Parallel Programming
Exercise Session 2

Spring 2026

anguhl@ethz.ch
annikaguhl.com

Schedule

Preparation assignment 2 5’

Coding Remarks 5’

Theory Recap 15’

Pre-Discussion assignment 2 20’

Break 15’

Quiz 15’

Empfehlung: Lernt den Umgang mit der Command Line

- Nicht relevant für PP ausserhalb vom Erhalten/Abgeben von Übungen, aber es wird euch im weiteren Studium helfen (SPCA, etc.)
- Einen guten Überblick findet ihr hier: https://www.codecademy.com/article/command-line-commands
- Um den Umgang mit der Command Line zu üben, empfehle ich OverTheWire Bandit: https://overthewire.org/wargames/bandit/.

Wie wird man TA?

Grundsätzlich in jedem Fach unterschiedlich

Meistens wird eine gute Note (5.75+) benötigt

- Nach Fach mehr oder weniger wichtig, in manchen Fächern ist gutes Unterrichten wichtiger

Manche Fächer schicken Anfragen per Mail an Leute mit guten Noten

Meine Empfehlung, falls ihr TA werden wollt: Schickt 1-2 Monate im neuen Semester ein Mail an die Head-TAs (und die Dozenten im CC), um zu fragen, wie der Bewerbungsprozess funktioniert

Preparation Exercise 2

Preparations

1. Fetch Assignment 2 from Gitlab upstream
2. Run the projects unit-tests in IntelliJ
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

Pulling Assignment 02

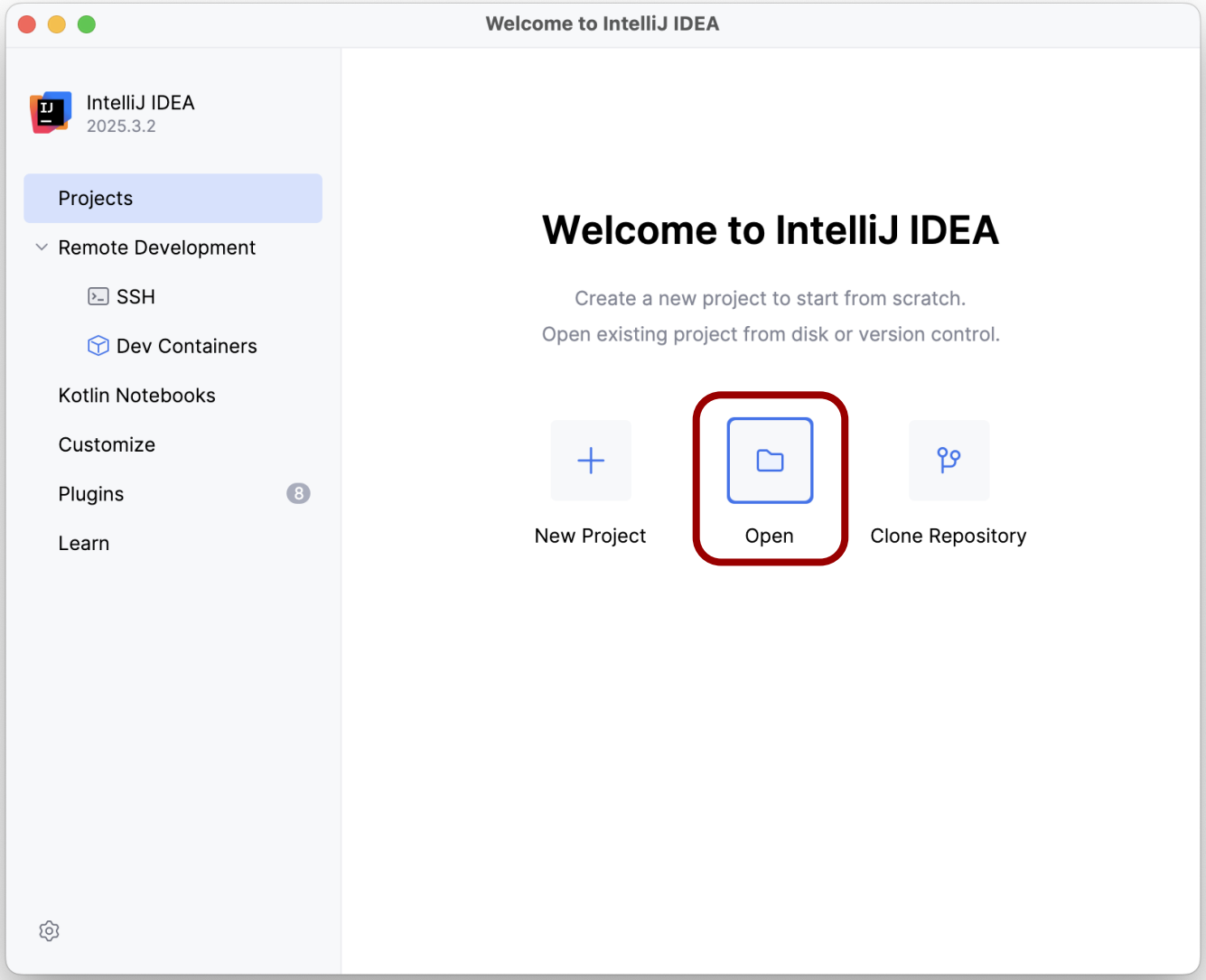
Each week, when new assignments are released, you will have to pull from an upstream repository.

