Parallel Programming
Exercise Session 2

Spring 2025

About me

- Erxuan Li
- Bachelor 4th semester
- erxuli@ethz.ch
- Discord: erxuanli
- https://erxuanli.com/teaching/fs25_pprog/

Schedule

Motivation: Why Parallel Programming?

Theory

Quiz

Preparation assignment 2

Pre-Discussion assignment 2

Coding Remarks

Why Parallel Programming?

Why Parallel Programming

- Solve problems faster
- Large problems → divided into smaller ones → executed in parallel

- Programs for Supercomputers / High-Performance Computing are highly parallel

HPE Cray
CSCS
ETH zürich
ALPS

Team RACKlette

- ETH Club about High-Performance Computing under Prof. T. Hoefler and in collaboration with CSCS
- Optimizing, compiling code for HPC clusters
- Understanding hardware and how to exploit that for speedup

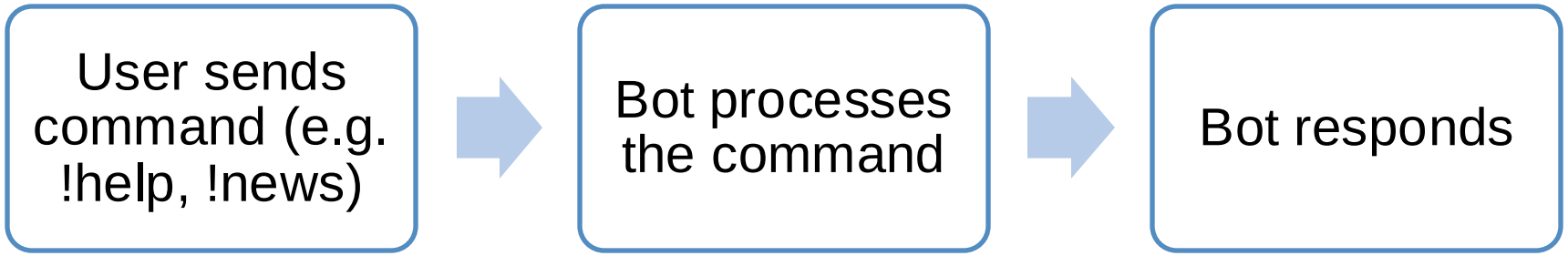
https://racklette.ethz.ch/

Interested? Join us!

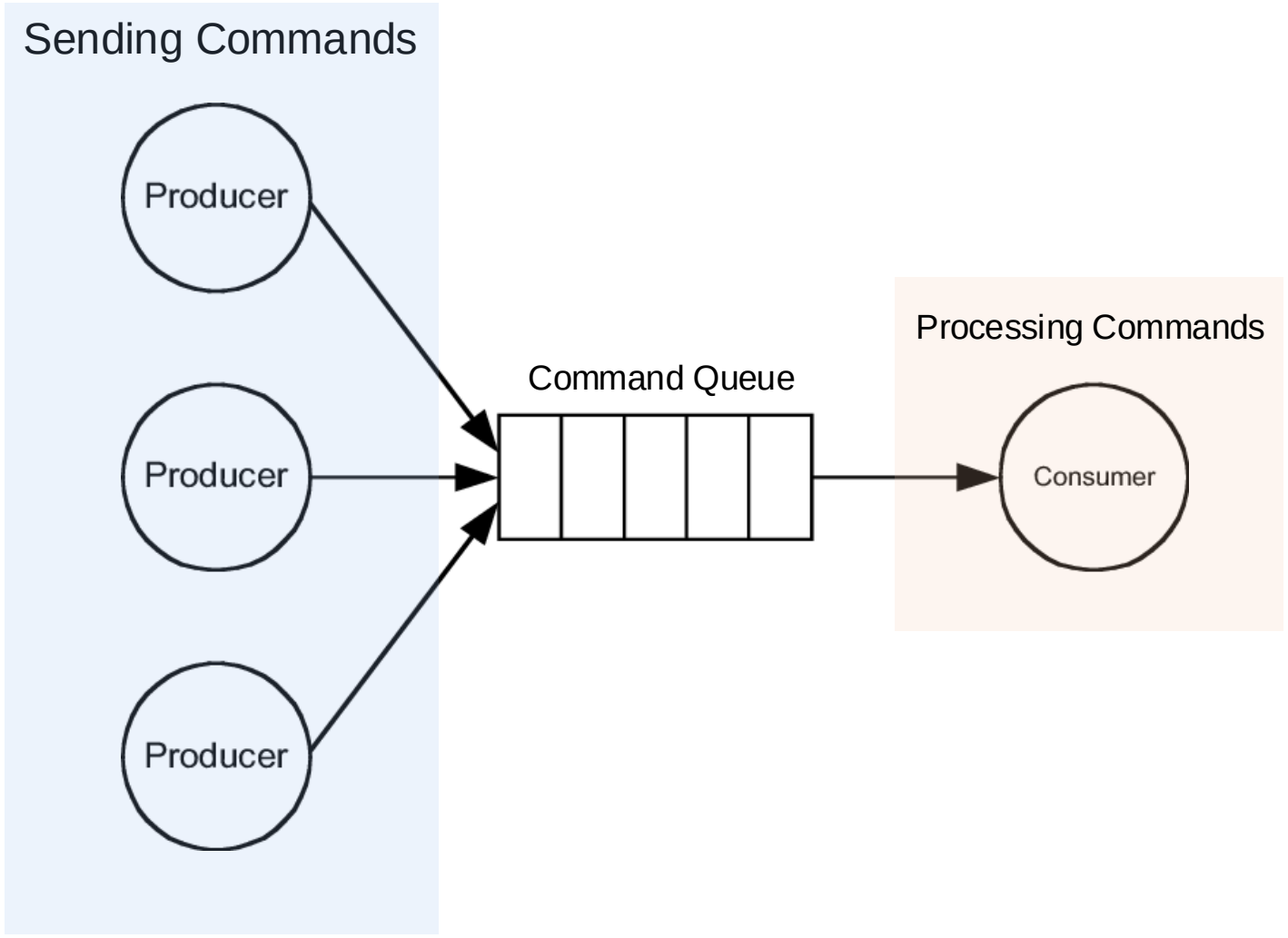
https://forms.gle/4zWgxXsMdd5DgCa76

Another example: Discord Bot

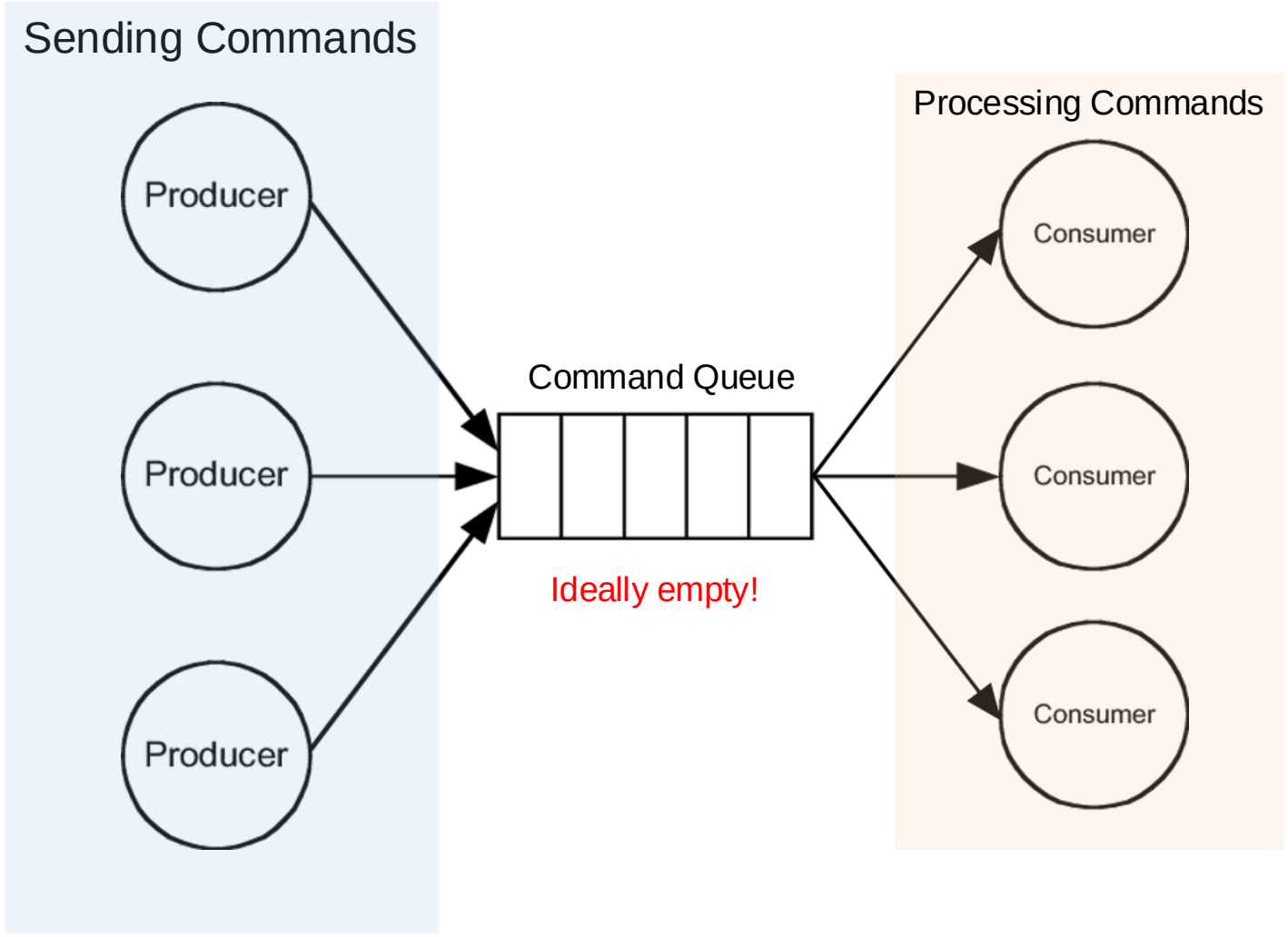
How a Discord Bot works (roughly):



Producer-Consumer Problem



Solution? More Threads!



