

6 Concurrency - 2

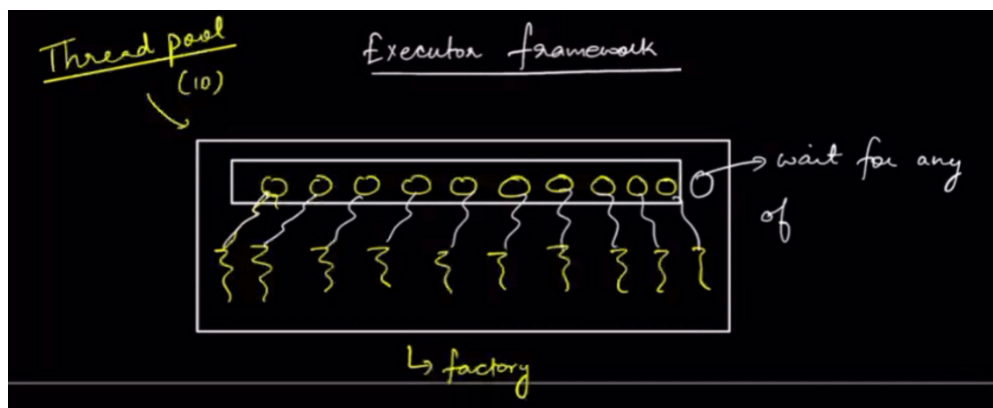
04 June 2024 21:15

Agenda :

1. Executor
2. Callable
3. Multithreaded Merge Sort
4. Intro to adder Subtractor Problem

1. Executors

- In a multithreaded environment, we divide the responsibilities into Parts
 - a. **Client** : knows What task to run
 - b. **Executors** : knows the best way to efficiently run the task, in order to achieve concurrency.



Waits for any other thread to finish to assign other task.

- Example of Executor Implementation

The screenshot shows an IDE with the following code:

```
new *
public class Client {
    new *
    public static void main(String[] args) {
        ExecutorService ex = Executors.newCachedThreadPool( threadFactory: 10);
        for(int i = 1 ; i <= 100 ; i++){
            PrintNumber t = new PrintNumber(i);
            ex.submit(t);
        }
    }
}
```

The output window shows the following text:

```
Printing 31 in thread : - pool-1-thread-7
Printing 32 in thread : - pool-1-thread-7
Printing 33 in thread : - pool-1-thread-7
Printing 34 in thread : - pool-1-thread-7
Printing 35 in thread : - pool-1-thread-7
Printing 36 in thread : - pool-1-thread-7
```

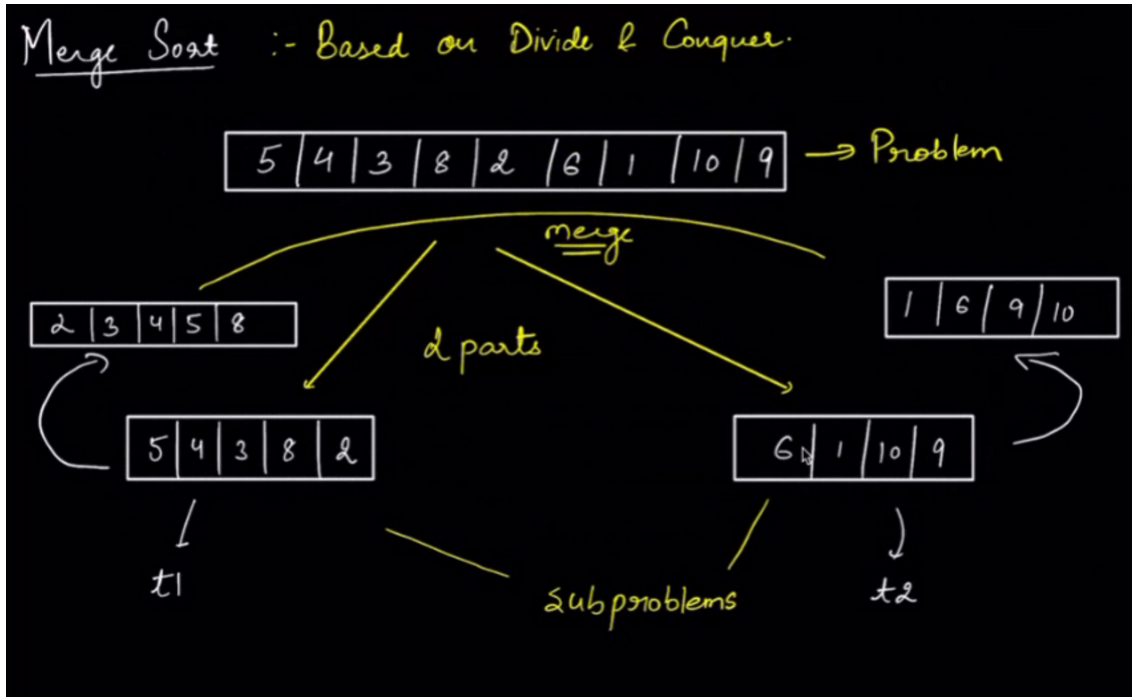
Newcachedthread pool passed with parameter creates that many threads.

