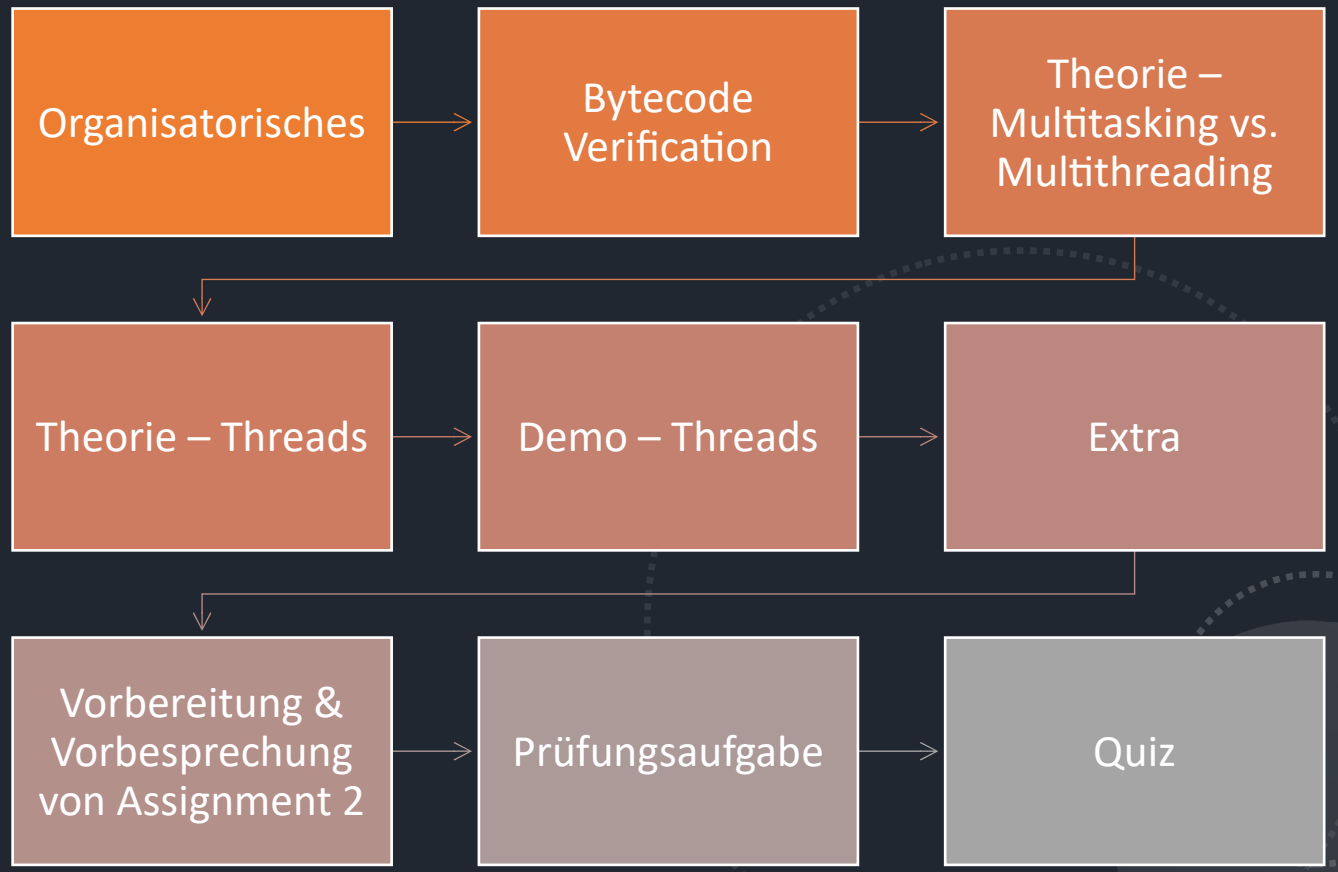
Parallel Programmierung Übungsstunde 2

FS2024

Plan für heute



Organisatorisches

Wie erreicht ihr mich?

- rlarisch@ethz.ch
- https://www.n.ethz.ch/~rlarisch

Bytecode Verification

Bytecode Verification

Java Source Code -> Java Bytecode –> Machine Code

Frage: Warum wird der Bytecode nochmals überprüft?

Sicherheit, da der Bytecode von einer anderen Quelle kommen kann.

Damit der Interpreter (der die JVM implementiert) schneller laufen kann, da er Garantien vom Bytecode Verifier bekommt, z.B. Alle Parameter haben korrekten Typ, Object Field accesses sind legal, etc.

Quelle: https://www.oracle.com/java/technologies/security-in-java.html#:~:text=The%20bytecode%20verifier%20acts%20as,has%20passed%20the%20verifier's%20tests

Theorie – Multitasking vs. Multithreading

Multitasking vs. Multithreading

Multitasking

Ausführung von mehreren Prozessen (processes).

Gemanaged vom Betriebssystem (OS).

Jeder Prozess hat seinen eigenen Kontext (context): address space, program counter (PC), register values, stack, heap, etc.

Das Betriebssystem bestimmt welche Prozesse laufen und teilt die Ressourcen unter den Prozessen auf (Siehe: context-switching)

Multitasking vs. Multithreading

Multithreading

Ausführung mehrerer Threads innerhalb eines OS-Prozess.

Diese Threads teilen sich die Ressourcen des OS-Prozess wie den Adressraum (address space). Das macht die Programme komplizierter.

Theorie- Threads

Was ist ein Thread?

