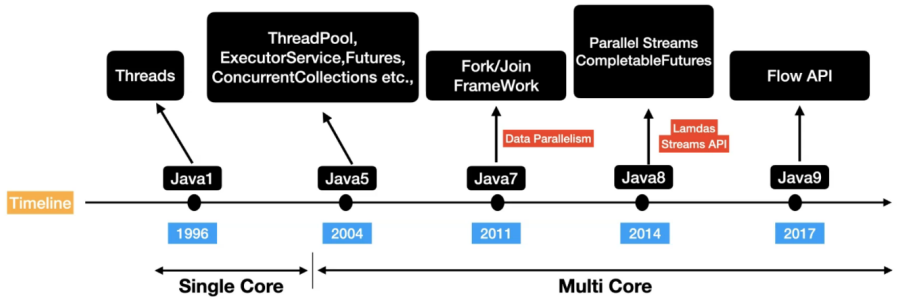
Technology Advancements

Hardware	Software
• Devices or computers comes up with Multiple cores	• MicroServices Architecture style
• Developer needs to learn programming patterns to maximize the use of multiple cores	• Blocking I/O calls are common in MicroServices Architecture. This also impacts the latency of the application
• Apply the Parallel Programming concepts	• Apply the Asynchronous Programming concepts
• **Parallel Streams**	• **CompletableFuture**

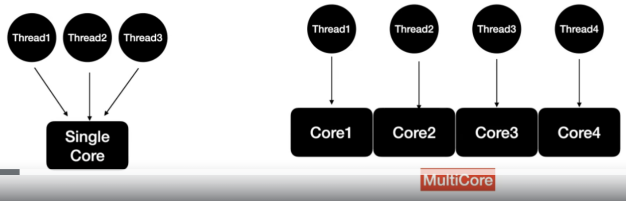
Threads Functional Style of Programming

Evolution of Concurrency/Parallelism APIs



Concurrency

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



Concurrency Example

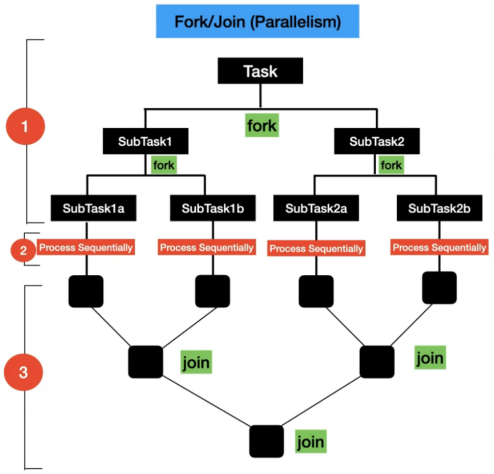
- In a real application, Threads normally need to interact with one another
  - Shared Objects or Messaging Queues
- Issues:
  - Race Condition
  - DeadLock and more
- Tools to handle these issues:
  - Synchronized Statements/Methods
  - Reentrant Locks, Semaphores
  - Concurrent Collections
  - Conditional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

1       Thread helloThread = new Thread(()-> hello());
        Thread worldThread = new Thread(()-> world());      Threads

2       //Starting the thread
        helloThread.start();
        worldThread.start();

3       //Joining the thread (Waiting for the threads to finish)
        helloThread.join();
        worldThread.join();

        System.out.println("Result is : " + result);
    }
}
```

Hello World

Parallelism

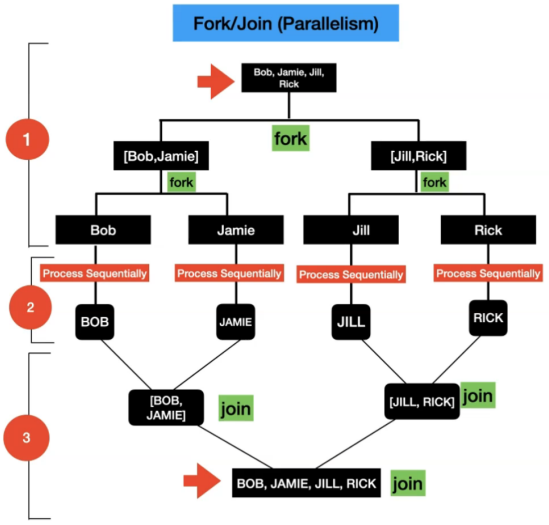
- Parallelism is a concept in which tasks are literally going to run in parallel
- Parallelism involves these steps:
  - Decomposing the tasks in to SubTasks(Forking)
  - Execute the subtasks in sequential
  - Joining the results of the tasks(Join)
- Whole process is also called **Fork/ Join**



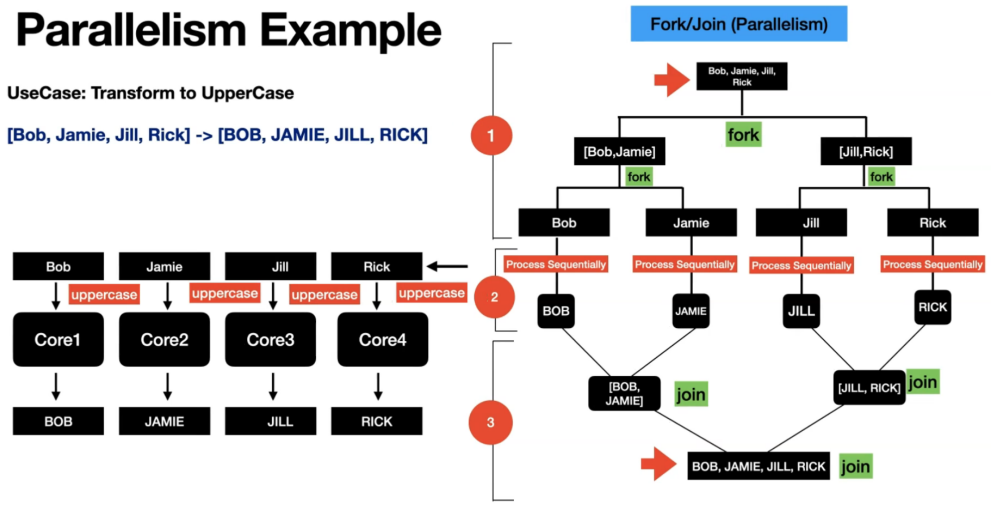
Parallelism Example

UseCase: Transform to UpperCase

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



What happening in point 2



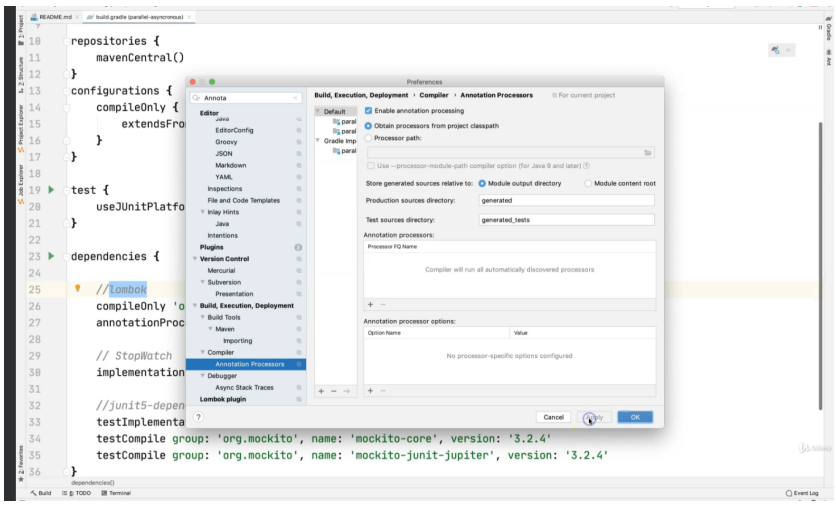
Parallelism Example

