Java Platform Threads vs. Virtual Threads (Part 2)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

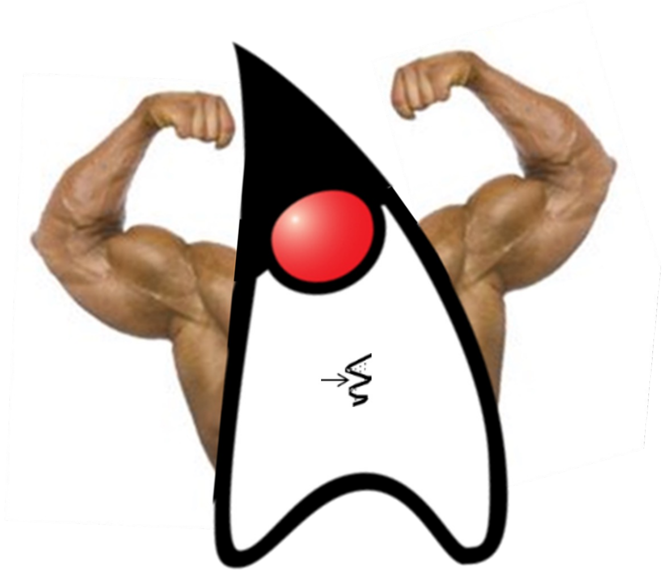
Institute for Software Integrated Systems

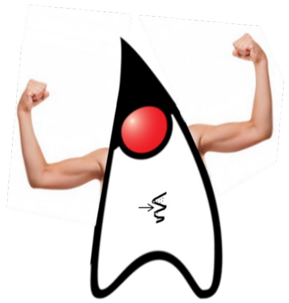
Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Know the differences between Java platform & virtual threads
  - Be aware of how to create Java platform & virtual threads





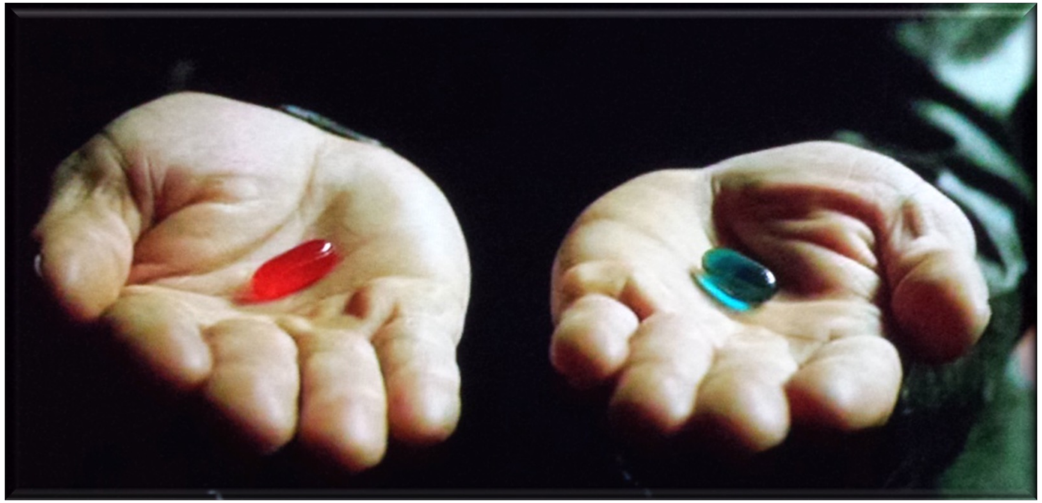
Learning Objectives in this Lesson

- Know the differences between Java platform & virtual threads
  - Be aware of how to create Java platform & virtual threads
  - Recognize virtual Thread best practices

Ways of Creating Java Platform Threads

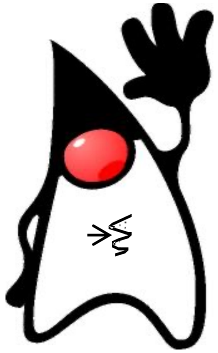
Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways



Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way



```
public class GCDThread
                   extends Thread {
  public void run()
  { /* code to run goes here */ }
}
```

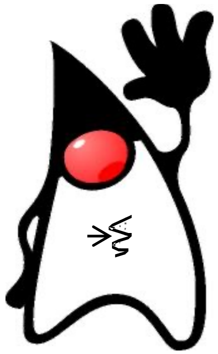
Create a new class that extends the Thread class

```
Thread gcdThread = new GCDThread();
gcdThread.start();
```