«Eine unabhängige Berechnungseinheit, die parallel ausgeführt werden kann. Das Konzept von Threads existiert auf mehreren Stufen: Hardware, OS, Programmiersprachen.»

Ein Thread ist wie ein sequentielles Programm, aber ein Thread kann neue Threads kreieren, die wiederum neue Threads kreieren…

In Java: Eine Instanz der Klasse Thread

Parallelism vs. Concurrency?

Parallelism: Mehrere Aufgaben zur gleichen Zeit erledigen

Concurrency: Mehrere Aufgaben gleichzeitig bewältigen.

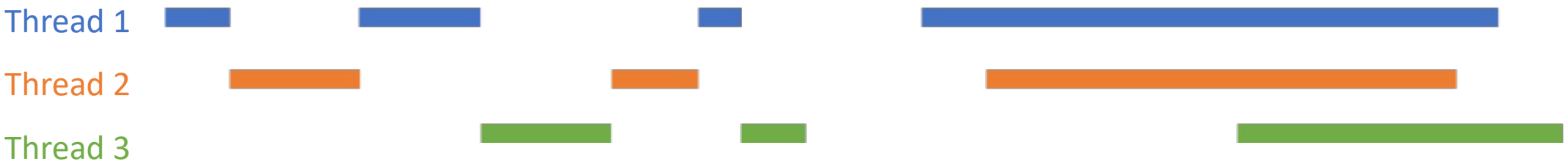
D.h. aber nicht, dass zu einem Zeitpunkt mehrere Aufgaben gleichzeitig bearbeitet werden.

Concurrency

Tasks start, run and complete in an interleaved fashion

Parallelism

Tasks run simultaneously



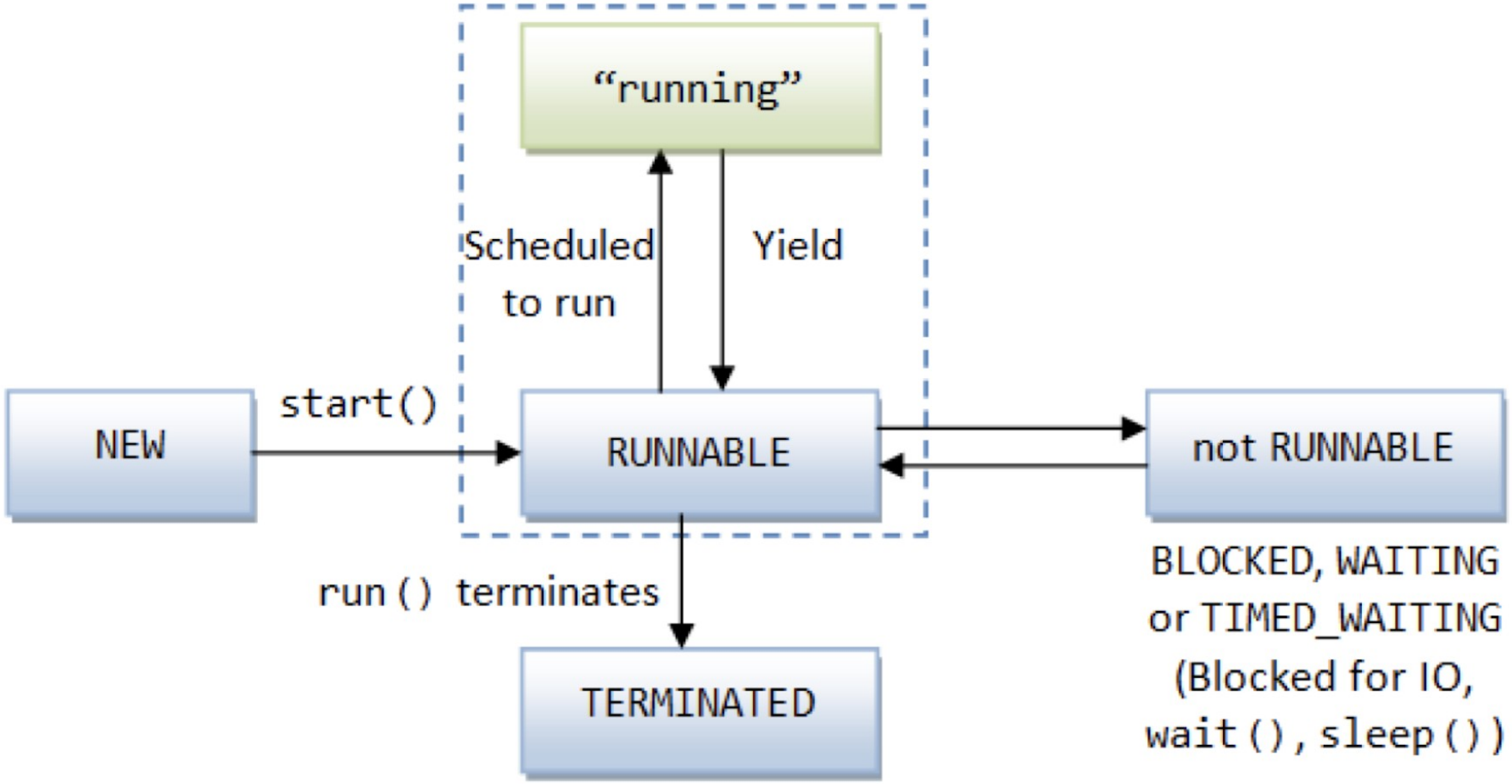
Concurrency klappt auch mit einem einzigen Thread!
Parallelism nicht.