```
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()
                .map(String::toUpperCase)
                .collect(Collectors.toList());

        System.out.println("namesListUpperCase : " + namesListUpperCase);

    }
}
```

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

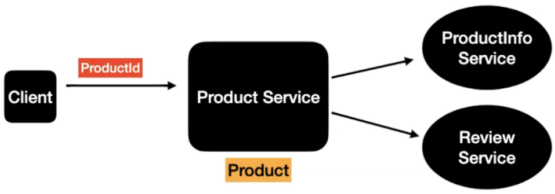


Enable lombok

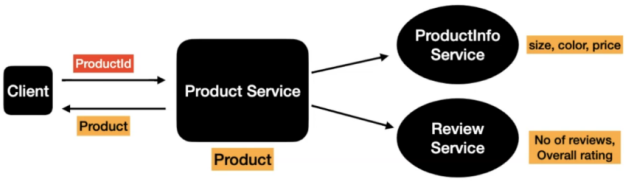
Section Overview

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

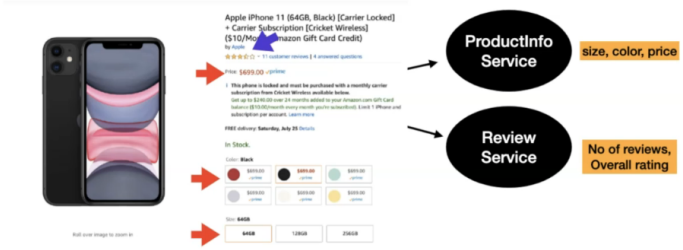
Product Service



Product Service



Product Service



```java
    }

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

        ProductInfo productInfo = productInfoService.retrieveProductInfo(productId); // blocking call
        Review review = reviewService.retrieveReviews(productId); // blocking call

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

    public static void main(String[] args) {

        ProductInfoService productInfoService = new ProductInfoService();
        ReviewService reviewService = new ReviewService();
        ProductService productService = new ProductService(productInfoService, reviewService);
        String productId = "ABC123";
        Product product = productService.retrieveProductDetails(productId);
        log("Product is " + product);

    }
}
```

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

```
Run:  ProductService
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=51673:/Applica
[main] - Total Time Taken : 2005
[main] - Product is Product(productId=ABC123, productInfo=ProductInfo(productId=ABC123, productOptions=[ProductOption(productionOptionId=1, size=64GB, color=

Process finished with exit code 0
```

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

```java
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);

}
```

ds On

```java
        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);

    }

    private class ProductInfoRunnable implements Runnable {
        private ProductInfo productInfo;
        private String productId;