The screenshot shows an IDE with the following code:

```
1 usage new *
PrintNumber(int noToPrint) { this.noToPrint = noToPrint; }

new *
@Override
public void run() {
    System.out.println("Printing" + noToPrint + " in thread : - " + Thr
}
```

Merge Sort with Threads to perform sorting concurrently



We cannot use `run()`, (`Runnable` interface) as it does not return anything.
So we need `Callable` interface here in this case.

2. Callable

Callable : `Runnable` + Returns some data.

```
class _____ implements Callable {  
    _____ call() {  
        _____  
    }  
}
```

3. Multithreaded Merge Sort

Client.java

```
package MergeSortMultiThreaded;  
  
import java.util.List;  
import java.util.concurrent.ExecutionException;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.Future;  
  
public class Client {  
    public static void main(String[] args) throws ExecutionException, InterruptedException {  
        ExecutorService ex = Executors.newCachedThreadPool();  
  
        List<Integer> ls = List.of(1,4,56,3,2,43,2,3,32,23,3);
```

```

        Sorter t = new Sorter(ls, ex);

        Future<List<Integer>> res = ex.submit(t);

        ls = res.get();

        System.out.println(ls);

    }
}

```

Sorter.java

```

package MergeSortMultiThreaded;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Future;

public class Sorter implements Callable<List<Integer>> {
    List<Integer> arrayToSort;
    ExecutorService ex;
    Sorter(List<Integer> arrayToSort, ExecutorService ex){
        this.arrayToSort = arrayToSort;
        this.ex = ex;
    }
    @Override
    public List<Integer> call() throws Exception {
        //Write the entire merge sort code
        if(arrayToSort.size() <= 1){
            return arrayToSort;
        }

        int mid = arrayToSort.size() / 2;
        /*
        First half - 0 to mid - 1
        second half - mid to size - 1
        */

        List<Integer> leftHalf = new ArrayList<>();
        for(int i = 0 ; i < mid ; i++){
            leftHalf.add(arrayToSort.get(i));
        }

        List<Integer> rightHalf = new ArrayList<>();
        for(int i = mid ; i < arrayToSort.size() ; i++){
            rightHalf.add(arrayToSort.get(i));
        }

        Sorter task1 = new Sorter(leftHalf, ex);
        Sorter task2 = new Sorter(rightHalf, ex);

        Future<List<Integer>> leftSortedArray = ex.submit(task1);
        Future<List<Integer>> rightSortedArray = ex.submit(task2);
        leftHalf = leftSortedArray.get();
        rightHalf = rightSortedArray.get();

        /*
        merge left and right half
        */
    }
}