Fragen bis jetzt?

Threads erstellen in Java (3 Optionen)

- Anonyme Klassen (für kleine Programme)
- Runnable Interface implementieren
- Thread Klasse extenden

Java Thread State Modell



https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.htm

Demo- Threads

Extra

Code Style

Probiert euren Code so leserlich wie möglich zu gestalten.

Fügt unbedingt high-level Kommentare hinzu.

Extra: Daemon Threads vs. non-daemon Threads

Daemon Threads sind LOW-PRIORITY Threads, die im Hintergrund Jobs wie Garbage Collection übernehmen.

Alle anderen «normalen» Threads sind non-daemon

Sobald alle non-daemon Threads fertig sind, terminiert die JVM das Programm.

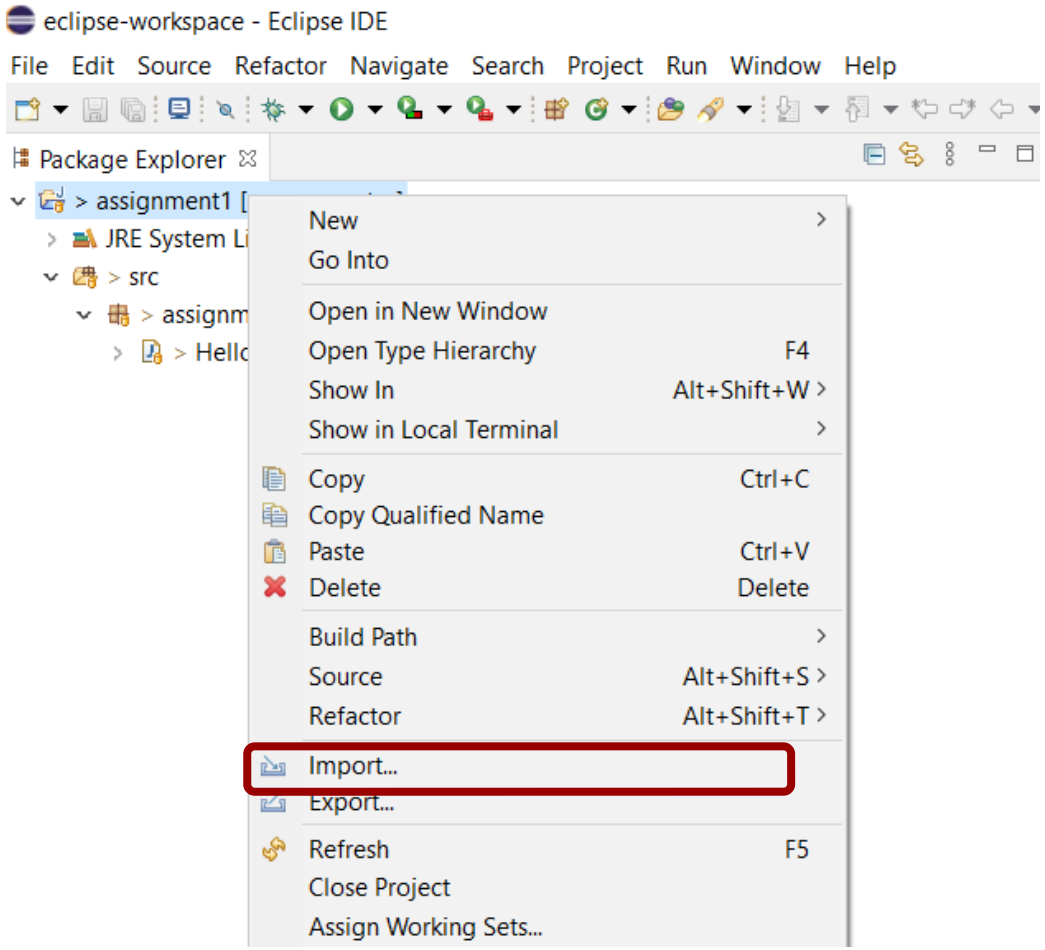
Daemon bzw. non-daemon wird vom Parent-Thread vererbt oder kann mit setDaemon(boolean) verändert werden.