        public ProductInfoRunnable(String productId) { this.productId = productId; }

        public ProductInfo getProductInfo() { return productInfo; }

        @Override
        public void run() {

            productInfo = productInfoService.retrieveProductInfo(productId);
        }
    }
```

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

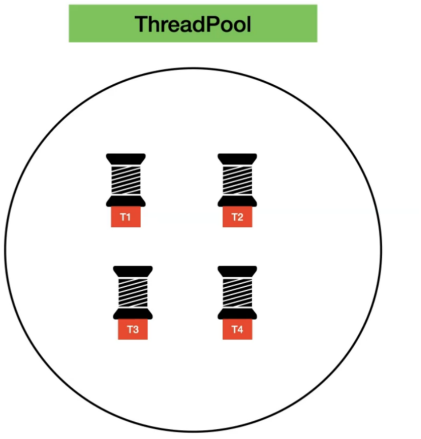
Limitations Of Thread

- 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

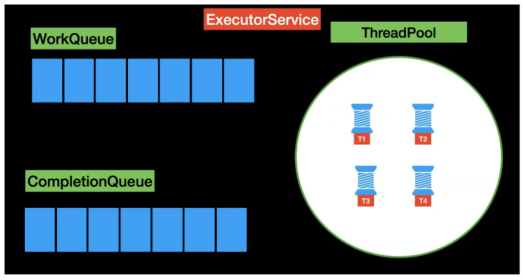


We no need to create threads

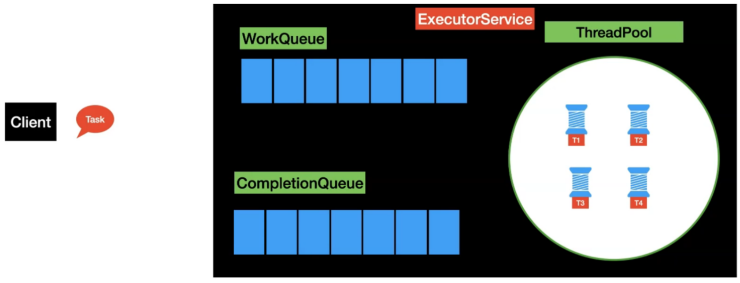
ExecutorService

- 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

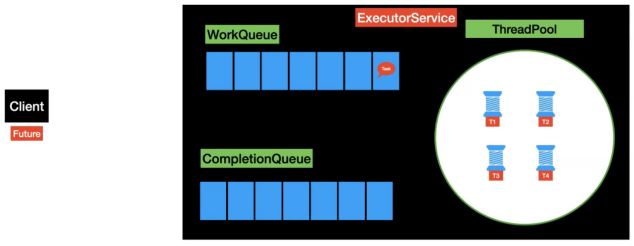
Working Of ExecutorService



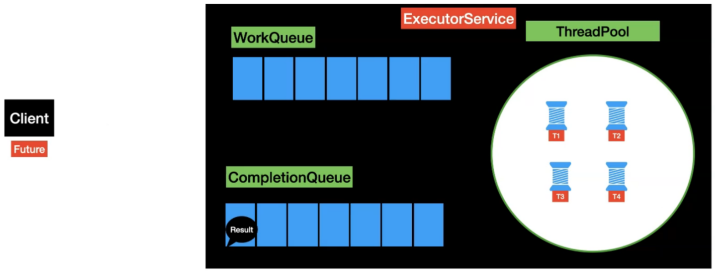
Working Of ExecutorService



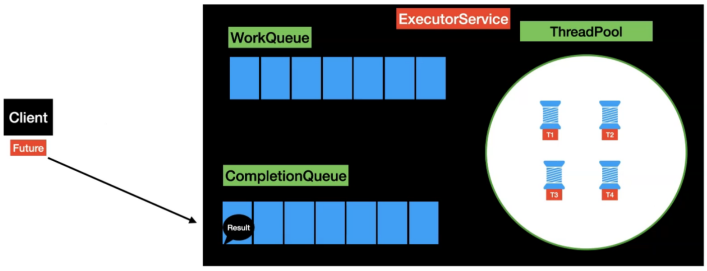
Working Of ExecutorService



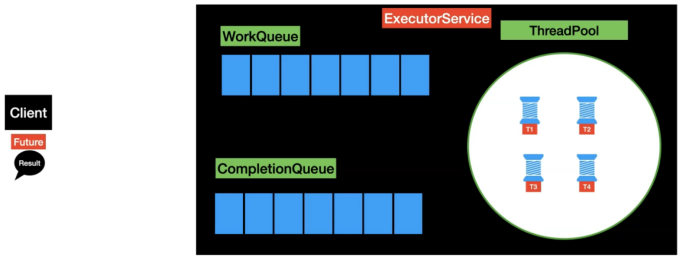
Working Of ExecutorService



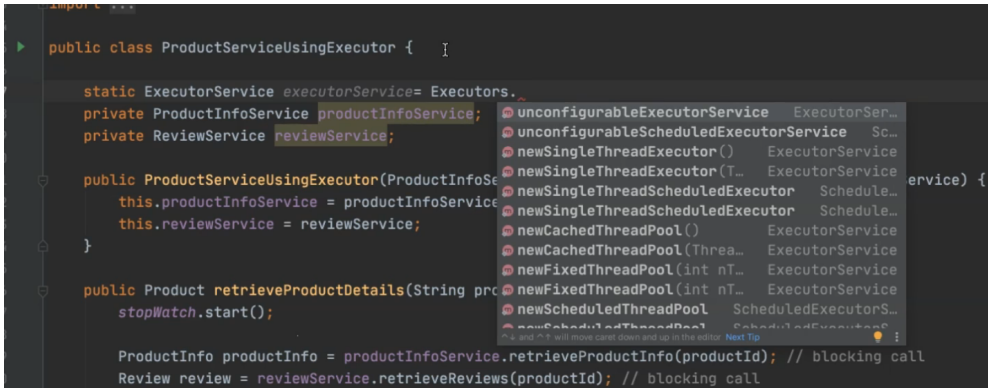
Working Of ExecutorService

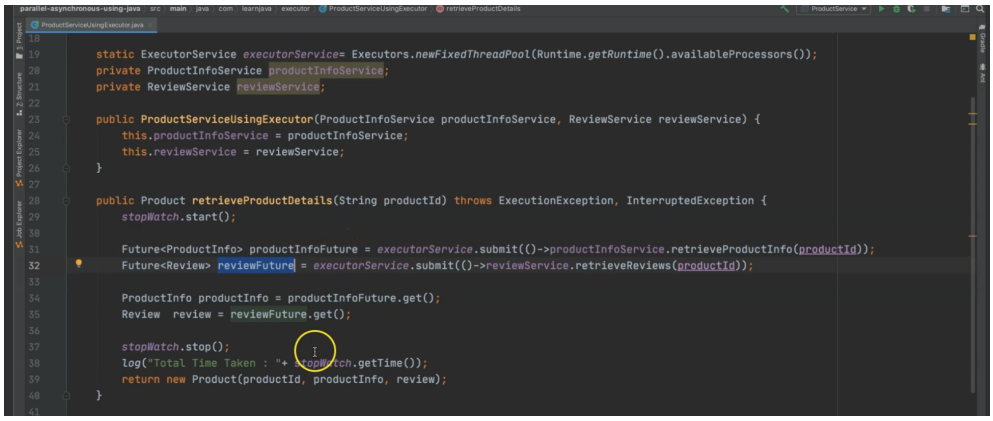


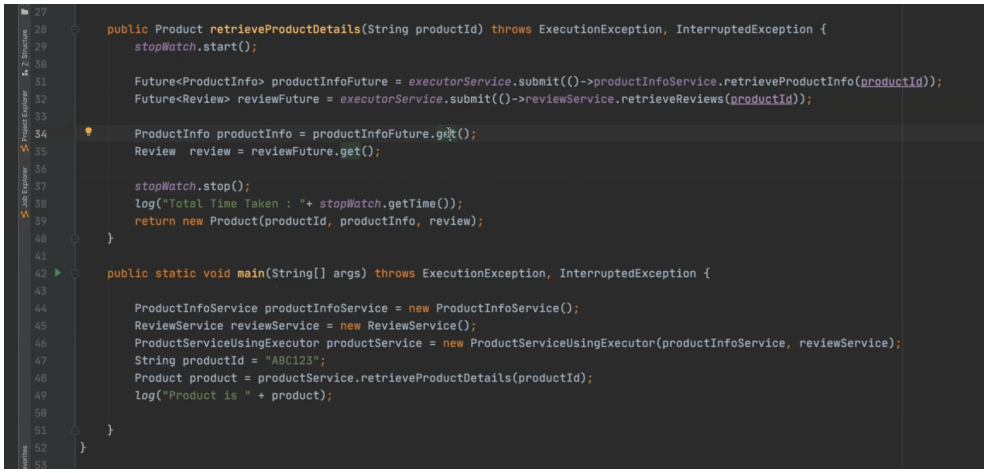
Working Of ExecutorService



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


import ...
public class ProductServiceUsingExecutor {
static ExecutorService executorService= Executors.
private ProductInfoService productInfoService;
private ReviewService reviewService;
public ProductServiceUsingExecutor(ProductInfoSe
this.productInfoService = productInfoService
this.reviewService = reviewService;
}
public Product retrieveProductDetails(String pro
stopWatch.start();
ProductInfo productInfo = productInfoService.retrieveProductInfo(productId); // blocking call
Review review = reviewService.retrieveReviews(productId); // blocking call
unconfigurableExecutorService ExecutorSer…
unconfigurableScheduledExecutorService Sc…
newSingleThreadExecutor() ExecutorService
newSingleThreadExecutor(T… ExecutorService
newSingleThreadScheduledExecutor Schedule…
newSingleThreadScheduledExecutor Schedule…
newCachedThreadPool() ExecutorService
newCachedThreadPool(Threa… ExecutorService
newFixedThreadPool(int nT… ExecutorService
newFixedThreadPool(int nT… ExecutorService
newScheduledThreadPool ScheduledExecutorS…
ervice) {


static ExecutorService executorService= Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
private ProductInfoService productInfoService;
private ReviewService reviewService;
public ProductServiceUsingExecutor(ProductInfoService productInfoService, ReviewService reviewService) {
this.productInfoService = productInfoService;
this.reviewService = reviewService;
}
public Product retrieveProductDetails(String productId) throws ExecutionException, InterruptedException {
