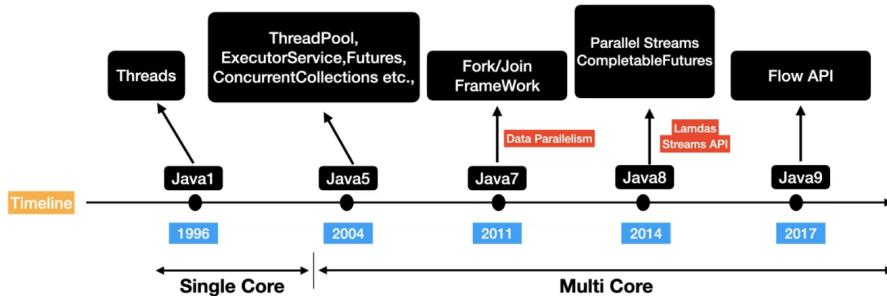


Technology Advancements

Hardware	Software
<ul style="list-style-type: none">Devices or computers comes up with Multiple coresDeveloper needs to learn programming patterns to maximize the use of multiple coresApply the Parallel Programming conceptsParallel Streams	<ul style="list-style-type: none">MicroServices Architecture styleBlocking I/O calls are common in MicroServices Architecture. This also impacts the latency of the applicationApply the Asynchronous Programming conceptsCompletableFuture

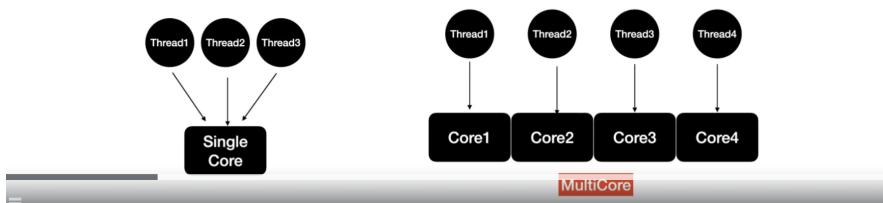
Threads **Functional Style of Programming**

Evolution of Concurrency/Parallelism APIs



Concurrency

- Concurrency is a concept where two or more tasks can run simultaneously
- In Java, Concurrency is achieved using **Threads**
 - Are the tasks running in interleaved fashion?
 - Are the tasks running simultaneously ?



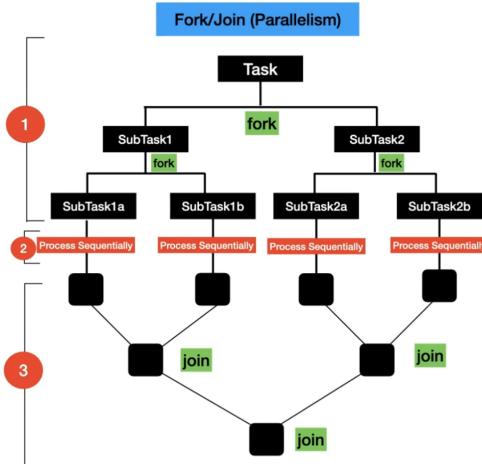
Concurrency Example

- In a real application, Threads normally need to interact with one another
 - Shared Objects or Messaging Queues
- Issues:
 - Race Condition
 - DeadLock and more
- Tools to handle these issues:
 - Synchronized Statements/Methods
 - Reentrant Locks, Semaphores
 - Concurrent Collections
 - Conditional Objects and More

```
public class HelloWorldThreadExample {  
    private static String result="";  
  
    private static void hello(){  
        delay(500);  
        result = result.concat("Hello");  
    }  
  
    private static void world(){  
        delay(600);  
        result = result.concat(" World");  
    }  
  
    public static void main(String[] args) throws InterruptedException {  
        Thread helloThread = new Thread(()-> hello());  
        Thread worldThread = new Thread(()-> world()); } Threads  
        //Starting the thread  
        helloThread.start();  
        worldThread.start();  
  
        //Joining the thread (Waiting for the threads to finish)  
        helloThread.join();  
        worldThread.join();  
  
        System.out.println("Result is : " + result);  
    }  
}  
  
Hello World
```

Parallelism

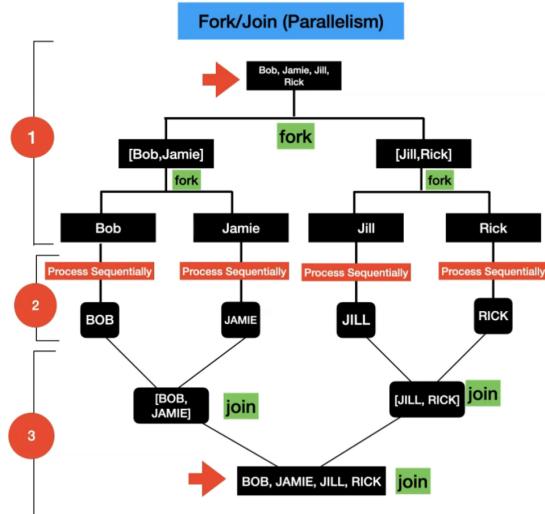
- Parallelism is a concept in which tasks are literally going to run in parallel
- Parallelism involves these steps:
 - Decomposing the tasks in to SubTasks(Forking)
 - Execute the subtasks in sequential
 - Joining the results of the tasks(Join)
- Whole process is also called **Fork/Join**



Parallelism Example

UseCase: Transform to UpperCase

[Bob, Jamie, Jill, Rick] -> [BOB, JAMIE, JILL, RICK]

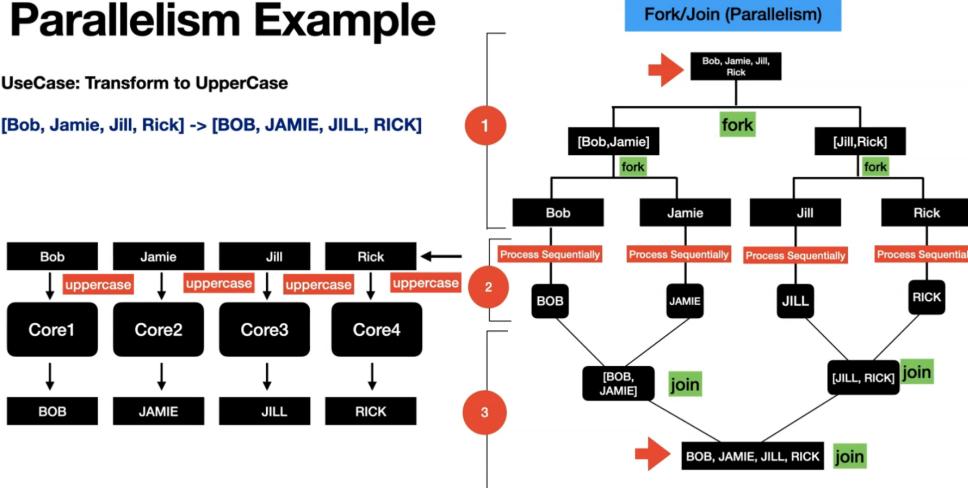


What happening in point 2

Parallelism Example

UseCase: Transform to UpperCase

[Bob, Jamie, Jill, Rick] -> [BOB, JAMIE, JILL, RICK]



