Upgrade to Modern Java

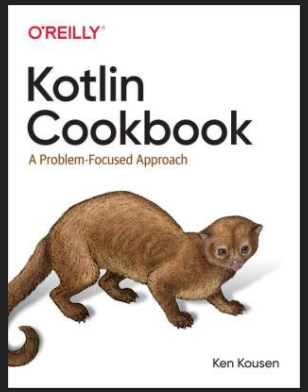
NFJS Webinar

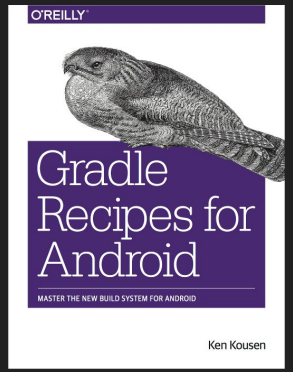
Contact Info

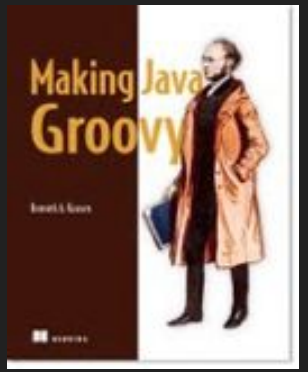
Ken Kousen
Kousen IT, Inc.

ken.kousen@kousenit.com
http://www.kousenit.com
http://kousenit.org (blog)
@kenkousen (twitter)

https://kenkousen.substack.com
(newsletter, *Tales from the jar side*)



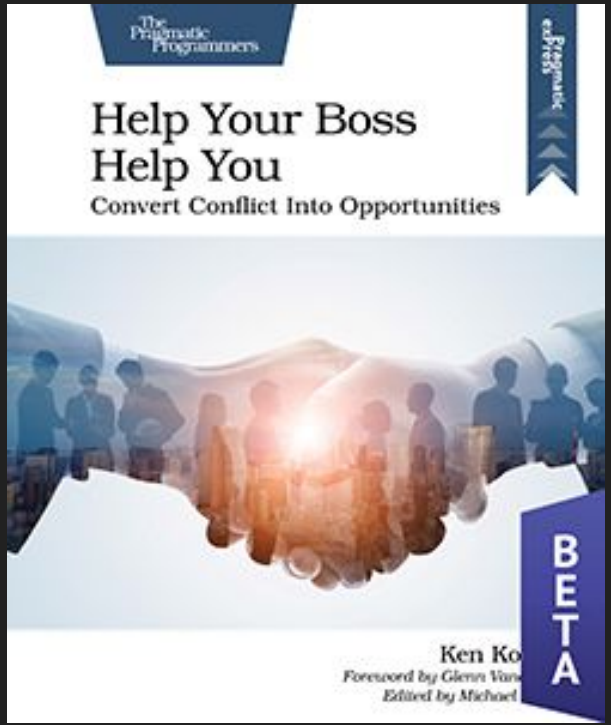




New Book

Help Your Boss Help You

https://pragprog.com/titles/kkmanage/help-your-boss-help-you/



GitHub Repository

https://github.com/kousen/java_latest

Contains examples from Java 8 through 16, including preview features

Uses Gradle to build and run tests

Updated whenever new Java version is available

The Basics

- Lambda Expressions: (args) → expression
- Method References: ref::method and Class::method
- Streams: pass data through a pipeline

Functional Interface

Interface with a Single Abstract Method

```
Runnable, FileFilter, ...
```

Lambdas can only be assigned to

functional interfaces

Functional Interface

- See `java.util.function` package
- Package added in Java 8
  - Handles cases of zero, one, or two arguments

Functional Interfaces

Consumer → single arg, no result

```
void accept(T t)
```

Predicate → returns boolean

```
boolean test(T t)
```

Supplier → no arg, returns single result

```
T get()
```

Function → single arg, returns result

```
R apply(T t)
```

Functional Interfaces

Primitive and binary variations

```
Consumer

    IntConsumer, LongConsumer,

    DoubleConsumer,

    BiConsumer<T,U>
```

Functional Interfaces

BiFunction → binary function from T and U to R

```
R apply(T, U)
```

UnaryOperator extends Function
(T and R same type)

BinaryOperator extends BiFunction
(T, U, and R same type)

Method References

Method references use :: notation

```
System.out::println
    x → System.out.println(x)
Math::max
    (x,y) → Math.max(x,y)
String::length
    x → x.length()
String::compareToIgnoreCase
    (x,y) → x.compareToIgnoreCase(y)
```

Default methods

- Default methods in interfaces
  - Use keyword `default`

Default methods

What if there is a conflict?

