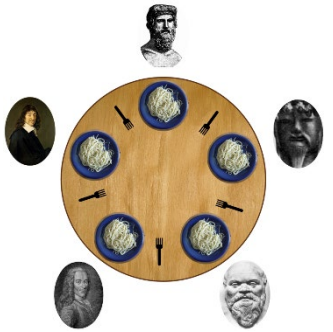
Objektorientierte Programmierung

Hochschule Bochum
WS 19/20
Dr.-Ing. Darius Malysiak

IV. Design Patterns



https://de.wikipedia.org/wiki/Philosophenproblem#/media/File:An_illustration_of_the_dining_philosophers_problem.png

Concurrency Patterns: Parallel-Muster

Dinning Philosophers Problem:

- 5 Philosophen sitzen am Tisch.
- Jeder handelt wie folgt: Denken, Essen, Denken, Essen, ….
- Beim Essen nimmt der jeweilige Philosoph zunächst die Gabel links von ihm, dann jene rechts von ihm.
- Zwischen jedem Philosoph liegt genau eine Gabel. -> Somit 5 Gabeln und leider eine zu wenig damit mindestens ein Philosoph essen kann. ☹
- Nach dem Essen legt der Philosoph die Gabeln wieder zurück, erst die rechte Gabel und dann die linke Gabel!

Was ist die triviale Lösung dieses Problems?

IV. Design Patterns

Concurrency Patterns: Parallel-Muster

Definition: **Semaphore**

- Eine Datenstruktur zur Kontrolle des parallelen Zugriffs auf ein Element.
- Besitzt einen internen Zähler.
- Besitzt jeweils eine Methode zum Inkrementieren bzw. Dekrementieren des Zählers.
- Blockiert den Aufrufer falls er dekrementieren möchte und der Zähler bei 0 ist.

Entwickelt von Edsger Dijkstra 1962-1963

Definition: Mutex

- Eine Semaphore, welche ihren Zähler nur bis 1 inkrementieren kann.

IV. Design Patterns

Concurrency Patterns: Parallel-Muster

https://de.wikipedia.org/wiki/Philosophenproblem#/media/File:An_illustration_of_the_dining_philosophers_problem.png

```
public class Philosopher extends Thread{

    final static int SLEEP_TIME_IN_MS = 300;
    final static int MAX_ITERATIONS = 10;
    private String name;
    private Fork leftFork;
    private Fork rightFork;


    public Philosopher(String name,Fork leftFork,Fork rightFork)
    {
        this.name = name;
        this.leftFork = leftFork;
        this.rightFork = rightFork;
    }

    @Override
    public void run()
    {
        IntStream.range(0,MAX_ITERATIONS).forEach( i -> {
            try {
                think();
                eat();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
    }

    ..................
}
```

IV. Design Patterns

Concurrency Patterns: Parallel-Muster

https://de.wikipedia.org/wiki/Philosophenproblem#/media/File:An_illustration_of_the_dining_philosophers_problem.png

```
public class Philosopher extends Thread{

    .....

    private void think() throws InterruptedException
    {
        System.out.println(name+" is thinking!");
        Thread.sleep(SLEEP_TIME_IN_MS);
    }

    private void eat() throws InterruptedException
    {
        leftFork.get();
        rightFork.get();
        System.out.println(name+" is eating!");
        leftFork.layBack();
        rightFork.layBack();
    }
}
```

IV. Design Patterns

Concurrency Patterns: Parallel-Muster

https://de.wikipedia.org/wiki/Philosophenproblem#/media/File:An_illustration_of_the_dining_philosophers_problem.png

```
public class Fork {
    Semaphore m = new Semaphore(1);

    public Fork()
    {
        m.release();
    }

    public void get() throws InterruptedException
    {
        m.acquire();
    }

    public void layBack()
    {
        m.release();
    }
}
```