Parallelism Example

```

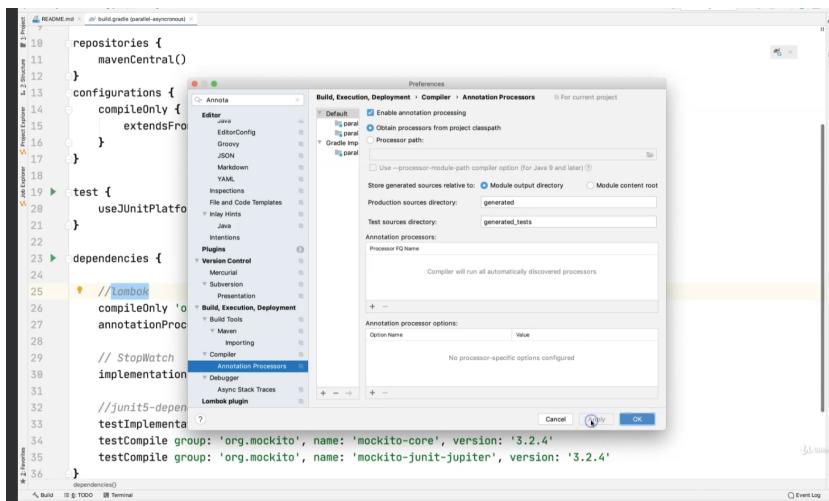
public class ParallelismExample {

    public static void main(String[] args) {
        List<String> namesList = List.of("Bob", "Jamie", "Jill", "Rick");
        System.out.println("namesList : " + namesList);
        List<String> namesListUpperCase = namesList
            .parallelStream() <-- .parallelStream()
            .map(String::toUpperCase) <-- .map(String::toUpperCase)
            .collect(Collectors.toList());
        System.out.println("namesListUpperCase : " + namesListUpperCase);
    }
}
  
```

A Java code snippet demonstrating parallel stream processing. It starts with a list of names: Bob, Jamie, Jill, Rick. This list is printed. Then, it is processed using a parallel stream. Inside the stream, the map operation converts each name to its uppercase equivalent using the String::toUpperCase method. Finally, the collect operation gathers the results into a new list, namesListUpperCase, which is then printed.

Concurrency vs Parallelism

- Concurrency is a concept where two or more tasks can run in simultaneously
- Concurrency can be implemented in single or multiple cores
- Concurrency is about correctly and efficiently controlling access to shared resources
- Parallelism is a concept where two or more tasks are literally running in parallel
- Parallelism can only be implemented in a multi-core machine
- Parallelism is about using more resources to access the result faster

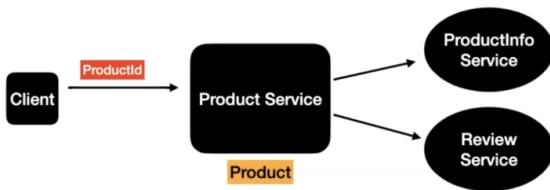


Enable lombok

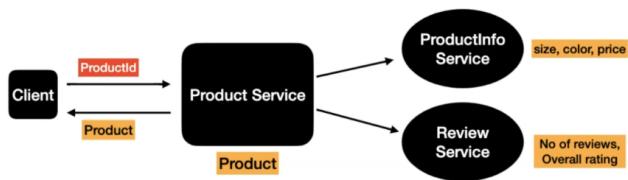
Section Overview

- Covers Asynchronous and Parallel Programming prior Java 8
- Threads, Futures and ForkJoin Framework and its limitations
- Covers Theory and Hands On

Product Service



Product Service

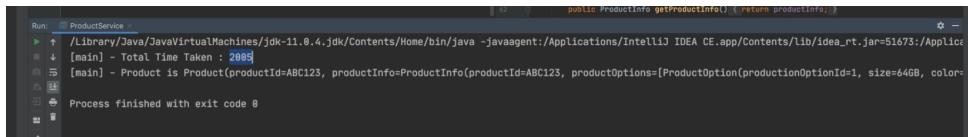


Product Service



```
15     }
16 
17     public Product retrieveProductDetails(String productId) {
18         stopWatch.start();
19 
20         ProductInfo productInfo = productInfoService.retrieveProductInfo(productId); // blocking call
21         Review review = reviewService.retrieveReviews(productId); // blocking call
22 
23         stopWatch.stop();
24         log("Total Time Taken : " + stopWatch.getTime());
25         return new Product(productId, productInfo, review);
26     }
27 
28 
29     public static void main(String[] args) {
30 
31         ProductInfoService productInfoService = new ProductInfoService();
32         ReviewService reviewService = new ReviewService();
33         ProductService productService = new ProductService(productInfoService, reviewService);
34         String productId = "ABC123";
35         Product product = productService.retrieveProductDetails(productId);
36         log("Product is " + product);
37     }
38 }
39 }
```

Regular way of writing the code. Review service (line number 22) won't invoke unless Product service completes (line number 21). The above code is taking 2 seconds to complete



