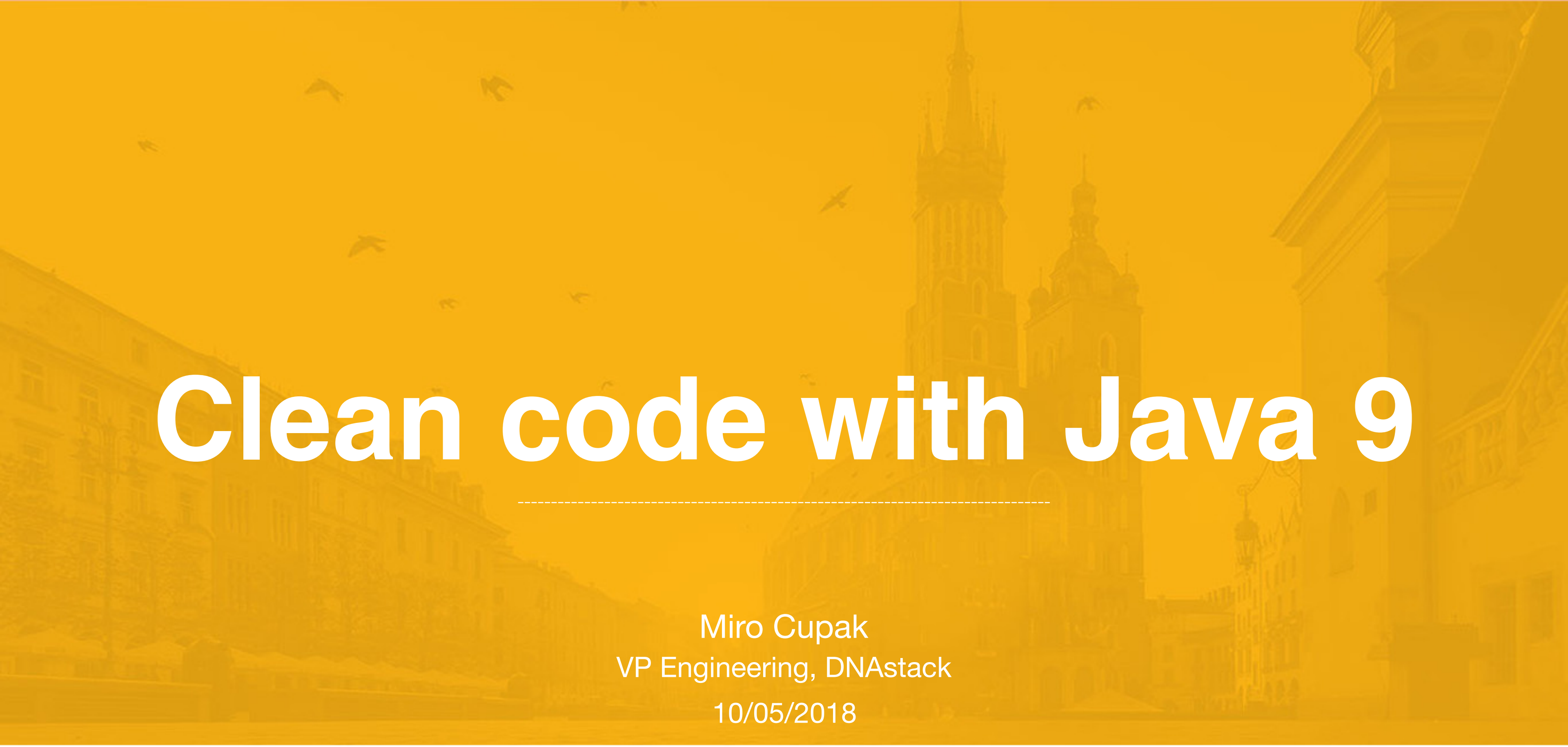


Clean code with Java 9

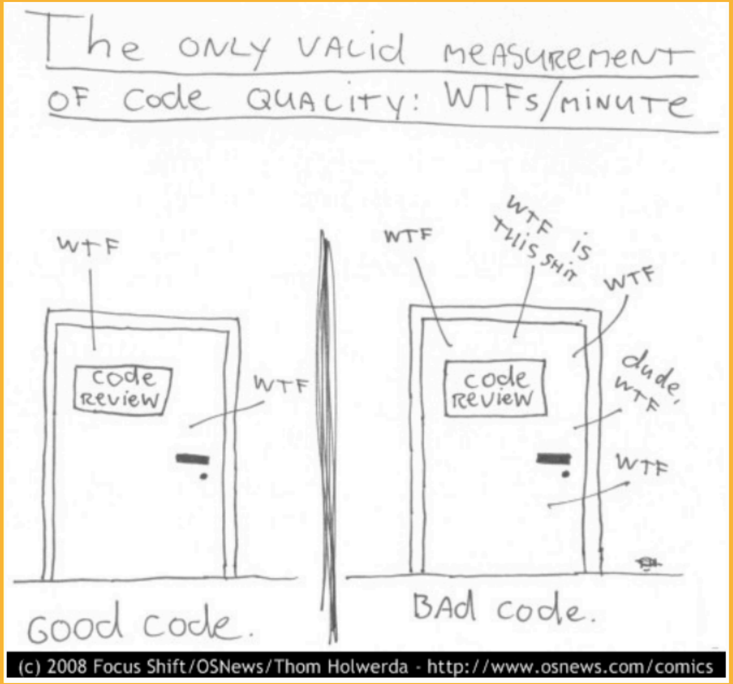
Miro Cupak
VP Engineering, DNAstack

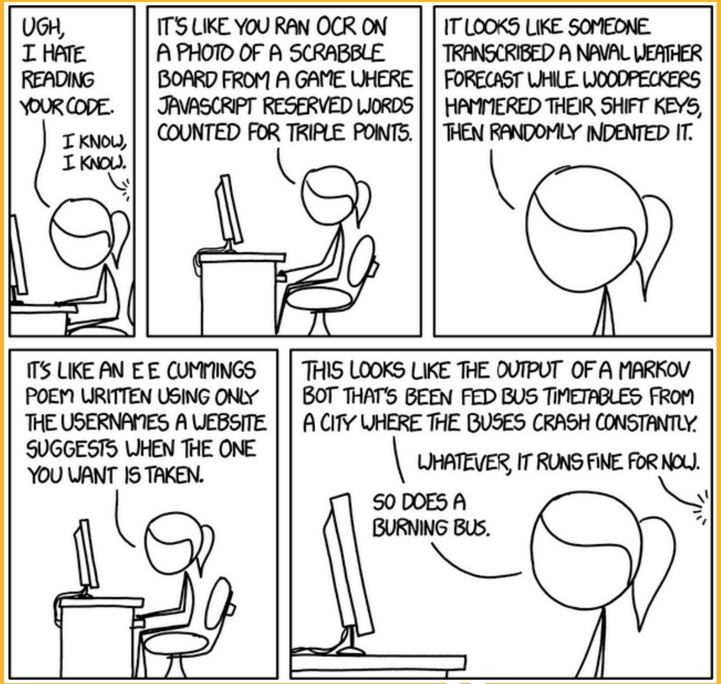