stopWatch.start();
Future<ProductInfo> productInfoFuture = executorService.submit(()->productInfoService.retrieveProductInfo(productId));
Future<Review> reviewFuture = executorService.submit(()->reviewService.retrieveReviews(productId));
ProductInfo productInfo = productInfoFuture.get();
Review review = reviewFuture.get();
stopWatch.stop();
log("Total Time Taken : "+ stopWatch.getTime());
return new Product(productId, productInfo, review);
}

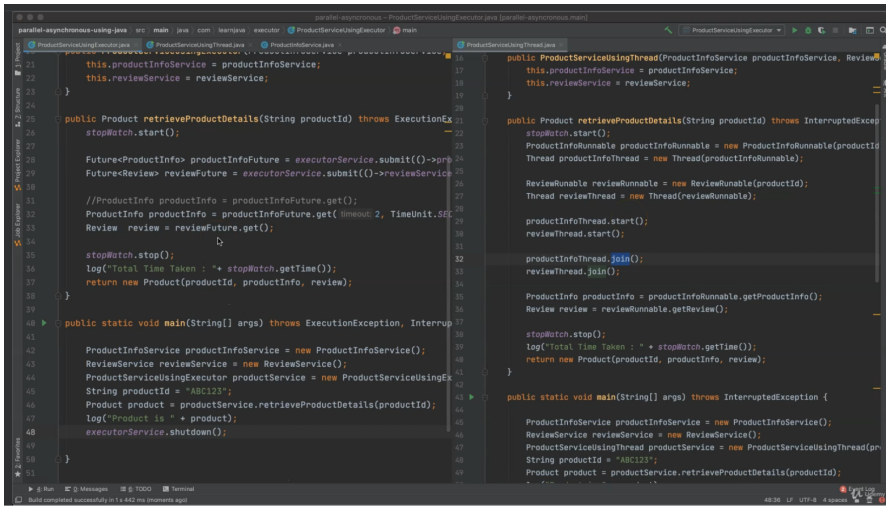

public Product retrieveProductDetails(String productId) throws ExecutionException, InterruptedException {
stopWatch.start();
Future<ProductInfo> productInfoFuture = executorService.submit(()->productInfoService.retrieveProductInfo(productId));
Future<Review> reviewFuture = executorService.submit(()->reviewService.retrieveReviews(productId));
ProductInfo productInfo = productInfoFuture.get();
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);
}
}


Run: ProductServiceUsingExecutor
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=59983:/Applica
[main] - Total Time Taken : 1003
[main] - Product is Product(productId=ABC123, productInfo=ProductInfo(productId=ABC123, productOptions=[ProductOption(productionOptionId=1, size=64GB, color=

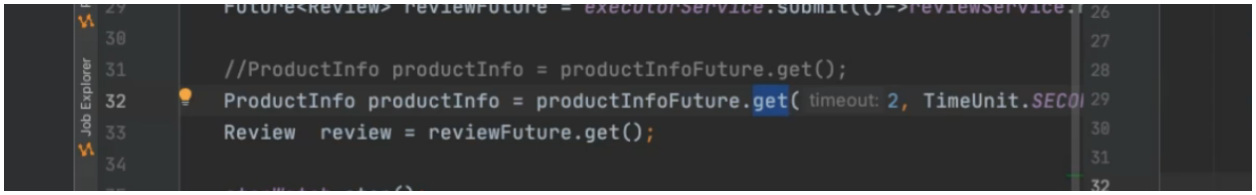
```
31          //ProductInfo productInfo = productInfoFuture.get();
32          ProductInfo productInfo = productInfoFuture.get( timeout: 2, TimeUnit.SECONDS);
33          Review  review = reviewFuture.get();
34
35          stopWatch.stop();
36          log("Total Time Taken : "+ stopWatch.getTime());
37          return new Product(productId, productInfo, review);
38      }
39
40      public static void main(String[] args) throws ExecutionException, InterruptedException, TimeoutException {
41
42          ProductInfoService productInfoService = new ProductInfoService();
43          ReviewService reviewService = new ReviewService();
44          ProductServiceUsingExecutor productService = new ProductServiceUsingExecutor(productInfoService, reviewService);
45          String productId = "ABC123";
46          Product product = productService.retrieveProductDetails(productId);
47          log("Product is " + product);
48          executorService.shutdown();
49
50      }
51  }
52
```

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



Disadvantages



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

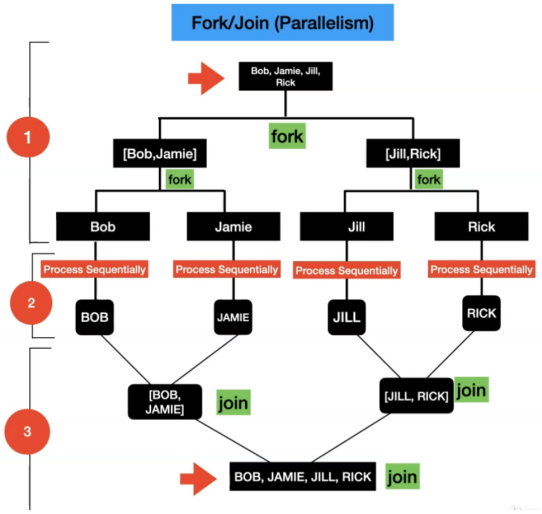
- This got introduced as part of **Java7**
- This is an extension of **ExecutorService**
- Fork/Join framework is designed to achieve **Data Parallelism**
- ExecutorService is designed to achieve **Task Based Parallelism**

