
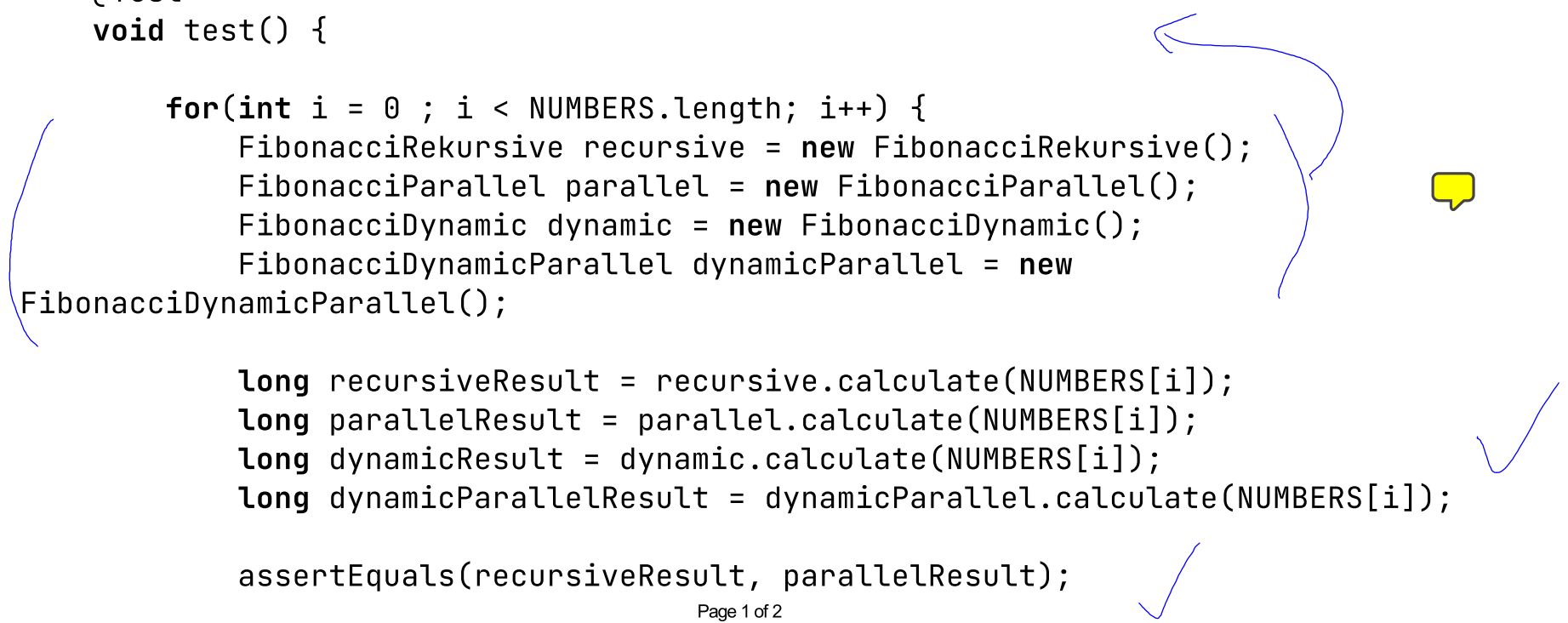


```
1 package uebung1111;  
2  
3 public class maintest {  
4     public static void main(String[] args) {  
5         FibonacciRekursive re = new FibonacciRekursive();  
6         FibonacciParallel pa = new FibonacciParallel();  
7         FibonacciDynamic dy = new FibonacciDynamic();  
8         FibonacciDynamicParallel dp = new FibonacciDynamicParallel();  
9         System.out.println("ergebnis recursive: " + re.calculate(16));  
10        System.out.println("ergebnis parallel: " + pa.calculate(16));  
11        System.out.println("ergebnis dynamic: " + dy.calculate(10));  
12        System.out.println("ergebnis dynamicparallel: " + dp.calculate(10));  
13    }  
14 }  
15
```



```
1 package uebung1111;  
2  
3 public abstract class Fibonacci {  
4  
5     public abstract long calculate(int n);  
6  
7 }  
8
```

```
1 package uebung1111;
2
3 import org.junit.jupiter.api.Test;
4
5 import static org.junit.jupiter.api.Assertions.assertEquals;
6
7 class FibonacciTest {
8     //Warum gibt es Fehler ab number 16 und höher????
9     private static final int[] NUMBERS = new int[] { 3, 5, 8, 12, 9, 15, 10, 7, 11
10 };
11
12 @Test
13 void test() {
14     for(int i = 0 ; i < NUMBERS.length; i++) {
15         FibonacciRekursive recursive = new FibonacciRekursive();
16         FibonacciParallel parallel = new FibonacciParallel();
17         FibonacciDynamic dynamic = new FibonacciDynamic();
18         FibonacciDynamicParallel dynamicParallel = new
19             FibonacciDynamicParallel();
20
21         long recursiveResult = recursive.calculate(NUMBERS[i]);
22         long parallelResult = parallel.calculate(NUMBERS[i]);
23         long dynamicResult = dynamic.calculate(NUMBERS[i]);
24         long dynamicParallelResult = dynamicParallel.calculate(NUMBERS[i]);
25
26         assertEquals(recursiveResult, parallelResult);
27     }
28 }
```



```

26         assertEquals(recursiveResult, dynamicResult);
27         assertEquals(recursiveResult, dynamicParallelResult);
28
29
30     }
31 }
32
33 }
34