```

```

List<Integer> finalMergedArray = new ArrayList<>();

int i = 0 , j = 0;
while(i < leftHalf.size() && j < rightHalf.size()){
    if(leftHalf.get(i) < rightHalf.get(j)){
        finalMergedArray.add(leftHalf.get(i));
        i++;
    }else{
        finalMergedArray.add(rightHalf.get(j));
        j++;
    }
}

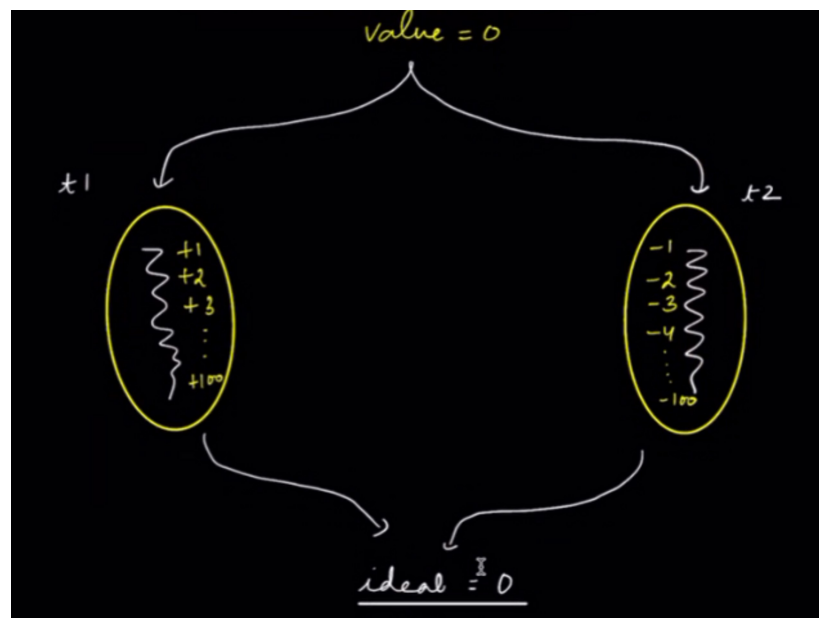
while(i < leftHalf.size()){
    finalMergedArray.add(leftHalf.get(i));
    i++;
}

while(j < rightHalf.size()){
    finalMergedArray.add(rightHalf.get(j));
    j++;
}

return finalMergedArray;
}
}

```

4. Synchronization Problem (Adder Subtractor Problem)



But since both the threads are not synchronized result need not to be 0.

Code :

```

2
3 import java.util.concurrent.Callable;
4
5 public class Adder implements Callable<Integer> {
6     private Count count;
7     Adder(Count count){
8         this.count = count;
9     }
10    @Override
11    public Integer call() throws Exception {
12        for(int i = 1 ; i <= 100 ; i++){
13            count.value += i;
14        }
15        return null;
16    }
17

```

```

2
3 import java.util.concurrent.Callable;
4
5 public class Subtractor implements Callable<Integer> {
6     private Count count;
7
8     Subtractor(Count count){
9         this.count = count;
10    }
11
12    @Override
13    public Integer call() throws Exception {
14        for(int i = 1 ; i <= 100 ; i++){
15            count.value -= i;
16        }
17        return null;
18    }
19

```

Fix return type to void in adder and subtractor.

Unsyncronised code to implement adder and subtractor using threads.

```

8 public class Client {
9     public static void main(String[] args) throws ExecutionException, Interrupt
10    {
11        Count count = new Count();
12
13        ExecutorService ex = Executors.newCachedThreadPool();
14
15        Adder t1 = new Adder(count);
16        Subtractor t2 = new Subtractor(count);
17
18        Future<Void> res1 = ex.submit(t1);
19        Future<Void> res2 = ex.submit(t2);
20
21        res1.get();
22        res2.get();
23
24        System.out.println(count.value);
25    }
26

```

This code can results in wrong value. So we need to synchronize the code.

