Latest Features In Java

•••

Additions to Java from 1.8 to 20

Contact Info

Ken Kousen

Kousen IT, Inc.

ken.kousen@kousenit.com

http://www.kousenit.org

http://kousenit.org (blog)

@kenkousen (twitter)

@kenkousen@mastodon.social (mastodon)

Tales from the jar side (free newsletter)

https://kenkousen.substack.com

https://youtube.com/@talesfromthejarside

GitHub Repository

Java Latest

https://github.com/kousen/java_latest

Documentation pages

https://docs.oracle.com/en/java/javase/17/ (replace 17 with any other version)

- Tools Reference
- JShell User Guide
- Javadoc Guide

Note: Actual API Javadocs are at:

https://docs.oracle.com/en/java/javase/17/docs/api/index.html

Java Licensing Is a Mess, But...

Java is Still Free 3.0.0 - Java Champions

Java 8

- End of life without commercial support (ended Jan 2019)
- Open JDK (and others) still provide updates

Java 11+

- Oracle JDK requires license for production use
- Open JDK (and others) are free

JDK Proposals

You can see which JEPs (Java Enhancement Proposals) in which version:

http://openjdk.org/projects/jdk/

Click on the JDK version number to see what was included in each

Features You Need To Know

Java Functional Features (JDK 8)

Streams, lambdas, method references

Lambda Expressions

Java lambda expressions

Assigned to functional interfaces

Parameter types inferred from context

```
Predicate<String> evenFilter = s → s.length() % 2 == 0
```

Predicate: functional interface with generic type
Lambda: RHS expression

Functional Interface

Interface with a Single Abstract Method

Lambdas (and method references) can only be assigned to functional interfaces

Functional Interfaces in the JDK

See `java.util.function` package

`@FunctionalInterface`

Not required, but used in library

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 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)
```

Streams

A sequence of elements

Does not store the elements

Does not change the source

Operations are lazy when possible

Closed when terminal expression reached

Streams

A stream carries values

from a source

through a pipeline

Pipelines

Okay, so what's a pipeline?

A source

Zero or more **intermediate** operations

A **terminal** operation

Reduction Operations

Reduction operations

Terminal operations that produce

one value from a stream

```
average, sum, max, min, count, ...
```

Transforming Streams

Process data from one stream into another

```
Stream<T> filter(Predicate<T> predicate)
```

Return only elements satisfying the predicate

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

Convert a `Stream<T>` into a `Stream<R>`

Transforming Streams

There's also flatMap:

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

Maps from single element of type T
to *wrapped* element of type `Stream<R>`

Removes internal wrapping

Using Collectors

```
Stream.of( ... )
```

`.collect( Collectors.toList() )` → creates an ArrayList

`.collect( Collectors.toSet() )` → creates a HashSet

`.collect( Collectors.toCollection( Supplier ))`

→ creates the supplier (LinkedList::new, TreeSet::new, etc)

`.collect( Collectors.toMap( Function, Function ))`

→ creates a map; first function is keys, second is values

Static And Default Methods in Interfaces (JDK 8)

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
  - Must have an implementation
  - See `Comparator.comparing`

Optional Type (JDK 8)

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)
```

The java.time Package (JDK 8)

LocalDate, LocalTime, ZonedDateTime, and more

LocalDate

A date without time zone info

contains year, month, day of month

```
LocalDate.of(2017, Month.FEBRUARY, 2)
```