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way



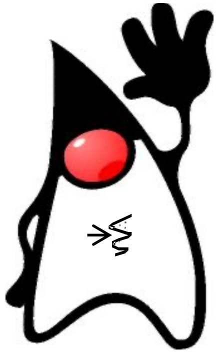
```
public class GCDThread
                   extends Thread {
  public void run()
  { /* code to run goes here */ }
}
```

Create & start a Thread using a new instance of GCDThread

```
Thread gcdThread = new GCDThread();
gcdThread.start();
```

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way



```
public class GCDRunnable
             implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

Create a new class that implements the Runnable interface

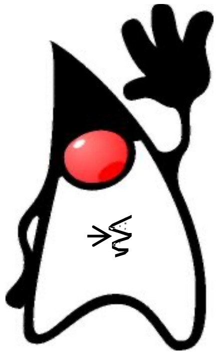
```
Runnable gcdRunnable =
  new GCDRunnable();

new Thread(gcdRunnable).start();
```

See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

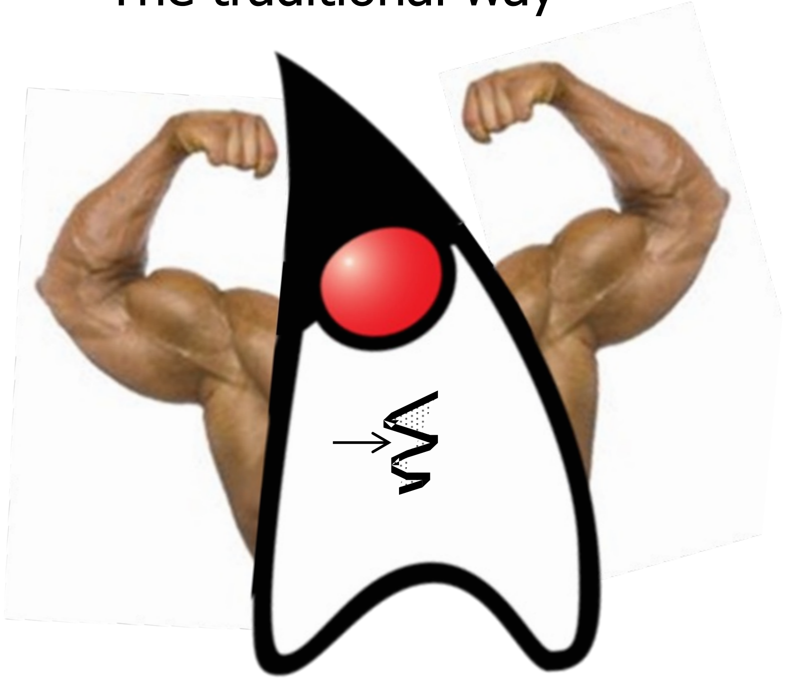
Create a new GCDRunnable, pass it to a Thread object, & start it

```
Runnable gcdRunnable =
  new GCDRunnable();

new Thread(gcdRunnable).start();
```

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

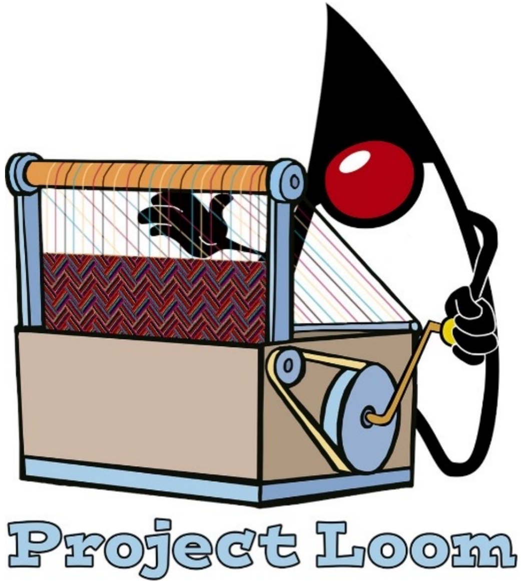
```
Runnable gcdRunnable =
  new GCDRunnable();

