Pitfalls in Multi-threading programming

Islam Khamis Galal Omar Ahmed
575449

Veranstaltung: B56 Projektstudium (P) - Big Data Management und Analytics | Dozent: Herr. Prof. Dr. Patrick Fuhrmann

Table of contents

Why is Multithreading Important?[1]

Multithreading allows the execution of multiple parts of a program at the same time.(Responsiveness)

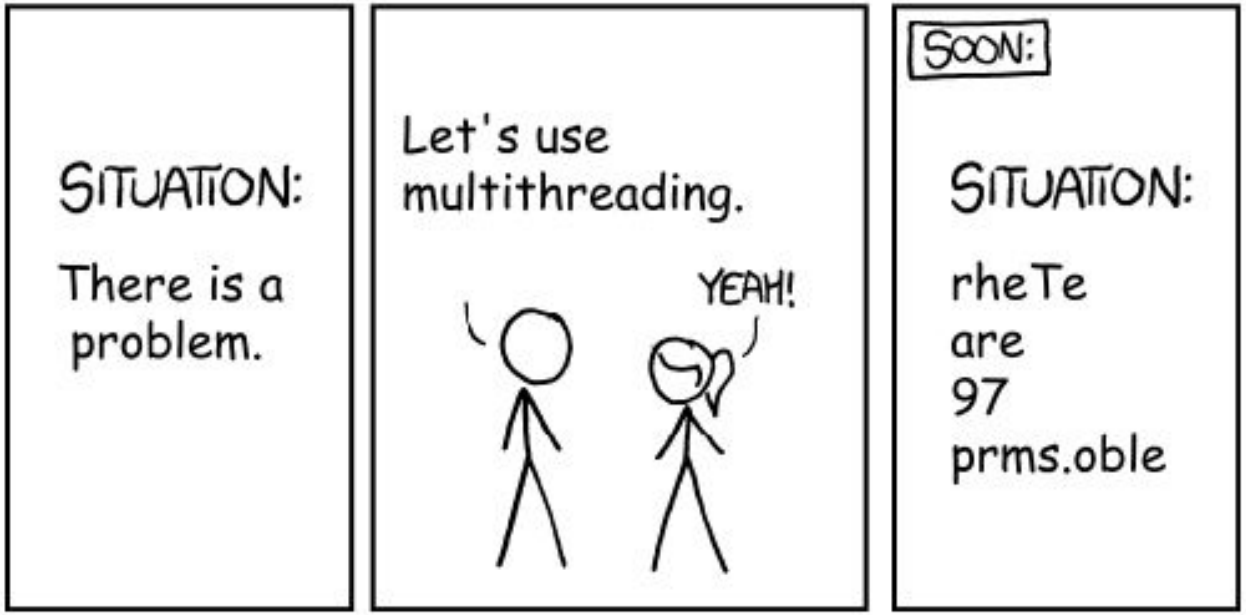
Threads are lightweight processes.

Threads share the same resources.(Economy)

Maximum utilization of the CPU by multitasking.

Quelle [1] https://www.tutorialspoint.com/the-benefits-of-multithreaded-programming

Pitfalls in Multi-threading programming



Quelle https://i.imgr.com/RbelQBq.png

Race Condition

```
public class SynchronizedCounter extends Thread {
     private static int cnt = 0;

     private static int inc() {
        int c=cnt;
        c=c+1;
        cnt=c;
        return cnt;
     }

     public void run() {
       while (5>cnt) System.out.println(this.getName()+" "+inc());
     }

     public static void main(String[] args) {
      SynchronizedCounter c1 = new SynchronizedCounter();
      SynchronizedCounter c2 = new SynchronizedCounter();
      c1.start();
      c2.start();
     }
}
```

Race Condition[2]

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Figure 1 https://en.wikipedia.org/wiki/Race_condition

Race Condition

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

Quelle https://en.wikipedia.org/wiki/Race_condition

⚠️ **The final value is 1 instead of the expected result of 2**

Race Condition

Two threads simultaneously have access to a global variable.

Difficult to determine in multithreaded programs because a program can often run correctly one, two, or even hundreds of times without ever running into a problem.

Race Condition[3]

Java offers a mechanism to avoid race conditions by synchronizing thread access to shared data.

The synchronized keyword can be used on different levels: Instance methods, Static methods, and Code blocks.

All synchronized blocks of the same object can have only one thread executing them at the same time.