```
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 it leaset possible size and execute those tasks in parallel
- Basically it uses the **divide and conquer** approach

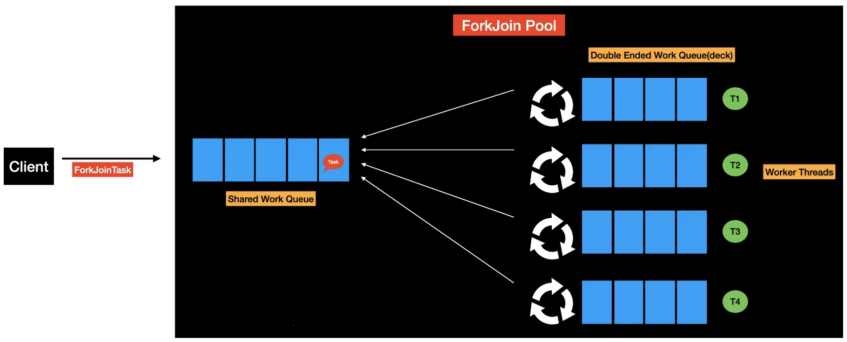
Watch "Concurrency vs Parallelism"



How does Fork/Join Framework Works ?

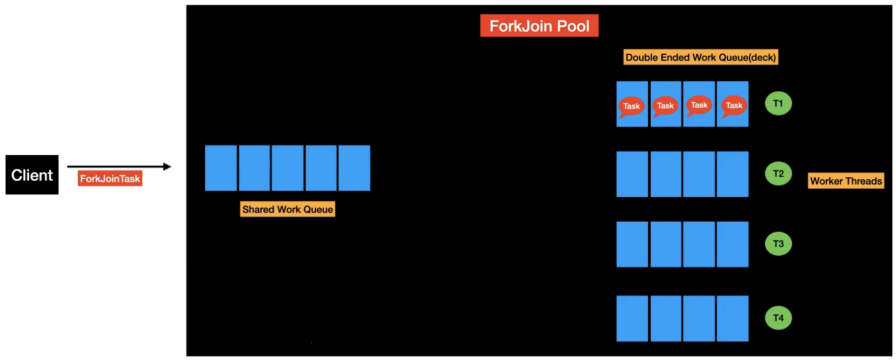
ForkJoin Pool to support Data Parallelism

ForkJoin Pool



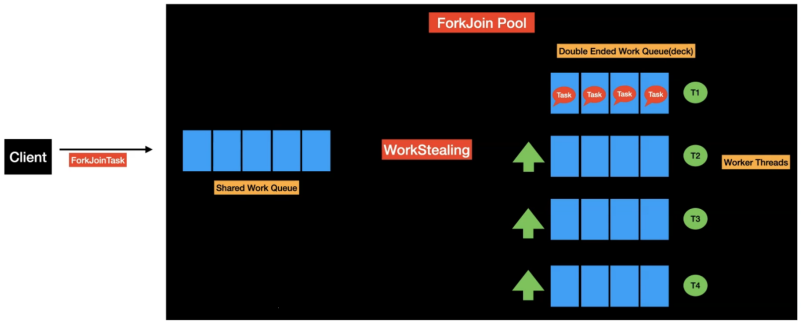
Polling happen and will take any of the thread

ForkJoin Pool

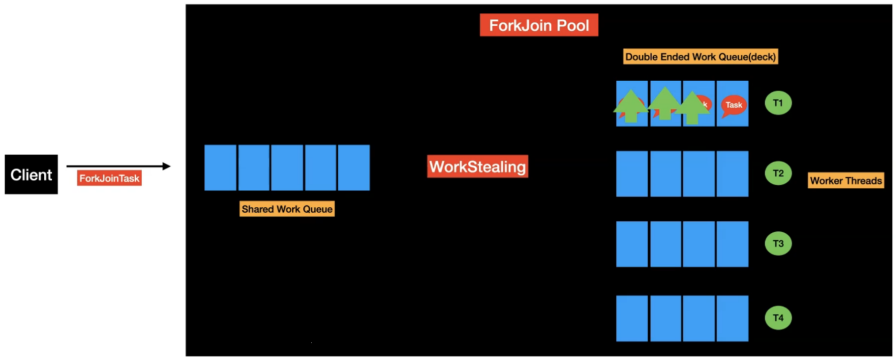


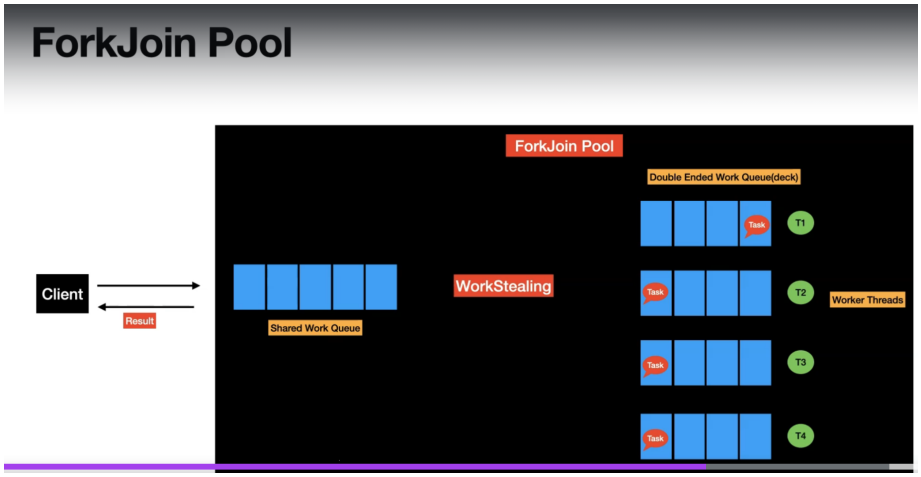
And it will check whether task can be further divisible and divide

