Übungsstunde 3

Programm für heute

- Loops (Slides von T. Gross)
- Kurze Wiederholung (Exceptions und Synchronisation) Bsp. von Reto Conconi
- Assignment3 besprechen
  - Fragen besprechen
- Thread management
- Part4 programmieren mit wait() und notifyAll()
- Assignment4 vorbesprechen

Loops – 7 Beispiele

Loop 1

```
System.out.println("Loop 1");
j = 0;
while (j<10) {
    j =  j + 1;
    System.out.print("  " + j);
}
```

- Output: Loop 1

  1 2 3 4 5 6 7 8 9 10

Loop 2

```
System.out.println("Loop 2");
j = 0;
while (j++ < 10) {
    System.out.print("  " + j);
}
```

- Output: Loop 2

  1 2 3 4 5 6 7 8 9 10

Loop 3

```
System.out.println("Loop 3");
j = 0;
while (++j < 10) {
    System.out.print("  " + j);
}
```

- Output: Loop 3

  1 2 3 4 5 6 7 8 9

Loop 4

```
System.out.println("Loop 4");
j = 0;
m = 0;
while (j++  < 10) {
      if ( ++m==j)
            System.out.print(" j= " + j + " m = " +
            m);
}
```

- Output: Loop 4

```
j= 1 m = 1 j= 2 m = 2 j= 3 m = 3 j= 4 m = 4 j= 5
m = 5 j= 6 m = 6 j= 7 m = 7 j= 8 m = 8 j= 9 m = 9
j= 10 m = 10
```

Loop 5

```
System.out.println("Loop 5");
j = 0;
m = 0;
while (j++  < 10) {
      if (m++ == j)
            System.out.print(" j= " + j + " m
            = " + m);
}
```

Output: Loop 5

Loop 6

```
System.out.println("Loop 6");
j = 0;
m = 0;
while (j < 10) {
      if ( j==(++m) )
            System.out.print(" j= " + j + " m =
            " + m);
       j++;
}
```

Output: Loop 6

Loop 7

```
System.out.println("Loop 7");
j = 0;
m = 0;
while (j < 10) {
      if ( m++==j)
            System.out.print(" j= " + j + " m = " +
  m);
            j++;
}
```

- Output: Loop 7

```
j= 0 m = 1 j= 1 m = 2 j= 2 m = 3 j= 3 m = 4 j= 4 m = 5
j= 5 m = 6 j= 6 m = 7 j= 7 m = 8 j= 8 m = 9 j= 9 m = 10
```

Exceptions - Wiederholung

- ExampleRC4
- Was ist der Output?

Exceptions - Lösung

- 1: main begins
- 6: IllegalArgumentException in foo caught
- 7: finally of foo
- 2: IllegalArgumentException of foo in main caught
- 3: IOException of bar in main caught
- 5: main ended

Synchronisation - Wiederholung

- ExampleRC5

- Es gibt 2 Threads. Ausserdem: Klasse ExampleRC5 und Klasse ExampleRC5_2 haben jeweils nur eine Instanz. Sind folgende Situationen möglich? (Pos Thread 1 – Position Thread 2)
- 1 – 1
- 1 – 2
- 1 – 3
- 2 – 3
- 1 – 4
- 5 – 6
- 3 – 5
- 2 – 4
- 2 – 5

Synchronisation - Lösung

- 1 – 1
  No, both need the lock of the instance ExampleRC5
- 1 – 2
  No, because of the same reason
- 1 – 3
  No, because of the same reason (3 needs actually two locks)
- 2 – 3
  No, both need the lock of the instance ExampleRC5
- 1 – 4
  Yes, both use ‘this’ as lock. But these refer to different objects
- 5 – 6
  Yes, because 6 is not synchronized at all
- 3 – 5
  No, they use the same lock
- 2 – 4
  Yes, different locks (like 1 – 4)
- 2 – 5
  Yes, different locks

Assignment3

Erste Zusatzfragen Part2

- Warum genügt es nicht, dass Schlüsselwort ‚synchronized' zu den Methoden read() und write() hinzuzufügen?

- Antwort: Die Synchronisation versichert, dass der Producer und der Consumer die Methoden nicht zur selben Zeit aufrufen. ABER: Es wird nicht verhindert, dass der Producer read() mehrfach ausführt, ohne dass der Consumer etwas produziert hat.