To do this, please use the following commands:

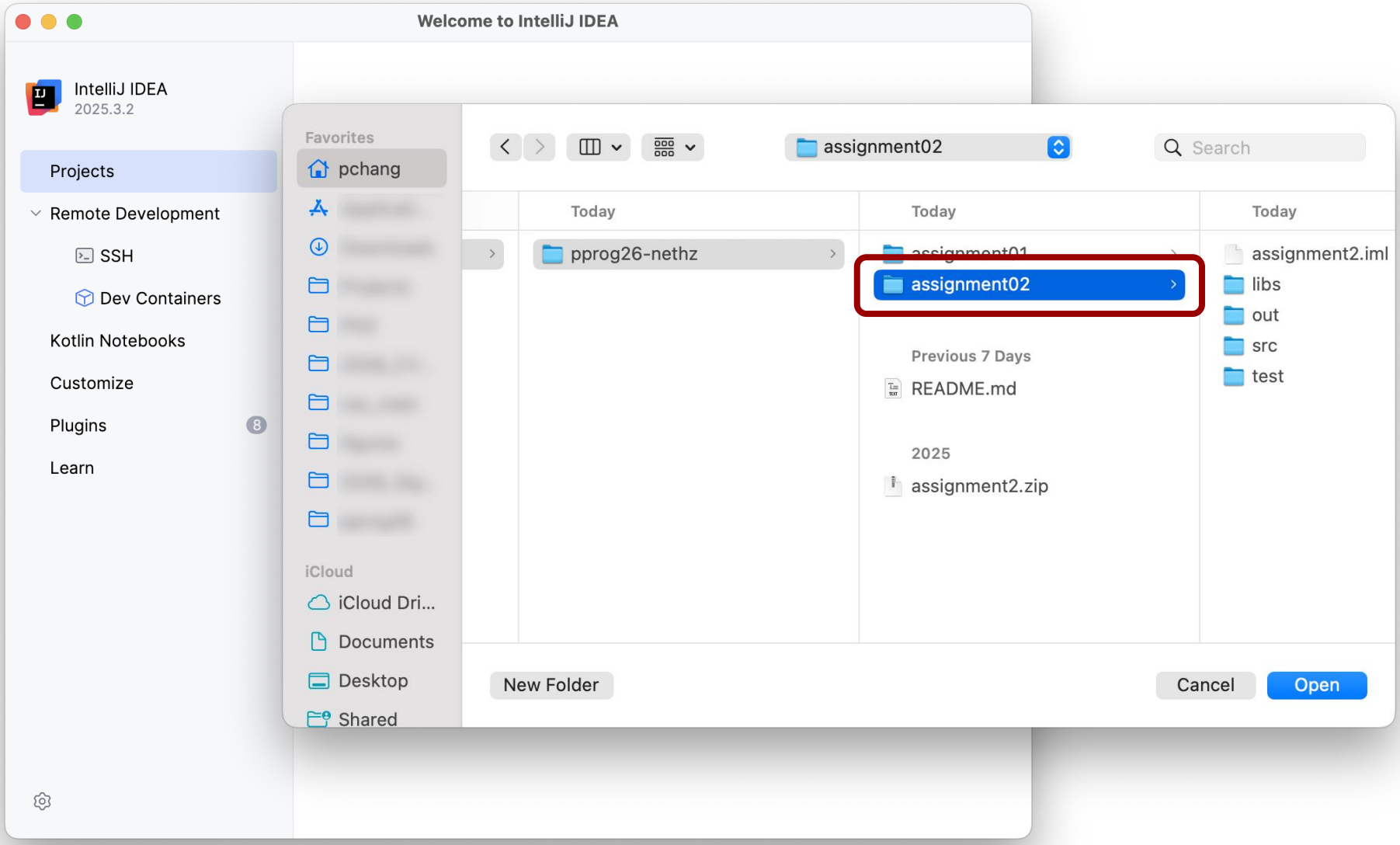
```
# check to see if there is a new assignment
git fetch upstream

# merge your assignment into your repository
git merge upstream/main
```