10/05/2018

geecon

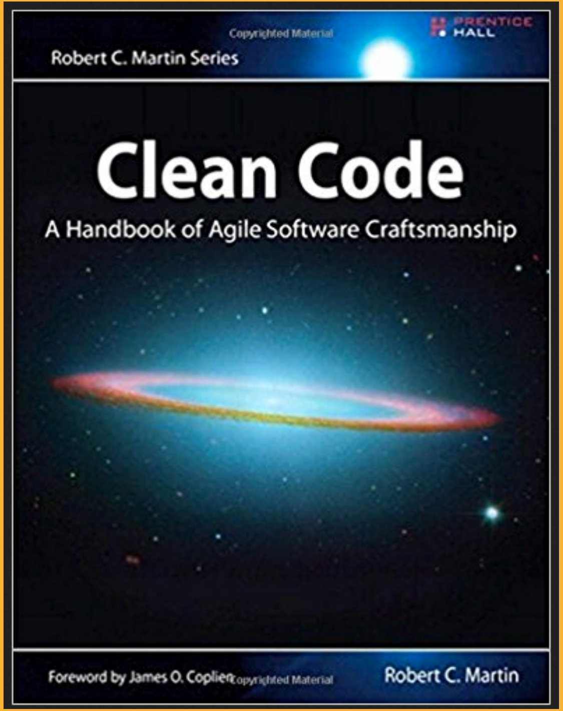
Kraków, 9-11 May 2018

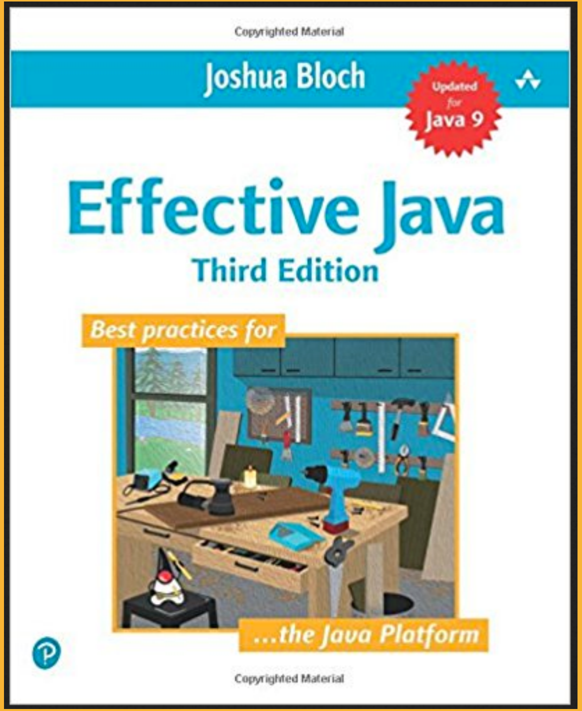
Clean code: clarity, simplicity, brevity, humanity