Theory

Terminology

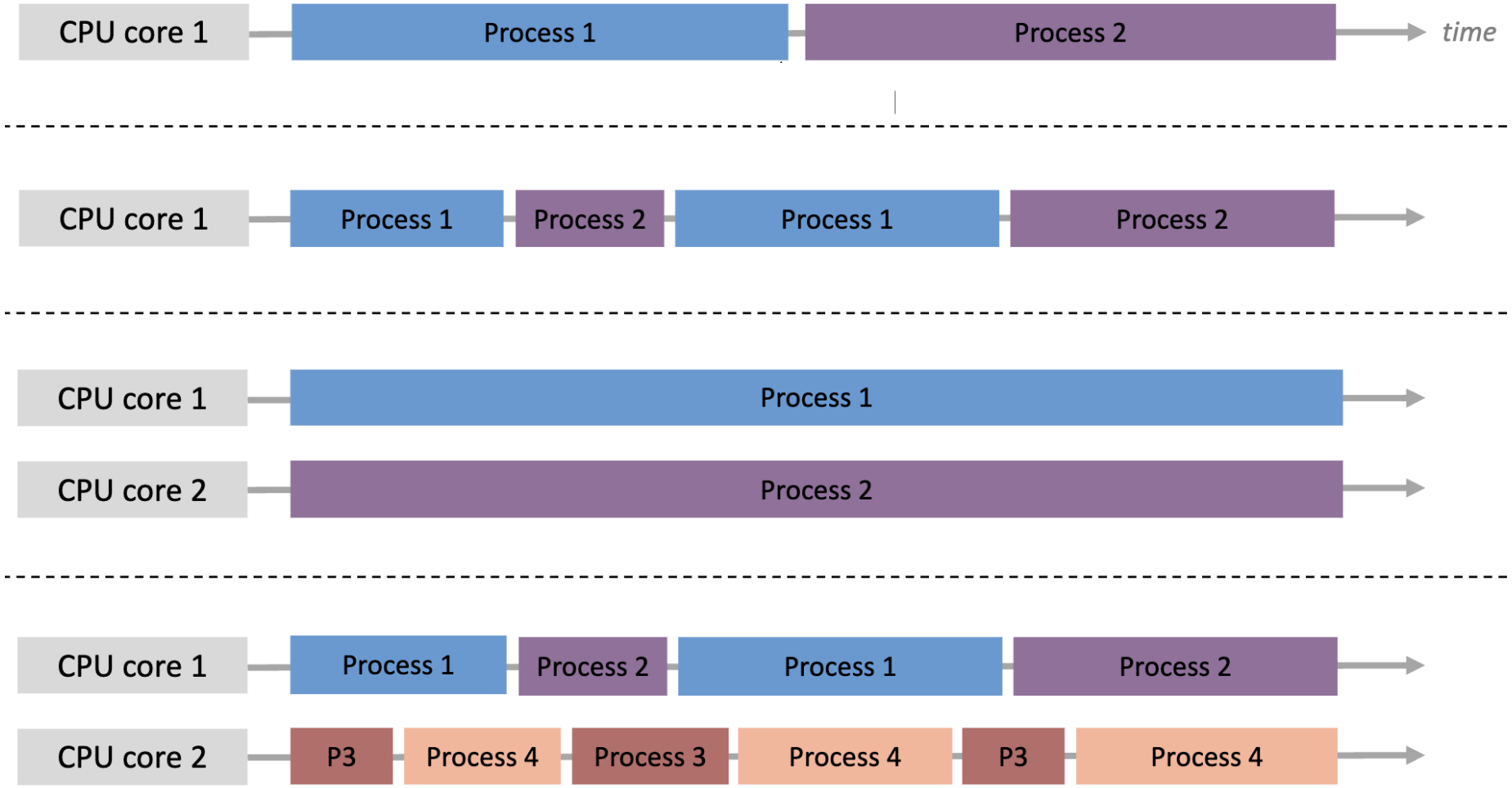
Overview:
https://cgl.ethz.ch/teaching/parallelprog25/pages/terminology.html

Sequential vs Concurrent vs Parallel

Concurrency:
Dealing with multiple things at the same time.

Parallelism:
Doing multiple things at the same time.

Sequential vs Concurrent vs Parallel



Thread Definition

An independent (i.e., capable of running in parallel) unit of computation that executes code.

Each thread is like a running sequential program,

but a thread can create other threads

that are then part of the same program.

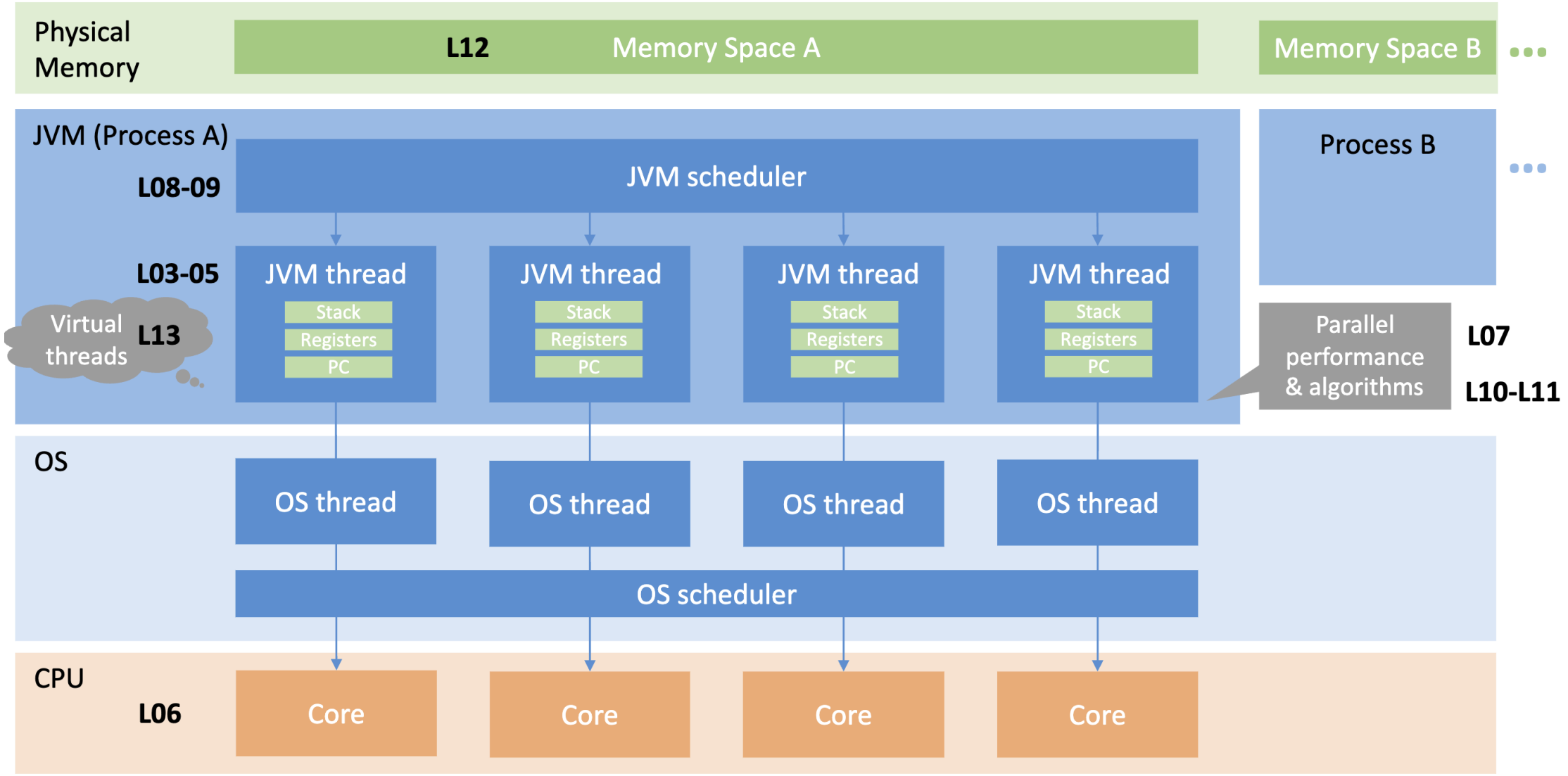
Those threads can create more threads etc.

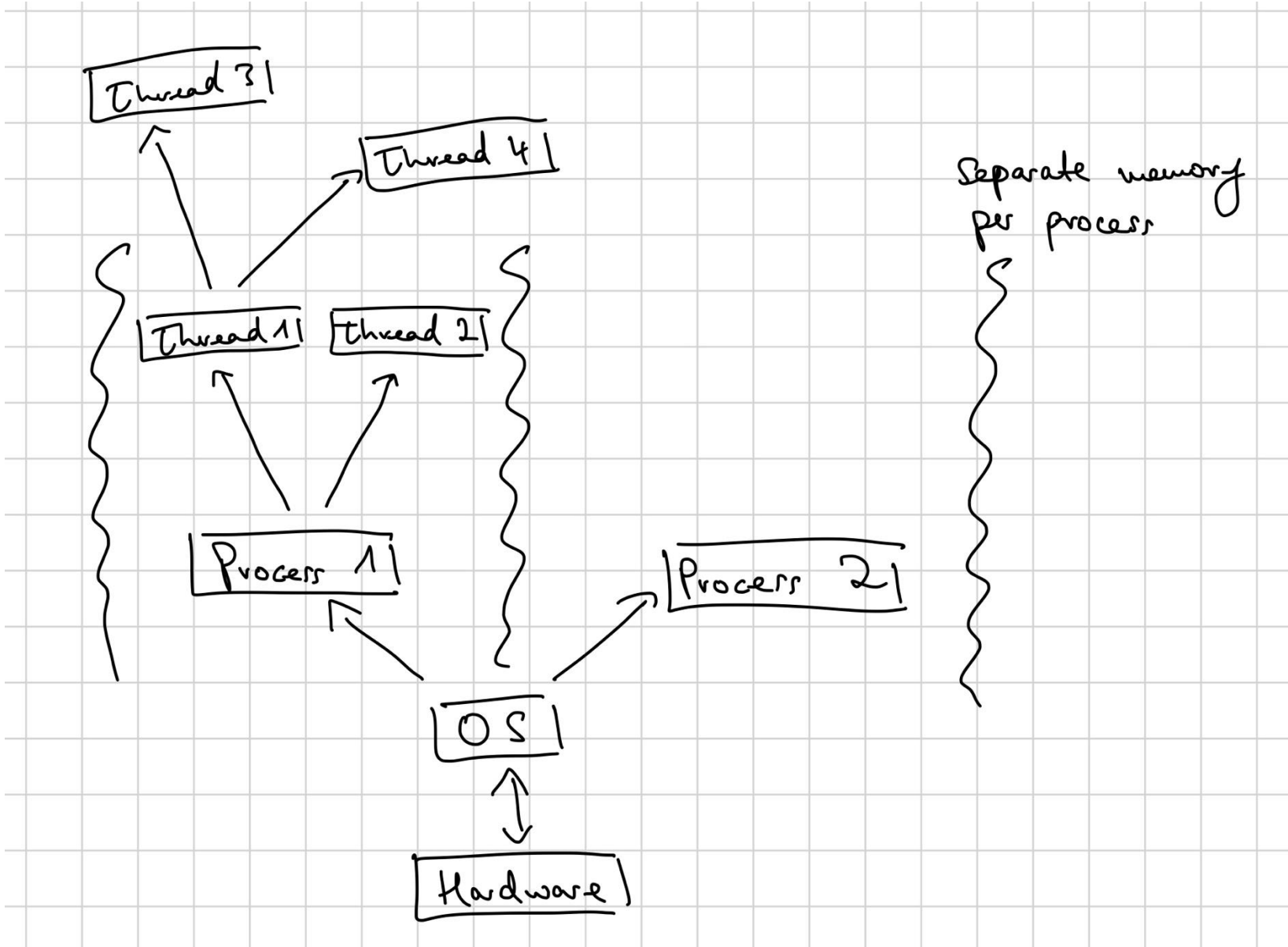
Thread Definition Advanced

Concept of threads exists on various levels:

- Hardware (CPU)
- Operating systems
- Programming languages
  - Java: `Thread` class

Big Picture


Physical Memory
L12
Memory Space A
Memory Space B
JVM (Process A)
L08-09
JVM scheduler
Process B
L03-05
JVM thread
Stack
Registers
PC
Virtual threads
L13
Parallel performance & algorithms
L07
L10-L11
OS
OS thread
OS scheduler
CPU
L06
Core


Thread 3
Thread 4
Thread 1
Thread 2
Process 1
Process 2
OS
Hardware
Separate memory per process

Thread Properties (in our course)

- Threads can create other threads
- Shared memory (changes to variables by threads are visible to other Threads)
- Threads (from same class) execute same program *but* with different arguments
- Communication between threads: Writing fields of shared objects

Daemon vs non-daemon threads

Daemon threads
low priority threads

Non-daemon / user threads
high priority threads

JVM process stops when all non-daemon threads terminate

Daemon vs non-daemon threads

- Creating a new thread from a daemon thread leads to a daemon thread
- Creating a new thread from a non-daemon thread leads to a non-daemon thread
- Manually set daemon / non-daemon status before .start() with .setDaemon([true | false])
- Check if a thread is daemon with .isDaemon();

Create Java Threads: Option 1 (oldest)

Instantiate a subclass of `java.lang.Thread` class

- Override run method (must be overridden)
- run() is called when execution of that thread begins
- A thread terminates when run() returns
- start() method invokes run()
- Calling run() does not create a new thread

```
class ConcurrWriter extends Thread { …
    public void run() {
        // code here executes concurrently with caller
    }
}
ConcurrWriter writerThread = new ConcurrWriter();
writerThread.start();   // calls ConcurrWriter.run()
```

Creating the Thread object does not start the thread!

Need to actually call start() to start it.

Create Java Threads: Option 2 (better)

Implement `java.lang.Runnable`

- Single method: public void run()
- Class implements Runnable

```
public class ConcurrWriter implements Runnable {
      …
      public void run() { …
         // code here executes concurrently with caller
       }
}

ConcurrWriter writerThread = new ConcurrWriter();
Thread t = new Thread(writerThread);
t.start();   // calls ConcurrWriter.run()
```

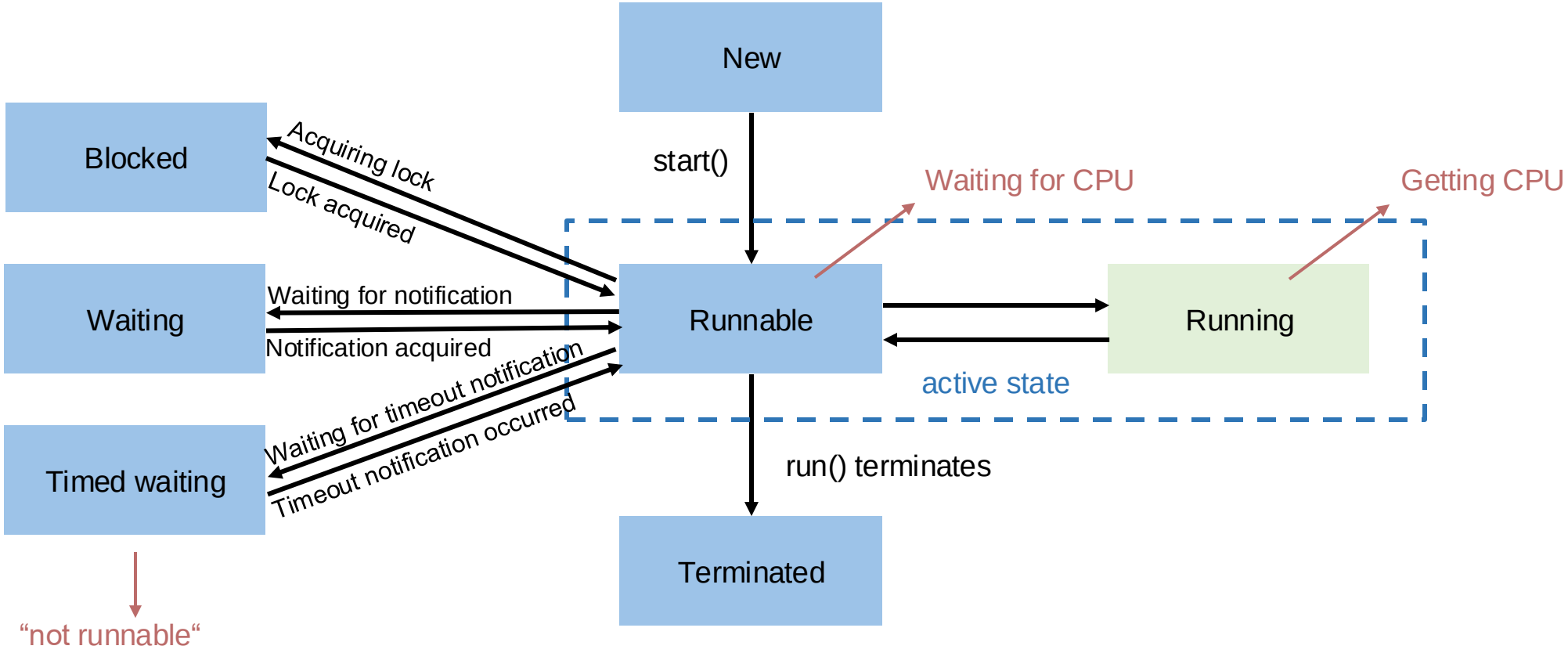
Create Java Threads: Option 3 (lazy)

Inline

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        // do something
    }
});

thread.start();
```

Life cycle of a Thread



Past Exam Task

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.

- ◯ Um eine eigene Thread-Klasse in Java zu definieren kann man das Runnable-Interface implementieren.

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

- ◯ 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.

Rep. Exam, FS 2023

Past Exam Task

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.

√ **Um eine eigene Thread-Klasse in Java zu definieren kann man das Runnable-Interface implementieren.**

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

√ **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.

Rep. Exam, FS 2023

What can go wrong?

Assume we have two threads executing increment() n-times concurrently.

```
public class Counter {
  int count = 0;