Let us solve this problem using Threads

Threads API

- Threads API got introduced in Java1
- Threads are basically used to offload the blocking tasks as **background** tasks
- Threads allowed the developers to write asynchronous style of code

```
public Product retrieveProductDetails(String productId) throws InterruptedException {
    stopWatch.start();
    ProductInfoRunnable productInfoRunnable = new ProductInfoRunnable(productId);
    Thread productInfoThread = new Thread(productInfoRunnable);

    ReviewRunnable reviewRunnable = new ReviewRunnable(productId);
    Thread reviewThread = new Thread(reviewRunnable);

    productInfoThread.start();
    reviewThread.start();

    productInfoThread.join();
    reviewThread.join();

    ProductInfo productInfo = productInfoRunnable.getProductInfo();
    Review review = reviewRunnable.getReview();

    stopWatch.stop();
    log("Total Time Taken : " + stopWatch.getTime());
    return new Product(productId, productInfo, review);
}

public static void main(String[] args) throws InterruptedException {
    ProductInfoService productInfoService = new ProductInfoService();
    ReviewService reviewService = new ReviewService();
    ProductServiceUsingThread productService = new ProductServiceUsingThread(productInfoService, reviewService);
    String productId = "ABC123";
    Product product = productService.retrieveProductDetails(productId);
    log("Product is " + product);
}
```

```
On
36     ProductInfo productInfo = productInfoRunnable.getProductInfo();
37     Review review = reviewRunnable.getReview();
38
39     stopWatch.stop();
40     log("Total Time Taken : " + stopWatch.getTime());
41     return new Product(productId, productInfo, review);
42
43 public static void main(String[] args) throws InterruptedException {
44
45     ProductInfoService productInfoService = new ProductInfoService();
46     ReviewService reviewService = new ReviewService();
47     ProductServiceUsingThread productService = new ProductServiceUsingThread(productInfoService, reviewService);
48     String productId = "ABC123";
49     Product product = productService.retrieveProductDetails(productId);
50     log("Product is " + product);
51
52 }
53
54     private class ProductInfoRunnable implements Runnable {
55         private ProductInfo pInfo;
56         private String productId;
57
58         public ProductInfoRunnable(String productId) { this.productId = productId; }
59
60         public ProductInfo getProductInfo() { return pInfo; }
61
62         @Override
63         public void run() {
64
65             pInfo = productInfoService.retrieveProductInfo(productId);
66
67         }
68     }
69 }
```

It took only 1 second. But for this we had to write more code. We have to introduce Threads/runnable interface for every asynchronous call. And the run method wont take any input and won't return any output . so we had to introduce additional properties to get the data.

ThreadPool, ExecutorService & Future

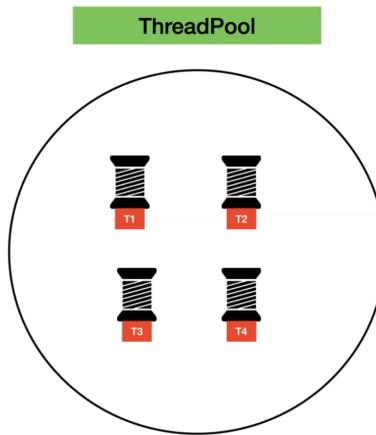
Limitations Of Thread

- Limitations of Thread:
 - Create the thread
 - Start the thread
 - Join the thread
- Threads are expensive
 - Threads have their own runtime-stack, memory, registers and more

Thread Pool was created specifically to solve this problem

Thread Pool

- Thread Pool is a group of threads created and readily available
- CPU Intensive Tasks
 - ThreadPool Size = No of Cores
- I/O task
 - ThreadPool Size > No of Cores
- What are the benefits of thread pool?
 - No need to manually create, start and join the threads
- Achieving Concurrency in your application

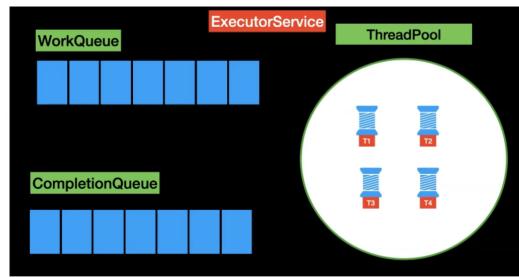


We no need to create threads

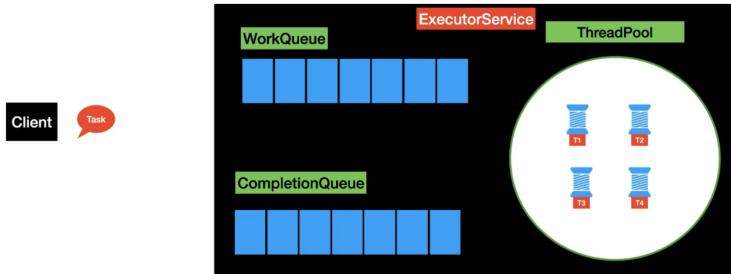
ExecutorService

- Released as part of Java5
- ExecutorService in Java is an **Asynchronous Task Execution Engine**
- It provides a way to asynchronously execute tasks and provides the results in a much simpler way compared to threads
- This enabled coarse-grained task based parallelism in Java