Class vs Interface → Class always wins

Interface vs Interface →

- Child overrides parent
- Otherwise compiler error

Static methods in interfaces

- Can add static methods to interfaces
  - See `Comparator.comparing`

Streams

A sequence of elements

- Does not store the elements
- Does not change the source
- Operations are lazy when possible
- Intermediate operations return a new stream
- Closed when terminal expression reached

Streams

Easy to parallelize

Replace `stream()` with

`parallelStream()`

Transforming Streams

Process data from one stream into another

```
filter(Predicate<T> p)

map(Function<T,R> mapper)

Stream<R> flatMap(Function<T, Stream<R>> mapper)
```

Optional

Alternative to returning object or null

`Optional<T>` value

- `isPresent()` → boolean
- `get()` → return the value

Goal is to return a default if value is null

Optional

`ifPresent()` accepts a consumer

```
optional.ifPresent( … do something …)
```

`orElse()` provides an alternative

```
optional.orElse(... default …)
optional.orElseGet(Supplier<? extends T> other)
optional.orElseThrow(Supplier<? extends X> exSupplier)
```

Deferred execution

Logging

```
log.info("x = " + x + ", y = " + y);
```

String formed even if not info level

```
log.info(() -> "x = " + x + ", y = " + y);
```

Only runs if at info level

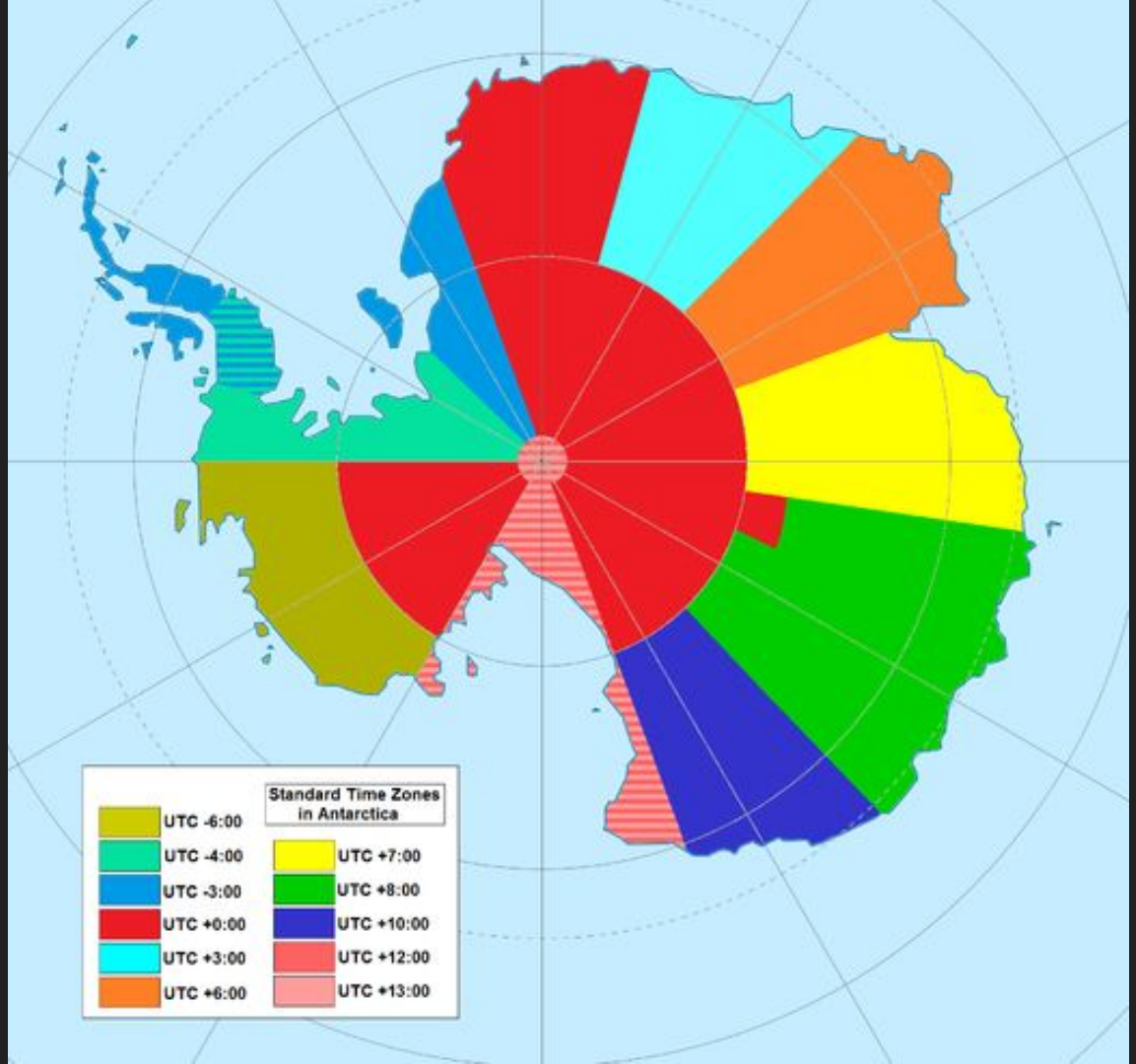
Arg is a `Supplier<String>`

Date and Time API

`java.util.Date` is a disaster

`java.util.Calendar` isn't much better

Now we have `java.time`


Standard Time Zones in Antarctica
UTC -6:00
UTC -4:00
UTC -3:00
UTC +0:00
UTC +3:00
UTC +6:00
UTC +7:00
UTC +8:00
UTC +10:00
UTC +12:00
UTC +13:00