new Thread(gcdRunnable).start();
```

Traditional Java Thread objects are relatively "heavyweight" & inflexible

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way
  - The very modern Java way



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

A familiar way to create & start a Java platform thread so it executes gcdRunnable

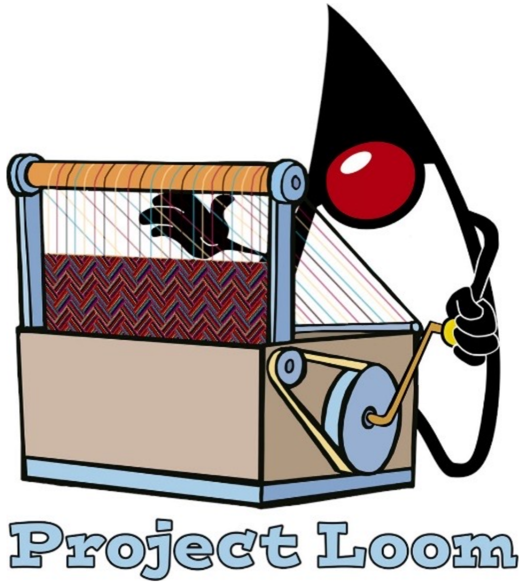
```
Runnable gcdRunnable =
  new GCDRunnable();

new Thread(gcdRunnable).start();
```

By default, a traditional Java Thread *is* a platform thread!

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way
  - The very modern Java way



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

A more flexible way to create & start a platform thread so it executes gcdRunnable

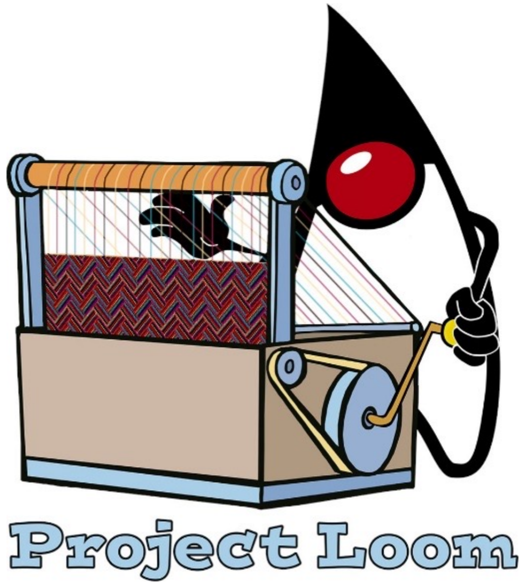
```
Runnable gcdRunnable =
  new GCDRunnable();

Thread
  .ofPlatform().start(gcdRunnable);
```

See docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/Thread.html#ofPlatform()

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way
  - The very modern Java way



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

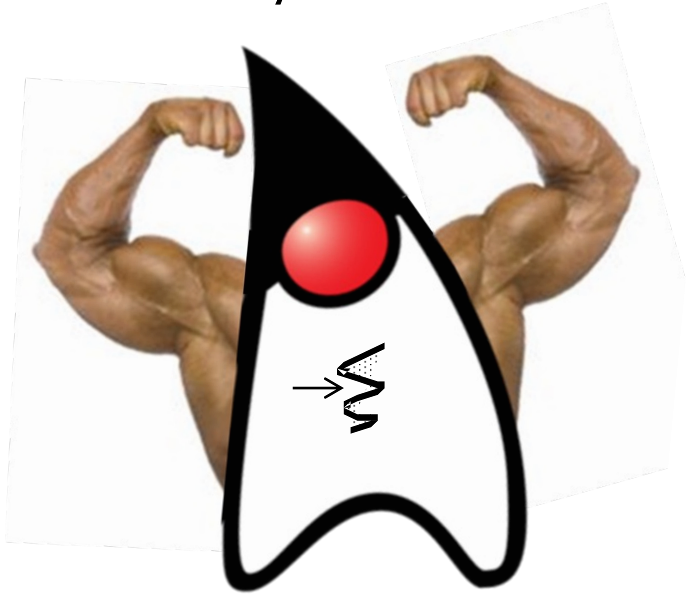
Create an "unstarted" platform thread & later start it so it executes gcdRunnable

```
Runnable gcdRunnable =
  new GCDRunnable();

Thread thread = Thread
  .ofPlatform().unstarted(gcdRunnable);
...
thread.start();
```

Ways of Creating Java Platform Threads

- Java platform threads can be created in two different ways
  - The traditional way
  - The very modern Java way



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

```
Runnable gcdRunnable =
  new GCDRunnable();