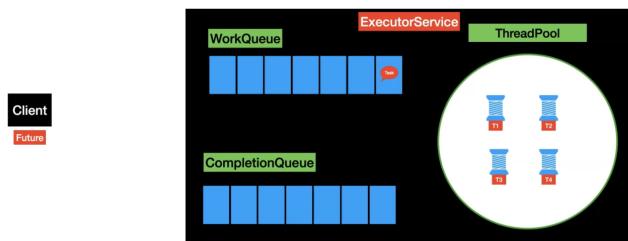
Working Of ExecutorService



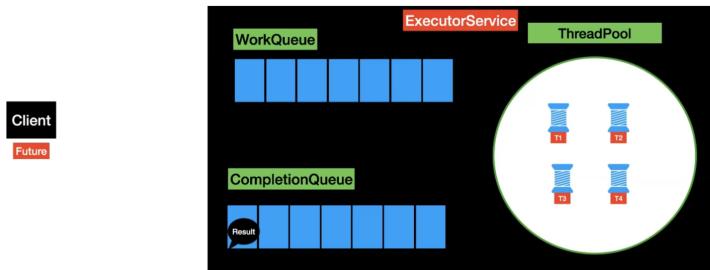
Working Of ExecutorService



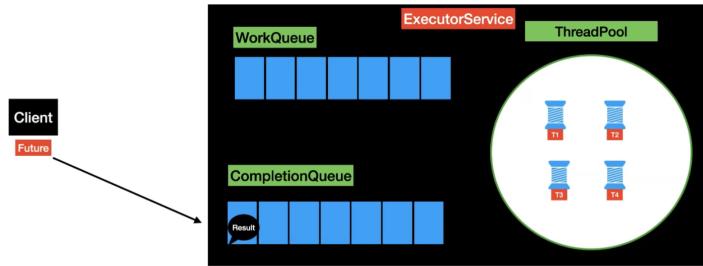
Working Of ExecutorService



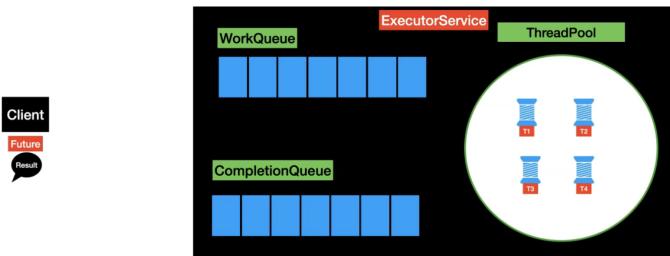
Working Of ExecutorService



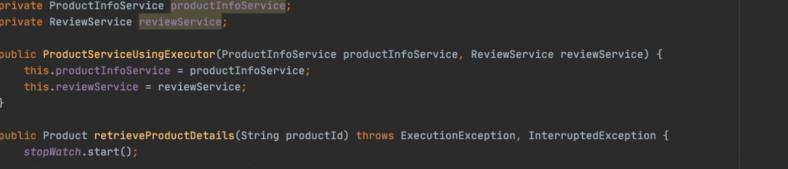
Working Of ExecutorService



Working Of ExecutorService



We have many different options to create executor service as shown below



```
parallel-asynchronous-using-java src main java com.learnjava.executor ProductServiceUsingExecutor RetrieveProductDetails

18
19     static ExecutorService executorService= Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
20     private ProductInfoService productInfoService;
21     private ReviewService reviewService;
22
23     public ProductServiceUsingExecutor(ProductInfoService productInfoService, ReviewService reviewService) {
24         this.productInfoService = productInfoService;
25         this.reviewService = reviewService;
26     }
27
28     public Product retrieveProductDetails(String productId) throws ExecutionException, InterruptedException {
29         stopWatch.start();
30
31         Future<ProductInfo> productInfoFuture = executorService.submit(()->productInfoService.retrieveProductInfo(productId));
32         Future<Review> reviewFuture = executorService.submit(()->reviewService.retrieveReviews(productId));
33
34         ProductInfo productInfo = productInfoFuture.get();
35         Review review = reviewFuture.get();
36
37         stopWatch.stop();
38         log("Total Time Taken : "+ stopWatch.getTime());
39         return new Product(productId, productInfo, review);
40     }
41 }
```

```
 27
 28     public Product retrieveProductDetails(String productId) throws ExecutionException, InterruptedException {
 29         stopWatch.start();
 30
 31         Future<ProductInfo> productInfoFuture = executorService.submit(() -> productService.retrieveProductInfo(productId));
 32         Future<Review> reviewFuture = executorService.submit(() -> reviewService.retrieveReviews(productId));
 33
 34         ProductInfo productInfo = productInfoFuture.get();
 35         Review review = reviewFuture.get();
 36
 37         stopWatch.stop();
 38         Log("Total Time Taken : " + stopWatch.getTime());
 39         return new Product(productId, productInfo, review);
 40     }
 41
 42     public static void main(String[] args) throws ExecutionException, InterruptedException {
 43
 44         ProductInfoService productInfoService = new ProductInfoService();
 45         ReviewService reviewService = new ReviewService();
 46         ProductServiceUsingExecutor productService = new ProductServiceUsingExecutor(productInfoService, reviewService);
 47         String productId = "ABC123";
 48         Product product = productService.retrieveProductDetails(productId);
 49         log("Product is " + product);
 50
 51     }
 52 }
```

```
    Product product = productService.retrieveProductDetails(productId);
}
Run: ProductServiceUsingExecutor
```

```

    //ProductInfo productInfo = productInfoFuture.get();
    ProductInfo productInfo = productInfoFuture.get( timeout: 2, TimeUnit.SECONDS);
    Review review = reviewFuture.get();

    stopWatch.stop();
    log("Total Time Taken : " + stopWatch.getTime());
    return new Product(productId, productInfo, review);
}

public static void main(String[] args) throws ExecutionException, InterruptedException {
    ProductInfoService productInfoService = new ProductInfoService();
    ReviewService reviewService = new ReviewService();
    ProductServiceUsingExecutor productService = new ProductServiceUsingExecutor(productInfoService, reviewService);
    String productId = "ABC123";
    Product product = productService.retrieveProductDetails(productId);
    log("Product is " + product);
    executorService.shutdown();
}
}

```

We have a timeout also line number 32 and we need to manually stop executor service (line number 48).

They are taking same time but number of lines reduced when we use threadpool

```

public Product retrieveProductDetails(String productId) throws ExecutionException, InterruptedException {
    this.productInfoService = productInfoService;
    this.reviewService = reviewService;

    Future<ProductInfo> productInfoFuture = executorService.submit(() -> productInfoService
        .retrieveProductDetails(productId));
    Future<Review> reviewFuture = executorService.submit(() -> reviewService
        .getReview(productId));

    ProductInfo productInfo = productInfoFuture.get( timeout: 2, TimeUnit.SECONDS);
    Review review = reviewFuture.get();

    stopWatch.stop();
    log("Total Time Taken : " + stopWatch.getTime());
    return new Product(productId, productInfo, review);
}

public static void main(String[] args) throws ExecutionException, InterruptedException {
    ProductInfoService productInfoService = new ProductInfoService();
    ReviewService reviewService = new ReviewService();
    ProductServiceUsingExecutor productService = new ProductServiceUsingExecutor();
    String productId = "ABC123";
    Product product = productService.retrieveProductDetails(productId);
    log("Product is " + product);
    executorService.shutdown();
}
}

```

Disadvantages

```

Future<Review> reviewFuture = executorService.submit(() -> reviewService
    .getReview(productId));

ProductInfo productInfo = productInfoFuture.get();
Review review = reviewFuture.get();

```

Get method block the rest of the code. Even if we have timeout - if times out which means we do not have results and we are not able to combine two futures.

Fork/Join Framework

Fork/Join Framework

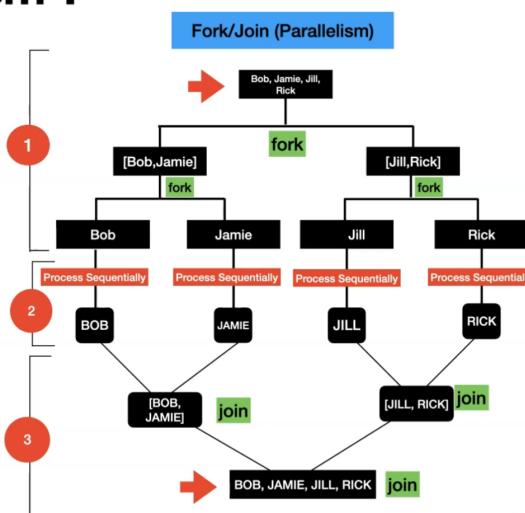
- This got introduced as part of Java7
- This is an extension of **ExecutorService**
- Fork/Join framework is designed to achieve **Data Parallelism**
- ExecutorService is designed to achieve **Task Based Parallelism**

```
Future<ProductInfo> productInfoFuture = executorService.submit(() -> productInfoService.retrieveProductInfo(productId)); ←  
Future<Review> reviewFuture = executorService.submit(() -> reviewService.retrieveReviews(productId)); ←
```

What is Data Parallelism ?

