Advanced Java T.T.

Threading Utilities

Develop Intelligence

Simon Roberts

Join Us in Making Learning Technology Easier

Develop Intelligence

Our mission...

Over 16 years ago, we embarked on a journey to improve the world by making learning technology easy and accessible to everyone.

...impacts everyone daily.

And it's working. Today, we're known for delivering customized tech learning programs that drive innovation and transform organizations.

In fact, when you talk on the phone, watch a movie, connect with friends on social media, drive a car, fly on a plane, shop online, and order a latte with your mobile app, you are experiencing the impact of our solutions.

Upskilling and Reskilling Offerings

Develop Intelligence

Intimately customized learning experiences just for your teams.

- **Workshop** — 2-3 day upskilling experiences
- **Fast Track** — 5-day reskilling experiences
- **Learning Spike** — 1-day technology overviews
- **Target Topics** — 90-minute instructor-led micro-learnings
- **Hack-a-thon** — Learn and build an MVP in 2-3 days

BACK END DEVELOPMENT

BIG DATA

CLOUD COMPUTING

DEVOPS

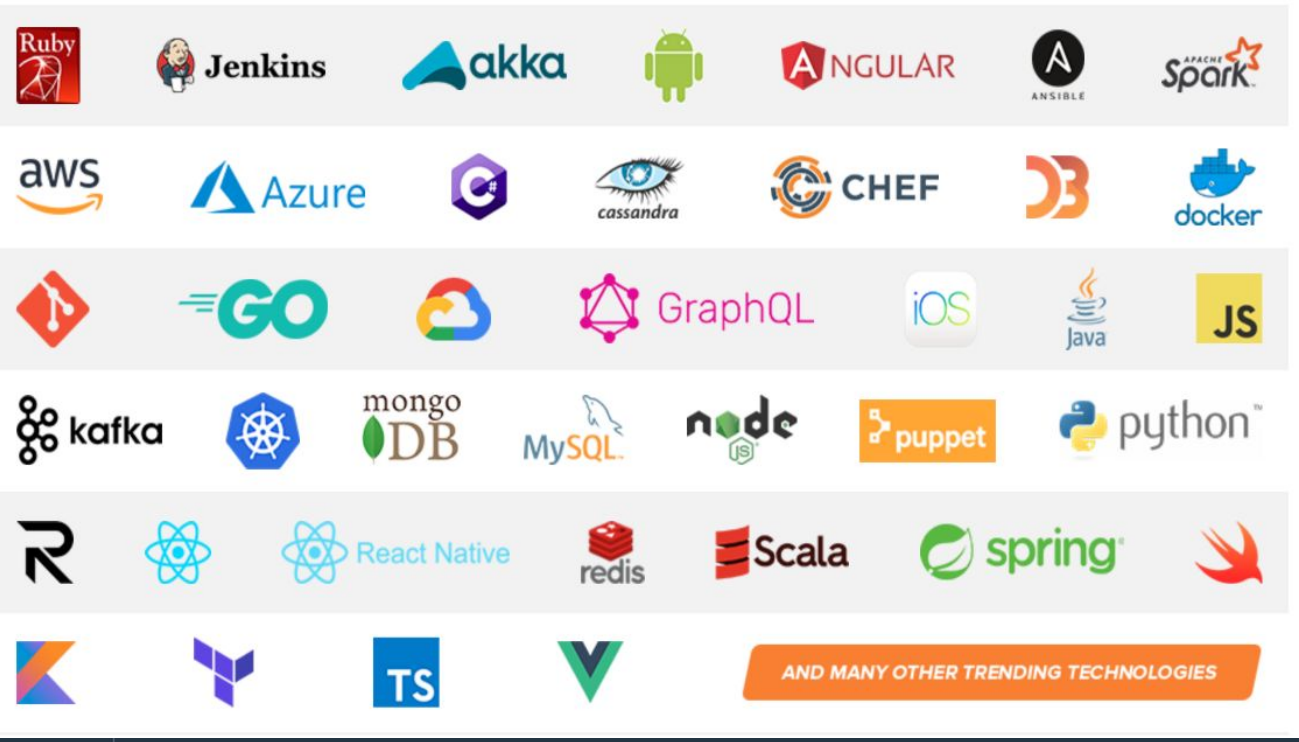
FRONT END DEVELOPMENT

MACHINE LEARNING

MOBILE APP DEVELOPMENT

SOFTWARE ENGINEERING

SYSTEM ADMINISTRATION



World Class Practitioners

Recording Policy

Recordings are provided to participants who have attended the training, in its entirety. Recordings are provided as a value-add to your training, and should not be utilized as a replacement to the classroom experience.

Participants can expect the following:

- Recordings will be provided the Monday after class is completed
- Recordings will be provided for 14 days
- Recordings will be accessible as “View Only” status and cannot be copied
- Sharing recordings is strictly prohibited

To request recordings, please fill out the form linked in Learn++.