months actually count from 1 now

LocalTime

`LocalTime` is just LocalDate for times

hh:mm:ss

`LocalDateTime` is both, but then you

might need time zones

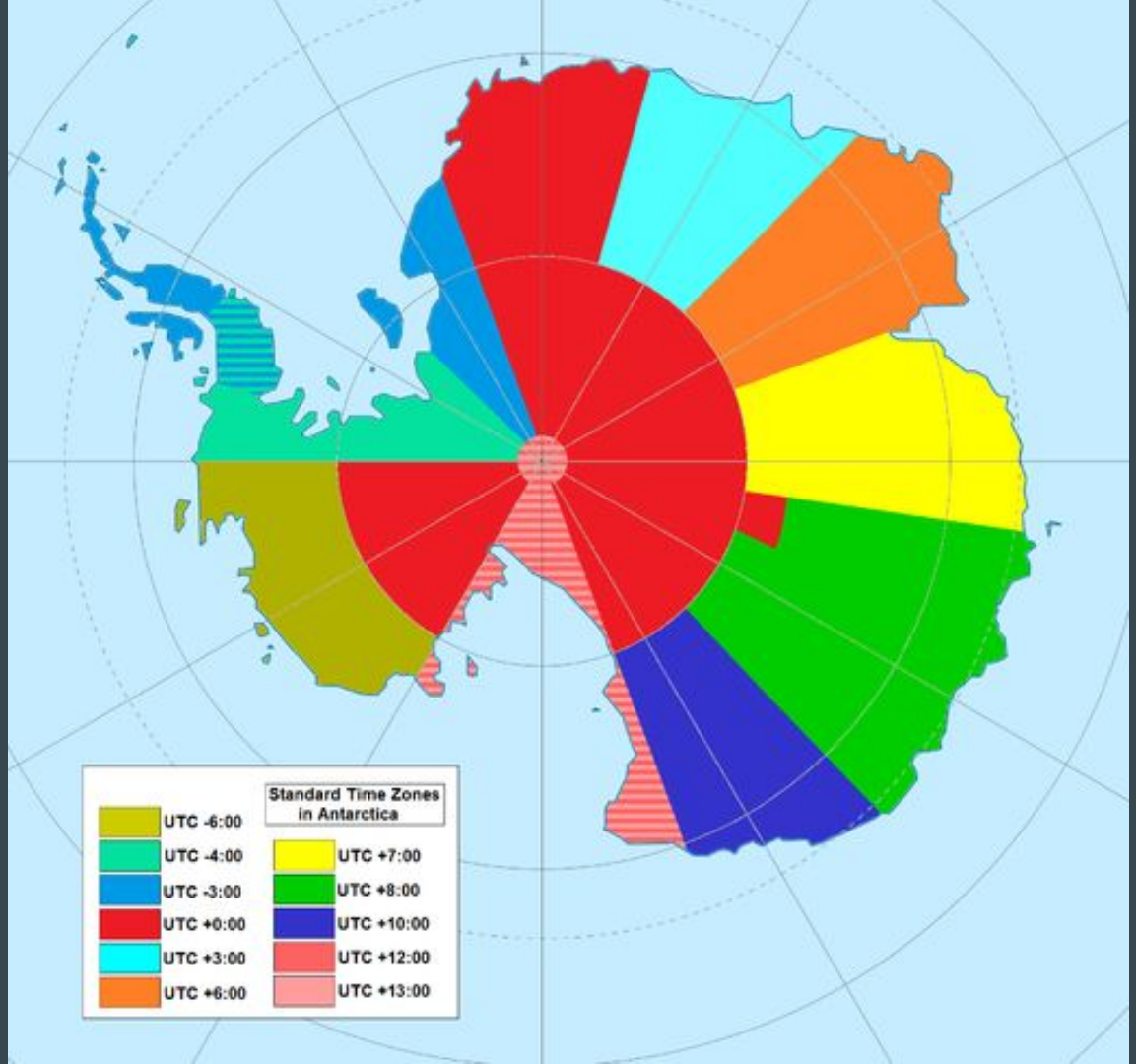
ZonedDateTime

Database of timezones from IANA

https://www.iana.org/time-zones

```
Set<String> ZoneId.getAvailableZoneIds()

ZoneId.of("... tz name ...")
```


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, c, ...)