```



```
1 package uebung1111;  
2  
3 public class FibonacciRunner extends Thread{  
4  
5  
6     private long n;  
7  
8     public long getResult() {  
9         return result;  
10    }  
11  
12    private long result;  
13    public FibonacciRunner(long n){  
14        this.n = n;  
15    }  
16    @Override  
17    public void run() {  
18        result = calculate(n);  
19    }  
20  
21    private long calculate(long n){  
22        long result;  
23        if(n == 0){  
24            return 0;  
25        }  
26        if(n == 1){  
27            return 1;
```

```
28
29     }
30     else{
31         FibonacciRunner nminus1 = new FibonacciRunner(n-1);
32         FibonacciRunner nminus2 = new FibonacciRunner(n-2);
33
34         nminus1.start();
35         nminus2.start();
36
37         try{
38             nminus1.join();
39             nminus2.join();
40         }
41         catch( InterruptedException e){
42
43         }
44
45         result = nminus1.getResult() + nminus2.getResult();
46         return result;
47     }
48 }
49 }
50
```

```
1 package uebung1111;  
2  
3 import java.util.HashMap;  
4 import java.util.Map;  
5  
6 public class FibonacciDynamic extends Fibonacci {  
7     private Map<Integer, Long> memo;  
8  
9     public FibonacciDynamic() {  
10         memo = new HashMap<>();  
11     }  
12  
13     @Override  
14     public long calculate(int n) {  
15         if(n == 0){  
16  
17             return 0;  
18         }  
19         if(n <= 1){  
20  
21             return 1;  
22  
23         }  
24         if (memo.containsKey(n)) {  
25             return memo.get(n);  
26         }  
27
```

n < 0 fehlt



```

28      long result = calculate(n -1) + calculate( n -2);
29      memo.put(n, result);
30
31      return result;
32  }
33
34 }
35

```




```
1 package uebung1111;  
2  
3 public class FibonacciParallel extends Fibonacci {  
4     @Override  
5     public long calculate(int n) {  
6         long result;  
7         if(n == 0){  
8             return 0;  
9         }  
10        if(n == 1){  
11            return 1;  
12  
13        }  
14        else{  
15            FibonacciRunner nminus1 = new FibonacciRunner(n-1);  
16            FibonacciRunner nminus2 = new FibonacciRunner(n-2);  
17  
18            nminus1.start();  
19            nminus2.start();  
20  
21            try{  
22                nminus1.join();  
23                nminus2.join();  
24            }  
25            catch( InterruptedException e){  
26  
27            }
```



28

29

30

31

32

33

34

35

36

37 }

38

}

```
result = nminus1.getResult() + nminus2.getResult();  
return result;
```

}



```

1 package uebung1111;
2
3 public class FibonacciRekursive extends Fibonacci {
4
5
6     @Override
7     public long calculate(int n) {
8         long result;
9         if(n == 0){
10
11             return 0;
12         }
13         if(n <= 1){
14             return 1;
15         }
16
17         //Gibt Zahlen in der Reihenfolge des Fibonacci-Baums von links nach rechts
18         //al slong zurück
19         result = calculate(n -1) +calculate( n -2);
20         return result;
21     }
22 }
23

```

```
1 package uebung1111;  
2  
3 import java.util.HashMap;  
4 import java.util.Map;  
5  
6 public class FibonacciDynamicParallel extends Fibonacci {  
7     private Map<Integer, Long> memo;  
8  
9     public FibonacciDynamicParallel() {  
10         memo = new HashMap<>();  
11     }  
12  
13     @Override  
14     public long calculate(int n) {  
15         long result;  
16         if(n == 0){  
17  
18             return 0;  
19         }  
20         if(n >= 1){  
21  
22             return 1;  
23  
24         }  
25         if (memo.containsKey(n)) {  
26             return memo.get(n);  
27         }
```

dynamic?

n < 0 fehlt

```
28     else{
29         FibonacciRunner nminus1 = new FibonacciRunner(n-1);
30         FibonacciRunner nminus2 = new FibonacciRunner(n-2);
31
32         nminus1.start();
33         nminus2.start();
34
35         try{
36             nminus1.join();
37             nminus2.join();
38         }
39         catch( InterruptedException e){
40
41         }
42
43         result = nminus1.getResult() + nminus2.getResult();
44
45         memo.put(n, result);
46     }
47
48     return result;
49 }
50 }
51
```

Wartezeit = 3

$x = \text{BSFPLWB}$

$x_{\text{best}} = \text{BFLPSWB}$

$q(\text{BFLPSWB}) = 4859$

Tabu - Liste:

$\text{BSLPFWB}, 1$

$\text{BSFWPLB}, 2$

$\text{BSFDLWB}, 3$

$N(x) = \text{BSFWLPB}, \text{BFSPLWB}$

$\text{BSFPWL B}, \text{BPFSLWB}, \text{BSPFLWB}$

$\text{BLPFSWB}, \text{BSLPFWB}, \text{BSFLPW B}$

BSWL PFB

$q(\text{BSFWLPB}) = 4847$

4927



$q(\text{BFSPLWB}) = 5076$

5417

$q(\text{BSFPWL B}) = 5464$

$q(\text{BPFSLWB}) = 5743$

$q(\text{BSPFLWB}) = 5535$

$q(\text{BLPFSWB}) = 4642$

$q(\text{BSFLPW B}) = 4864$

$q(\text{BSWL PFB}) = 4526$

~~$q(\text{BSLPFWB})$~~

b) je nachdem, wie die Nachbarschaft definiert ist.

Entweder die erste (4847), die beste (4526) oder zufällig eine auswählen.