Thread thread = Thread
  .ofPlatform().unstarted(gcdRunnable);
...
thread.start();
```

However, Java platform threads are also relatively "heavyweight"

Ways of Creating Java Virtual Threads

Ways of Creating Java Virtual Threads

- Virtual threads can also be created in very modern Java

```
public class GCDRunnable
             implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

Use the same GCDRunnable class as before

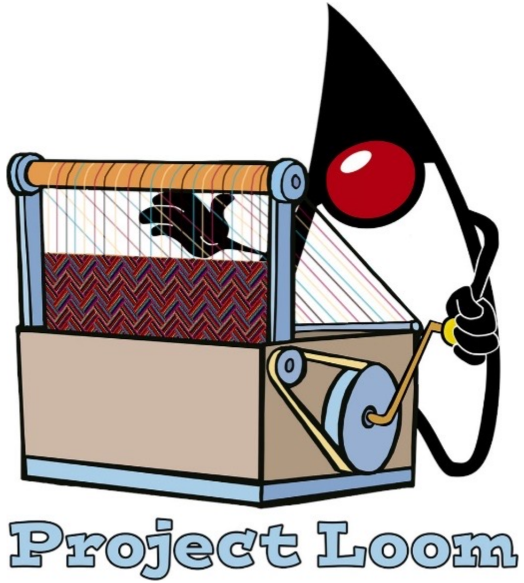
```
Runnable gcdRunnable =
  new GCDRunnable();

Thread.startVirtualThread
                   (gcdRunnable);
```

See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

Ways of Creating Java Virtual Threads

- Virtual threads can also be created in very modern Java



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

A concise way to create & start a Java virtual thread so it executes gcdRunnable

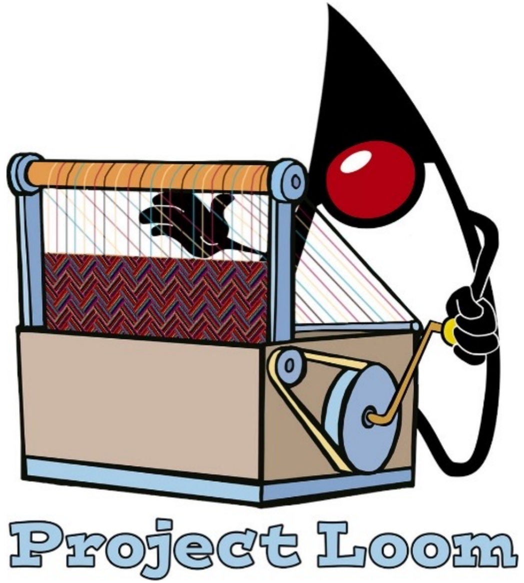
```
Runnable gcdRunnable =
  new GCDRunnable();

Thread.startVirtualThread
           (gcdRunnable);
```

See docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/Thread.html#startVirtualThread

Ways of Creating Java Virtual Threads

- Virtual threads can also be created in very modern Java



```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

A more flexible way to create & start a virtual thread so it executes gcdRunnable

```
Runnable gcdRunnable =
  new GCDRunnable();

Thread.ofVirtual()
      .start(gcdRunnable);
```

See docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/Thread.html#ofVirtual()

Ways of Creating Java Virtual Threads

- Virtual threads can also be created in very modern Java

```
public class GCDRunnable
       implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

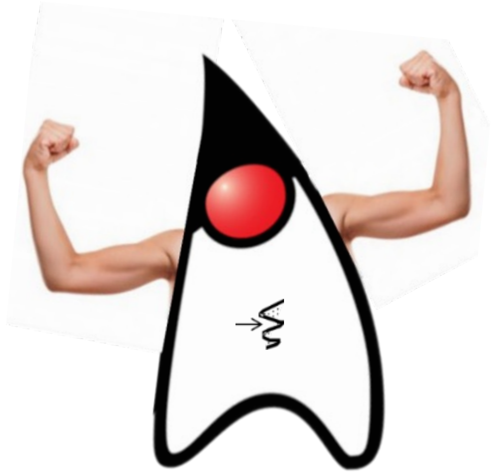
Create an "unstarted" virtual thread & later start it so it executes gcdRunnable

```
Runnable gcdRunnable =
  new GCDRunnable();

Thread thread = Thread
  .ofVirtual().unstarted(gcdRunnable);
...
thread.start();
```