Quelle [3]https://www.baeldung.com/java-synchronized

Avoid Race Condition

```
public class SynchronizedCounter extends Thread {
     private static int cnt = 0;

     private static synchronized int inc() {
        int c=cnt;
        c=c+1;
        cnt=c;
        return cnt;
     }

     public void run() {
       while (5>cnt) System.out.println(this.getName()+" "+inc());
     }

     public static void main(String[] args) {
      SynchronizedCounter c1 = new SynchronizedCounter();
      SynchronizedCounter c2 = new SynchronizedCounter();
      c1.start();
      c2.start();
     }
}
```

Deadlock in Real life

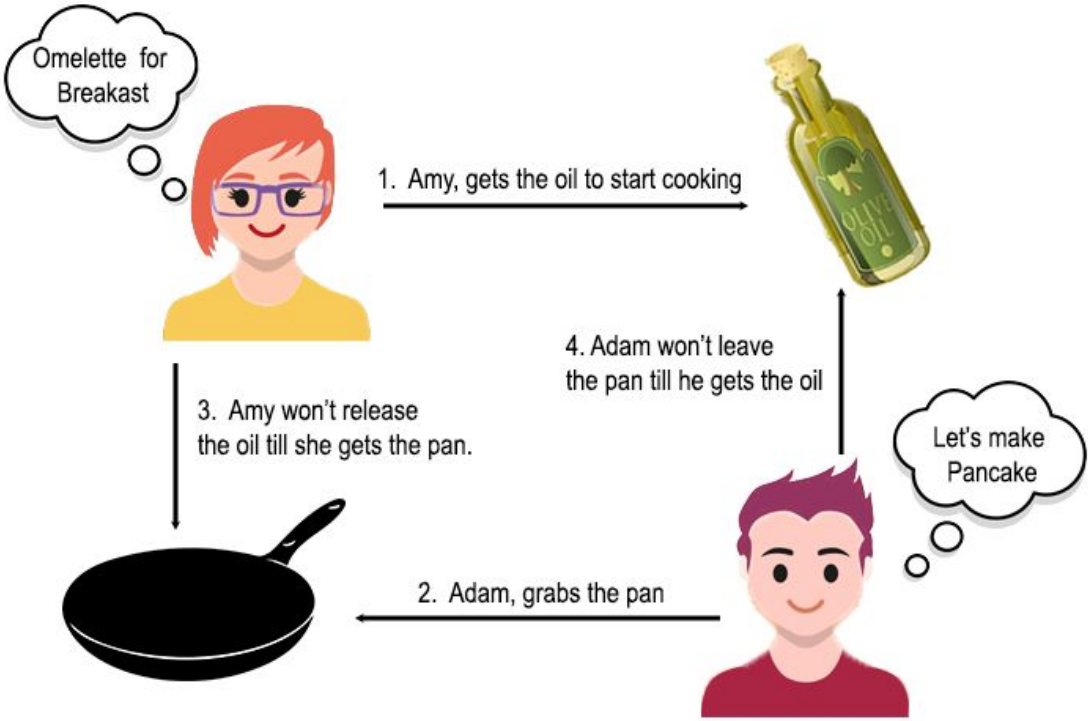


Figure 2 [4]https://www.cleantutorials.com/jconsole/deadlock-example-and-how-to-detect-it-using-jconsole

Deadlock in OS

A set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Deadlock Example	Thread A	Thread B	Resources
Time 1	Accesses 1 & 2	Accesses 3 & 4	Resource 1
Time 2	Attempts to Access 3	Denies Access to 3	Resource 2
Time 3	Denies Access to 1	Attempts Access to 1	Resource 3
Time 4	Try Again (Deadlock)	Try Again (Deadlock)	Resource 4

Quelle [5]https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/
[6]https://austingwalters.com/multithreading-common-pitfalls/

Necessary Conditions for deadlock

Mutual Exclusion: One or more than one resource are non-shareable (Only one process can use at a time)

Hold and Wait: A process is holding at least one resource and waiting for resources.

No Preemption: A resource cannot be taken from a process unless the process releases the resource.

Circular Wait: A set of processes are waiting for each other in circular form.

