



AARHUS UNIVERSITET

Software Engineering and Architecture

Concurrency

A Classic Java Take on it



Disclaimer...

- In Danish "Tomme tønder buldrer mest..." 😊
- I have *(almost) no experience* in large scale, realistic, development of parallel and concurrent programs ☹️
- The 'handling concurrency' scene is a vast topic, and has transformed considerably over the last decade!
 - Multicore processors
 - And OS/Libraries to take advantage of them !
- **We will only treat *classic and basic issues and solutions!***
 - So, read up on the material once you are 'out there'...
 - The problems are the same, but solutions become better...




- Concurrency = many 'objects' executing at the *same* time
- Why?
- Modelling: This is how the world is!
 - Many people working in parallel, collaborating, sharing...
- Quality Attributes of our architecture
 - Performance
 - Responsiveness / Availability

Responsiveness

- *Sometimes* our computations take quite a while to complete
- Example:
 - User 1 searches for all flights to Bali
 - Server is busy requesting a lot of external booking systems
 - *Meanwhile*
 - User 2 wants to search for flights to Tokyo
- *But what happens here?*

```
while(! isStopped()){
    Socket clientSocket = null;
    try {
        clientSocket = this.serverSocket.accept();
    } catch (IOException e) {
        if(isStopped()) {
            System.out.println("Server Stopped.");
            return;
        }
        throw new RuntimeException(
            "Error accepting client connection", e);
    }
    try {
        processClientRequest(clientSocket);
    } catch (Exception e) {
        //log exception and go on to next request.
    }
}
```





- Just like one cashier in Føtex can only handle a limited number of customers at the same time; so can a single thread
- Solution: *Employ more cashiers / threads!*
- *However... Poses its own set of challenges!*

Analyzing Code...

- ... is based upon a *sequential execution of statements*

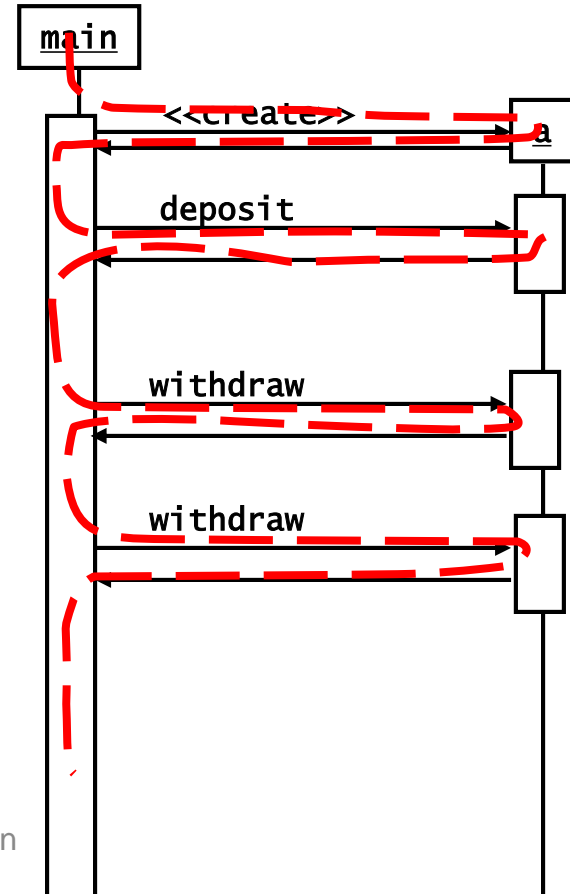
```
interface Account {  
    public boolean deposit(long amount);  
    public boolean withdraw(long amount);  
    public long getBalance();  
}
```

```
class SingleThread {  
    public static void main(String[] args) {  
        Account a = new AccountImpl();  
  
        a.deposit(500);  
        a.withdraw(100);  
        a.withdraw(100);  
    }  
}
```

- Exercise: What is 'a.balance()' after the last withdraw()?
 - Assuming the balance is 0.00 at the start...