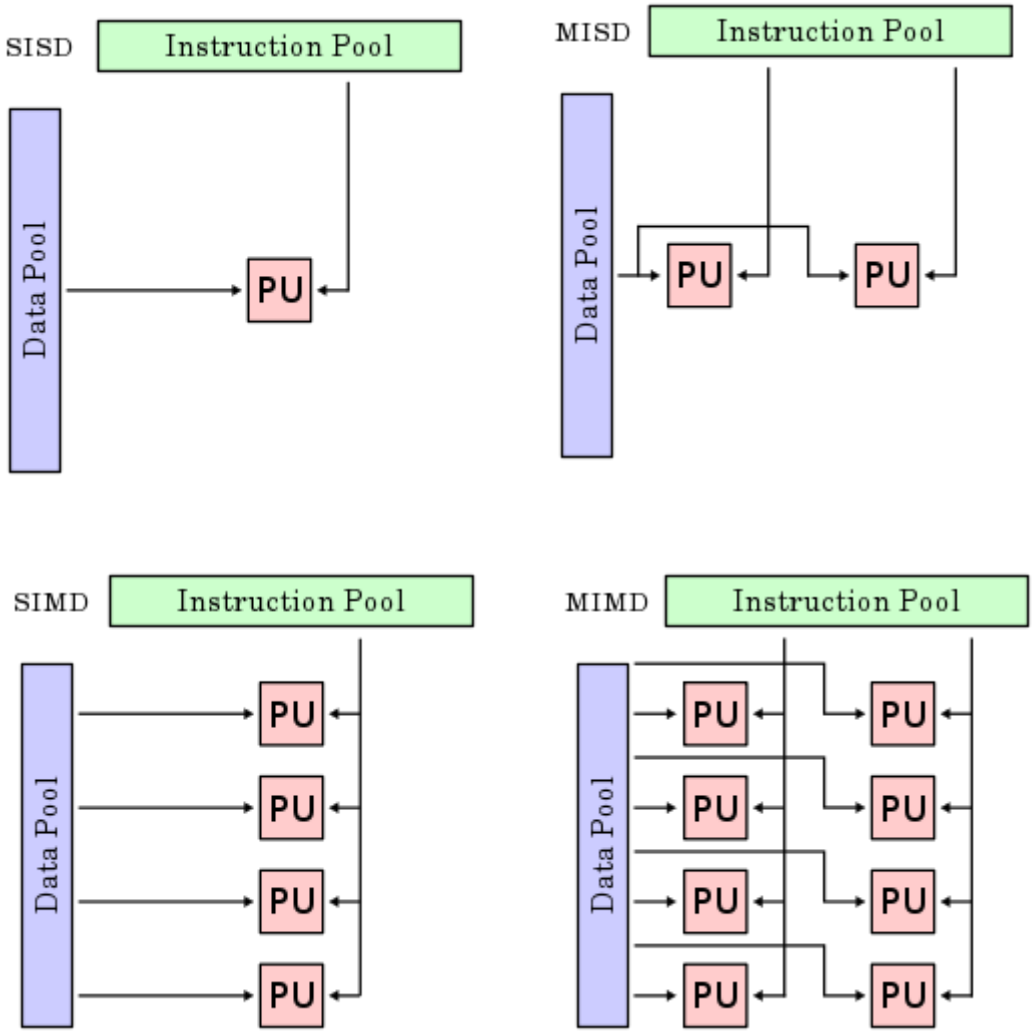
Inkrementieren

Dekrementieren

IV. Design Patterns

Flynn‘s Taxonomie:

- Klassifikation von Computersystemen im Hinblick auf parallele Verarbeitung.
- Klassische x86 Systeme waren häufig SISD Architekturen.
- Moderne x64 Multicore Systeme sind häufig MIMD Architekturen.
- GPU‘s arbeiten als SIMD Architekturen.