- Data Parallelism is a concept where a given **Task** is recursively split in to **SubTasks** until it reaches its least possible size and execute those tasks in parallel
- Basically it uses the **divide and conquer** approach

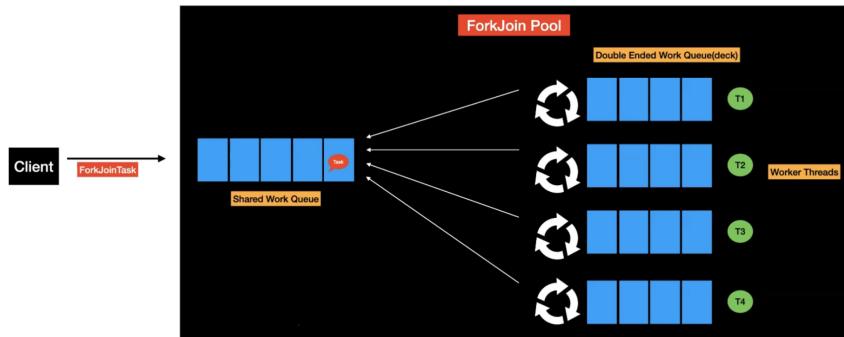
Watch "Concurrency vs Parallelism"



How does Fork/Join Framework Works ?

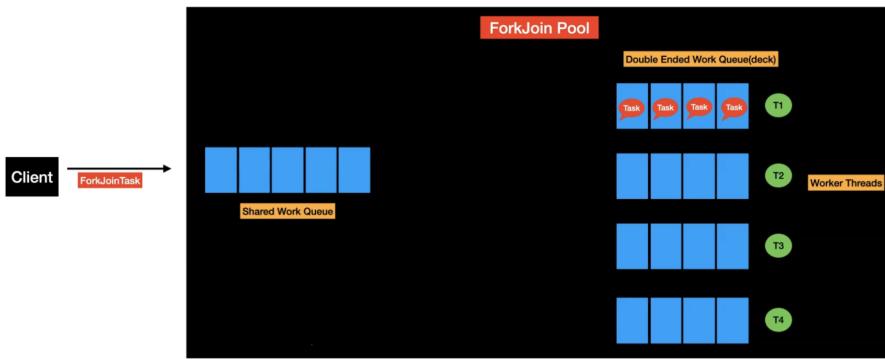
ForkJoin Pool to support Data Parallelism

ForkJoin Pool



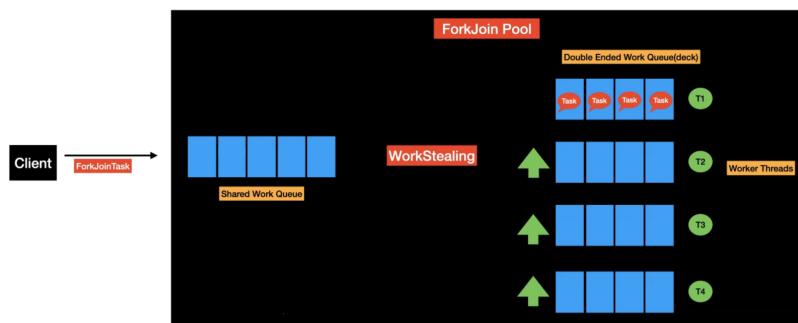
Polling happen and will take any of the thread

ForkJoin Pool

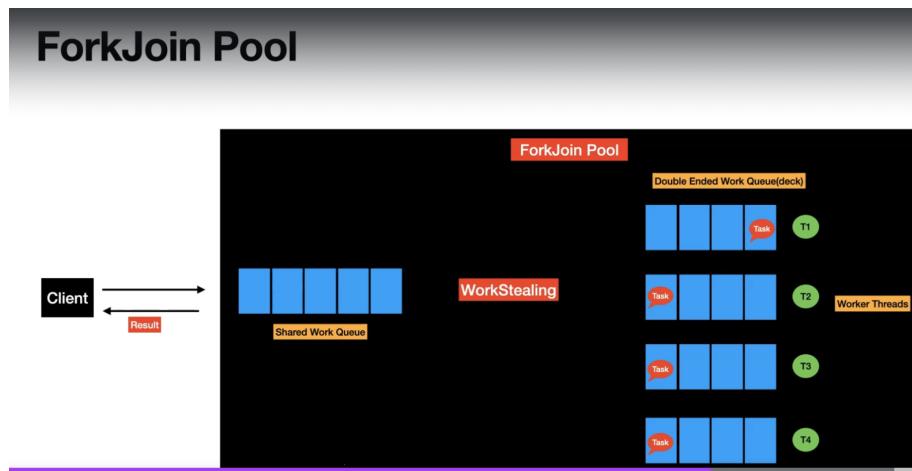
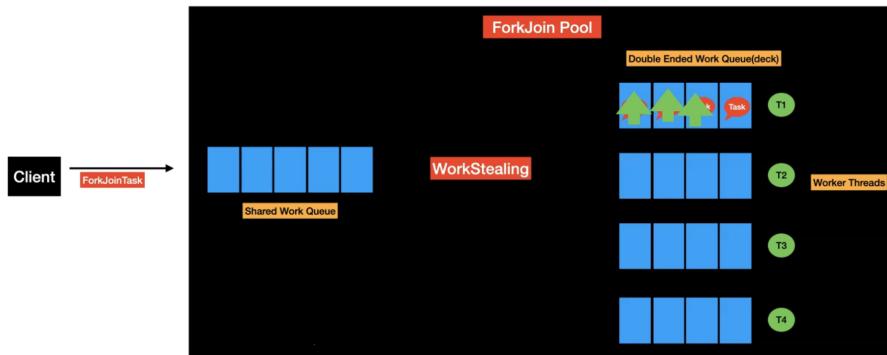


And it will check whether task can be further divisible and divide

ForkJoin Pool

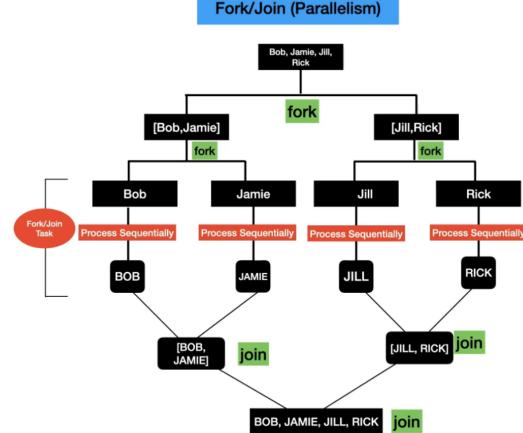


ForkJoin Pool



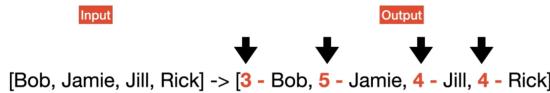
ForkJoin Task

- ForkJoin Task represents part of the data and its computation
- Type of tasks to submit to ForkJoin Pool
 - ForkJoinTask
 - RecursiveTask -> Task that returns a value
 - RecursiveAction -> Task that does not return a value



ForkJoin Task mainly used by java company.