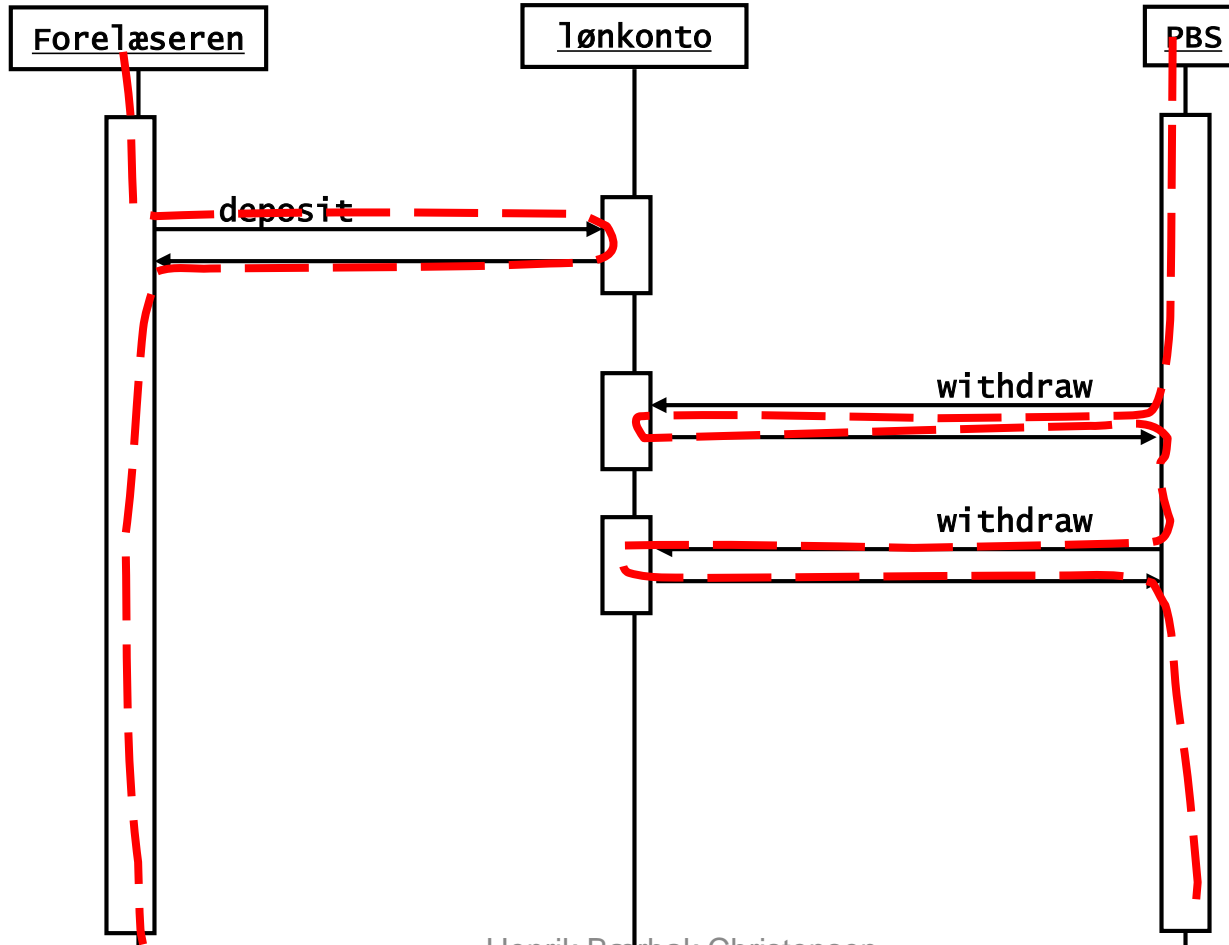
Program Thread...

- The *program thread* weaves through methods and statements...
- In machine code
 - Register PC
 - Program counter
 - Increments for every instruction meet
 - Some instructions change PC
 - JMP 47 =
 - Change PC to address 47
 - I.e. a method call...





Two Threads!





Three Types of Concurrency

- When more than one thread executes in a program, we say that it is *concurrently executed*, it is a *concurrent* program.
- Three categories of concurrent programs
 - Independent threads
 - Like running your Media player program while coding in IntelliJ
 - Shared resources
 - Like two threads reading/writing to the *same* account object
 - Collaborating threads
 - Like one thread inserting into a buffer and assuming some other thread will remove those items from the buffer

- Java was one of the first mainstream languages to have threads as part of the language!