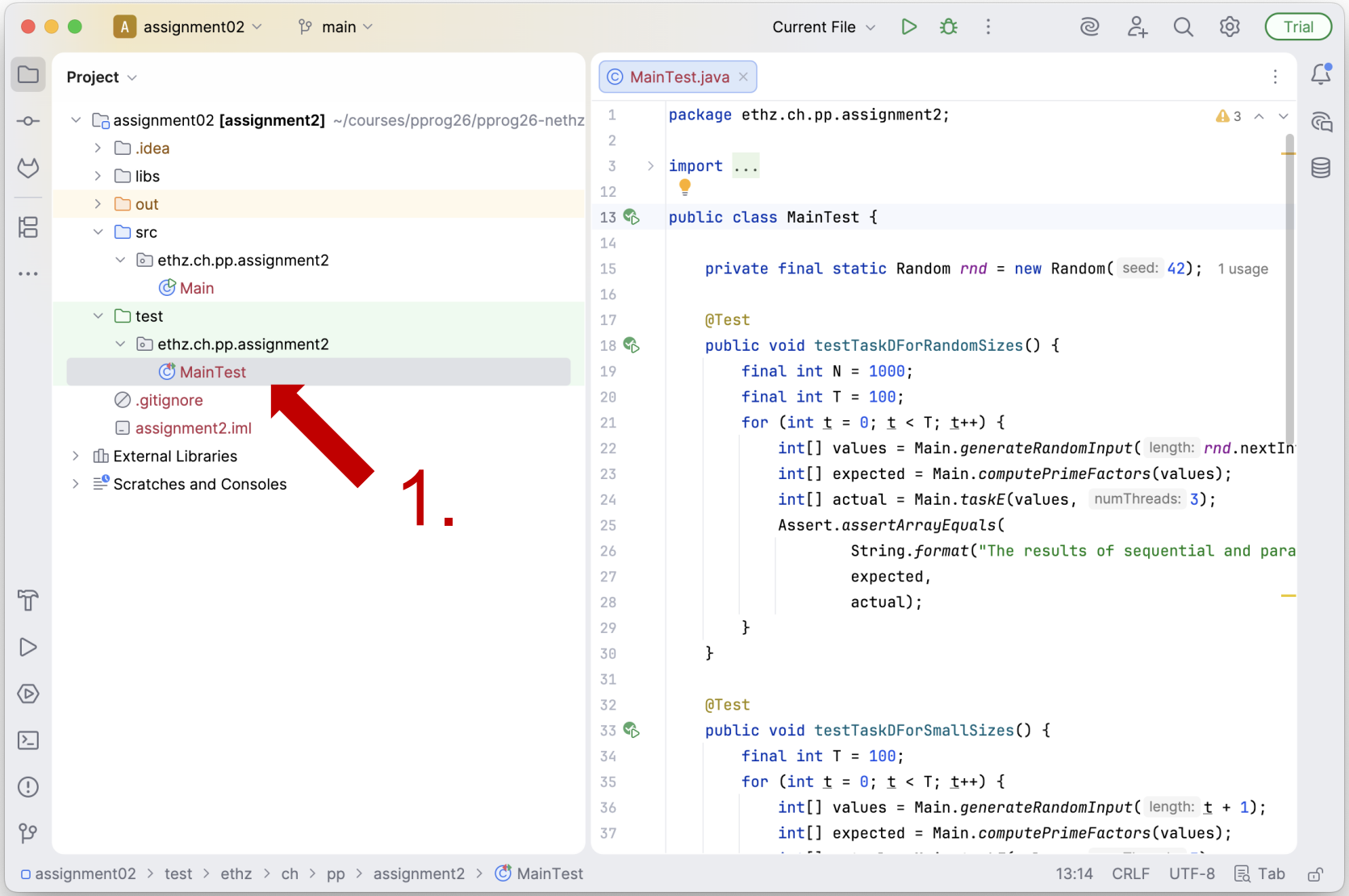
IntelliJ IDEA: Import Project



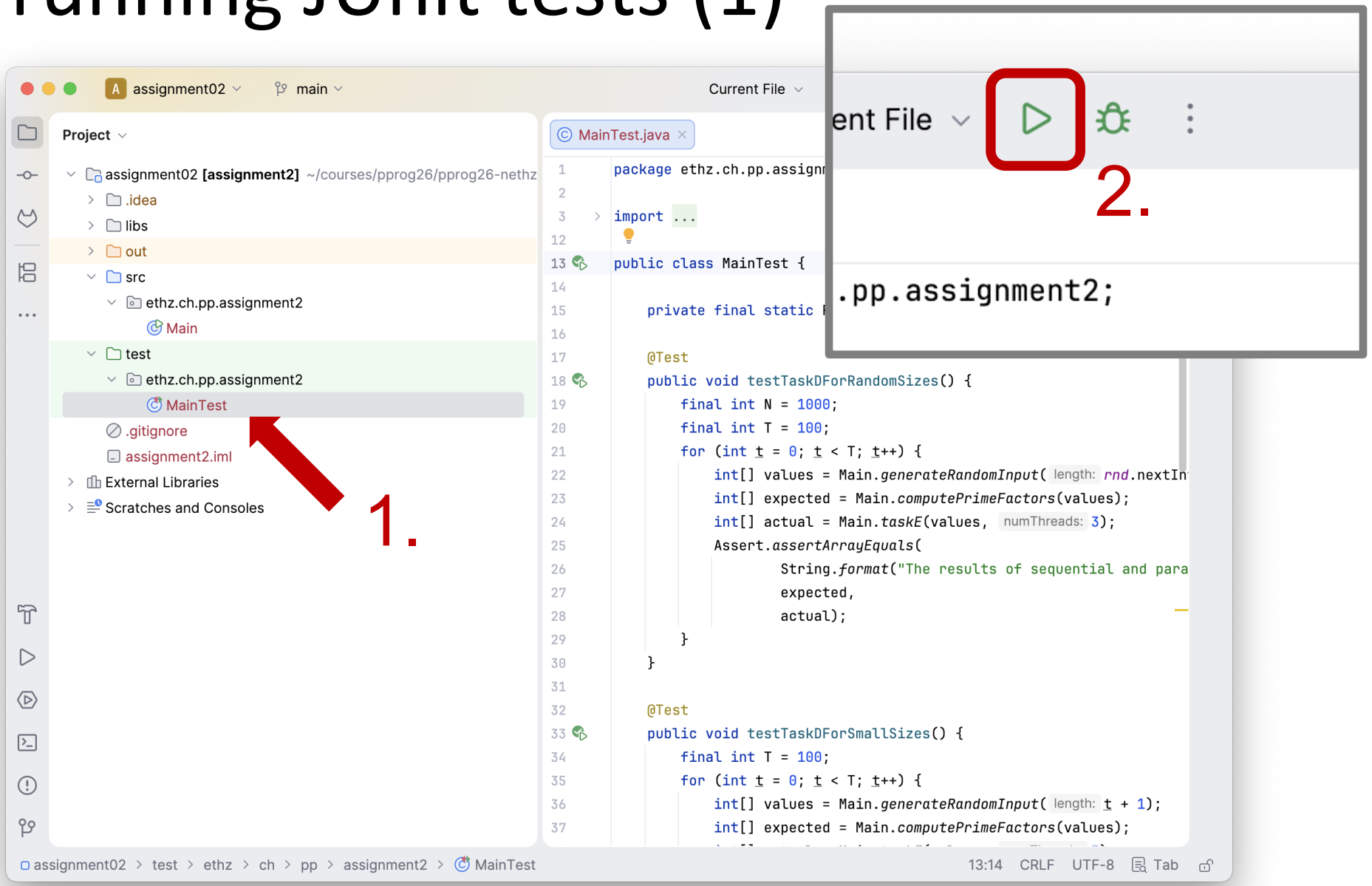
IntelliJ IDEA: Import Project



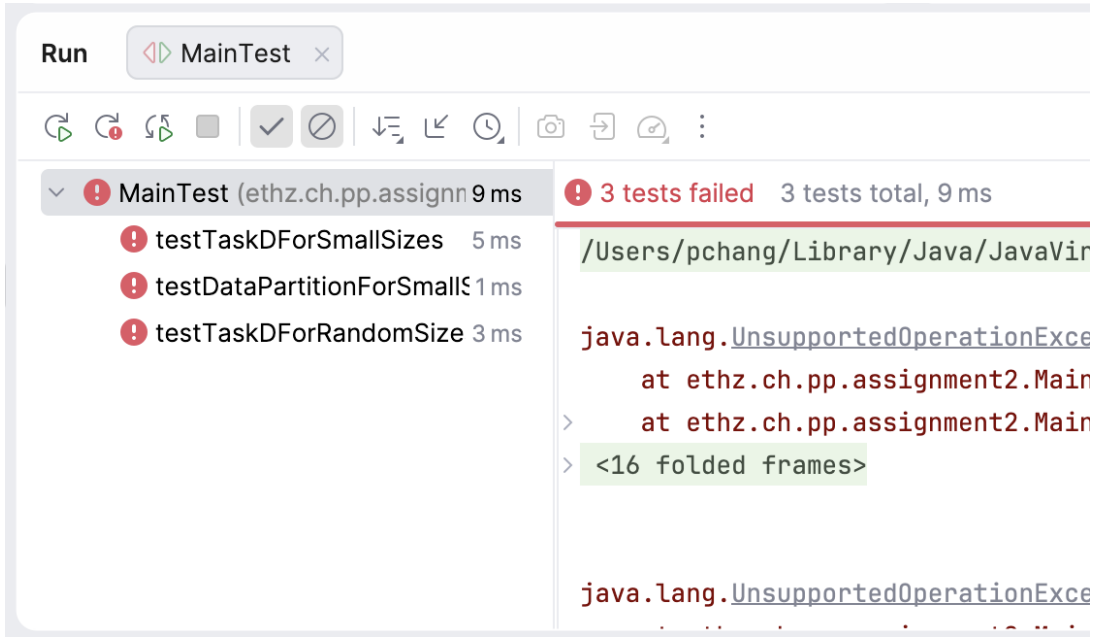
IntelliJ: running JUnit tests (1)