ForkJoin - UseCase



A screenshot of an IDE showing Java code. The code defines a main method that starts a stopwatch, creates a result list, and iterates over a names list, applying a transform to each name before adding it to the result list. It then stops the stopwatch and logs the final result and total time taken.

```
13
14 ► public static void main(String[] args) {
15
16     stopWatch.start();
17     List<String> resultList = new ArrayList<>();
18     List<String> names = DataSet.namesList();
19     names.forEach((name)->{
20         String newValue = addNameLengthTransform(name);
21         resultList.add(newValue);
22     });
23     stopWatch.stop();
24     log("Final Result : "+resultList);
25     log("Total Time Taken : "+stopWatch.getTime());
26 }
27
28
```

Run: StringTransformExample

```
27
28
29 @ private static String addNameLengthTransform(String name) {
30     delay( delayMilliseconds: 500 );
31     return name.length() + " - " + name ;
32 }
33 }
```

Run: StringTransformExample

```
▶ /Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/lib/idea_rt.jar@11.0.4 -Dfile.encoding=UTF-8 -jar /Users/ashishgupta/IdeaProjects/StringTransformExample/target/StringTransformExample.jar
[main] - Final Result : [3 - Bob, 5 - Jamie, 4 - Jill, 4 - Rick]
[main] - Total Time Taken : 2024
Process finished with exit code 0
```

It took almost 2024 milliseconds (keep in mind we have 500 milliseconds delay in processing every string).

The below one is using forkjoin where data parallelism is happening to complete the same above task. It took only 524 milliseconds even though we have a 500 millisecond delay time.

```

22
23 public static void main(String[] args) {
24
25     stopWatch.start();
26     List<String> resultList = new ArrayList<>();
27     List<String> names = DataSet.namesList();
28     ForkJoinPool forkJoinPool = new ForkJoinPool();
29     ForkJoinUsingRecursion forkJoinUsingRecursion = new ForkJoinUsingRecursion(names);
30     resultList = forkJoinPool.invoke(forkJoinUsingRecursion);
31
32     stopWatch.stop();
33     log("Final Result : " + resultList);
34     log("Total Time Taken : " + stopWatch.getTime());
35 }
36
37
38 @
39 private static String addNameLengthTransform(String name) {
40     delay(delayMilliSeconds, 500);
41     return name.length() + "-" + name;
42 }
43
44 @Override
45 protected List<String> compute() {
46
47     if(inputList.size()<=1){
48         List<String> resultList = new ArrayList<>();
49         inputList.forEach(name-> resultList.add(addNameLengthTransform(name)));
50         return resultList;
51     }
52     int midpoint = inputList.size()/2;
53     ForkJoinTask<String> leftInputList= new ForkJoinUsingRecursion(inputList.subList(0,midpoint)).fork();
54     inputList = inputList.sublist(midpoint, inputList.size());
55     List<String> rightResult = compute(); //recursion happens
56     List<String> leftResult = leftInputList.join();
57     leftResult.addAll(rightResult);
58     return leftResult;
59 }
60

```

Run: ForkJoinUsingRecursion x
 /Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java
 [main] - Final Result : [3 - Bob, 5 - Jamie, 4 - Jill, 4 - Rick]
 [main] - Total Time Taken : 542
 Process finished with exit code 0

Disadvantages are that it is very difficult to understand this complex algorithm unless we are proactively practicing. From java 8 onwards this is more simplified.

Something let us solve it using the stream api.

Streams API & Parallel Streams

Streams API

- Streams API got introduced in **Java 8**
- Streams API is used to process a collection of Objects
- Streams in Java are created by using the **stream()** method

```
public List<String> stringTransform_upperCase(List<String> namesList){  
    return namesList  
        .stream()  
        .map(String::toUpperCase)  
        .collect(Collectors.toList());  
}
```

Parallel Streams

- This allows your code to run in parallel
- Parallel Streams are designed to solve **Data Parallelism**

```
public List<String> stringTransform_upperCase(List<String> namesList){  
    return namesList  
        .parallelStream() // ←  
        .map(String::toUpperCase) // ←  
        .collect(Collectors.toList());  
}
```

Stream/Parallel Stream

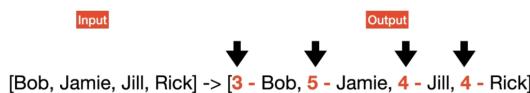
Stream

```
public List<String> stringTransform_upperCase(List<String> namesList){  
    return namesList  
        .stream() // ←  
        .map(String::toUpperCase)  
        .collect(Collectors.toList());  
}
```

Parallel Stream

```
public List<String> stringTransform_upperCase(List<String> namesList){  
    return namesList  
        .parallelStream() // ←  
        .map(String::toUpperCase)  
        .collect(Collectors.toList());  
}
```

Parallel Streams - UseCase



The screenshot shows two code files in an IDE:

ParallelStreamsExample.java

```
1 package com.example;
2
3 public class ParallelStreamsExample {
4
5     public List<String> stringTransform(List<String> namesList) {
6
7         return namesList
8             .stream()
9             .map(this::addNameLengthTransform)
10            .collect(Collectors.toList());
11     }
12
13     public static void main(String[] args) {
14
15         List<String> namesList = dataSet.namesList();
16         ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
17         startTimer();
18         List<String> resultlist = parallelStreamsExample.stringTransform(namesList);
19         log(resultList : " + resultlist);
20         timeTaken();
21     }
22
23     private String addNameLengthTransform(String name) {
24         delay(delayInMilliseconds 500);
25         return name.length() + "-" + name ;
26     }
27
28 }
29
```

CommonUtil.java

```
1 package com.example;
2
3 import java.util.concurrent.TimeUnit;
4
5 public class CommonUtil {
6
7     public static StopWatch stopWatch = new StopWatch();
8
9     public static void startTimer(){
10        stopWatch.start();
11    }
12
13    public static void timeTaken(){
14        stopWatch.stop();
15        log("Total Time Taken : " +stopWatch.getTime());
16    }
17
18    public static void delay(long delayMilliSeconds) {
19        try{
20            sleep(delayMilliSeconds);
21        }catch (Exception e){
22            log("Exception is :" + e.getMessage());
23        }
24    }
25
26 }
27
28 }
```