Vorbereitung & Vorbesprechung Assignment 2

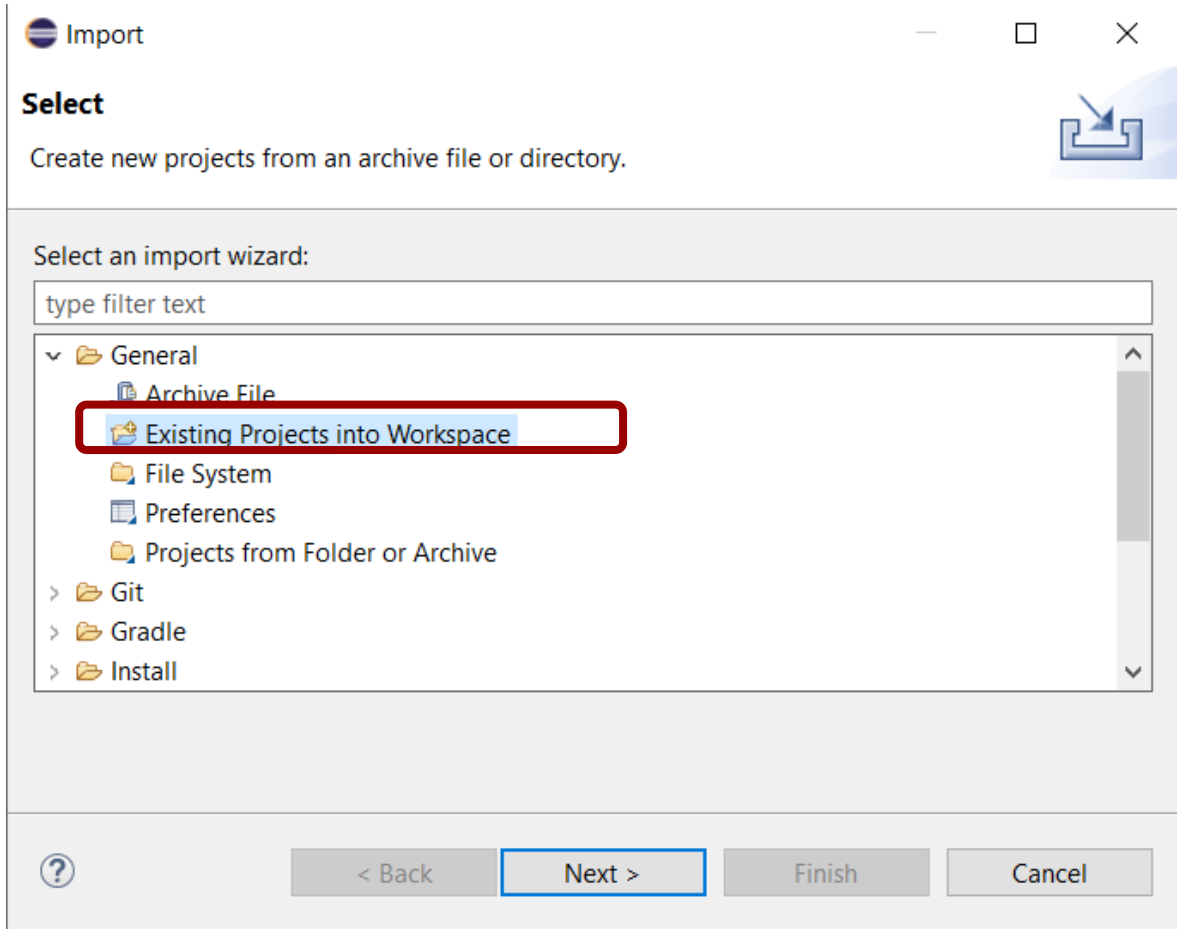
Vorbereitung

1. assignment2.zip in Eclipse importieren
2. JUnit Tests ausführen

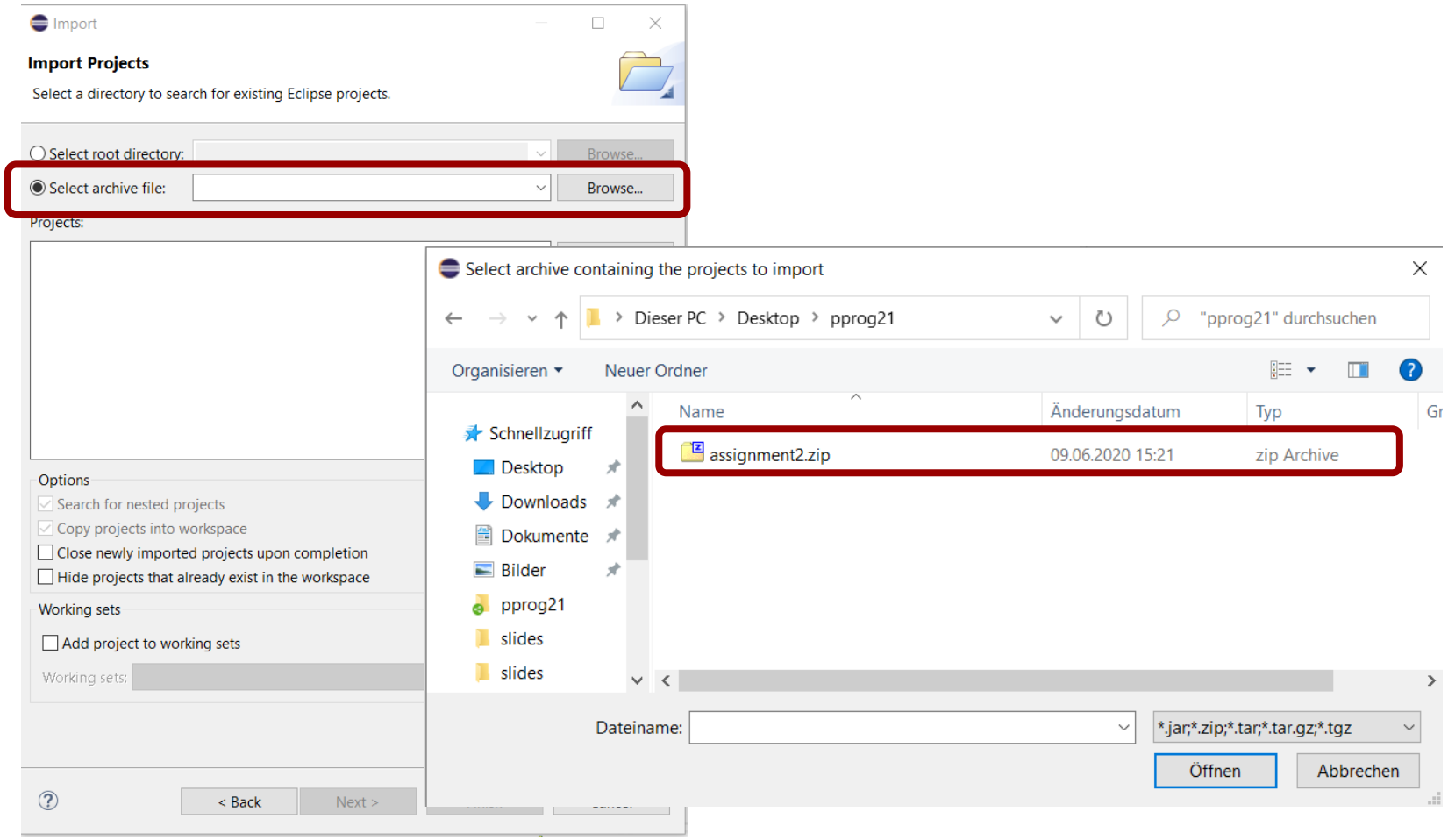