  public void increment() {
     count = count + 1; // or count++;
  }
}
```

Data Race

count == 0

Thread A	**Thread B**
Read count == 0 tmp = count + 1 count = tmp	Read count == 0 tmp = count + 1 count = tmp

count == 1

Demo

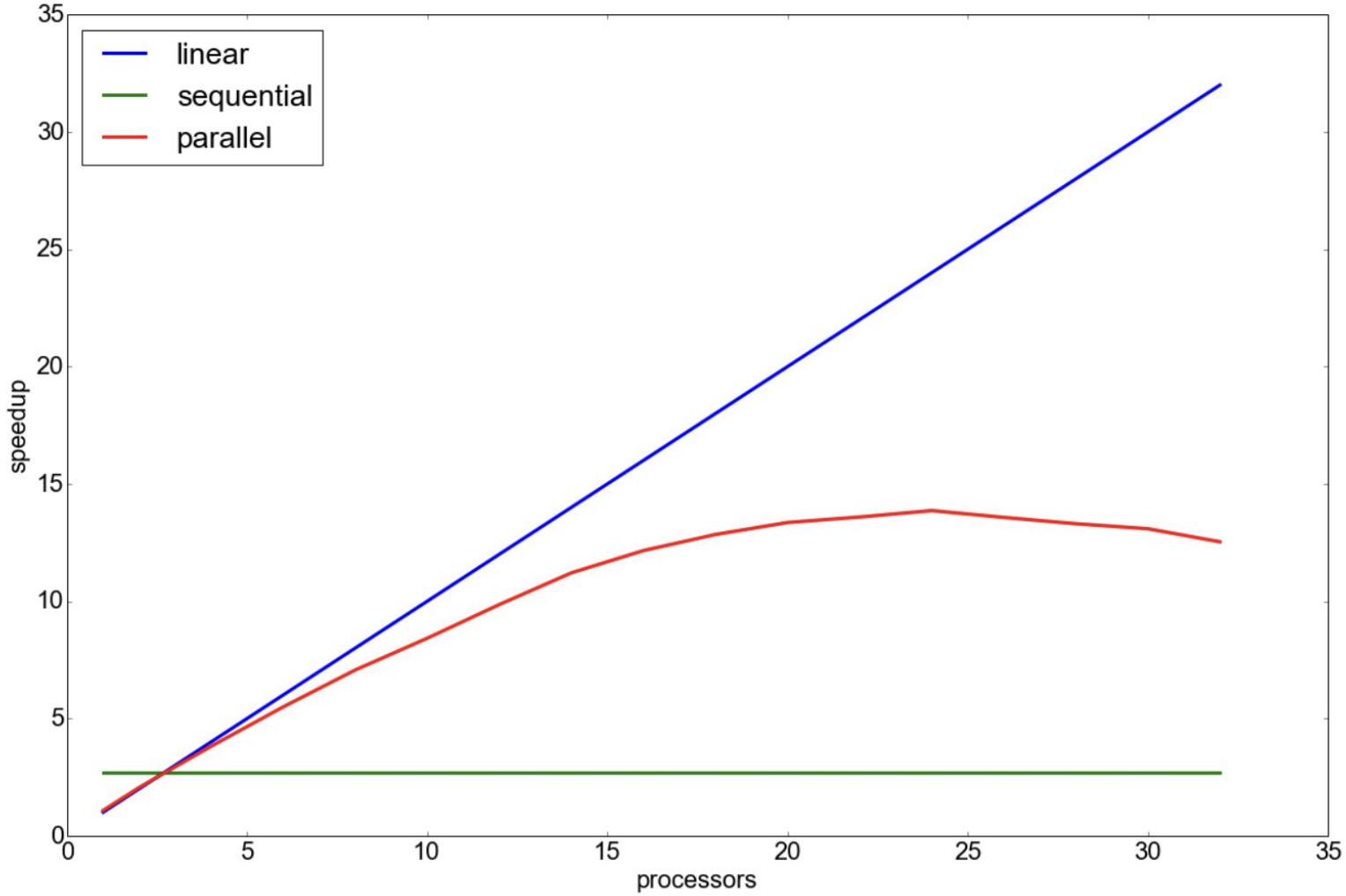
Speedup

Sub-linear: usually

Super-linear: not possible in theory, *but*

- Modern hardware properties (local/remote memory)
- Bug (this course assumes this)

Speedup


linear
sequential
parallel
speedup
processors
0
5
10
15
20
25
30
35

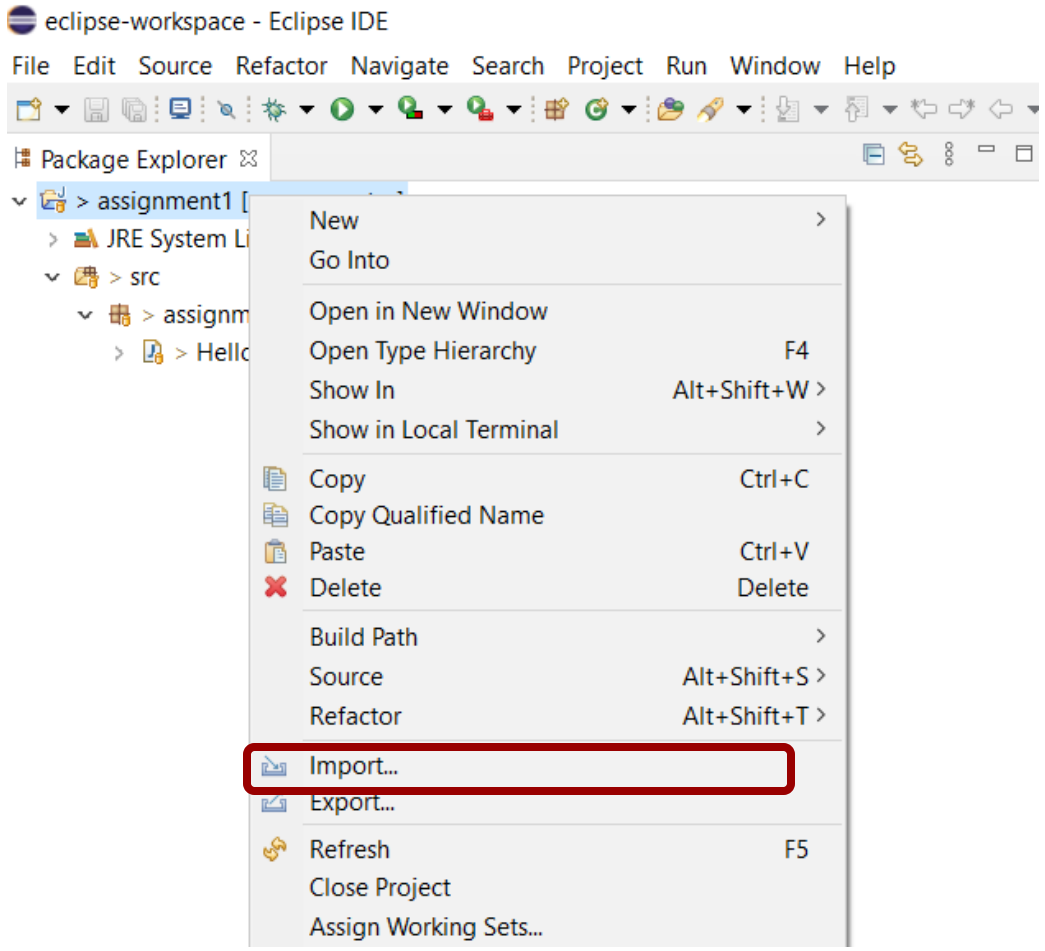
QUIZ

Preparation Exercise 2

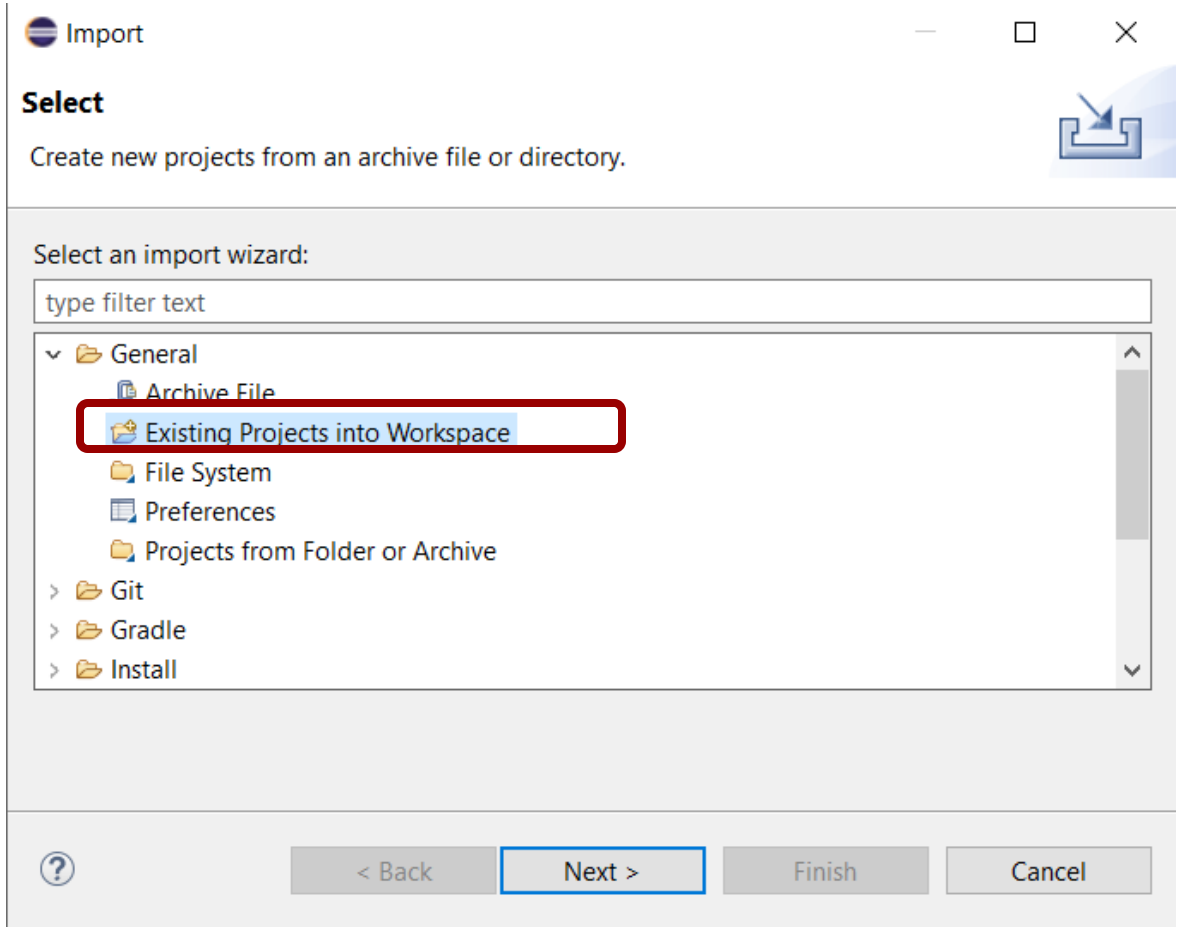
Preparations

1. Import assignment2.zip in Eclipse
2. Run the projects unit-tests in Eclipse
3. Understand output of unit-tests
    - Did the test fail or succeed?
    - Why did the test fail?
4. Start coding and keep checking if tests pass

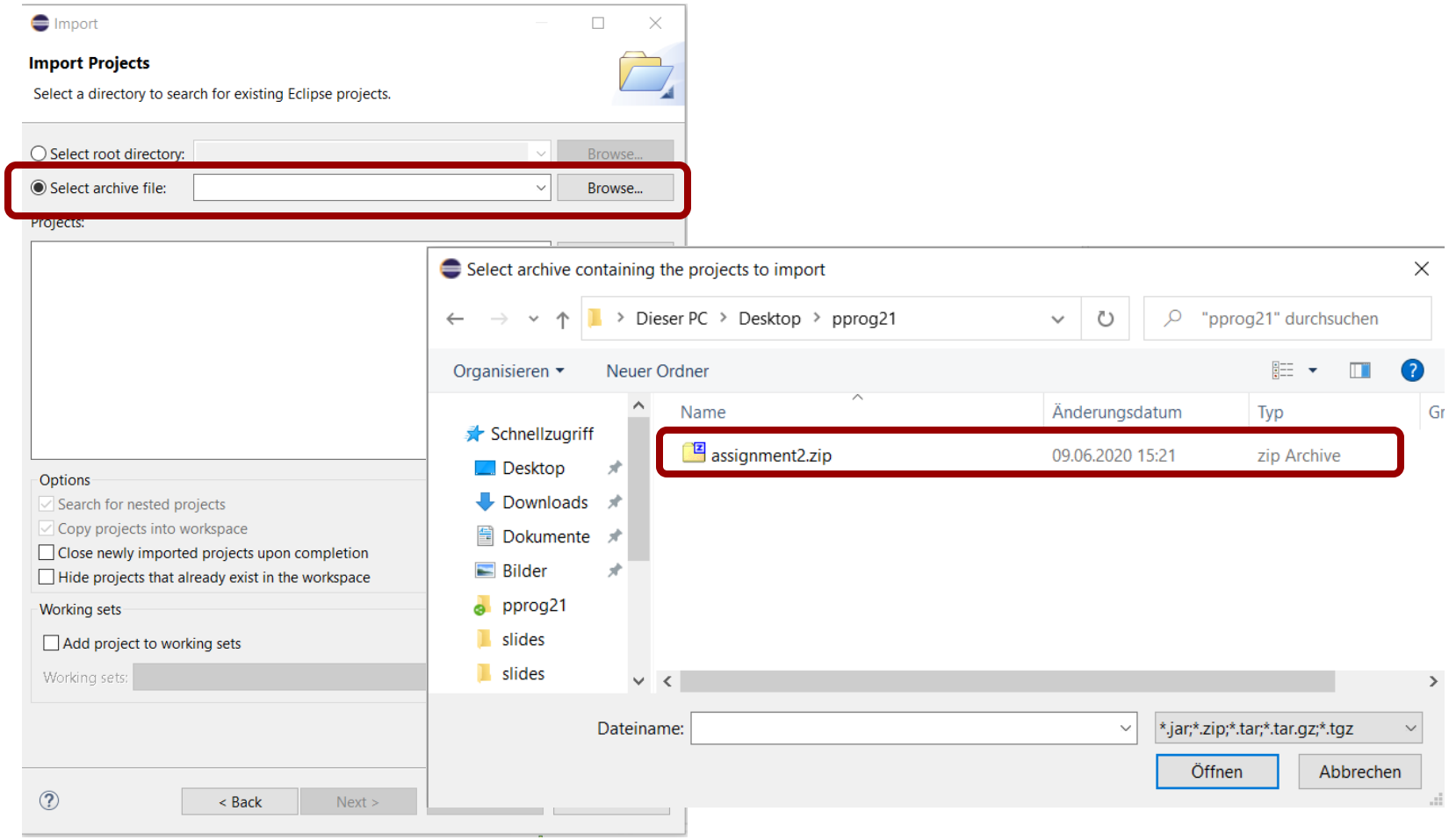
Eclipse: import project



Eclipse: import project

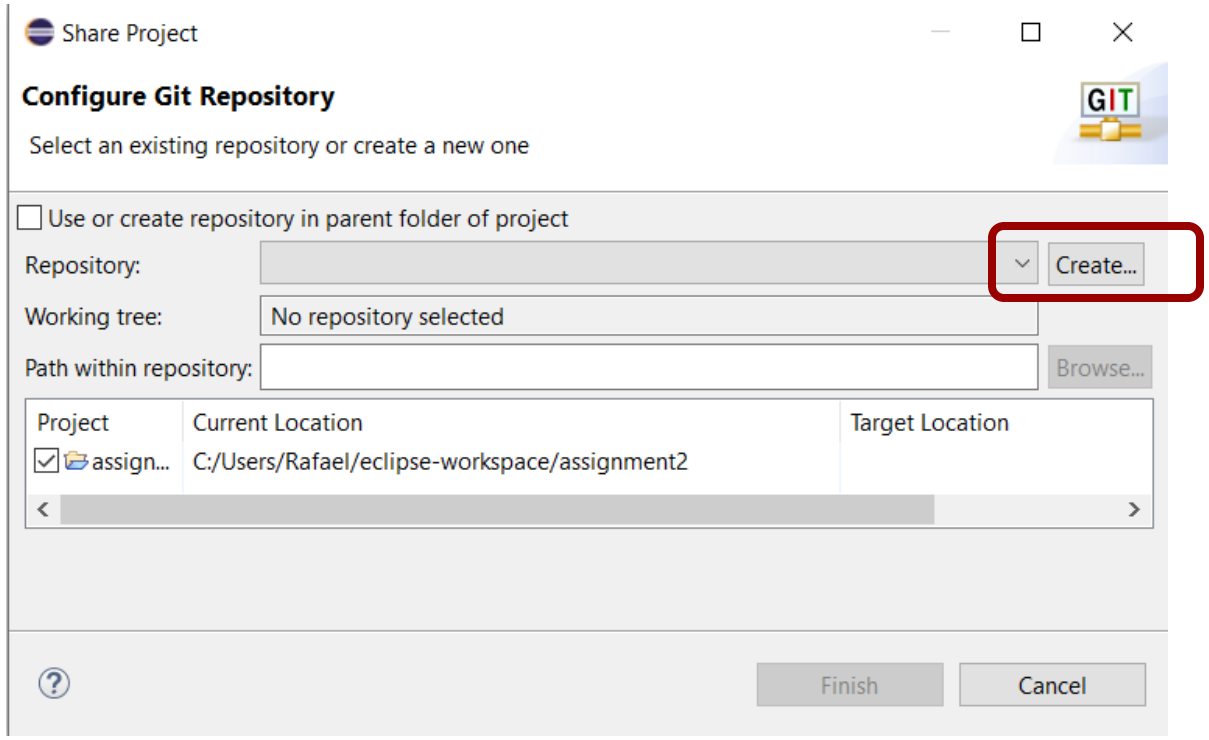


Eclipse: import project

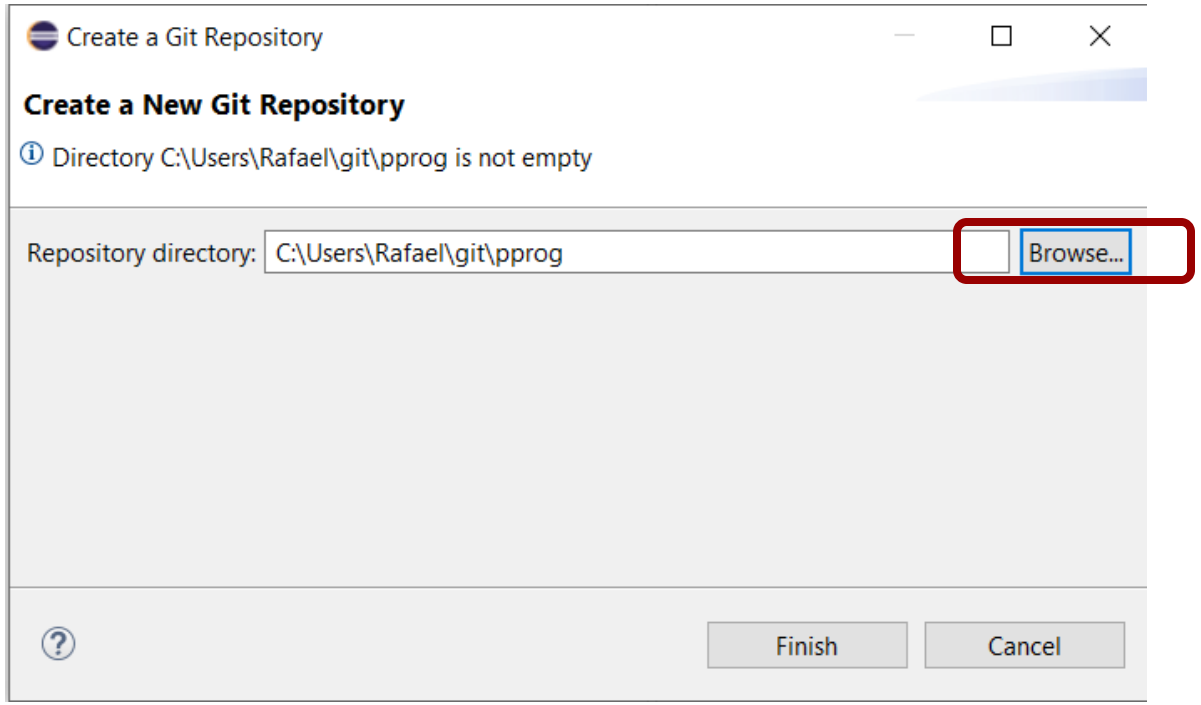


Eclipse: add to git

Team -> Share Project ...