Ways of Creating Java Virtual Threads

- Virtual threads can also be created in very modern Java



```
public class GCDRunnable
           implements Runnable {
  public void run()
  { /* code to run goes here */ }
}
```

```
Runnable gcdRunnable =
  new GCDRunnable();

Thread thread = Thread
  .ofVirtual().unstarted(gcdRunnable);
...
thread.start();
```

Java virtual threads are relatively "lightweight"

Virtual Thread Best Practices

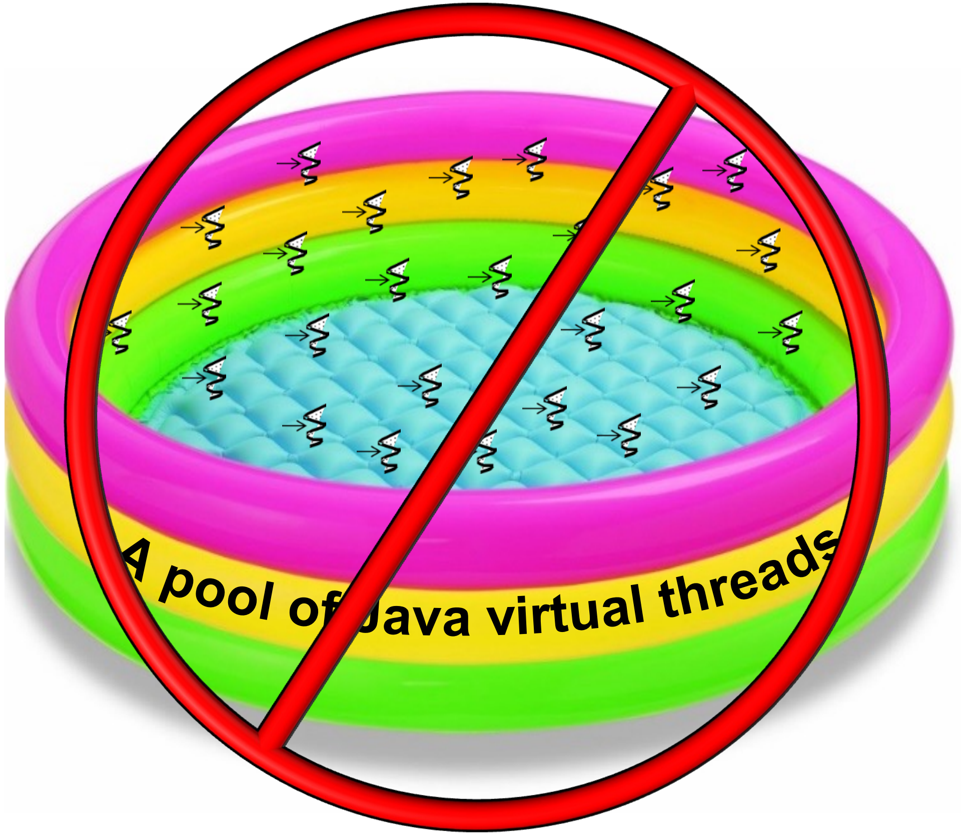
Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads

See howtodoinjava.com/java/multi-threading/virtual-threads/#5-best-practices

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!



See virtual-threads/#51-do-not-pool-the-virtual-threads

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!
    - Creating virtual threads is inexpensive, so there is never a need to pool them