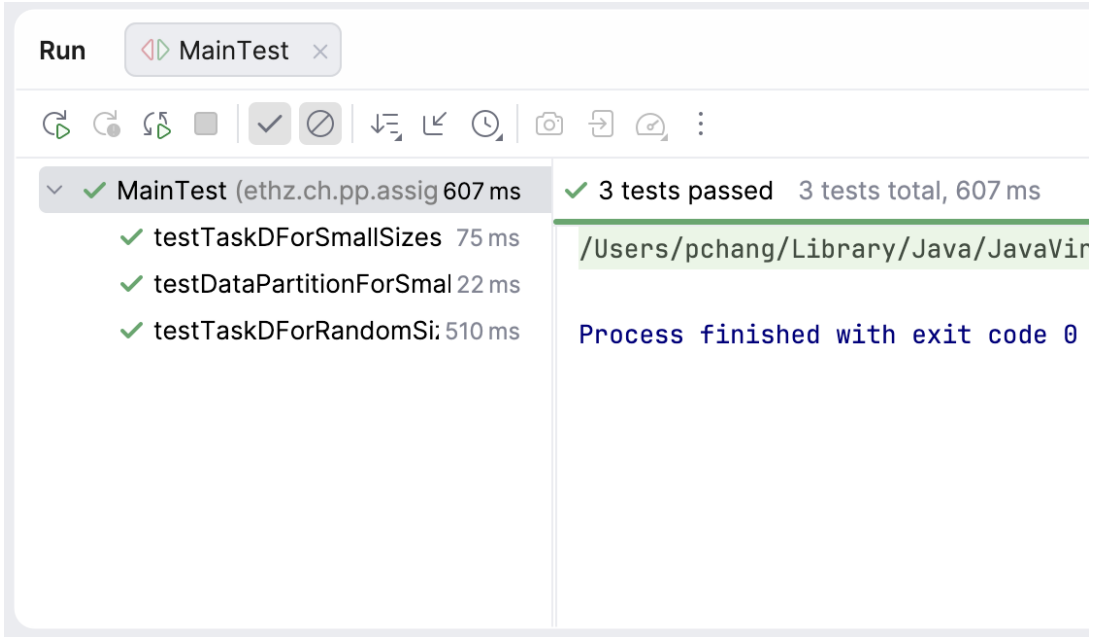


IntelliJ: running JUnit tests (1)



IntelliJ: running JUnit tests (2)





Template

Your solution (ideally)