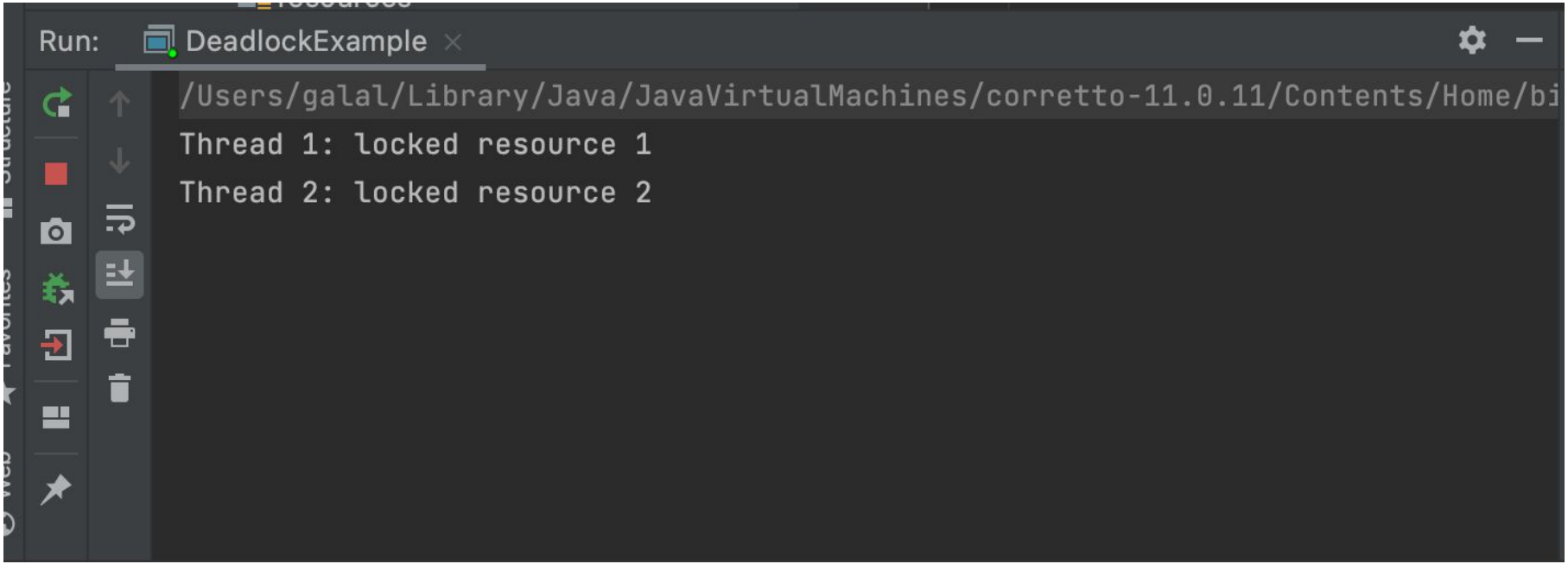
Quelle [5] https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/

Deadlock example

```
public static void main(String[] args) {
   final String resource1 = "R1";
   final String resource2 = "R2";

   Thread t1 = new Thread() {
       public void run() {
           synchronized (resource1) {
               System.out.println("Thread 1: locked
                                        resource 1");
               try { Thread.sleep(100);}
               catch (Exception e) {}
               synchronized (resource2) {
                   System.out.println("Thread 1: locked
                                            resource 2");
               }
           }
       }
   };
```

```
Thread t2 = new Thread() {
       public void run() {
           synchronized (resource2) {
               System.out.println("Thread 2: locked
                                        resource 2");
               try { Thread.sleep(100);}
                 catch (Exception e) {}

               synchronized (resource1) {

                   System.out.println("Thread 2: locked
                                        resource 1");
               }
           }
       }
   };

   t1.start();
   t2.start();
}
```

Console output



Detect Deadlock

```
public static void main(String[] args) {
   final String resource1 = "R1";
   final String resource2 = "R2";

   Thread t1 = new Thread() {
       public void run() {
           synchronized (resource1) {
               System.out.println("Thread 1: locked
                                                resource 1");
               try { Thread.sleep(100);}
               catch (Exception e) {}
               synchronized (resource2) {
                   System.out.println("Thread 1: locked
                                                 resource 2");
               }
           }
       }
   };
```

```
Thread t2 = new Thread() {
       public void run() {
           synchronized (resource2) {
               System.out.println("Thread 2: locked
                                                resource 2");
               try { Thread.sleep(100);}
               catch (Exception e) {}
               synchronized (resource1) {
                   System.out.println("Thread 2: locked
                                                resource 1");
               }
           }
       }
   };

   t1.start();
   t2.start();
}
```

⚠️ Nested synchronized block

How to avoid deadlock in Java?