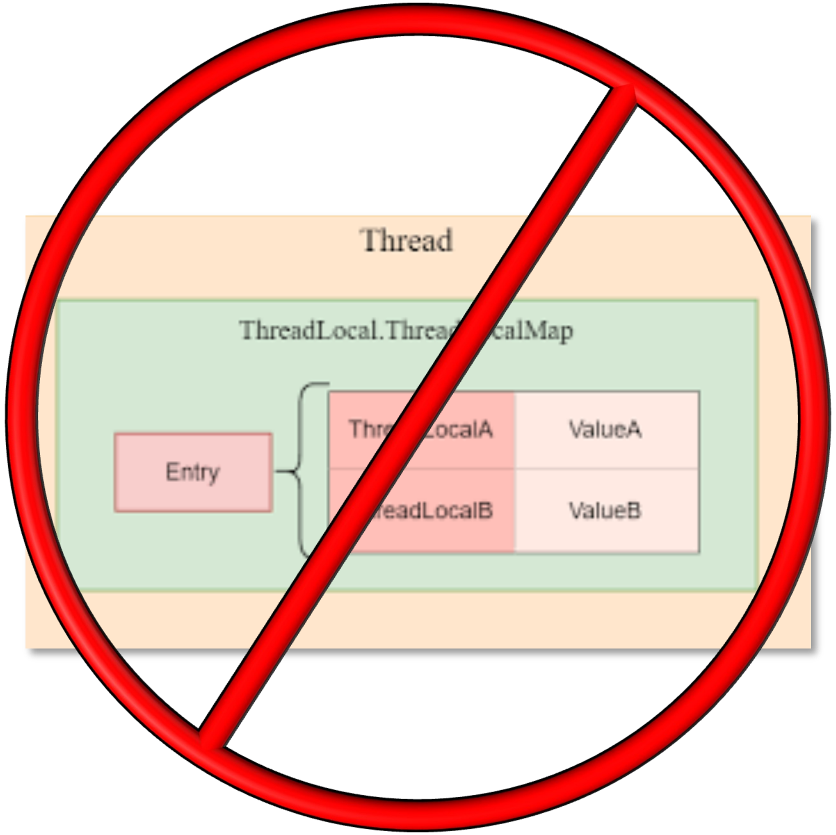
```
Runnable runnable =
  () -> doWork();

for (int i = 0;
     i < 20_000_000;
     i++)
  Thread.startVirtualThread
    (runnable);

...
```

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!
  - Avoid using thread-local variables



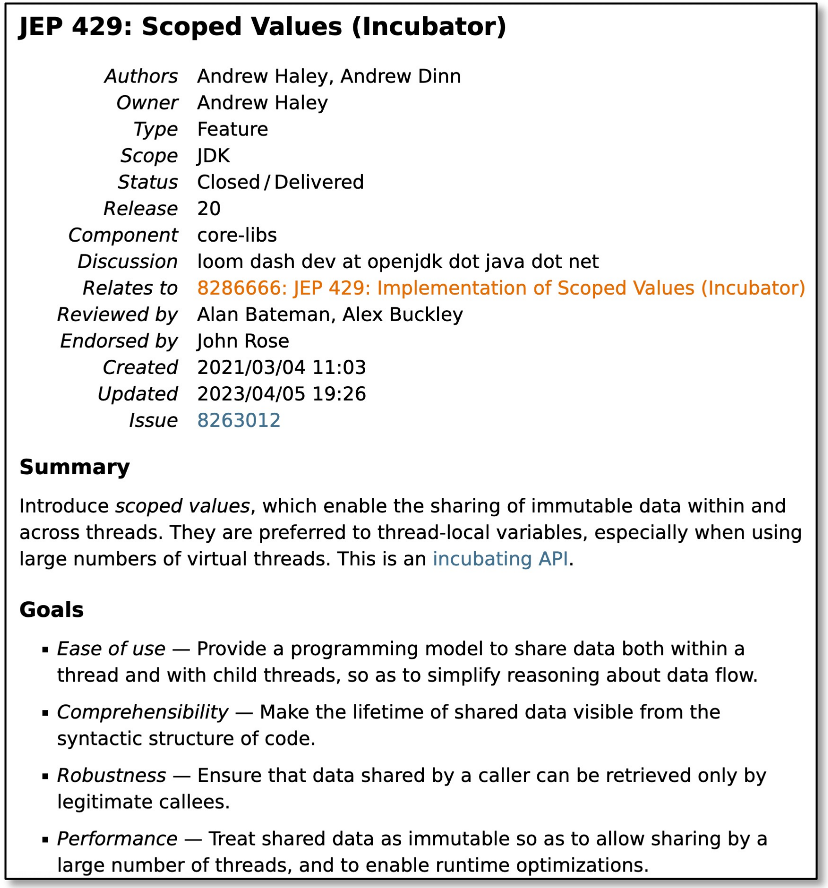
See virtual-threads/#52-avoid-using-thread-local-variables

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!
  - Avoid using thread-local variables
    - If an app uses ThreadLocal & creates 1 million virtual threads then 1 million Thread Local instances are created!