Coding Remarks

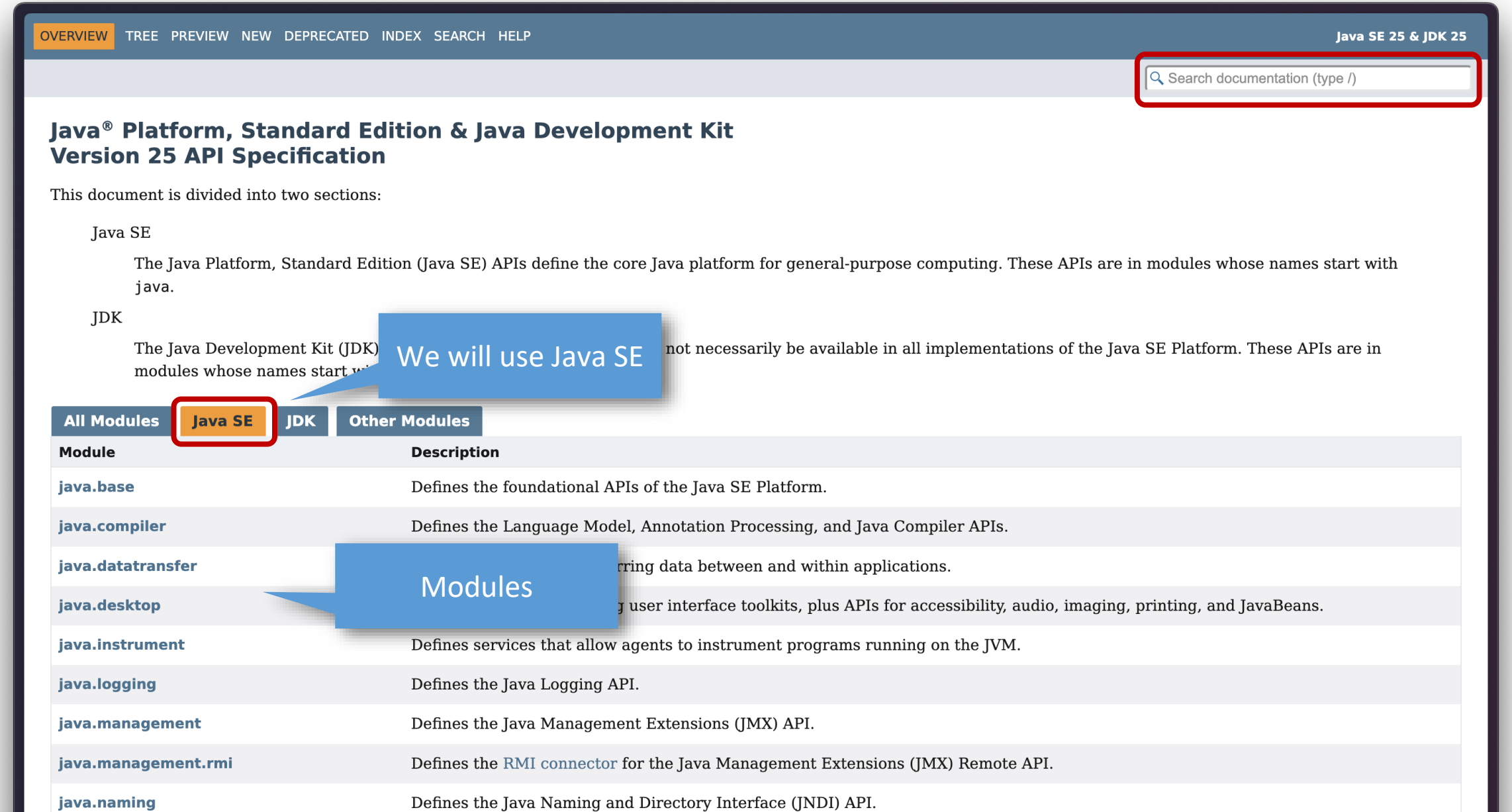
Code Style

- 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)

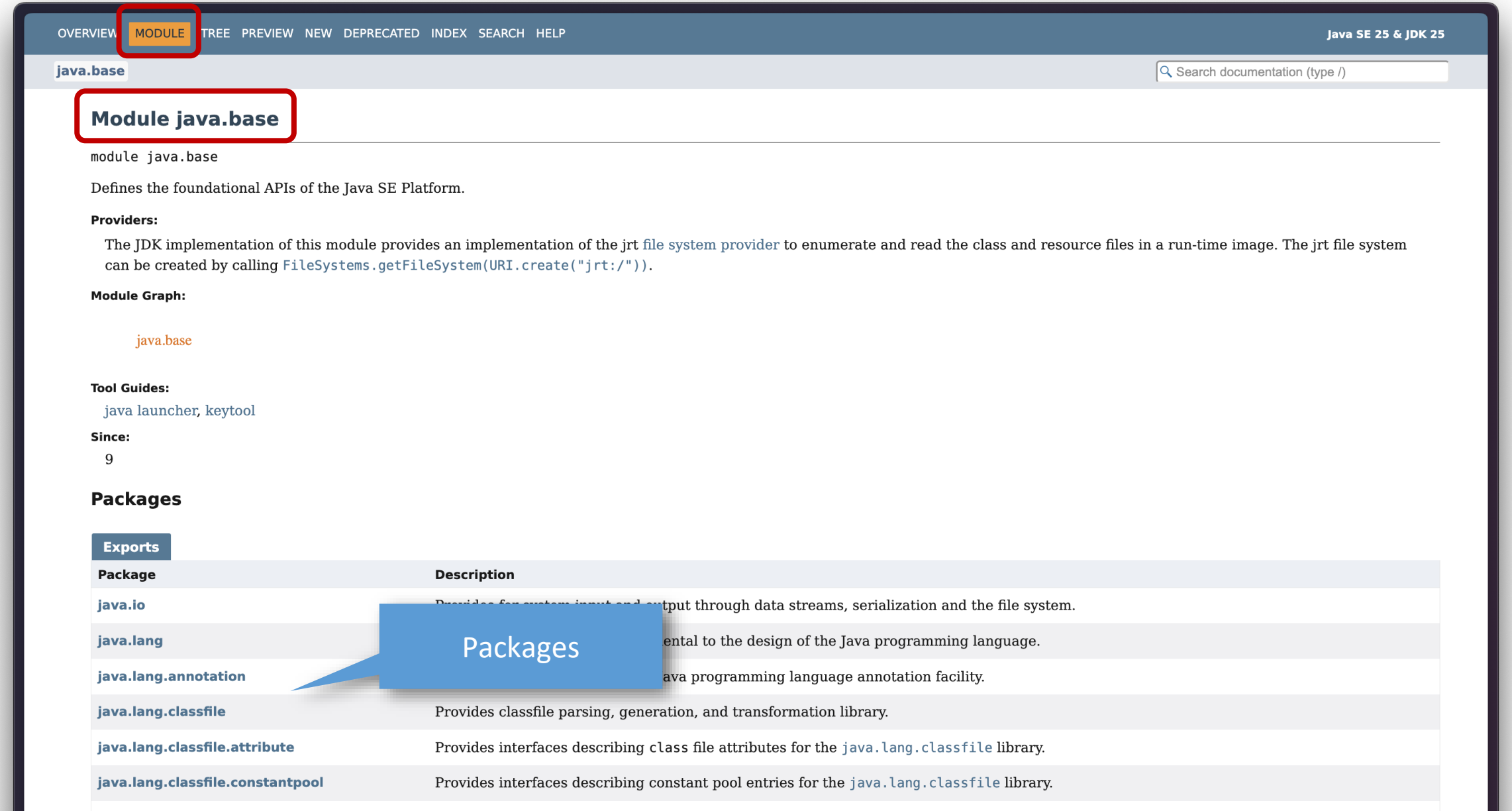
Code Style / Errors

Keep attention what IntelliJ reports:

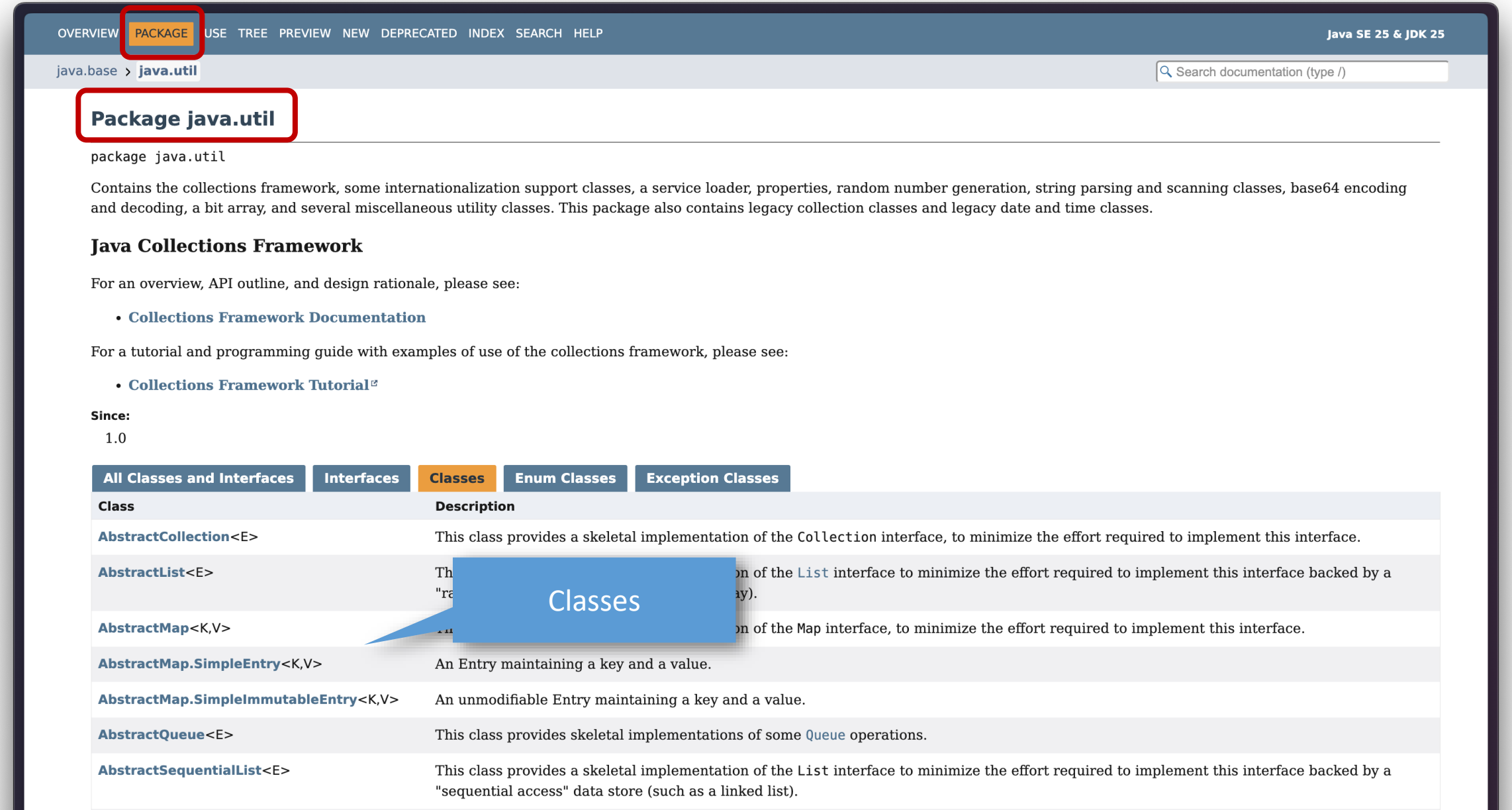
Java Doc (https://docs.oracle.com/en/java/javase/25/docs/api/index.html)



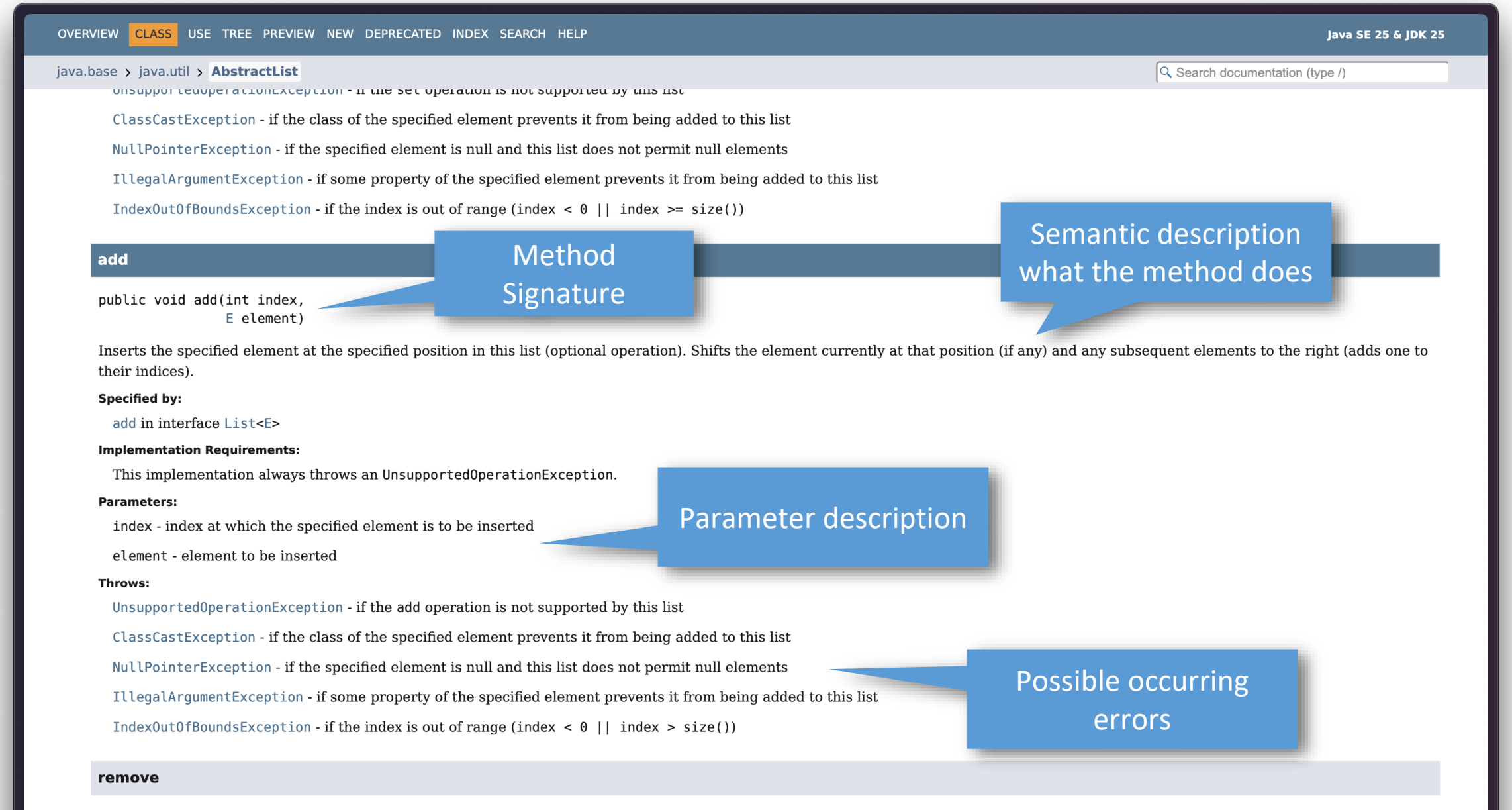
Java Doc (https://docs.oracle.com/en/java/javase/25/docs/api/index.html)