The screenshot shows the IDE's Run tab with the following output:

```
Run: ParallelStreamsExample
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java -javaagent:/
[main] - resultlist : [3 - Bob, 5 - Jamie, 4 - Jill, 4 - Rick]
[main] - Total Time Taken : 2052
Process finished with exit code 0
```

Change the stream to parallel stream you get quicker response.

```

12
13     @Override
14     public List<String> stringTransform(List<String> namesList){
15
16         return namesList
17             //.stream()
18             .parallelStream()
19             .map(this::addNameLengthTransform)
20             .collect(Collectors.toList());
21     }
22
23     public static void main(String[] args) {
24
25         List<String> namesList = DataSet.namesList();
26         ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
27         startTimer();
28         List<String> resultList = parallelStreamsExample.stringTransform(namesList);
29         log("resultList : " + resultList);
30         timeTaken();
31     }

```

Run: ParallelStreamsExample
 /Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar
 [main] - resultList : [3 - Bob, 5 - Jamie, 4 - Jill, 4 - Rick]
 [main] - Total Time Taken : 559
 [main] -
 Process finished with exit code 0

Java 7 Vs Java 8

```

16     public class ParallelStreamsExample {
17
18         @Override
19         public List<String> stringTransform(List<String> namesList){
20
21             return namesList
22                 //.stream()
23                 .parallelStream()
24                 .map(this::addNameLengthTransform)
25                 .collect(Collectors.toList());
26         }
27
28         public static void main(String[] args) {
29
30             List<String> namesList = DataSet.namesList();
31             ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
32             startTimer();
33             List<String> resultList = parallelStreamsExample.stringTransform(namesList);
34             log("resultList : " + resultList);
35             timeTaken();
36         }
37
38         private String addNameLengthTransform(String name) {
39             delay(delayMilliseconds: 500);
40             return name.length() + "-" + name;
41         }
42
43     }

```

```

16     private List<String> inputList;
17
18     public ForkJoinUsingRecursion(List<String> inputList) { this.inputList = inputList; }
19
20     public static void main(String[] args) {
21
22         stopWatch.start();
23         List<String> resultList = new ArrayList<>();
24         List<String> names = DataSet.namesList();
25         ForkJoinPool forkJoinPool = new ForkJoinPool();
26         ForkJoinUsingRecursion forkJoinUsingRecursion = new ForkJoinUsingRecursion(names);
27         resultList = forkJoinPool.invoke(forkJoinUsingRecursion);
28
29         stopWatch.stop();
30         log("Final Result : " + resultList);
31         log("Total Time Taken : " + stopWatch.getTime());
32     }
33
34     private static String addNameLengthTransform(String name) {
35         delay(delayMilliseconds: 500);
36         return name.length() + "-" + name;
37     }
38
39     @Override
40     protected List<String> compute() {
41
42         if(inputList.size()<=1){
43             List<String> resultList = new ArrayList<>();
44             inputList.forEach(name-> resultList.add(addNameLengthTransform(name)));
45             return resultList;
46         }
47         int midpoint = inputList.size()/2;
48         ForkJoinTask<List<String>> leftInputList = new ForkJoinUsingRecursion(inputList.subList(0, midpoint));
49         inputList = inputList.subList(midpoint, inputList.size());
50         List<String> rightResult = compute(); //recursion happens
51         List<String> leftResult = leftInputList.invoke();
52
53         for(int i=0; i<leftResult.size(); i++){
54             resultList.add(leftResult.get(i));
55             resultList.add(rightResult.get(i));
56         }
57
58         return resultList;
59     }

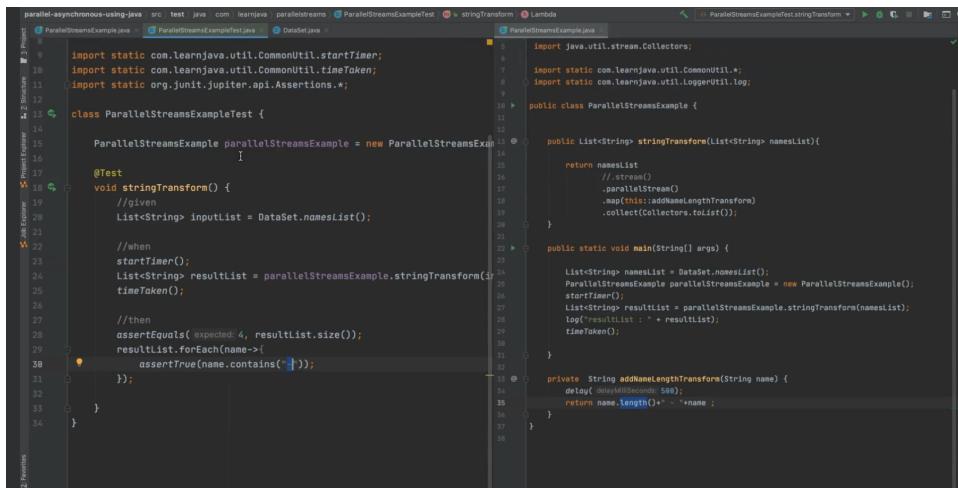
```

Run: ForkJoinUsingRecursion
 Build completed successfully in 2 414 ms (3 minutes ago)

Unit Testing Parallel Streams Using JUnit5

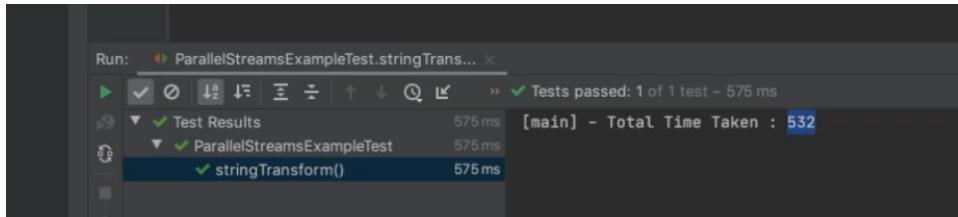
Why Unit Tests ?