Dates and Times

Java 8 Date-Time: `java.time` package

AntarcticaTimeZones.java

Collection Factory Methods (JDK 9)

List.of, Set.of, Map.of, Map.ofEntries

Collection Factory Methods

```
List.of(a, b, b, c, ...)

Set.of(a, b, b, c, ...)

Map.of(k1, v1, k2, v2, k3, v3, ...)

Map.ofEntries(
    Map.entry(k1, v1),
    Map.entry(k2, v2),
    Map.entry(k3, v3), …)
```

Local Variable Type Inference (JDK 10 and 11)

The var reserved type name

var Data Type

Local variables only

- No fields
- No method parameters
- No method return types

var is a "reserved type name", not a keyword

Can also use on

- for loops
- try-with-resources blocks

Features You Should Probably Know

HTTP Client (JDK 11)

Built-in sync and async networking

HTTP 2 Client

- New HTTP Client API
  - Supports HTTP/2 and websockets
  - Replaces HTTPURLConnection
- Both synchronous and asynchronous modes

JShell (JDK 9)

The Java REPL

JShell

Java interpreter

https://docs.oracle.com/en/java/javase/11/jshell/introduction-jshell.html

> `jshell` (or add -v for verbose)

jshell>

/exit to leave

No semicolons needed

Enhanced Switch (JDK 14)

Makes switch useable

Enhanced Switch

- Expressions return a value
- Arrow rather than colon → no fall through
- Multiple case labels
- Statement blocks → yield
- Exhaustive

Text Blocks (JDK 15)

Multiline Strings

Text Blocks

- Use "triple double" quotes (""") *and a newline*
- Indentation based on closing """
- `stripIndent`, `indent`, `translateEscapes`

Records (JDK16)

Data classes

Records

- Intended to hold data
- Add attributes using constructor syntax
- generates getter methods (but not following the convention!)
- final
- extends java.lang.Record
- generates toString, equals, and hashCode
- can add static fields

Pattern Matching (JDK 16)

Modifies instanceof

Pattern matching

- Enhances the `instanceof` operator
- `if (shape instanceof Square s)` → use square methods on s
- Like a "smart cast"

See also "Java Feature Spotlight: Pattern Matching"
by Brian Goetz in InfoQ
https://www.infoq.com/articles/java-pattern-matching/

Sealed Classes (JDK 15 and 16 preview)

- Sealed classes (or interfaces!) can only be extended by permission
- Use sealed modifier
- Use permits clause for subclasses
- Use non-sealed to open up children
  - First ever Java keyword with a hyphen :)
- Children are open unless final
- All classes must be in the same module
  - If unnamed module, same package

Sealed classes

```
1  package com.example.geometry;
2
3  public abstract sealed class Shape
4      permits Circle, Rectangle, Square {...}
5
6  public final class Circle extends Shape {...}
7
8  public sealed class Rectangle extends Shape
9      permits TransparentRectangle, FilledRectangle {...}
10 public final class TransparentRectangle extends Rectangle {...}
11 public final class FilledRectangle extends Rectangle {...}
12
13 public non-sealed class Square extends Shape {...}
```

Miscellaneous Features

Deprecated Annotation

`@Deprecated` now has fields:

- `forRemoval`
- `since`

Tool `jdeprscan` to scan a jar file for deprecated uses

SafeVarargs (JDK 9)

Until Java 8, `@SafeVarargs` could only be applied to:

- static methods
- final methods
- constructors

In Java 9, can add @SafeVarargs to private methods

Features You Can Probably Skip

The Module System (JDK 9)

JPMS

Module descriptors

`module-info.java`

exports, requires, opens, …

Quick start guide:

http://openjdk.java.net/projects/jigsaw/quick-start

State of the Module System

http://openjdk.java.net/projects/jigsaw/spec/sotms/

Summary

- Functional programming
  - Streams with map / filter / reduce
  - Lambda expressions
  - Method references
  - Concurrent, parallel streams
- Optional type
- Collectors and Comparators
  - Conversion from stream back to collections
  - Enable sorting, partitioning, and grouping
- Date/Time API
  - Good reason to upgrade