ForkJoin Pool



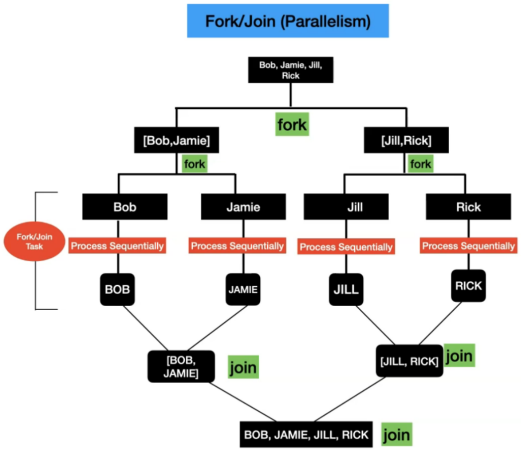
ForkJoin Pool





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


```
    public static void main(String[] args) {

        stopWatch.start();
        List<String> resultList = new ArrayList<>();
        List<String> names = DataSet.namesList();
        names.forEach((name)->{
            String newValue = addNameLengthTransform(name);
            resultList.add(newValue);
        });
        stopWatch.stop();
        log("Final Result : "+ resultList);
        log("Total Time Taken : "+ stopWatch.getTime());
    }
```

```
    private static String addNameLengthTransform(String name) {
        delay( delayMilliSeconds: 500);
        return name.length()+" - "+name ;
    }
}
```

```
Run: StringTransformExample
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java -javaagent:/Applications/Int
[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.

```java
public static void main(String[] args) {

    stopWatch.start();
    List<String> resultList = new ArrayList<>();
    List<String> names = DataSet.namesList();
    ForkJoinPool forkJoinPool = new ForkJoinPool();
    ForkJoinUsingRecursion forkJoinUsingRecursion = new ForkJoinUsingRecursion(names);
    resultList = forkJoinPool.invoke(forkJoinUsingRecursion);

    stopWatch.stop();
    log("Final Result : "+ resultList);
    log("Total Time Taken : "+ stopWatch.getTime());
}


private static String addNameLengthTransform(String name) {
    delay( delayMilliSeconds: 500);
    return name.length()+" - "+name ;
}
```

```java
@Override
protected List<String> compute() {

    if(inputList.size()<=1){
       List<String> resultList = new ArrayList<>();
       inputList.forEach(name-> resultList.add(addNameLengthTransform(name)));
       return resultList;
    }
    int midpoint = inputList.size()/2;
    ForkJoinTask<List<String>> leftInputList=  new ForkJoinUsingRecursion(inputList.subList(0,midpoint)).fork();
    inputList = inputList.subList(midpoint, inputList.size());
    List<String> rightResult = compute(); //recursion happens
    List<String> leftResult = leftInputList.join();
    leftResult.addAll(rightResult);
    return leftResult;
}
}
```

```
Run:  ForkJoinUsingRecursion
/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.

Samething 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());
}
```

ParallelStreams

- This allows your code to run in parallel
- ParallelStreams 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());
}
```

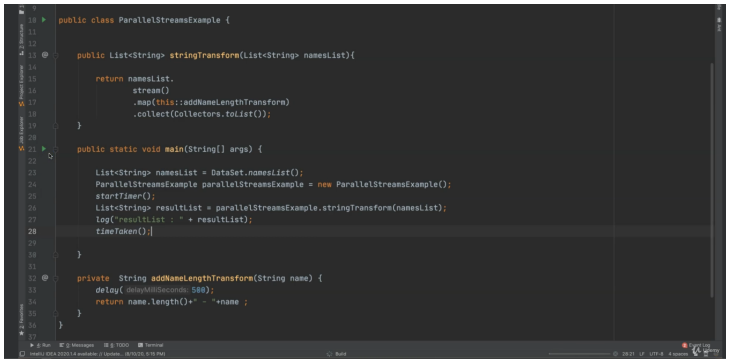
Parallel Stream

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

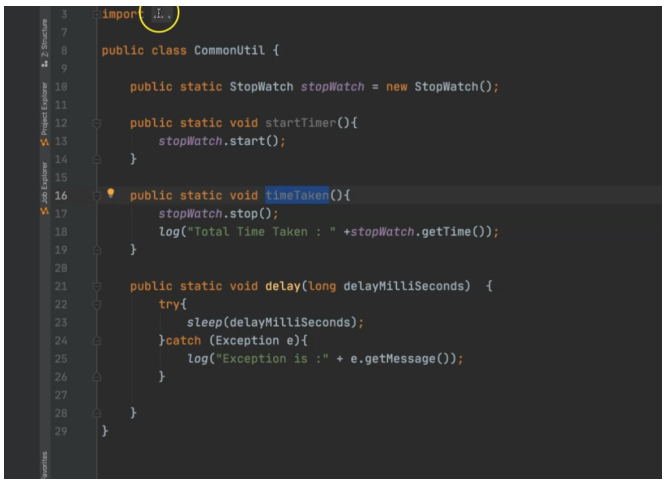
Parallel Streams - UseCase

Input Output

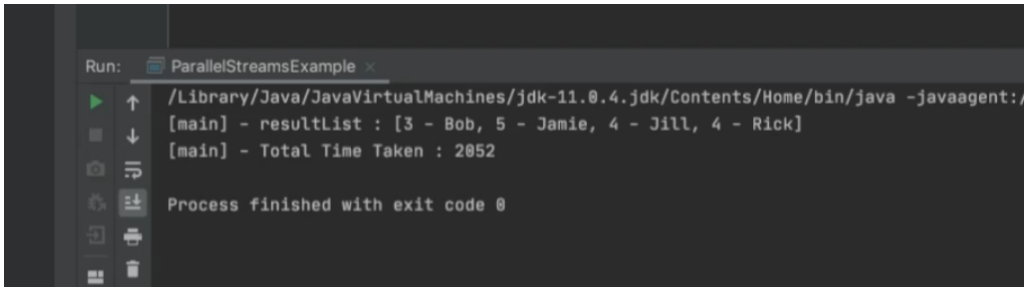
[Bob, Jamie, Jill, Rick] -> [3 - Bob, 5 - Jamie, 4 - Jill, 4 - Rick]



```java
public class ParallelStreamsExample {

    public List<String> stringTransform(List<String> namesList){

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

    public static void main(String[] args) {

        List<String> namesList = DataSet.namesList();
        ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
        startTimer();
        List<String> resultList = parallelStreamsExample.stringTransform(namesList);
        log("resultList : " + resultList);
        timeTaken();

    }

    private  String addNameLengthTransform(String name) {
        delay( delayMilliSeconds: 500);
        return name.length()+" - "+name ;
    }
}
```