Java Doc (https://docs.oracle.com/en/java/javase/25/docs/api/index.html)



Java Doc (https://docs.oracle.com/en/java/javase/25/docs/api/index.html)



Theory Recap

Terminology

Overview:
https://cgl.ethz.ch/teaching/parallelprog26/pages/terminology.html

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

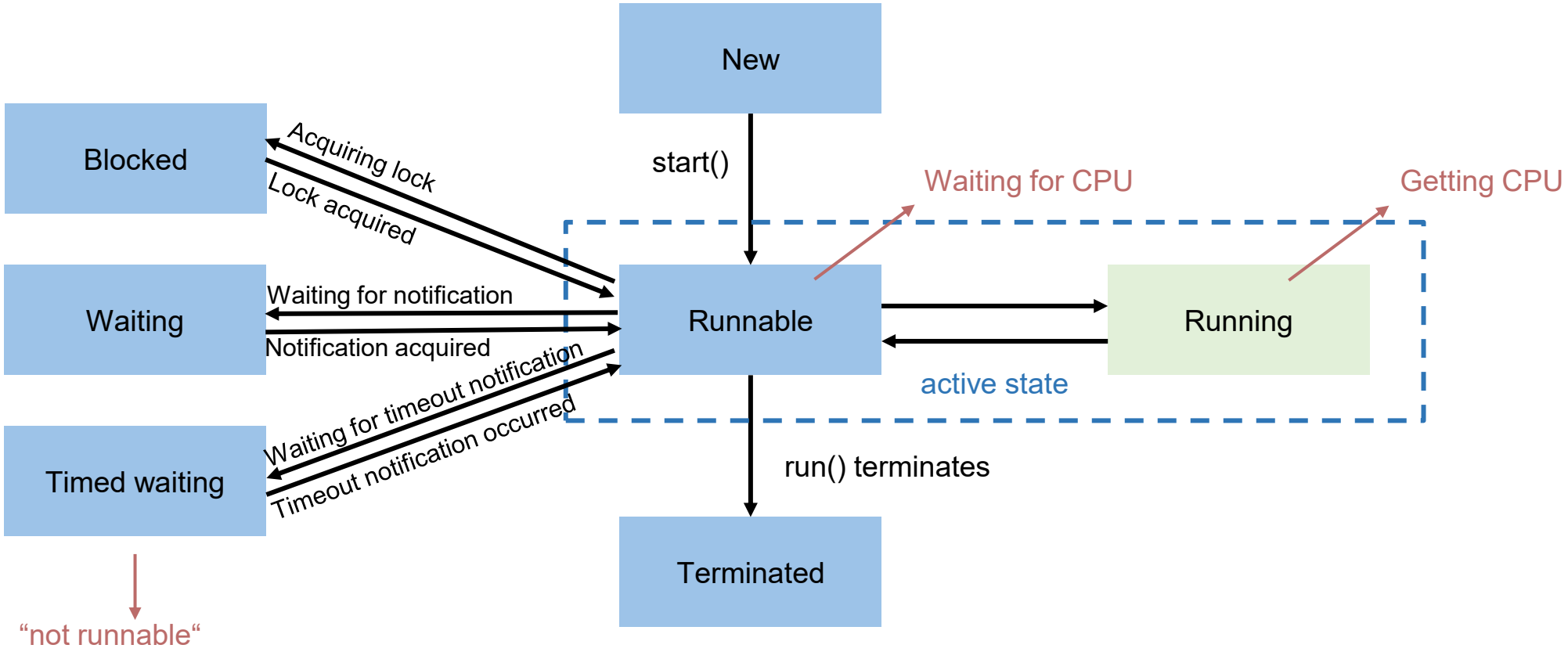
Concept of threads exists on various levels:

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

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

Life cycle of a Thread



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 A: How to create and start a new thread?

option 1: Extend class Thread

```
class ConcurrWriter extends Thread { …
     public void run() { … }
}
ConcurrWriter writerThread = new ConcurrWriter();
writerThread.start();   // calls ConcurrWriter.run()
```