Eclipse: Projekt importieren



Eclipse: Projekt importieren

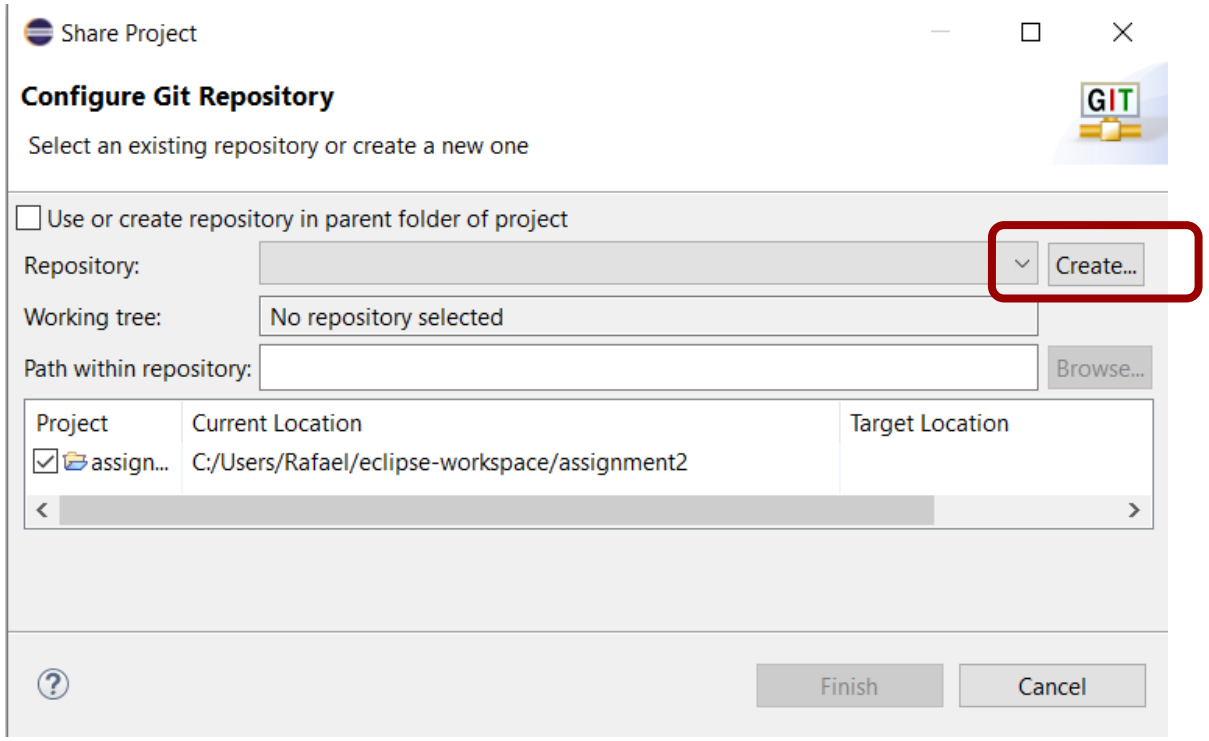


Eclipse: Projekt importieren

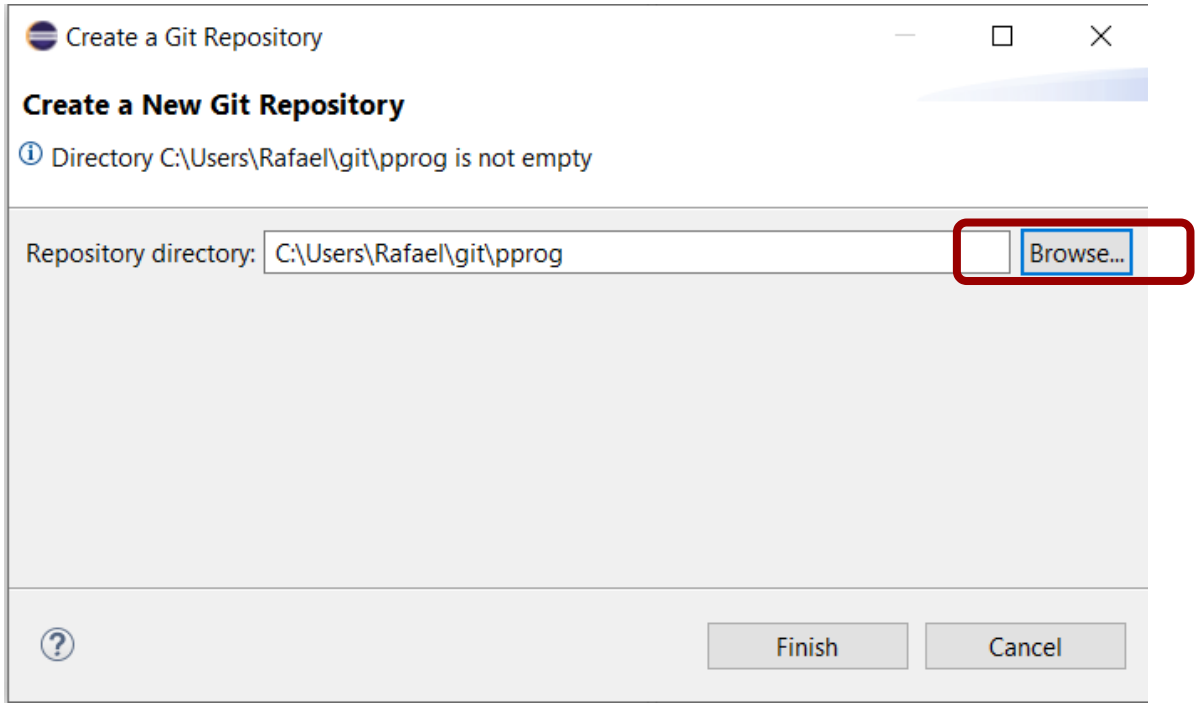


Eclipse: Git hinzufügen

Team -> Share Project ...