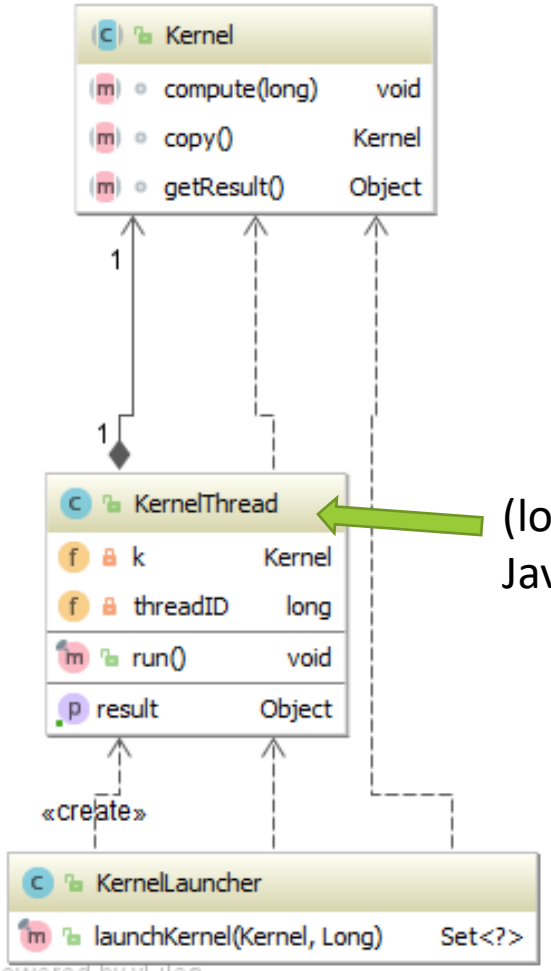


https://upload.wikimedia.org/wikipedia/commons/thumb/a/ae/SISD.svg/250px-SISD.svg.png

IV. Design Patterns

Compute Kernel – Pattern (Single Instruction Multiple Threads ‚SIMT'): Parallel-Muster

Software Komplement zu SIMD



(lokaler) Java Thread

Grundidee:

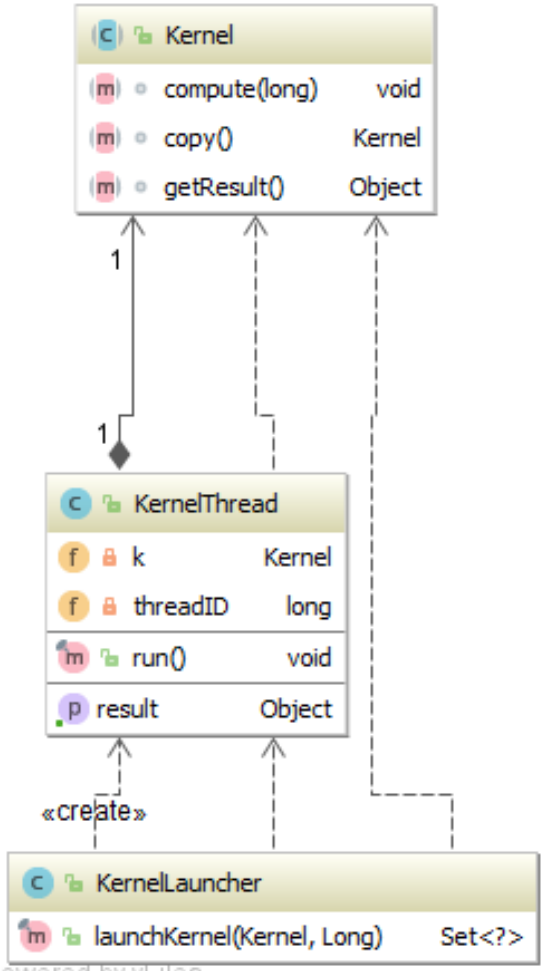
- Jeweils ein KernelThread führt genau einen Kernel aus.
- Die Methode ‚compute' erhält eine Thread ID von 0 beginnend.
- Die Methode ‚launchKernel' bekommt als Parameter:
  - Einen Kernel als Muster, welcher anschließend kopiert wird.
  - Die Anzahl der zu startenden Threads.
- Die Methode ‚launchKernel' sammelt die Resultate jedes Kernels und gibt diese als Set zurück.

- Verallgemeinert müssen die KernelThread Instanzen nicht lokal laufen.

IV. Design Patterns

Compute Kernel – Pattern (Single Instruction Multiple Threads ‚SIMT'): Parallel-Muster

Software Komplement zu SIMD



Vorteile:

- Einfache Möglichkeit Workloads für eine parallele Verarbeitung zu verteilen.
- Ausführungs-Art sowie -Ort sind transparent durch die Implementierung von ‚Kernelthread' -> Einfacher Austausch möglich z.B. durch Erweiterung des Interfaces von ‚KernelLauncher'.

