Pitfalls and Tips

Watch out

- **case** is a reserved keyword in Java
- mind the **IOException** when you use try with resources
- Mind the type used in lambda, there ain't auto boxing.

I/O -NIO

- mind the checked exception within lambda, example: `toRealPath()`
- mark, check if the stream supports mark: `inputStream.markSupported()`
- What I/O classes to use? Reader/Writer for text, Stream for primitives and String values
- Check if the `System.out` or `System.err` is closed within the try-with-resource
- `relativize()` needs both Path to be either absolute or relative
- it's `System.console()` NOT ~~new Console()~~

I/O - IO

- `file.delete()` returns a boolean and does not throw an exception if the file does not exist

Localization

- there ain't ~~DateFormatter~~ only `DateTimeFormatter`

Concurrency

- the Concurrent classes order read/write access such that access to the class is consistent across all threads and processes, while the synchronized classes do not.
- `Future.get()` throws `InterruptedException` and a `ExecutionException`, and they block execution !!!!!!
- scheduled methods: `newScheduledThreadPool(10)`/`newSingleThreadScheduledExecutor()`
- when the method is `synchronized`, check if it's static or instance

Stream

- `IntSummaryStatistics`, not ~~IntegerSummaryStatistics~~.
- Do not confuse `IntStream` with `Stream<Integer>`
- `Stream.concat(s1, s2)` works only with 2 streams, not more parameters. !!!!!
- `Optional.orElseThrow()`
- check if there is more than one terminal operation
- Names: Source / Intermediate operation / Terminal operation. ~~Not Origin~~ !
- Mind `forEach(System.out::println)`; not ~~forEach(System.out.println);~~
- check if the stream ends with an Optional
- `parallelStream()` is defined on Collection, ~~NOT on Stream~~
- Collectors.groupingBy() - requires a Function !
- Collectors.partitioningBy() - requires a Predicate !
-

IntStream

- IntStream.sum() returns an int (not a long)

Lambda

- `var x = ()->"hello"` does not compile! - **you cannot assign a lambda to a var**
- check the right symbol! it's -> NOT =>
- check lambda does not throw a Checked exception
- mind the semicolon when you assign a variable lambda. Example: Predicate predicate = s->true;
- when using curly brackets, check if there is return!

FunctionalInterface

- `IntUnaryOperator` is not with generics. So this does not compile: `IntUnaryOperator<String>`

Various

- check if a field is final in the code
- `enum`: check if the constructor is private, or protected!
- `instanceof` : a class can always be an instance of an Interface, so the instanceof in that case is allowed. !!!!
- mind the scope of variables in the `instanceof`
- variable names: _ can be allowed but not as a char single: `var _` does not compile
- multilines: check if a variable hasn't been used previously
- mind the curley brackets in the do/while

- Local variable type inference requires a value so that the type can be inferred. `var x;` //does not compile
- Arrays Sort String (natural order)
    1. Numbers
    2. Uppercase
    3. Lowercase
- local variables need to be initialized before being used!!
- Do NOT confuse local with instance variables when they have the same name!

Class Design

- an instance variable final either is initialized in the constructor or in the code
- in the setX() check there is this.x=x

check

- `switch` expression: You CANNOT use `return` but you can use `yield`.
- you need the `case` within a `switch` !!!

Collection

- `Deque.offer()` insert element at the end of the queue
- Comparator: `comparing(...).thenComparing()`
- pop()/poll() when Deque is empty: `pop() throws java.util.NoSuchElementException` but `poll()` returns null

Interfaces

- default method: `Plant.super.leaves()` NOT ~~super.Plant.leaves()~~

Classes

- if there is an abstract method makes sure the class is declared `abstract`.!!!!

Nested Classes

- A local class can access only **final** and **effectively final** local variables
- anonymous class needs to have **semicolon** at the end.!!!!!!!!!!!
- MIND THE SEMICOLON!!! `;`

Exception

- Check if the variable for the exception, **e**, is not already defined!!!
- check that main throws the checked exception declared in the method, especially with the AutoClosable resources
- in the try-with-resources check round parenthesis is used, not curly.
- try with resources: check in the variable is in scope

JDBC

- ResultSet rs: `rs.next()` NOT ~~rs.hasNext()~~
- `ResultSet.getInt(1)` NOT ~~ResultSet.getInteger(1)~~
- watch out the overlap of `conn.savepoint()` and `conn.rollback()`
- PreparedStatement will use the existing/previous parameter set if you don't replace it.
- watch out! `conn.prepareCall(sql)` returns a `CallableStatement` !

Serialization

- always check the class implements `Serializable`

java.time

- only `ZonedDateTime` has the `toInstant()` method!!
- You cannot add Duration to LocalDate
- Instant: you can only add days or smaller, otherwise you have exception