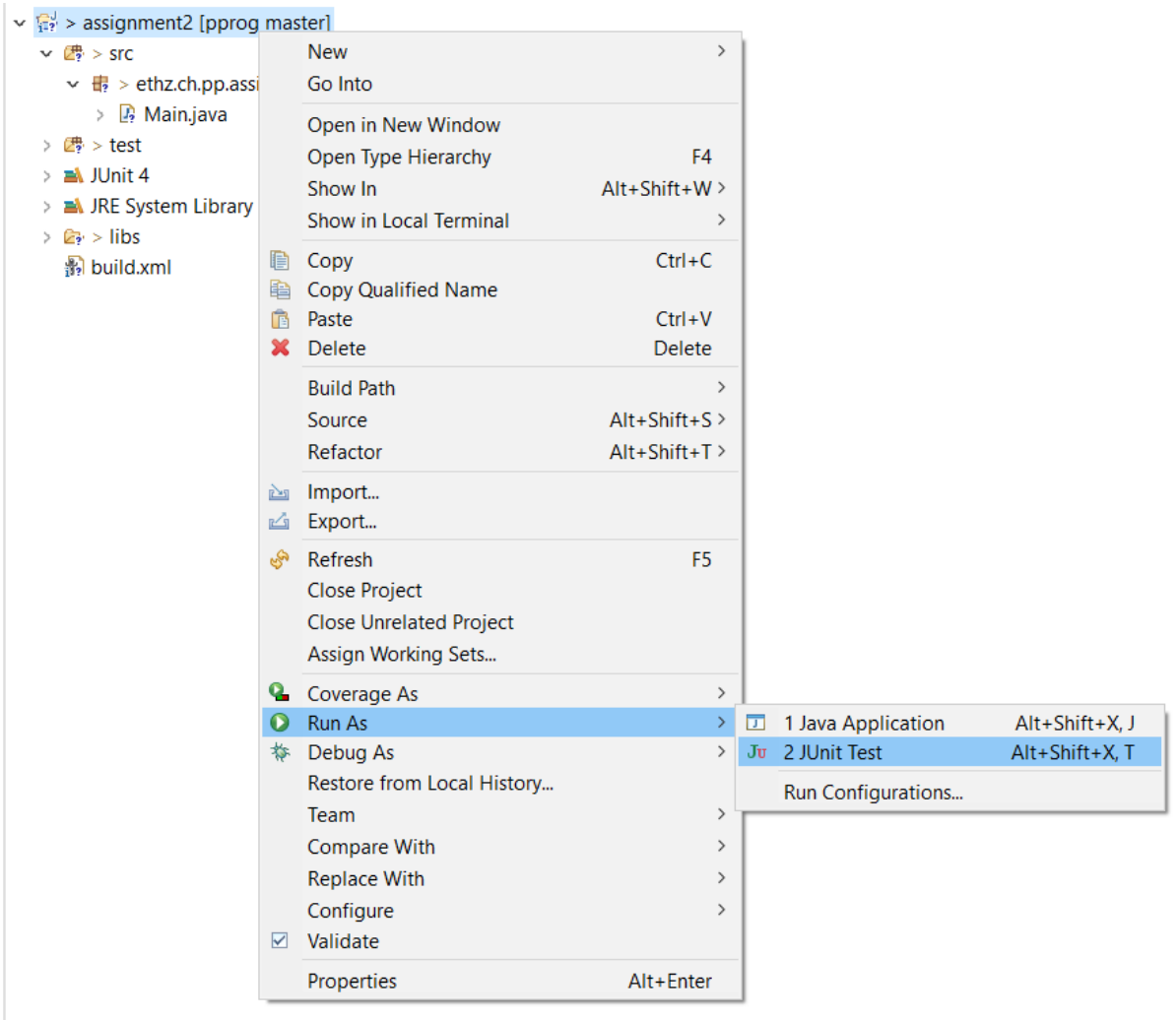


Eclipse: Git hinzufügen



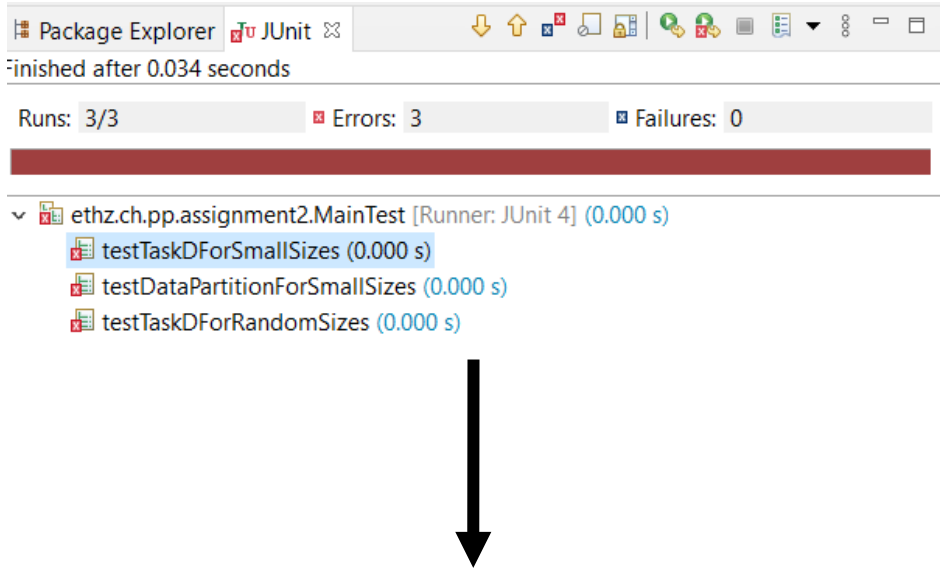
Wichtig: Wählt den gleichen Ordner wie für assignment 1 aus. Falls ihr noch kein Repository habt, meldet euch!

Eclipse: JUnit-Tests ausführen

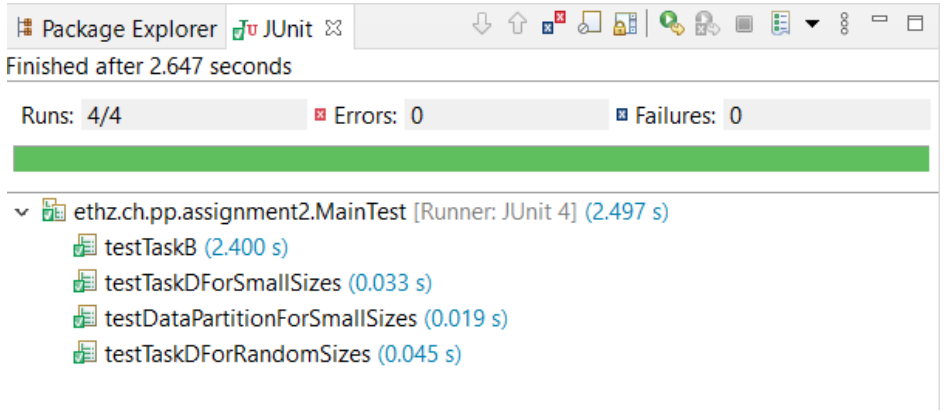


Eclipse: JUnit-Tests ausführen

Startzustand



Eure Lösung (idealerweise)



Task A

Startet einen neuen Thread, der “Hello Thread!” ausgibt.

Wie checkt ihr, dass tatsächlich ein neuer Thread kreiert wurde?

Stellt sicher, dass das Program erst terminiert wenn der neue Thread fertig ist.

Task B

Description: Our goal in this exercise will be to parallelize the execution of the following loop defined in `computePrimeFactors` method:

```
for (int i = 0; i < values.length; i++) {
  factors[i] = numPrimeFactors(values[i]);
}
```

which computes the number of prime factors for each element in an given array. For example, for number `12` the number of prime factors is `numPrimeFactors(12) = 3` since `12 = 2*2*3`. The implementation of `numPrimeFactors` is already provided for you in the assignment template and should not be changed.

Task B

Führt computePrimeFactors() im main Thread aus und vergleicht diese Zeit mit der Dauer, die ein neu erstellter Thread braucht (inkl. erstellen des Threads).

Ist eine grosse Differenz zu beobachten?

Task C

Führt ein Experiment aus, mit dem ihr den Overhead messen könnt, die das Erstellen eines neuen Threads verursacht.

Task C

Option 1: Misst die Dauer mit der Zeit die der Thread nicht läuft.

```
long time = System.nanoTime();
//compute something
time =  System.nanoTime() - time;
```

Option 2: Misst die Dauer ohne der Zeit die der Thread nicht läuft.

```
ThreadMXBean tmxb = ManagementFactory.getThreadMXBean();
long time = tmxb.getCurrentThreadCpuTime();
//compute something
time = tmxb.getCurrentThreadCpuTime()-time;
```

Task C

Die gemessene Zeit wird nicht immer die gleiche sein.

Berechnet den Durchschnitt über mehrere Versuche (je mehr desto besser)

Berechnet die Varianz (Streuung) euer Daten.