- Before that,
it was the job of the OS
 - *Processes*
- *Coded using OS libraries*

- Core class:
 - Thread

```
class ThreadDemo {
    public static void main(String[] args) {
        Thread a = new OutputThread('a');
        Thread b = new OutputThread('b');
        a.start(); b.start();
    }
}

class OutputThread extends Thread {
    private char c;

    OutputThread(char outputChar) {
        c = outputChar;
    }

    public void run() {
        for (int i=0; i<100; i++) {
            System.out.print(c); System.out.flush();
        }
    }
}
```

- Anatomy
 - Create
 - Call start();
 - Will run 'run()'
- Exercise
 - How many threads?
 - What does it do?
 - *And what is the output?*

```
class ThreadDemo {
    public static void main(String[] args) {
        Thread a = new OutputThread('a');
        Thread b = new OutputThread('b');
        a.start(); b.start();
    }
}

class OutputThread extends Thread {
    private char c;

    OutputThread(char outputChar) {
        c = outputChar;
    }

    public void run() {
        for (int i=0; i<100; i++) {
            System.out.print(c); System.out.flush();
        }
    }
}
```



- The hallmark of *concurrent programs*: ***non-determinism***

```
PS D:\work\teaching\SWEA-E17\code\lab\threads\thread-demo1> java ThreadDemo
aaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
PS D:\work\teaching\SWEA-E17\code\lab\threads\thread-demo1> java ThreadDemo
aaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbaabbbbbbbbbbbbbbbbbb
baaaaabbbaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
PS D:\work\teaching\SWEA-E17\code\lab\threads\thread-demo1> java ThreadDemo
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaabbaabbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
PS D:\work\teaching\SWEA-E17\code\lab\threads\thread-demo1>
```

- *Welcome to debugging hell!!!*
- *Welcome to testing hell!!!*

- Threads execute concurrently
 - Abstractly speaking, even if they do not always in practice !
- In my youth we had *one CPU*
 - Today you 4, 8, 12, ..., and several thousands in your GFX card
- Concurrency is (partly) *simulated* by
- *Thread scheduling*
 - *Preemptive:*
 - *Thread runs for n milliseconds, is interrupted and the scheduler then picks the next thread for execution*
 - *(Non-preemptive): Thread 'yields()' to signal thread change...*