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
Bad code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics


UGH, I HATE READING YOUR CODE.
I KNOW, I KNOW.
IT'S LIKE YOU RAN OCR ON A PHOTO OF A SCRABBLE BOARD FROM A GAME WHERE JAVASCRIPT RESERVED WORDS COUNTED FOR TRIPLE POINTS.
IT LOOKS LIKE SOMEONE TRANSCRIBED A NAVAL WEATHER FORECAST WHILE WOODPECKERS HAMMERED THEIR SHIFT KEYS, THEN RANDOMLY INDENTED IT.
IT'S LIKE AN E E CUMMINGS POEM WRITTEN USING ONLY THE USERNAMES A WEBSITE SUGGESTS WHEN THE ONE YOU WANT IS TAKEN.
THIS LOOKS LIKE THE OUTPUT OF A MARKOV BOT THAT'S BEEN FED BUS TIMETABLES FROM A CITY WHERE THE BUSES CRASH CONSTANTLY.
WHATEVER, IT RUNS FINE FOR NOW.
SO DOES A BURNING BUS.

https://theamericanscholar.org/writing-english-as-a-second-language/, http://www.osnews.com/story/19266/WTFs_m, https://xkcd.com/1695/





https://www.amazon.ca/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882/

https://www.amazon.ca/Effective-Java-3rd-Joshua-Bloch/dp/0134685997/

Features

- Factory methods for collections.
- Improved try-with-resources.
- Private methods in interfaces.
- Diamond operator with anonymous classes.
- Stream API enhancements.
- Extensions to Optional.
- Stackwalker.
- HTTP/2 client.

Factory methods for collections

Factory methods for collections

- Obtain immutable collections via `of`/`ofEntries` methods.
- Static import `java.util.Map.entry`.
- Less verbose, no static initializer blocks.
- Don't use `Arrays.asList` or `Stream.of` as shortcuts for creating collections.
- Don't use external libraries if you only need immutable collections (Guava).
- No need to worry about leaving references to underlying collections.
- Thread-safe.
- Can be shared freely (no need for defensive copies).
- Good performance.
- Don't create mutable collections unless necessary.

Improved try-with-resources

Improved try-with-resources

- Always prefer try-with-resources, don’t use try-finally and definitely don’t use finalizers to close resources.
- Be aware of convenience methods, such as `InputStream.transferTo`.
- Don’t create unnecessary helper objects.

Private methods in interfaces

Private methods in interfaces

- Default methods, like all other methods, should be short.
- DRY.
- Use private methods to keep default methods short.
- Use private methods to extract shared core of default methods.
- Don’t use default methods to extend existing interfaces unless necessary.
- Use default methods for providing standard implementations for new interface code.

Diamond operator with anonymous classes

Diamond operator with anonymous classes

- Improved readability and consistency.
- Just use as everywhere else.
- Prefer lambdas to anonymous classes.
- Anonymous classes are not legacy (abstract classes, interfaces with multiple abstract methods…).

Stream API enhancements

Stream API enhancements

- Be aware of new stream methods: `takeWhile`, `dropWhile`, `iterate`.
- Check for convenience stream methods before converting to streams manually (e.g. `LocalDate`, `Matcher`).
- Avoid unnecessary null checks with `ofNullable`.
- Streams are suitable for more use cases now, but not all use cases.
- Don’t overuse streams as they can make code hard to read and difficult to maintain.

Extensions to Optional

Extensions to Optional

- Use `ifPresentOrElse()` instead of if-isPresent construct.
- `or()` provides a clean fluent way of chaining behaviour on Optionals.
- Use `stream()` to take advantage of the lazy nature of streams and handle streams of Optionals.
- Remember that `isPresent` is rarely the answer.

Stackwalker

Stackwalker

- Prefer collections and streams to arrays.
- Access stacktraces lazily via `walk()`.
- Take advantage of the Stream API to access only certain elements.
- Be aware of `StackWalker.Option`. Don't resolve classes manually.

HTTP/2 client

HTTP/2 client

- Clean separation: `HttpClient`, `HttpRequest`, `HttpResponse`.
- `HttpURLConnection` is not pleasant to use.
- Avoid APIs with side effects.
- The new client API is versatile, flexible and clean.
- Prefer functionality in the JDK to external libraries.
- But aware it's an incubator module.

Q&A

More info:

Session notes on Twitter.

Blog: https://mirocupak.com/