Task D

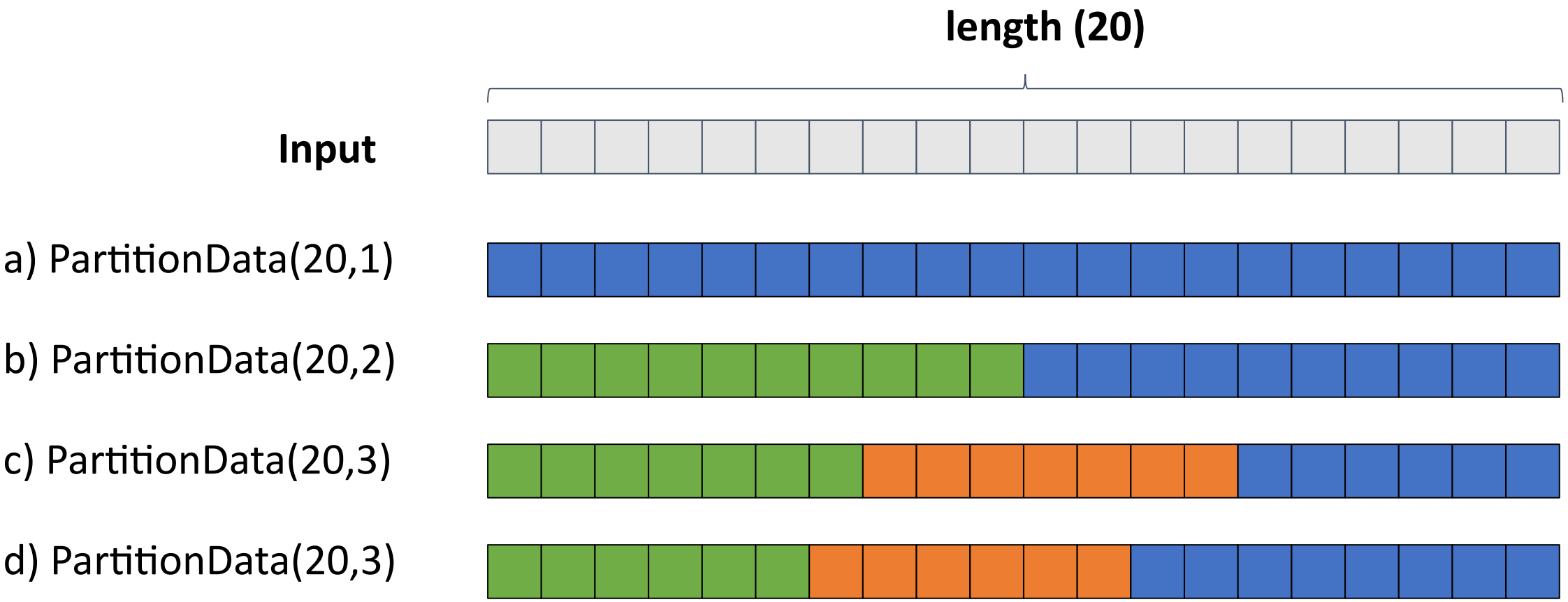
Bevor ihr den Loop in Task E parallelisiert, denkt nach wie die Arbeit auf die verschiedenen Threads verteilt werden soll.

Jeder Thread sollte ungefähr gleich viel Arbeit erhalten.

Beschreibt eure Lösung + andere Möglichkeiten.

Task D: Aufteilung der Arbeit zw. Threads

PartitionData(int length, int numPartitions) { … }



c) & d) sind für diese Aufgabe beide valide Lösungen

Task D

Verschiedene Möglichkeiten mit unterschiedlicher Performance je nach Struktur der Daten.

Für Random Inputs: Reicht es das Array in gleich grosse Stücke zu teilen

Für sortierte Inputs: Werden die ersten Teilstücke schneller fertig sein als die Letzten.

Task D

Edge Cases nicht vergessen. Was soll passieren wenn:
Array length < numPartitions ???
Length <= 0 ???
numPartitions <= 0 ???

Schreibt eure Annahmen in die Kommentare (oder JavaDoc)

```
PartitionData(int length, int numPartitions) {  …  }
```

Task E

Parallelisiert die Ausführung des Loops in computePrimeFactors() mit einer konfigurierbaren Anzahl an Threads.

Task F

Bevor ihr die Experimente durchführt:

Probiert zu erraten wie der Plot, der den Speedup abhängig von der Anzahl Threads und der Array Länge darstellt, aussehen wird.

Misst den Speedup für die folgenden Werte:

Anzahl Threads und
Array Länge

Task G

Misst die Runtime für alle Experimente und besprecht den Unterschied zwischen den Resultaten aus diesem Task und dem Task F

Speedup

Linear: D.h. Speedup ist proportional zu den Anzahl Threads. Auf den ersten Blick logisch.

Sub-Linear: Meistens der Fall, da nicht das ganze Programm parallelisierbar ist (sequential Bottleneck -> später: Amdahl's Law) und Overheads dazukommen.

Super-linear: Möglich in manchen Fällen (z.B. da mehr Cache dank mehreren Cores zur Verfügung steht)

Prüfungsaufgabe

Kreuzen Sie alle korrekten Aussagen über das Erstellen von Java `Threads` an.

Mark all correct statements regarding the creation of Java `Threads`.

- [ ] Beim Aufteilen eines Workloads sollte man soviele Threads erstellen wie möglich, bis nur noch elementare Operationen pro Thread ausgeführt werden.

  When splitting a workload, as many threads as possible should be created until only elementary operations are performed per thread.

- [x] Um einen eigenen Thread in Java zu definieren kann man das Runnable-Interface implementieren.

  To define a custom thread in Java, one can implement the Runnable interface.

- [x] Um eine eigene Thread-Klasse in Java zu definieren kann man die Thread-Klasse erweitern.

  To define a custom thread class in Java, one can extend the Thread class.

- [ ] Threads werden fast ausschliesslich genutzt um eine rekursive Implementation zu beschleunigen.

  Threads are used almost exclusively to speed up a recursive implementation.

Quiz!