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!
  - Avoid using thread-local variables
    - If an app uses ThreadLocal & creates 1 million virtual threads then 1 million Thread Local instances are created!
    - Consider using "scoped values" instead



JEP 429: Scoped Values (Incubator)

Authors	Andrew Haley, Andrew Dinn
Owner	Andrew Haley
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	20
Component	core-libs
Discussion	loom dash dev at openjdk dot java dot net
Relates to	8286666: JEP 429: Implementation of Scoped Values (Incubator)
Reviewed by	Alan Bateman, Alex Buckley
Endorsed by	John Rose
Created	2021/03/04 11:03
Updated	2023/04/05 19:26
Issue	8263012

Summary

Introduce *scoped values*, which enable the sharing of immutable data within and across threads. They are preferred to thread-local variables, especially when using large numbers of virtual threads. This is an incubating API.

Goals

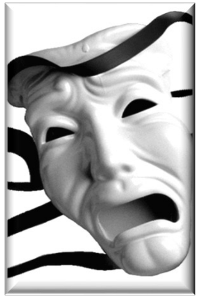
- *Ease of use* — Provide a programming model to share data both within a thread and with child threads, so as to simplify reasoning about data flow.
- *Comprehensibility* — Make the lifetime of shared data visible from the syntactic structure of code.
- *Robustness* — Ensure that data shared by a caller can be retrieved only by legitimate callees.
- *Performance* — Treat shared data as immutable so as to allow sharing by a large number of threads, and to enable runtime optimizations.

See openjdk.org/jeps/429

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!
  - Avoid using thread-local variables
  - Avoid using synchronized blocks
    - Synchronized blocks "pin" a virtual thread to a platform thread..

```
public synchronized void m() {
  // ... access resource
}
...
```



See virtual-threads/#53-use-reentrantlock-instead-of-synchronized-blocks

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!
  - Avoid using thread-local variables
  - Avoid using synchronized blocks
    - Synchronized blocks "pin" a virtual thread to a platform thread..
    - Use ReentrantLocks instead

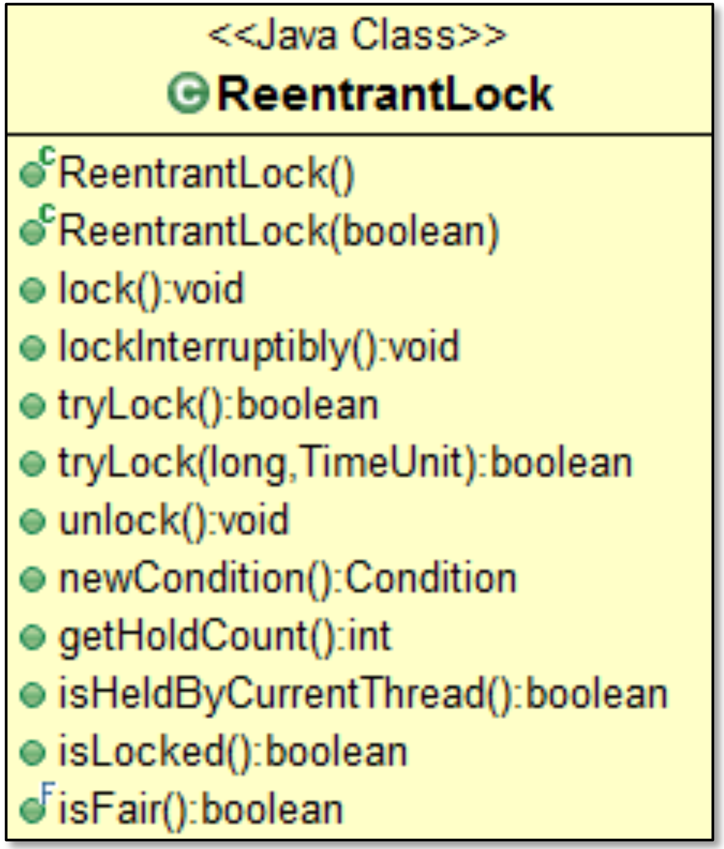
```
private final ReentrantLock lock
  = new ReentrantLock();

public void m() {
  lock.lock();
  try {
    // ... access resource
  } finally {
    lock.unlock();
  }
}
...
```

See www.geeksforgeeks.org/reentrant-lock-java

Virtual Thread Best Practices

- Follow certain "best practices" when using Java virtual threads
  - Do not pool virtual threads!
  - Avoid using thread-local variables
  - Avoid using synchronized blocks
    - Synchronized blocks "pin" a virtual thread to a platform thread..
    - Use ReentrantLocks instead
      - These locks also provide many more features than synchronized blocks!



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReentrantLock.html

End of Java Platform Threads vs. Virtual Threads (Part 2)