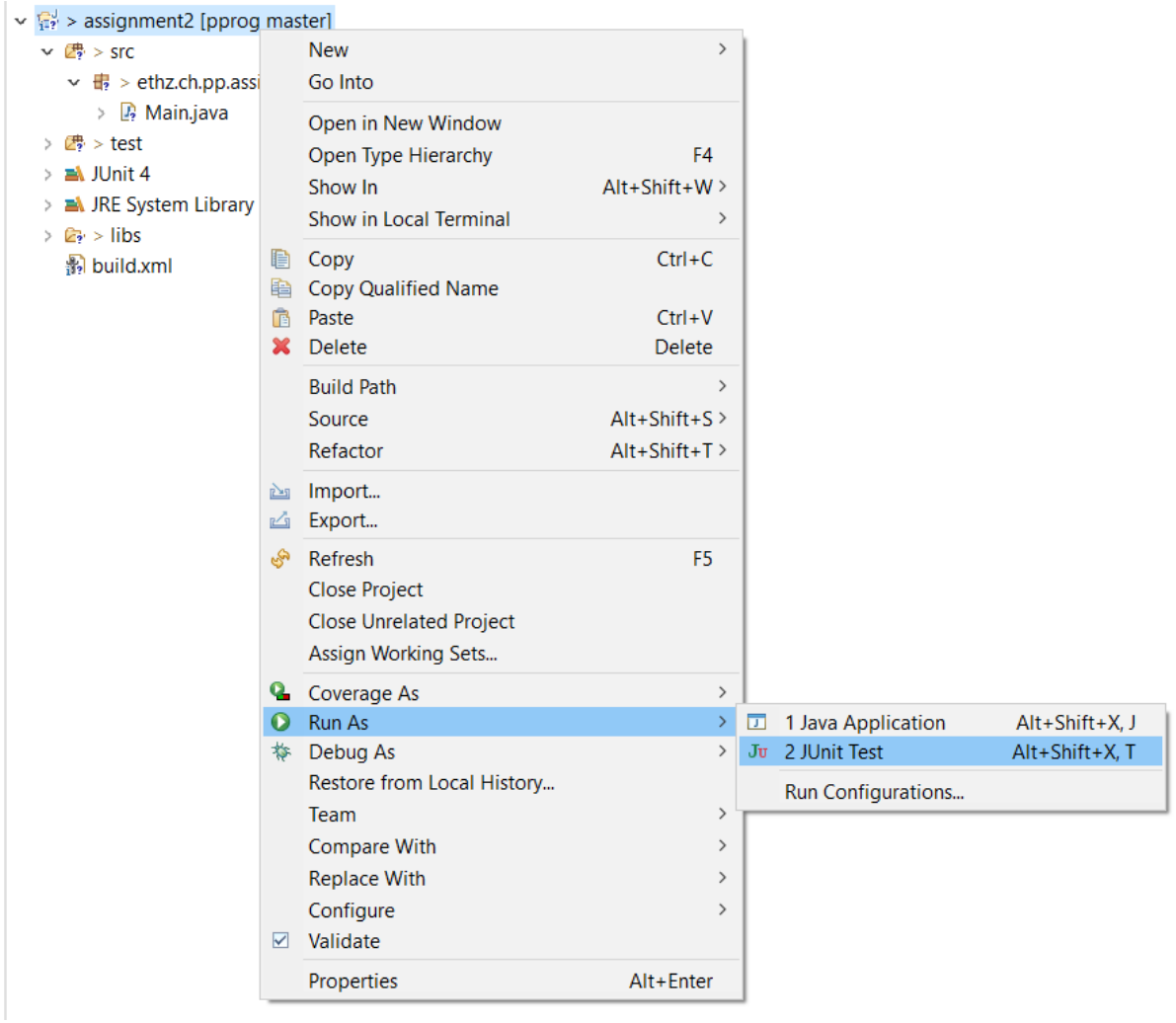


Eclipse: add to git



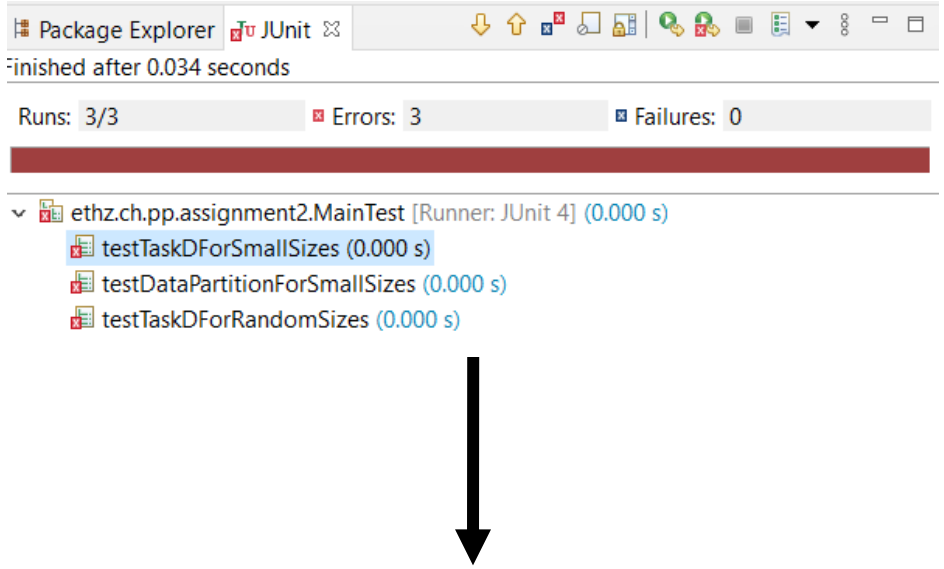
Important: Select same directory as for assignment 1
If you don't have a repo yet, contact your TA

Eclipse: running JUnit tests (1)

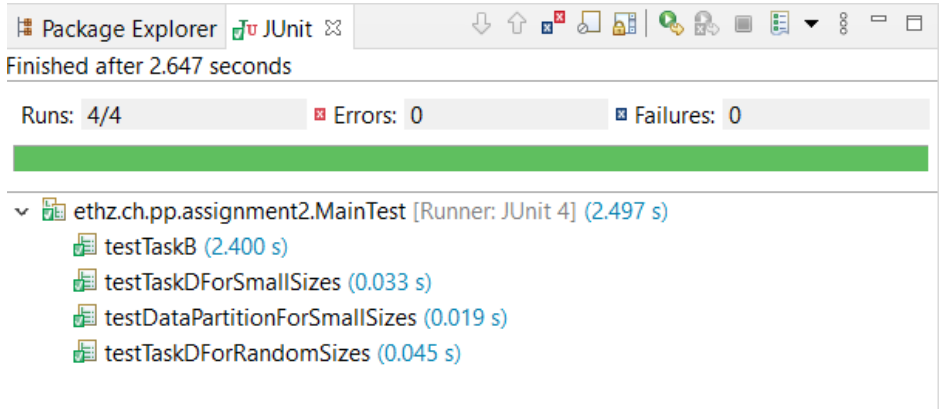


Eclipse: running JUnit tests (2)

Template



Your solution (ideally)



Pre-Discussion Exercise 2

Task A

To start with, print to the console "Hello Thread!" from a new thread. How do you check that the statement was indeed printed from a thread that is different to the main thread of your application? Furthermore, ensure that your program (i.e., the execution of main thread) finishes only after the thread execution finishes.

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

Run the method computePrimeFactors in a single thread other than the main thread. Measure the execution time of sequential execution (on the main thread) and execution using a single thread. Is there any noticeable difference?

Task C

Design and run an experiment that would measure the overhead of creating and executing a thread.