Zweite Zusatzfrage Part2

- Wäre es genug, eine bool‘sche Variable als „Wächter“ in der read() und write() Methode zu benutzen und das synchronized weglassen?

- Antwort: Nein, denn es kann sein, dass der Wert der bool‘schen Variable vom Thread1 verändert wird und der Thread2 diese Änderung nicht sieht (da der Vorgang nicht atomisch ist). Races und Deadlocks können die Folge sein.

Erste Zusatzfrage Part3

- Wäre es genug ein synchronized(this) in der run()-Methode vom Producer und Consumer zu verwenden um das Update des buffers zu bewachen?

- Antwort: Nein, der Producer und der Consumer sind unterschiedliche Objekte. Somit würden die Codeblocks von verschiedenen Monitoren geschützt, was rein gar nichts bringen würde.

Zweite Zusatzfrage Part3

- Welches Objekt dient als Monitor? Also, welches Objekt wird zum synchronisieren verwendet?

- Antwort: Die geteilte Instanz vom UnsafeBuffer. Es gäbe hier auch noch andere Möglichkeiten, in der Mulö wird es aber so gemacht.

Dritte Zusatzfrage Part3

- Was sind die Vor- und Nachteile wenn man die Klassen Producer und Consumer synchronisiert, anstatt den Buffer zu synchronisieren?

- Antwort:
  - Vorteile: Man kann unsichere Buffer Implementationen verwenden (falls man diese z.Bsp. nicht verändern kann)
  - Nachteile: Der synchronisierte Code verteilt sich auf Producer und Consumer. Ausserdem muss man sich überlegen, wie der Producer dem Consumer signalisiert, dass ein Wert anliegt,…

  → Einfacher und weniger fehleranfällig den Buffer zu synchronisieren.

Thread Management

Wait() und notifyAll()

- Organisiert Warteliste (eher Wartemenge) für Threads
- Jedes Objekt besitzt diese Methoden
- Darf nur innerhalb synchronized Methode oder Block stehen

Wie kommt man in die Wartemenge?

- Mit wait()
  - Objekt kann sein Lock mittels wait() temporär aufgeben
  - Wird geweckt durch ein notifyAll() von einem anderen Thread
- Wenn der Thread eine synchronized Methode aufrufen möchte, die bereits „besetzt“ ist

notifyAll()

- Die Methode public final void notifyAll() der Klasse Object benachrichtigt alle Threads aus der Warteliste
- Keine Garantien, welcher Thread tatsächlich vom Scheduler ausgewählt wird

Beispiel

```
public synchronized read() {
        //Wait until empty is false.
        while (empty) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        //Toggle status.
        empty = true;
        //Notify threads that status has changed.
        notifyAll();
        //do something
    }
```

Part4 programmieren

Assignment4

Folgende Folien von Mitra Purandare

MergeSort

Problem: Given a sequence of ‘n’ numbers, arrange them in ascending order.

Example: n=10

Input: 0,5,3,4,55,55,45,77,99,24

Output: 0,3,4,5,24,45,55,55,77,99

How does it work?