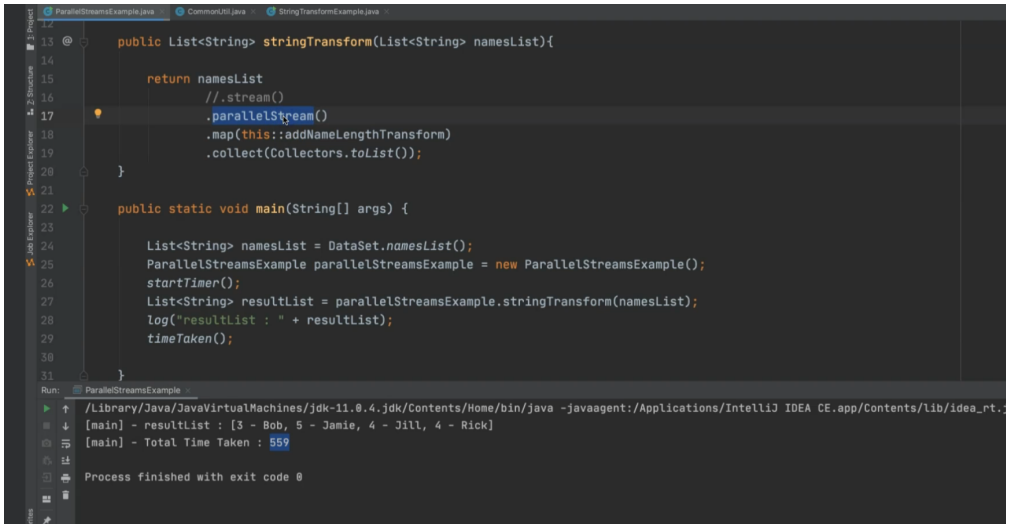


```java
public class CommonUtil {

    public static StopWatch stopWatch = new StopWatch();

    public static void startTimer(){
        stopWatch.start();
    }

    public static void timeTaken(){
        stopWatch.stop();
        log("Total Time Taken : " +stopWatch.getTime());
    }

    public static void delay(long delayMilliSeconds)  {
        try{
            sleep(delayMilliSeconds);
        }catch (Exception e){
            log("Exception is :" + e.getMessage());
        }

    }
}
```



```
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
```

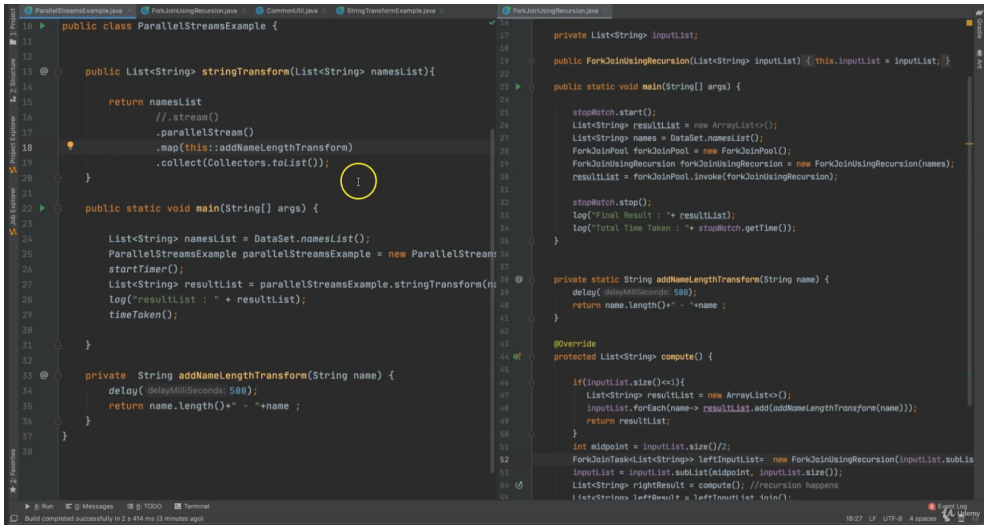
Change the steam to parallel stream you get quicker response.



```
public List<String> stringTransform(List<String> namesList){

    return namesList
            //.stream()
            .parallelStream()
            .map(this::addNameLengthTransform)
            .collect(Collectors.toList());
}

public static void main(String[] args) {

    List<String> namesList = DataSet.namesList();
    ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
    startTimer();
    List<String> resultList = parallelStreamsExample.stringTransform(namesList);
    log("resultList : " + resultList);
    timeTaken();

}
```

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

Process finished with exit code 0
```

Java 7 Vs Java 8



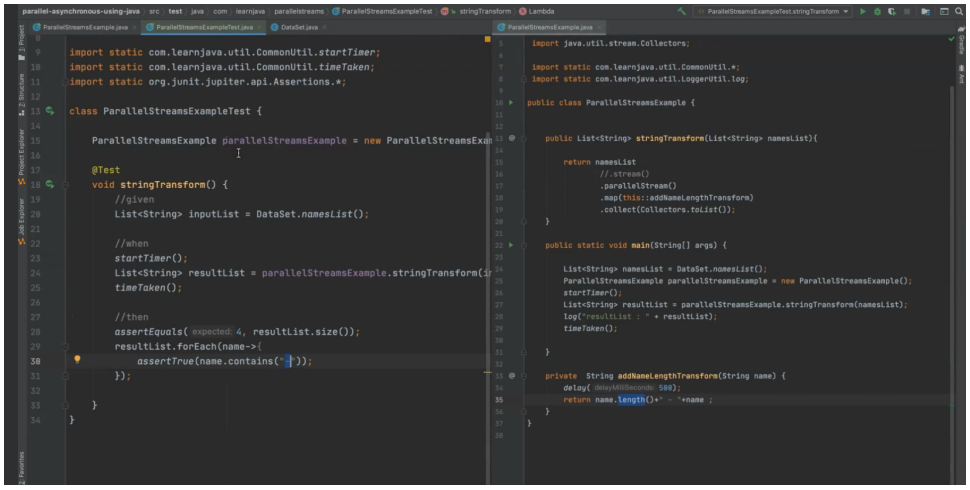
```
public class ParallelStreamsExample {