Task C

option 1: Measures real time elapsed including time when the thread is not running.

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

option 2: Measures thread cpu time excluding time when the thread is not running.

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

Task C

Measured execution time not always the same

→ Average over multiple runs (the more the better)

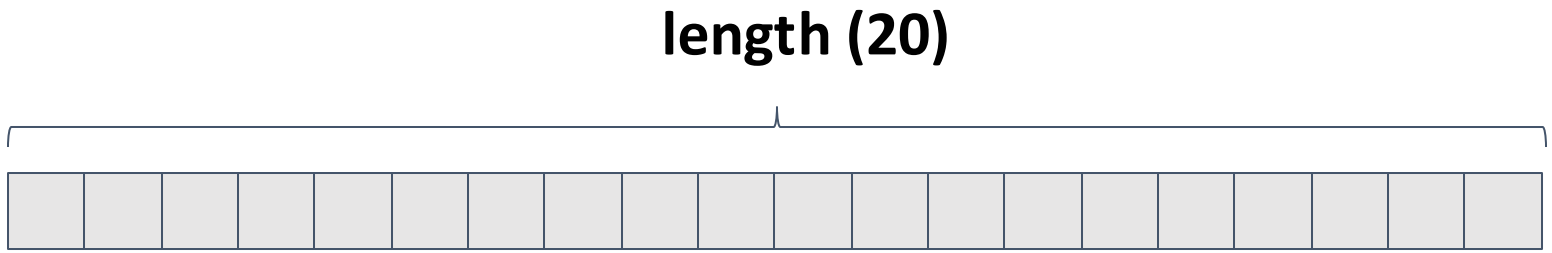
→ Calculate variance

Task D

Before you parallelize the loop in Task E, design how the work should be split between the threads by implementing method PartitionData. Each thread should process roughly equal amount of elements. Briefly describe you solution and discuss alternative ways to split the work.

Task D: Split the work between the threads

PartitionData(int length, int numPartitions) { … }



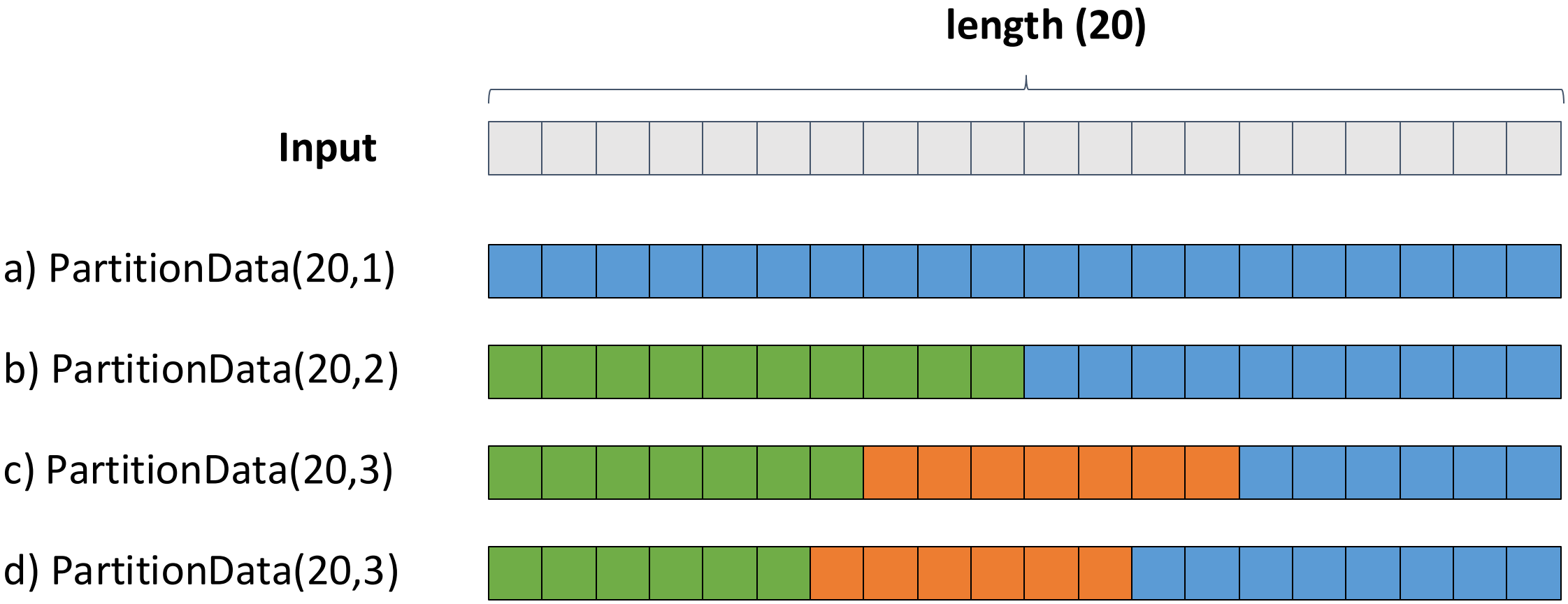
a) PartitionData(20,1) ?

b) PartitionData(20,2) ?

c) PartitionData(20,3) ?

Task D: Split the work between the threads

PartitionData(int length, int numPartitions) { … }



both c) and d) are correct solutions for this exercise