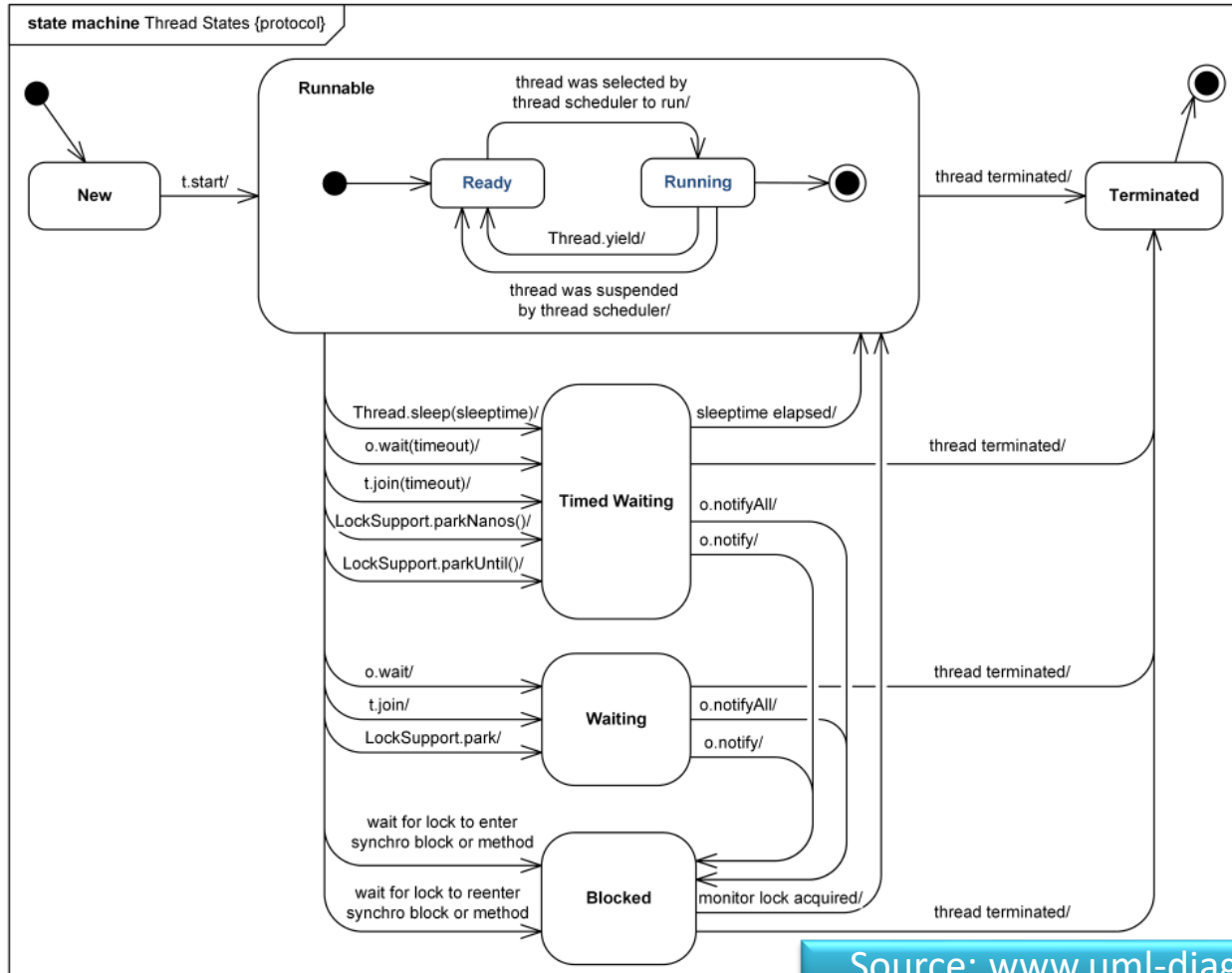


Thread States

- Any thread in a program are in one of several *states*
- *An incomplete list for Java includes*
 - *RUNNABLE: running or able to run ('running'/'ready')*
 - 100 threads may be runnable but only 1 [2, 4, 8] are *actually* executing code, the others are waiting for the scheduler to switch to them
 - *BLOCKED: not executing, but waiting for a lock*
 - Used to handle 'shared resources', see later...
 - *WAITING: not executing, but in a wait-set, waiting*
 - Used to handle 'producer-consumer' / collaborating threads



Thread States



Subclassing? No no no 😊

- *Program to an interface!*
- Process
 - Provide Thread object with the Runnable instance
- Exercise:
 - What design pattern?

Runnable interface

```
public class ThreadDemo2 {
    public static void main(String[] args) {
        Thread a = new Thread(new OutputRunnable('a') );
        Thread b = new Thread(new OutputRunnable('b') );
        a.start(); b.start();
    }
}

class OutputRunnable implements Runnable {
    private char c;
    OutputRunnable(char outputchar) {
        c = outputchar;
    }
    public void run() {
        for (int i=0; i<100; ++i) {
            System.out.print(c); System.out.flush();
        }
    }
}
```




Overviewing Threads

- Oracle Java comes with JVisualVM

The screenshot shows the JVisualVM 1.3.3 interface. The main window displays the 'Threads' tab for the application 'cloud.cave.main.CaveDaemon (pid 6441)'. The 'Threads visualization' checkbox is checked. The 'Live threads: 22' and 'Daemon threads: 10' are shown. The 'Timeline' view is active, showing a grid of thread activity from 2:30 to 2:40. The threads listed are pool-1-thread-1 through pool-1-thread-10. The legend indicates Running (green), Sleeping (blue), Wait (yellow), and Monitor (red). A red arrow points from the left sidebar to the Threads tab.

Threads	2:30	2:40	[ms]
pool-1-thread-1	Wait	Wait	
pool-1-thread-4	Wait	Wait	
pool-1-thread-3	Wait	Wait	
pool-1-thread-2	Wait	Wait	
pool-1-thread-10	Wait	Wait	
pool-1-thread-9	Wait	Wait	
pool-1-thread-8	Wait	Wait	
pool-1-thread-7	Wait	Wait	
pool-1-thread-6	Wait	Wait	
pool-1-thread-5	Wait	Wait	