Thank you for your understanding and adhering to the policy.

Note About Virtual Trainings

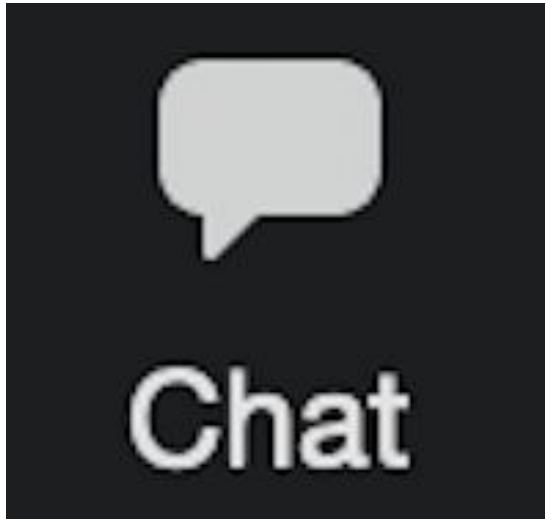
What we want

...what we've got

Virtual Training Expectations for You

Arrive on time / return on time

Mute unless speaking



Use chat or ask questions verbally

Virtual Training Expectations for Me

I pledge to:

- Make this as interesting and interactive as possible
- Ask questions in order to stimulate discussion
- Use whatever resources I have at hand to explain the material
- Try my best to manage verbal responses so that everyone who wants to speak can do so
- Use an on-screen timer for breaks so you know when to be back

Prerequisites

- Good understanding of the Java programming language to Java 8
- Basic understanding of threads in Java

Objectives

Develop Intelligence

At the end of this course you will be able to:

- Create and use atomic types and accumulators
- Choose between synchronized and concurrent data structures to suit the task at hand
- Use concurrent collections
- Use synchronized collections

About you

In 90 seconds!

- What you hope to learn
- What your background level is

THANK YOU

Develop Intelligence

Concurrent utilities

Develop Intelligence

Concurrent utilities typically provide thread safety, high scalability, or perhaps both.

Be sure which type of behavior a utility provides and use it appropriately.

Prefer library provided utilities over home-brewed ones; it's far too easy to make a small mistake, and debugging concurrency problems is exceptionally hard, at least in part because they tend to be non-deterministic (not repeatable) in nature.

Atomic types

Develop Intelligence

Atomic types provide indivisible read-modify-write cycles at a library API level.

The package `java.util.concurrent.atomic` is "lock free" which ais to provide very high scalability.

A few sample operations on `AtomicInteger` illustrates general concepts. Most create a *happens-before* relationship as though the variable were `volatile`:

- `get()` : returns the current value
- `addAndGet(int x)` : add `x` to the current value, and return the updated value.
- decrementAndGet() : reduce value by 1 and return the result.

Atomic array types

Develop Intelligence

Provide atomic operations along with *happens-before* relationships on array elements.

The array may be created with a length, or by duplicating an existing array.

Example operation:

```
int accumulateAndGet(int i, int x, IntBinaryOperator accFn)
```

Atomically updates `array[i]` with the result of `accFn.apply(array[i], x)`

Accumulators

Develop Intelligence

If a value is subject to concurrent updates, but very rare writes, and the update operations are entirely independent of one another (for example, simple increments) then it's possible to gain scalability by giving each thread a thread-local variable, so it can have unrestricted updates.

On reading the multiple values must be collected, aggregated, and returned.

Initialize an accumulator with a binary operator suited to the data type. This is used to apply the updates.

The operation should be commutative, associative, and free of side-effects, or the behavior of the accumulator will likely be unpredictable.

Concurrent data structures

These data structures provide thread safety, but are primarily focused on scalability, minimizing or eliminating locks in their operation.

`ConcurrentHashMap`, `ConcurrentSkipListMap`, `ConcurrentSkipListSet`, `CopyOnWriteArrayList`, and `CopyOnWriteArraySet`

ConcurrentHashMap locks only one "bucket" of the map rather than the entire map.

When significant contention is expected concurrent data structures are preferred, for non-contented concurrent situations, synchronized structures are likely better. Entirely single-threaded access favors normal structures.

CopyOnWriteArrayList

This structure is intended for heavily concurrent reading with occasional updates.

Read operations are lock free, however writing is accomplished in two steps:

- Read and duplicate the entire array (this is clearly expensive)
- Modify the copy
- Redirect all reads to the new version allowing the old array to be garbage collectable.

Synchronized structures

Develop Intelligence

If contention is rarely expected, a simple lock is likely the best approach.

The `Collections` class provides static factory methods for proxy objects that wrap around the main interfaces of the collections API, such that access through that proxy is serialized using simple locking.

E.g.

```
List<String> syncList =
  Collections.synchronizedList(new ArrayList<String>());
```