Task D

Several ways with different performance depending on task and data

If input is random: Splitting the input into half works well

If input is sorted: 1. half finishes faster than 2. half
→ maybe split on odd/even indices

Task D

- What about (length>0 and numPartitions>0) and length<numPartitions?
  - ??
  - ??
- And (length<=0 or numPartitions<=0)?
  - ??
  - ??

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

Task D

- What about (length>0 and numPartitions>0) and length<numPartitions?
  - Throw an exception?
  - Return m = min(m,n) splits?
- And (length<=0 or numPartitions<=0)?
  - Throw an exception?
  - Create a default return value (e.g. new ArraySplit[0])?
- In any case, write your assumptions in JavaDoc

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

Task E

Parallelize the loop execution in computePrimeFactors using a configurable number of threads.

Task F

Think of how would a plot that shows the execution speed-up of your implementation, for n = 1, 2, 4, 8, 16, 32, 64, 128 threads and the input array size of 100, 1000, 10000, 100000 look like.

Task G

Measure the execution time of your parallel implementation for n = 1, 2, 4, 8, 16, 32, 64, 128 threads and the input array size of input.length = 100, 1000, 10000, 100000. Discuss the differences in the two plots from task F and G.

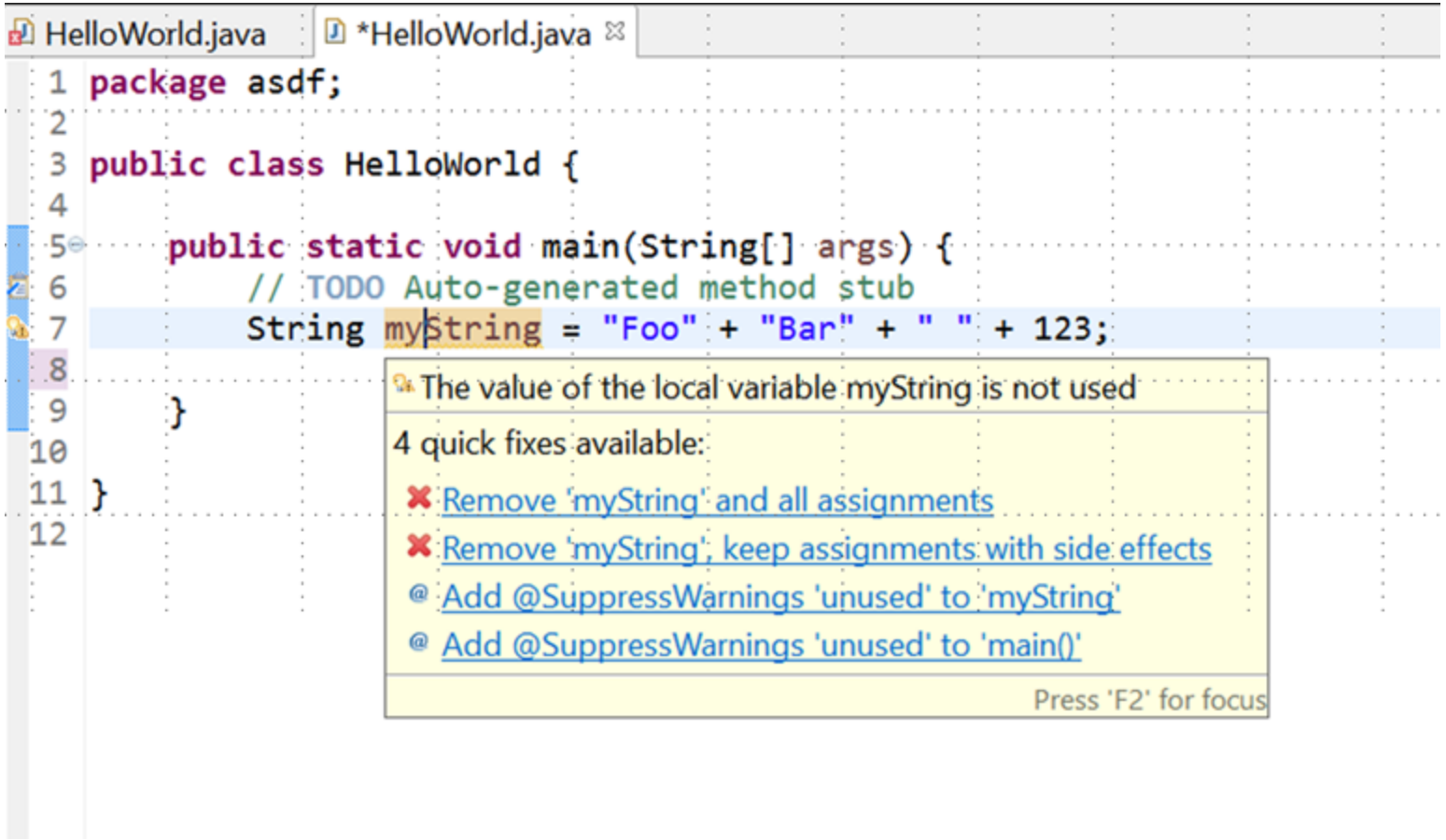
Coding Remarks

Code Style

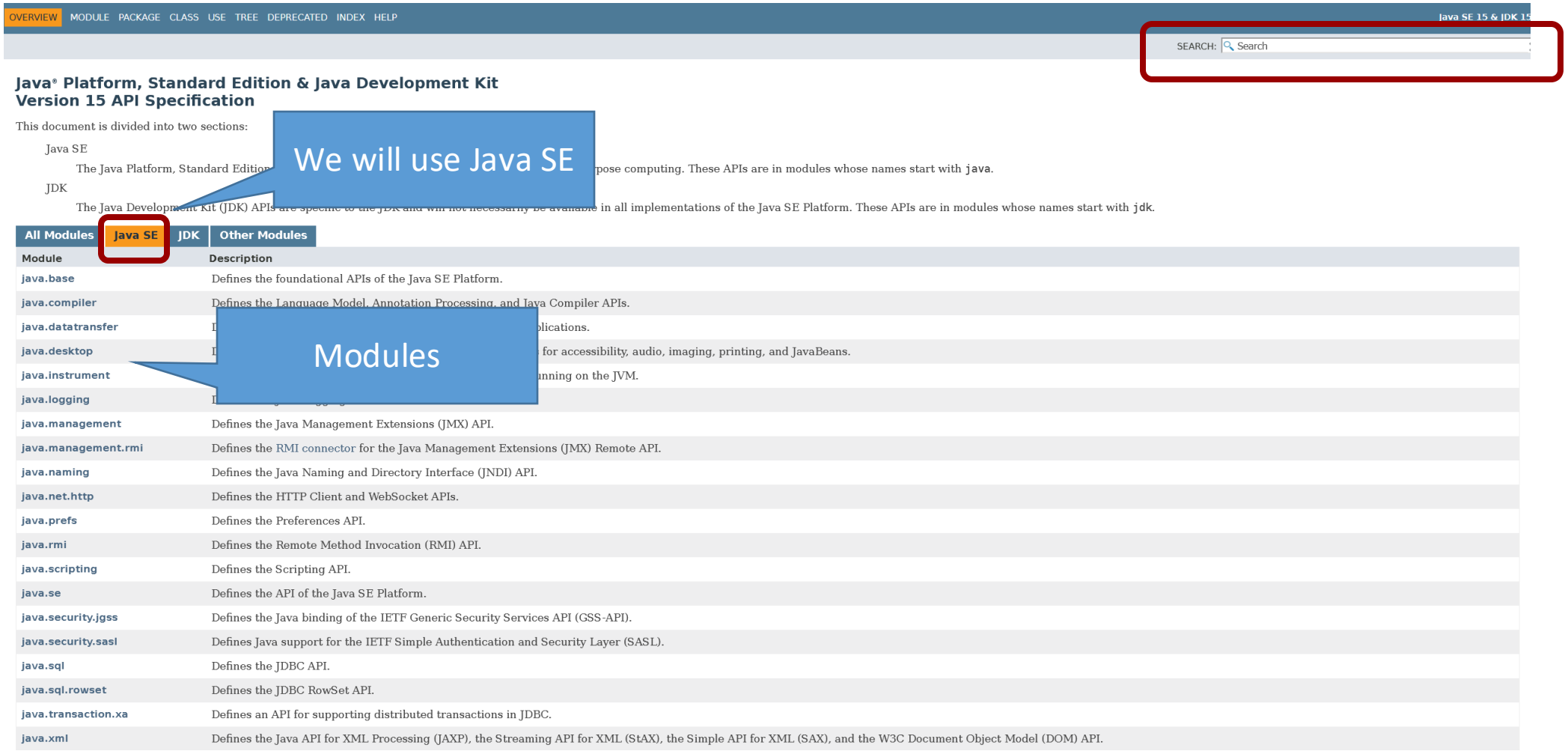
- Try to make your code as readable as possible
- Include high-level comments that explain why you are doing something (much better than a line-by-line commentary of your code)

Code Style / Errors

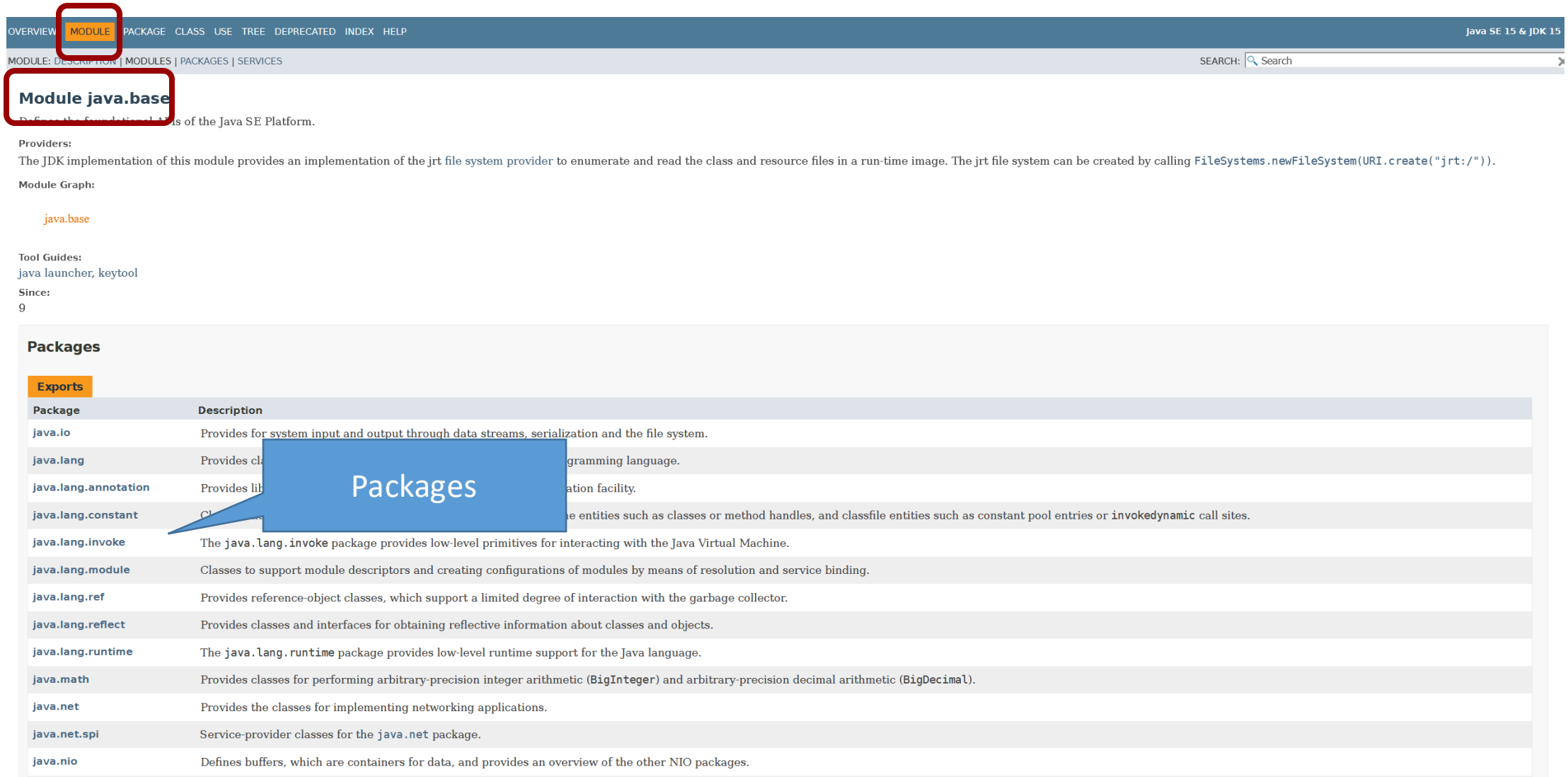
Keep attention what Eclipse reports:



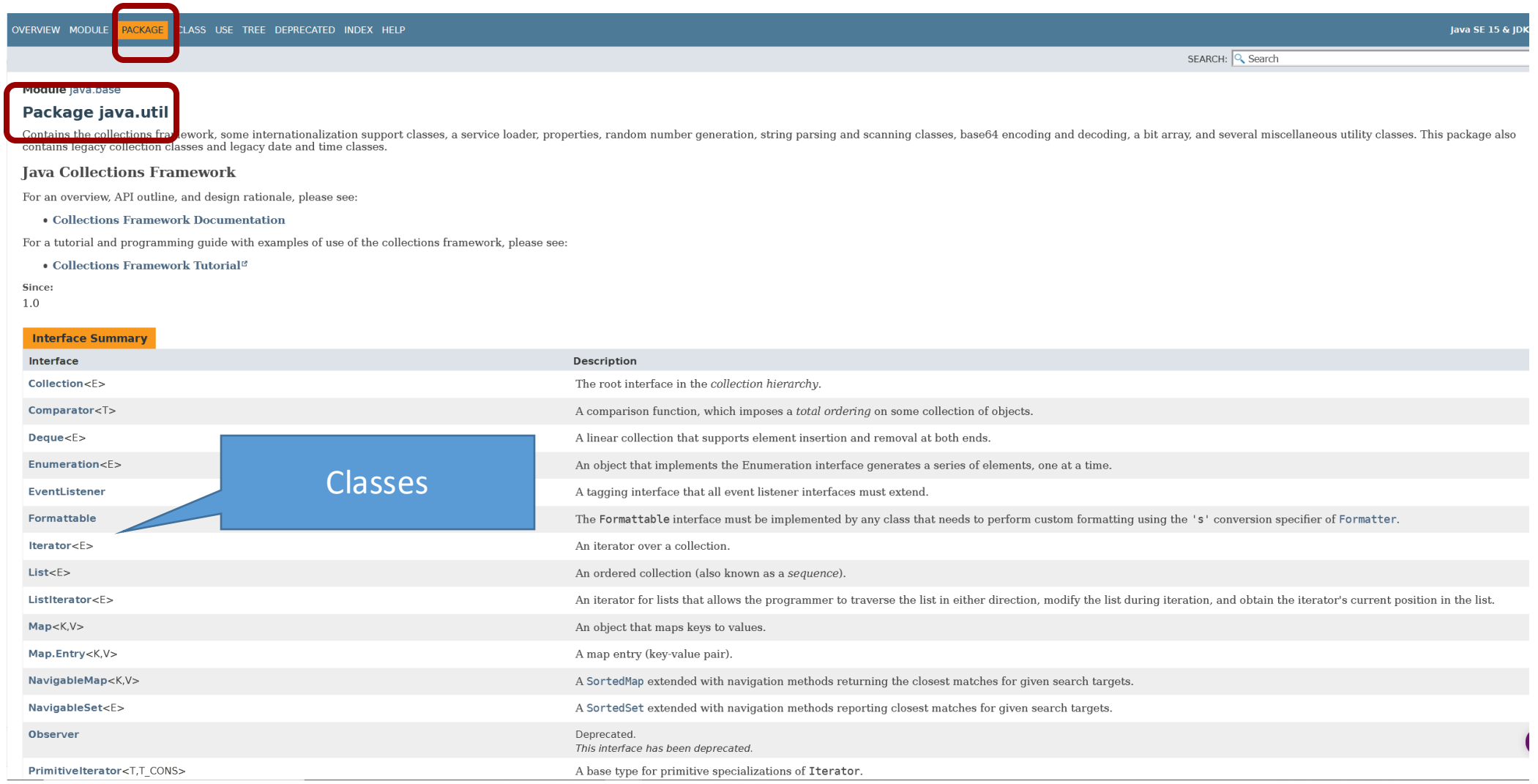
Java Doc (https://docs.oracle.com/en/java/javase/21/docs/api/index.html)



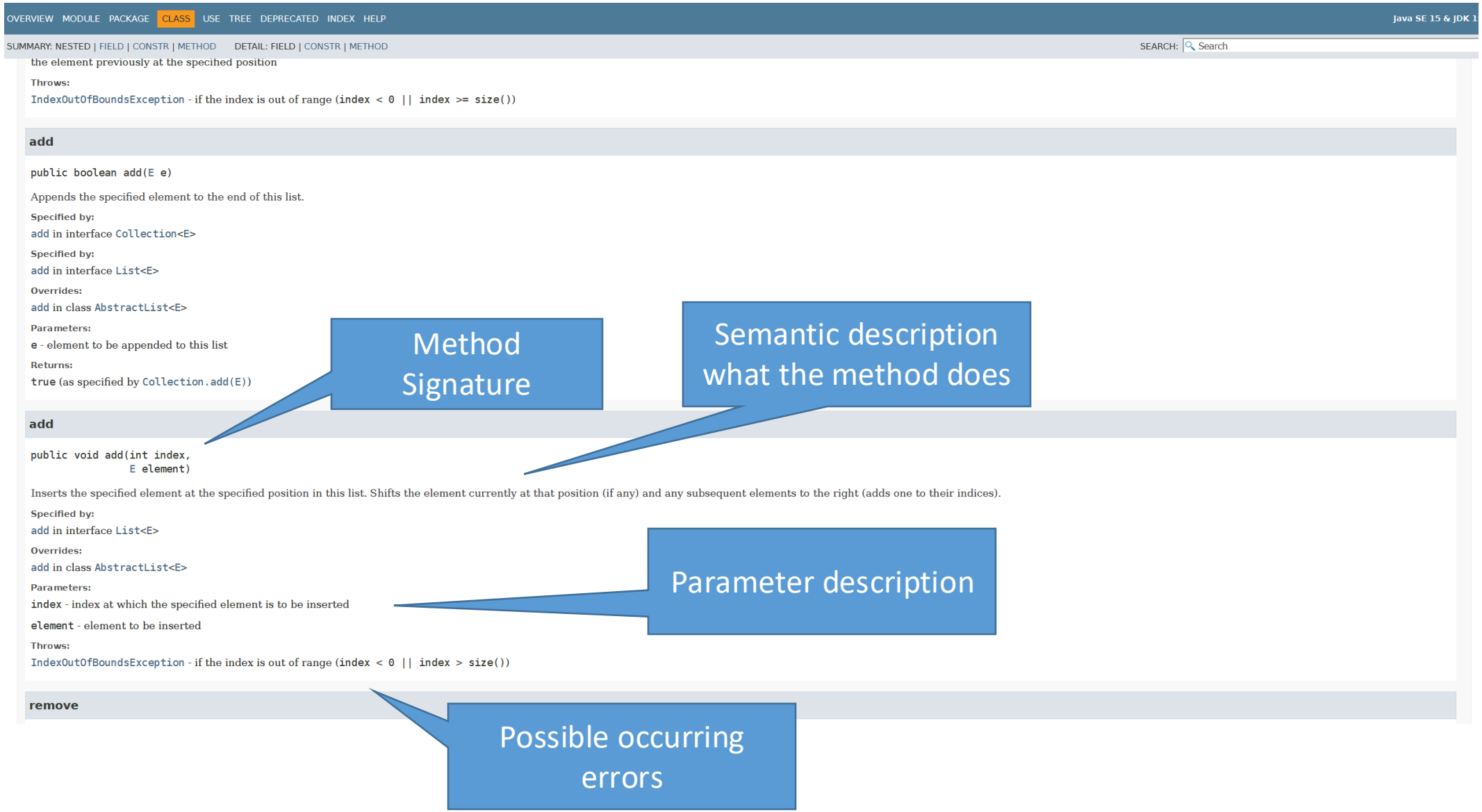
Java Doc (https://docs.oracle.com/en/java/javase/21/docs/api/index.html)



Java Doc (https://docs.oracle.com/en/java/javase/21/docs/api/index.html)



Java Doc (https://docs.oracle.com/en/java/javase/21/docs/api/index.html)



Feedback fürs Assignment

- Auf Gitlab pushen und mir dann eine Nachricht schreiben