Nachteile:

- Code muss häufig angepasst werden um parallel berechnet zu werden.
- Overhead zur (Daten/Thread)-Synchronisation kann schnell einen Flaschenhals darstellen.
- Anzahl von lokalen Threads ist stark begrenzt für hochparallele Algorithmen.

IV. Design Patterns

Local-Max Problem:

Bsp: n=3

3 6 5 6 2 3

Gesucht: Maxima in adjazenten sowie disjunkten Array-Segmenten der Größe **n** eines Arrays **a**.

Parameter:

- Größe der Segmente **n**
- Array **a** der Werte

V. OOP Software Engineering

Maven:

https://upload.wikimedia.org/wikipedia/commons/thumb/0/0b/Maven_logo.svg/220px-Maven_logo.svg.png

- Werkzeug zur Konfiguration von Software Projekten.
- Unabhängig von IDE!
- Unabhängig von Betriebssystem!
- Verwaltet Build-Pipelines.
- Verwaltet externe Abhängigkeiten: z.B.
  - Java Module werden in gewünschter Version heruntergeladen.
  - Module werden in die jeweiligen Javac Aufrufe integriert.
- Funktionalität kann durch Plugins erweitert werden.
- Konfiguration durch XML Datei namens ‚pom.xml' (Project Object Model).
- Erste Version 2004, aktuelle Version 3.6.0 (10/2018).
- De facto Standard für Java Projekte.
- Inoffizieller Vorgänger / Parallelprojekt / Alternative: ANT
- Inoffizieller Nachfolger / Parallelprojekt / Alternative: Gradle.
- IDE's bieten häufig Maven-Integration an.

V. OOP Software Engineering

Maven:

Struktur einer einfachen POM File:

XML Header

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>playground</groupId>
    <artifactId>playground</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>RELEASE</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
</project>
```

Artifact ID (Koordinaten)

Externe Abhängikeiten

V. OOP Software Engineering

Maven:

Erzeugen eines Maven Projektes:

```
mvn archetype:generate "-DgroupId=com.oop2018" "-DartifactId=helloworld"
        "-DarchetypeArtifactId=maven-archetype-quickstart" "-DarchetypeVersion=1.3" "-DinteractiveMode=false"
```

Kompilieren eines Maven Projektes:

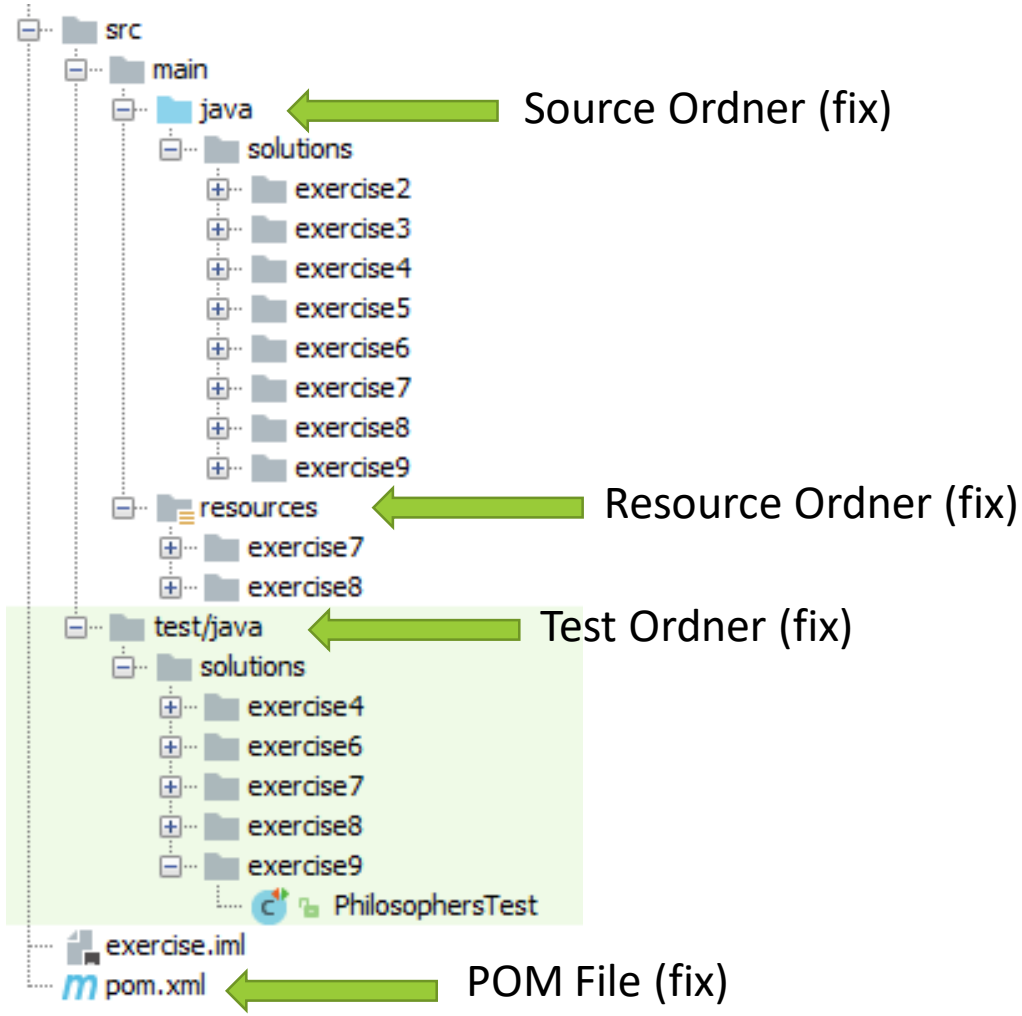
```
mvn compile
```

Packen eines Maven Projektes:

```
mvn package
```

V. OOP Software Engineering

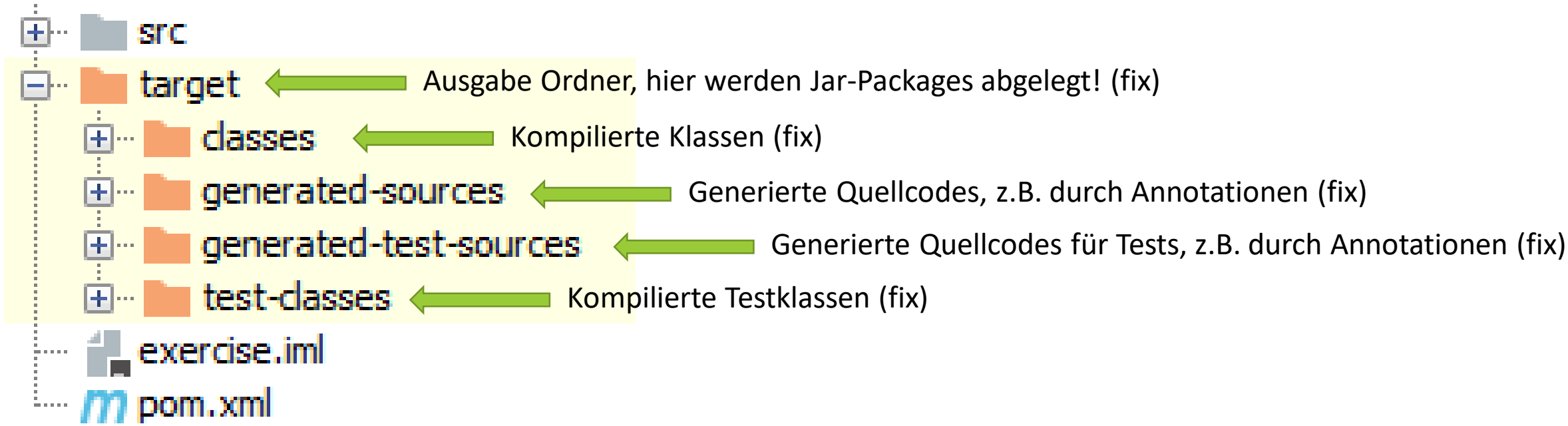
Maven: Struktur eines Maven Java Projekts:



- Maven definiert für Java Projekte eine Grundstruktur.
- Dies ist ein Beispiel für *Convention over Configuration.*
- Die Projektstruktur kann nachträglich umkonfiguriert werden. Hierfür sind aber sehr gute Gründe nötig!

V. OOP Software Engineering

Maven: Struktur eines Maven Java Projekts:



IV. Design Patterns

Nächste Vorlesung:

Maven Teil 2, Lombok, Hibernate, …