- Unit Testing allows you to programmatically test your code
- Manual Testing slows down the development and delivery
- Unit Testing allows the developer or the app team to make enhancements to the existing code easily and faster



The screenshot shows an IDE interface with the following details:

- Project Explorer:** Shows a project named "parallel-asynchronous-using-Java" containing "ParallelStreamsExample.java", "ParallelStreamsExampleTest.java", and "DataSet.java".
- Code Editor:** Displays the content of `ParallelStreamsExampleTest.java`. The code includes imports for `com.learnjava.util.CommonUtil`, `com.learnjava.util.CommonUtil.timeTaken`, and `org.junit.jupiter.api.Assertions`. It defines a test class `ParallelStreamsExampleTest` with a test method `void stringTransform()` that uses a `ParallelStreamsExample` instance to transform a list of strings and assert the result.
- Code Snippet:** The code also includes a `main` method and a private helper method `String addNameLengthTransform(String name)`.



sequential() and parallel()

- Streams API are **sequential** by default
 - `sequential()` -> Executes the stream in sequential
 - `parallel()` -> Executes the stream in parallel
- Both the functions() changes the behavior of the whole pipeline

sequential()

- Changing the **parallelStream()** behavior to sequential

```
public List<String> stringTransform(List<String> namesList){  
    return namesList  
        .parallelStream()  
        .map(this::transform)  
        .sequential() ← Sequential  
        .collect(Collectors.toList());  
}
```

parallel()

- Changing the **stream()** behavior to parallel

```
public List<String> stringTransform(List<String> namesList){  
    return namesList  
        .stream()  
        .map(this::transform)  
        .parallel() ← Parallel  
        .collect(Collectors.toList());  
}
```

```
exampleTest {  
    @Test  
    public void testParallelStreams() {  
        List<String> namesList = DataSet.namesList();  
  
        long start = System.currentTimeMillis();  
        List<String> resultList = parallelStreamsExample.stringTransform(namesList);  
        long end = System.currentTimeMillis();  
  
        assertEquals(4, resultList.size());  
        assertEquals("Parallel streams example passed", "Parallel streams example passed");  
    }  
}  
  
public List<String> stringTransform(List<String> namesList){  
    return namesList  
        .parallelStream()  
        .map(this::addNameLengthTransform)  
        .sequential() [Override]  
        .collect(Collectors.toList());  
}  
  
public static void main(String[] args) {  
    List<String> namesList = DataSet.namesList();  
    ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();  
    parallelStreamsExample.startTimer();  
    parallelStreamsExample.stringTransform(namesList);  
    parallelStreamsExample.stopTimer();  
    System.out.println("Total Time Taken : " + parallelStreamsExample.getTotalTimeTaken());  
}
```

Tests passed: 1 of 1 test - 2 ms
2 ms
26 ms
[main] - Total Time Taken : 2022
Process finished with exit code 0

Notice the performance above sequential and below parallel!

One method overrides the other method. It will take only the latest one, in the below example - parallel.

```

    @Test
    public void parallelStreamsExampleTest {
        ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
        List<String> inputList = DataSet.namesList();
        List<String> resultlist = parallelStreamsExample.stringTransform(inputList);
        assertEquals(4, resultlist.size());
    }
}

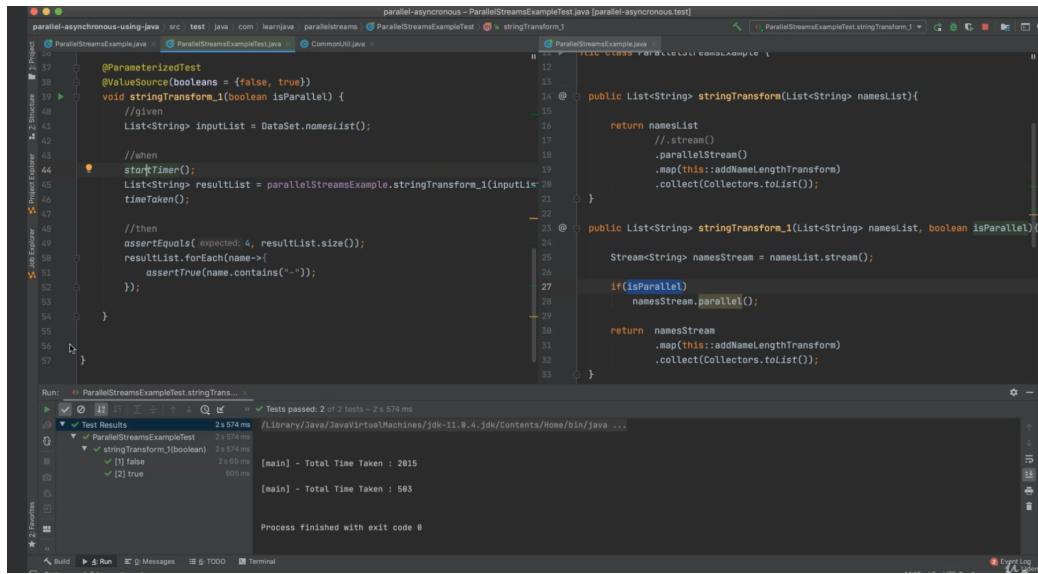
public class ParallelStreamsExample {
    public List<String> stringTransform(List<String> namesList) {
        return namesList
            .stream()
            .parallelStream()
            .map(this::addNameLengthTransform)
            .sequential()
            .parallel()
            .collect(Collectors.toList());
    }

    public static void main(String[] args) {
        List<String> namesList = DataSet.namesList();
        ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
        parallelStreamsExample.stringTransform(namesList);
    }
}

```

Tests passed: 1 of 1 test – 558 ms
 558 ms /Library/Java/JavaVirtualMachines/jdk-11.8.4.jdk/Contents/Home/bin/java ...
 558 ms [main] - Total Time Taken : 525
 Process finished with exit code 0

Test case parameterized test case



```

    @ParameterizedTest
    @ValueSource booleans = {false, true}
    void stringTransform_1(boolean isParallel) {
        //given
        List<String> inputList = DataSet.namesList();

        //when
        startTimer();
        List<String> resultlist = parallelStreamsExample.stringTransform_1(inputList, isParallel);
        timeTaken();

        //then
        assertEquals(expected: 4, resultlist.size());
        resultlist.forEach(name->
            assertTrue(name.contains("-"));
        );
    }
}

public class ParallelStreamsExample {
    public List<String> stringTransform_1(List<String> namesList, boolean isParallel) {
        Stream<String> namesStream = namesList.stream();

        if(isParallel)
            namesStream.parallel();

        return namesStream
            .map(this::addNameLengthTransform)
            .collect(Collectors.toList());
    }

    public static void main(String[] args) {
        List<String> namesList = DataSet.namesList();
        ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
        parallelStreamsExample.stringTransform(namesList);
    }
}

```

Tests passed: 2 of 2 tests – 2 s 674 ms
 2 s 674 ms /Library/Java/JavaVirtualMachines/jdk-11.8.4.jdk/Contents/Home/bin/java ...
 [1] false 2 s 69 ms [main] - Total Time Taken : 2015
 [2] true 605 ms [main] - Total Time Taken : 583
 Process finished with exit code 0