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

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

Collection Factory Methods

What to remember:

- No nulls
- Produces unmodifiable collections
- `Set.of` does not allow duplicates

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 still have variable called "var")

Can also use on

- for loops
- try-with-resources blocks

var Data Type

Stuart Marks: Style Guidelines for Local Variable Type Inference in Java

https://openjdk.org/projects/amber/guides/lvti-style-guide

Local variables only

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/17/jshell/introduction-jshell.html

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

jshell>

/exit to leave

No semicolons needed

Single File Code Execution (JDK 11)

Run without explicit compilation

Single File Code Execution

- Run source code from command line
  - java HelloWorld.java
  - Compiles, but doesn't save compiled class
- First top-level class in file is executed
  - Must contain a standard main method
- Can make "script" using shebang notation, #!
  - Make file executable, then add this line:
  - #!/path/to/java --source version

Enhanced Switch Statement (JDK 14)

Makes switch useable

Enhanced Switch

- Expressions return a value
- Arrow rather than colon, so no fall through
- Multiple case labels
- Statement blocks use yield to return from block only
- Must be 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

For instanceof (JDK 16)
For switch (JDK 17, 18, 19, 20 preview)
For records (20 preview)

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/

Pattern Matching for switch

from JEP 406:

```
1 static String formatterPatternSwitch(Object o) {
2     return switch (o) {
3         case Integer i -> String.format("int %d", i);
4         case Long l    -> String.format("long %d", l);
5         case Double d  -> String.format("double %f", d);
6         case String s  -> String.format("String %s", s);
7         default        -> o.toString();
8     };
9 }
```

Sealed Classes (JDK 17)

Restricted type hierarchies

Sealed Classes (JDK 17)

- 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

```java
package com.example.geometry;

public abstract sealed class Shape
    permits Circle, Rectangle, Square {...}

public final class Circle extends Shape {...}

public sealed class Rectangle extends Shape
    permits TransparentRectangle, FilledRectangle {...}
public final class TransparentRectangle extends Rectangle {...}
public final class FilledRectangle extends Rectangle {...}

public non-sealed class Square extends Shape {...}
```

Simple Web Server (JDK 18)

Trivial web server

Simple Web Server (JDK 18)

- jwebserver
  - Alias for java -m jdk.httpserver
- Runs on port 8000 (change with -p flag)
- Supports GET and HEAD requests only
- No https
- Supports HTTP/1.1 only
- Serves files and folder listings only
- See details in com.sun.net.httpserver package

Miscellaneous Features

Private Methods in Interfaces (JDK 9)

- Both `default` and `static` methods in interfaces
  - can call `private` methods

Deprecated Annotation

`@Deprecated` now has fields:

- `forRemoval`
- `since`

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

See also The Java Version Almanac

- Lets you compare the Java API between any two versions

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)

The Good and Bad of JPMS

JPMS

Module descriptors

`module-info.java`

exports, requires, opens, ...

Quick start guide:

https://openjdk.org/projects/jigsaw/quick-start

State of the Module System

https://openjdk.org/projects/jigsaw/spec/sotms/

JPMS

module name → use "reverse dns" (like packages)

- `requires` → add a module to the "module path"
  - `java.base` added automatically
  - `transitive` → any package using this module can read the arg
- `exports` → list of packages exported by a module
  - can export to selected modules

Modularization Benefits

- True encapsulation
- Smaller libraries using only needed modules
  - jlink on JDK
- Ordering dependencies on the module path

Is that worth changing the fundamental meaning of public and private?

- If you are making a library, maybe
- If you're not, probably not

Summary

- Need to know functional features
  - Streams with map / filter / reduce
  - Lambda expressions
  - Method references
  - Concurrent, parallel streams
- Need to use Optional
- See also Should I Upgrade My Java GitHub repo for performance tests
- Helpful to know preview features
  - Enhanced switch
  - Text blocks
  - Records
  - Pattern matching
- Can probably ignore modules (unless you're a library developer)