    public List<String> stringTransform(List<String> namesList){

        return namesList
                //.stream()
                .parallelStream()
                .map(this::addNameLengthTransform)
                .collect(Collectors.toList());
    }

    public static void main(String[] args) {

        List<String> namesList = DataSet.namesList();
        ParallelStreamsExample parallelStreamsExample = new ParallelStreams
        startTimer();
        List<String> resultList = parallelStreamsExample.stringTransform(n
        log("resultList : " + resultList);
        timeTaken();

    }

    private  String addNameLengthTransform(String name) {
        delay( delayMilliSeconds: 500);
        return name.length()+" - "+name ;
    }
}
```

```
    private List<String> inputList;

    public ForkJoinUsingRecursion(List<String> inputList) { this.inputList = inputList; }

    public static void main(String[] args) {

        stopWatch.start();
        List<String> resultList = new ArrayList<>();
        List<String> names = DataSet.namesList();
        ForkJoinPool forkJoinPool = new ForkJoinPool();
        ForkJoinUsingRecursion forkJoinUsingRecursion = new ForkJoinUsingRecursion(names);
        resultList = forkJoinPool.invoke(forkJoinUsingRecursion);

        stopWatch.stop();
        log("Final Result : "+ resultList);
        log("Total Time Taken : "+ stopWatch.getTime());
    }


    private static String addNameLengthTransform(String name) {
        delay( delayMilliSeconds: 500);
        return name.length()+" - "+name ;
    }

    @Override
    protected List<String> compute() {

        if(inputList.size()<=1){
            List<String> resultList = new ArrayList<>();
            inputList.forEach(name-> resultList.add(addNameLengthTransform(name)));
            return resultList;
        }
        int midpoint = inputList.size()/2;
        ForkJoinTask<List<String>> leftInputList=  new ForkJoinUsingRecursion(inputList.subLis
        inputList = inputList.subList(midpoint, inputList.size());
        List<String> rightResult = compute(); //recursion happens
```

Unit Testing Parallel Streams Using JUnit5

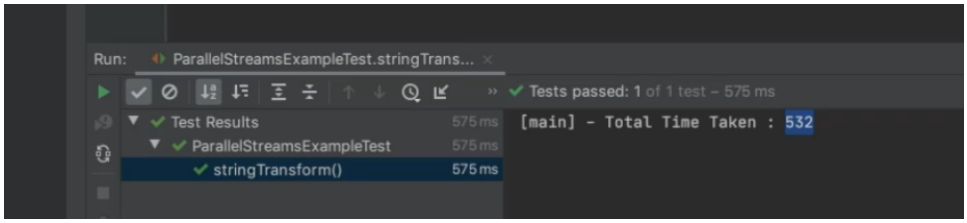
Why Unit Tests ?

- 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



```
import static com.learnjava.util.CommonUtil.startTimer;
import static com.learnjava.util.CommonUtil.timeTaken;
import static org.junit.jupiter.api.Assertions.*;

class ParallelStreamsExampleTest {

    ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExa

    @Test
    void stringTransform() {
        //given
        List<String> inputList = DataSet.namesList();

        //when
        startTimer();
        List<String> resultList = parallelStreamsExample.stringTransform(i
        timeTaken();

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

    }
}
```

```
import java.util.stream.Collectors;

import static com.learnjava.util.CommonUtil.*;
import static com.learnjava.util.LoggerUtil.log;

public class ParallelStreamsExample {

    public List<String> stringTransform(List<String> namesList){

        return namesList
                //.stream()
                .parallelStream()
                .map(this::addNameLengthTransform)
                .collect(Collectors.toList());
    }

    public static void main(String[] args) {

        List<String> namesList = DataSet.namesList();
        ParallelStreamsExample parallelStreamsExample = new ParallelStreamsExample();
        startTimer();
        List<String> resultList = parallelStreamsExample.stringTransform(namesList);
        log("resultList : " + resultList);
        timeTaken();

    }

    private String addNameLengthTransform(String name) {
        delay( delayMilliSeconds: 500);
        return name.length()+" - "+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()
            .collect(Collectors.toList());
}
```

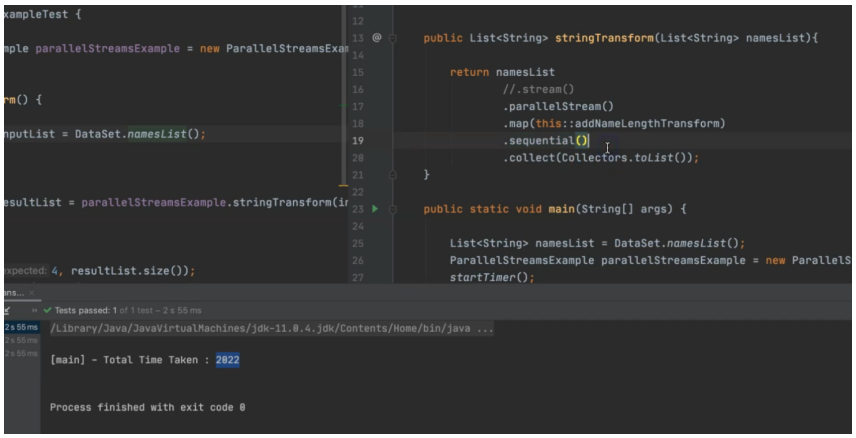
Sequential

parallel()

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

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

Parallel



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.

```
sExampleTest {

xample parallelStreamsExample = new ParallelStreamsExa

form() {

inputList = DataSet.namesList();

;

resultList = parallelStreamsExample.stringTransform(ir

( expected: 4, resultList.size());
```

```
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 Parall
```

```
Tests passed: 1 of 1 test – 558 ms
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...
[main] - Total Time Taken : 525

Process finished with exit code 0
```

Test case parameterized test case