- Avoid Unnecessary Locks
- Avoid Nested Locks (avoid giving a lock to multiple threads)
- Be careful with using Thread.join() Method
  - use join with maximum time
- Use Lock Ordering
- Lock Time-out

Quelle[7] https://www.javatpoint.com/how-to-avoid-deadlock-in-java

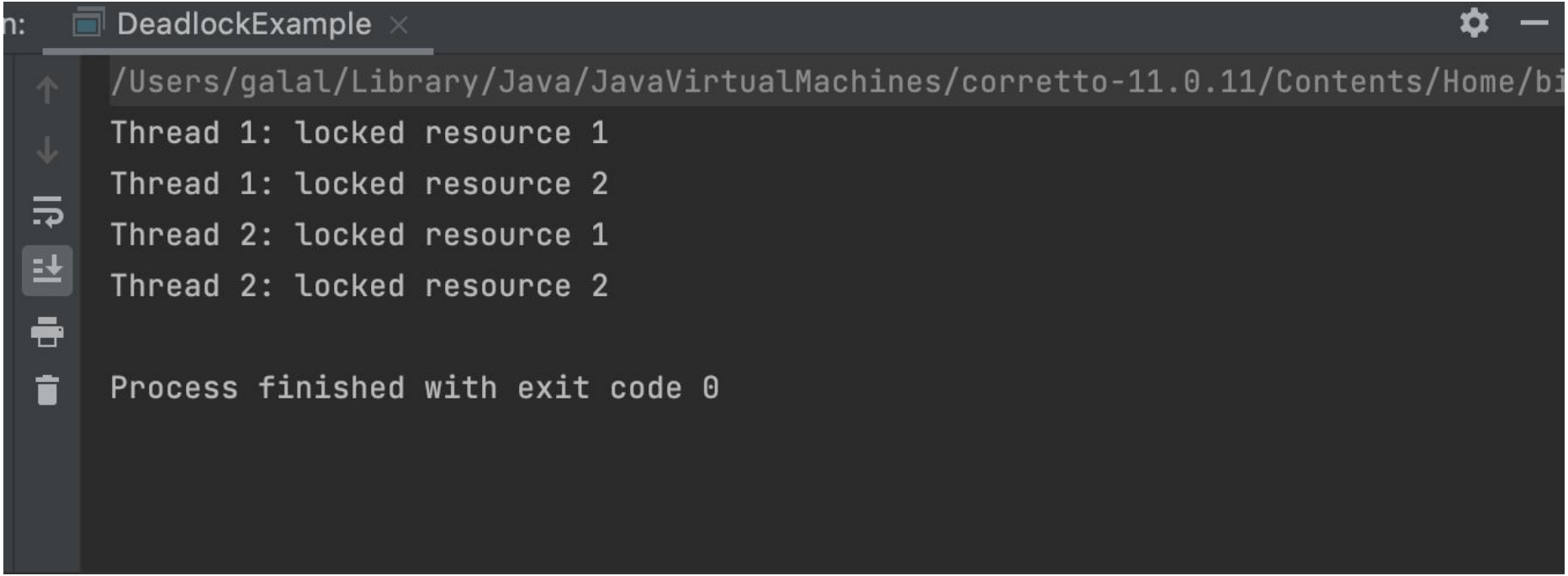
Avoid Deadlock

```
public static void main(String[] args) {
   final String resource1 = "R1";
   final String resource2 = "R2";

   Thread t1 = new Thread() {
       public void run() {
           synchronized (resource1) {
               System.out.println("Thread 1: locked
                                                resource 1");
               try { Thread.sleep(100);}
               catch (Exception e) {}
               synchronized (resource2) {
                   System.out.println("Thread 1: locked
                                                 resource 2");
               }
           }
       }
   };
```

```
Thread t2 = new Thread() {
       public void run() {
           synchronized (resource1) {
               System.out.println("Thread 2: locked
                                                resource 1");
               try { Thread.sleep(100);}
               catch (Exception e) {}
               synchronized (resource2) {
                   System.out.println("Thread 2: locked
                                                resource 2");
               }
           }
       }
   };

   t1.start();
   t2.start();
}
```

🎉 Lock Ordering

Console output



Pitfalls in Multi-threading programming

Fragen?

Quelle kohlenhydrate-tabellen.com/5-fragen-vor-einer-diaet/

Reference

1. https://www.tutorialspoint.com/the-benefits-of-multithreaded-programming
2. https://en.wikipedia.org/wiki/Race_condition
3. https://www.baeldung.com/java-synchronized
4. https://www.cleantutorials.com/jconsole/deadlock-example-and-how-to-detect-it-using-jconsole
5. https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/
6. https://austingwalters.com/multithreading-common-pitfalls/
7. https://www.javatpoint.com/how-to-avoid-deadlock-in-java