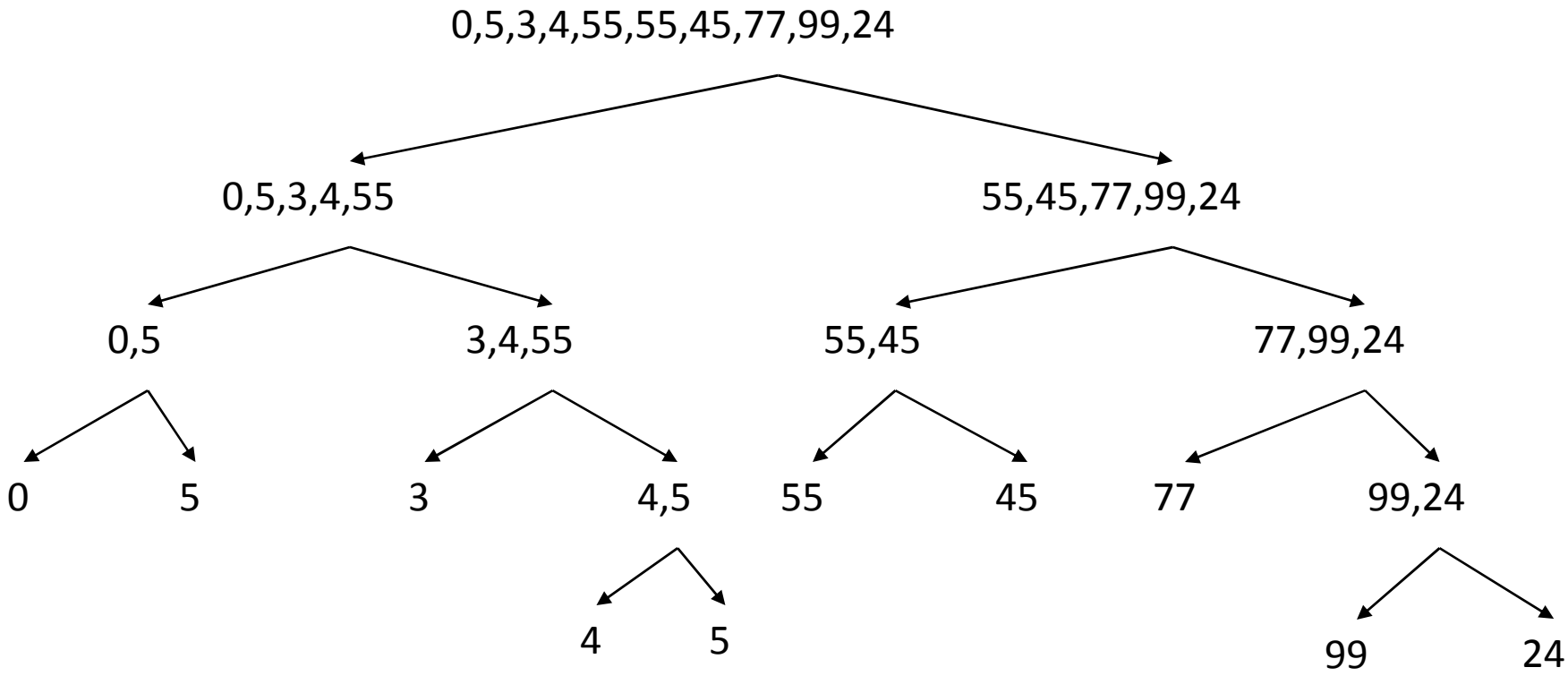
- Divide: Divide the sequence into 2 subsequences of size n/2.
- Conquer: Sort the 2 subsequences recursively using merge sort.
- Combine: Merge the 2 sorted subsequences.

End of Recursion

- Size of the subsequence becomes 1.
- Every 1 element sequence is already sorted.
- If recursion stopped when size of subsequence is > 1, sorting needed!

Proceed to Merging!

Recursive Combine



Merging

- Problem: Combine 2 sorted sequences into one sorted sequence.

- Example:

Input: Sequence 1: 0,5

Sequence 2: 3,4,45

Output: 0,3,4,5,45

How To Merge?

- Example: Seq 1: 0,5 Seq 2: 3,4,45

Create a new sequence A of size 5.