option 2: Implement Runnable

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

ConcurrReader readerThread = new ConcurrReader();
Thread t = new Thread(readerThread);
t.start();   // calls ConcurrReader.run() automatically
```

Demo

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

Mögliche Implementationen:

Measured execution time not always the same

→ Average over multiple runs (the more the better)

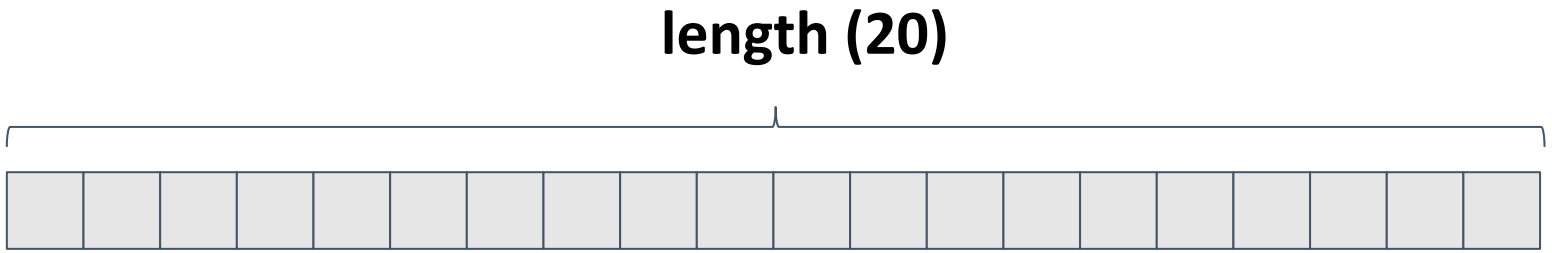
→ Calculate variance

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 an 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) { … }



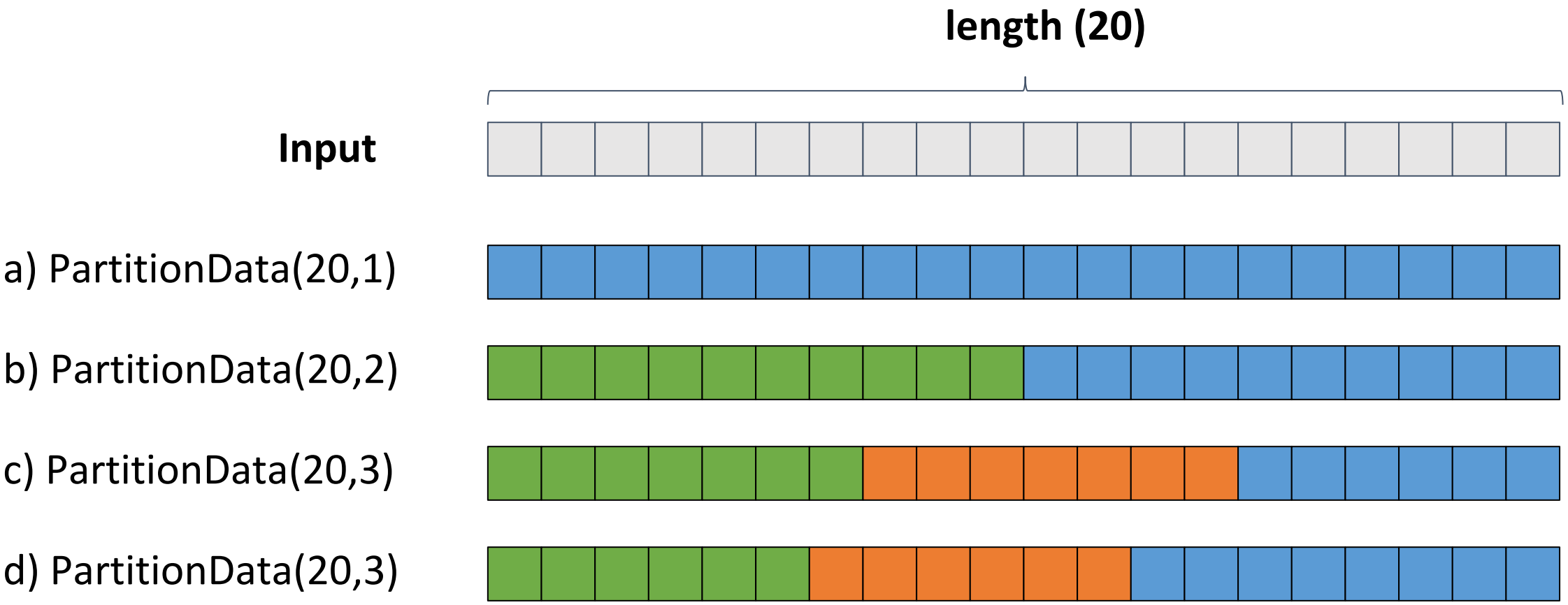
a) PartitionData(20,1) ?

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

Mögliche Implementationen:

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.

Empfehlung

Falls ihr nicht die Zeit habt, alle Aufgaben zu lösen, löst A-E.

F und G sind spannend, falls ihr dazu Zeit habt.

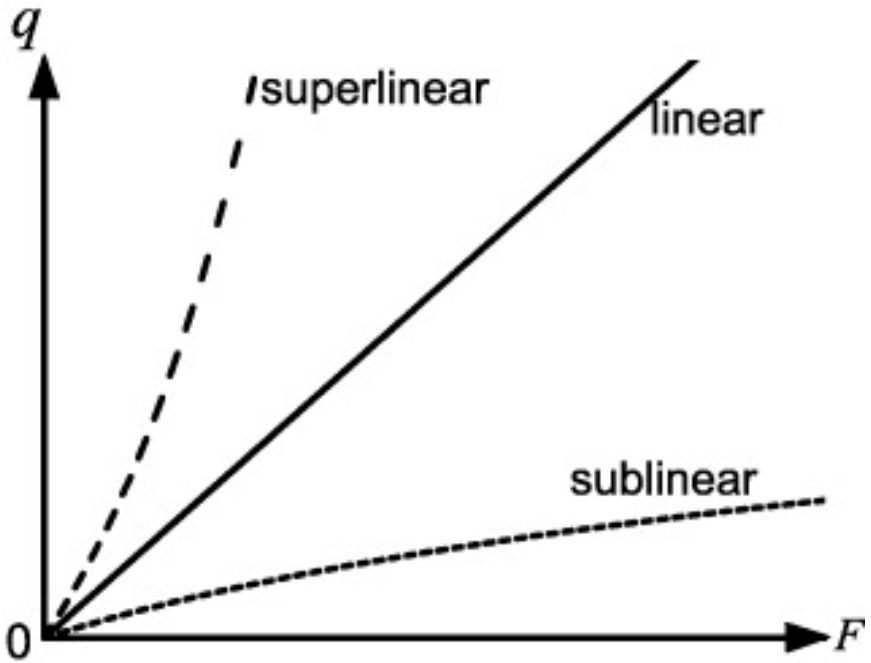
Ihr müsst einige Implementationsdetails selbst entscheiden, nicht alles ist vorgegeben!

Speedup

Sub-linear: usually

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

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



https://softwareengineering.stackexchange.com/questions/347748/can-sub-linear-still-be-a-straight-line

Feedback-Umfrage

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.

Exam, HS 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.

Exam, HS 2023

https://quizizz.com/admin/assessment/65d34c567875b472cb3fa7be?source=lesson_share

Replace link with link to quiz