# Replacing ThreadLocals with ScopedValues

**José Paumard**

PhD, Java Champion, JavaOne Rockstar

@JosePaumard | https://github.com/JosePaumard

# Why would you need to fix a new model for ThreadLocal variables?

So far, you saw two principles:
- blocking virtual thread is OK and cheap
- a virtual thread has a bounded life time

# Agenda

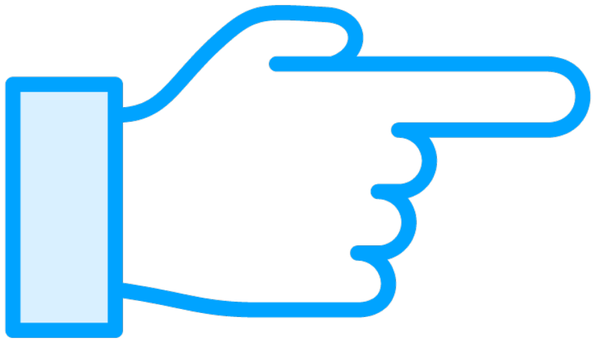
- Why do you need ThreadLocal variables?
- ScopedValues in action
- Adding ScopedValues to the Flight Booking application

# Why Do You Need ThreadLocals?

ThreadLocal variables have been created to pass arguments

From one place to the other

Without using method parameters

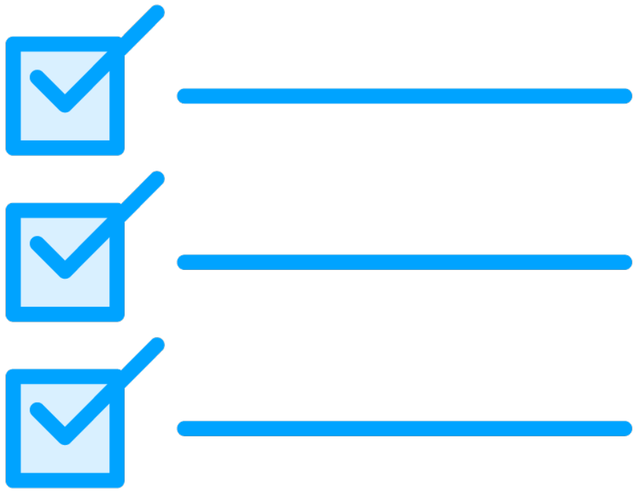


ThreadLocal variables were introduced in Java 2, in 1998

6 years before ExecutorService introduced in Java 5 in 2004

Virtual threads support ThreadLocal variables

But you can do better!

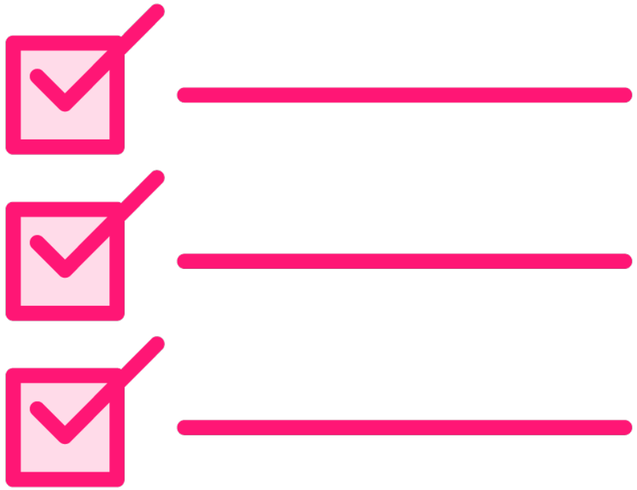


A ThreadLocal variable is attached to a thread

A thread is taken from an ExecutorService

That shares the life cycle of your application

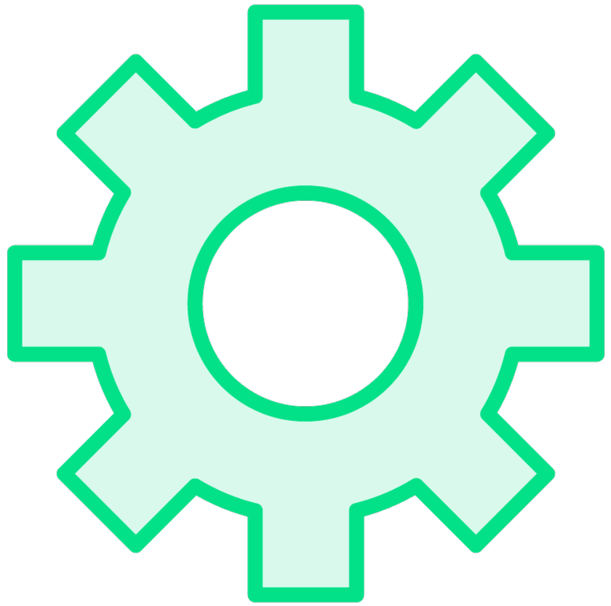
So a ThreadLocal variable is unbound



A ThreadLocal is mutable

It prevents many optimizations

And makes them costly memory-wise



ScopedValues are there to fix this

They have a bound life cycle

They are not bound to any thread

They are immutable

# ScopedValue in Action

# Creating Scoped Values

Let us create scoped values

Read them in tasks

And check how they are inherited when you create threads

**ScopedValues**
**are bound to the execution of a method**

**They are not transmitted to any unbound resource: thread or virtual thread**

**They are transmitted to bound resources: StructuredTaskScope**

# Using ScopedValue in a Real Application

# Validating a Licence Key with a Scoped Value

Let us use scope values in the application you built

Suppose that you need a licence key to access the external servers

ScopedValues can be used in
the same way as ThreadLocal variables

You can pass a value
from one place to another

Without using method parameters

In a much safer way!

# Module Wrap Up

ThreadLocal variables vs. ScopedValue

ThreadLocal are mutable, costly, unbound, and unsafe

ScopedValue are immutable, much cheaper, and have a bound life cycle

Virtual threads work with ThreadLocal, but you should move to ScopedValue

# Course Wrap Up

Virtual Threads are a new way to optimize the throughput of your application

They encourage you to write simple, imperative, and blocking code

They prevent you from writing unsafe code

No loose virtual threads with scopes

No loose ThreadLocal with ScopedValue

# Course Wrap Up

Comparing reactive programming and virtual threads:

From your platform thread perspective, everything looks the same

Writing blocking code in reactive programming is a costly bug

A virtual thread based application is much easier to maintain

Up Next:

# Write efficient and easy to maintain applications!