0,5 3,4,45 0<3, insert 0 in A

0,5 3,4,45 3<5, insert 3 in A

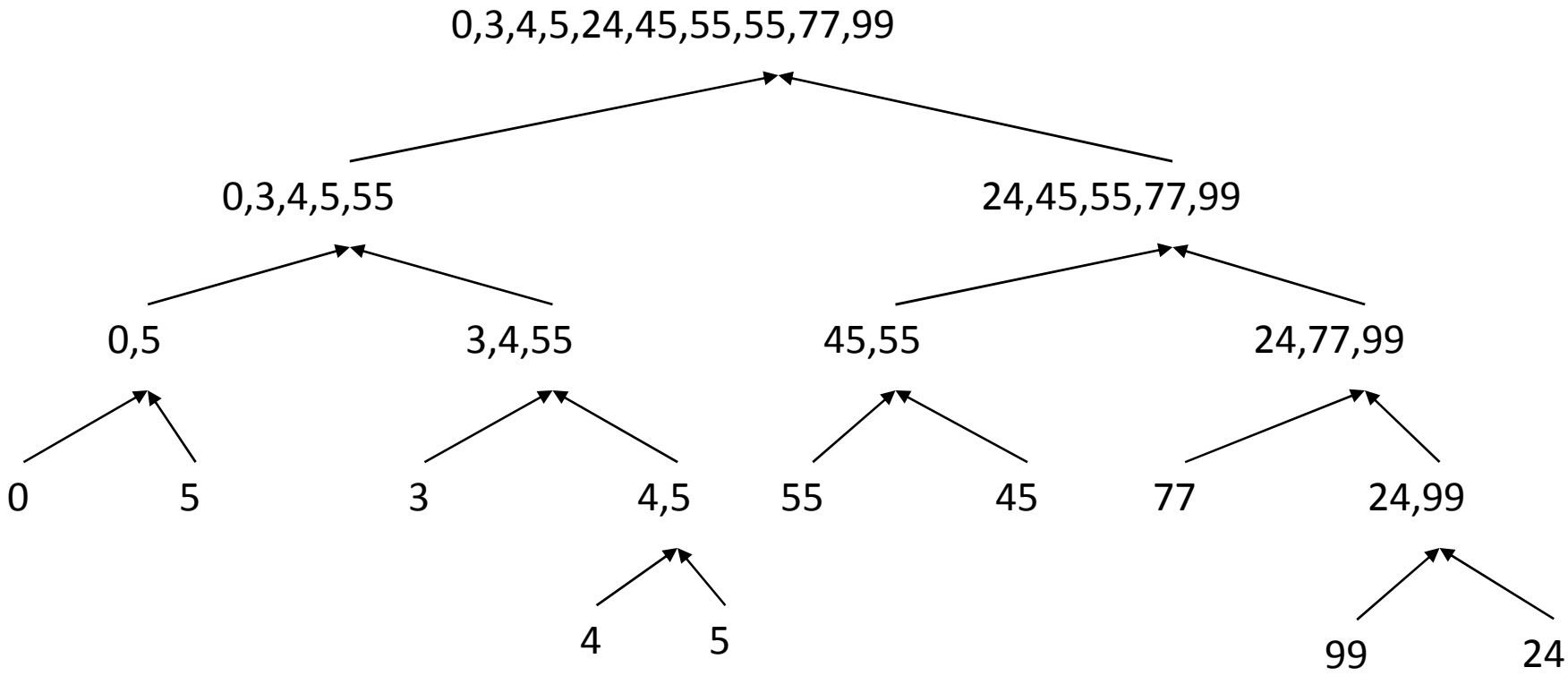
0,5 3,4,45 4<5, insert 4 in A

0,5 3,4,45 5<45, insert 5 in A

End of Seq 1, insert 45 in A

A = 0,3,4,5,45

Merging


0,3,4,5,24,45,55,55,77,99
0,3,4,5,55
24,45,55,77,99
0,5
3,4,55
45,55
24,77,99
0
5
3
4,5
55
45
77
24,99
4
5
99
24

SeqMergeSort

```
mergesort(int[]A, int start, int end) {
If(end-start >= 1) {
        int middle  = (start +end)/2;
        mergesort(A, start, middle);
        mergesort(A, middle+1, end);
        merge(A, start, middle, middle+1, end);
} }
```

Merge

```
Merge(int[]A, int start1, int
    end1, int start2, int end2)
    ){
```

- While(start1 <= end1 && start2 <= end2)

```
{
  If(A[start1] >= A[start2])
    T[i++] = A[start2++]
  else
   T[i++] = A[start1++]
}
```

- If(start1 > end1)
  Append A[start2:end2] to T

- if (start2 > end2)
  Append A[start1:end1] to T

- Copy T back to A at resp. places

} //end Merge

What we want to do?

- Some sort operations are independent of others
- Some merge operations are independent of others
- Exploit parallelism
- Implement a multithreaded merge sort

MultiThreaded Merge Sort

- Input: number of elements to sort (m) and number sort threads (n)
- Generate m random numbers
- For m=10, n=4
- Seq: 0,5,3,4,55,55,45,77,99,24
- Divide Seq in 4 subsequences

  Seq1: 0,5 Seq2: 3,4,

  Seq3: 55,55 Seq4: 45,77,99,24

MergeSort

4 Sort threads

0,5, 3,4, 55,55, 45,77,99,24

Sort

0,5 3,4 55,55 24,45,77,99

Merge Subsequences

0,3,4,5 24,45,55,55,77,99

Merge Further

0,3,4,5,24,45,55,55,77,99

Parallelism

- Parallelism in Sorting:

  Seq1, Seq2, Seq3 and Seq4 can be sorted at the same time.
- Parallelism in Merging

  merge(Seq1, Seq2) and merge(Seq3, Seq4) can be done at the same time.
- Constraint

  merge(Seq1, Seq2) can not be done before sort(Seq1) and sort(Seq2) are done.