Intermediate Java

WELCOME!

Develop Intelligence

Simon Roberts

WELCOME

Develop Intelligence

Mark Your Attendance:

- Go to the Course Calendar Invite
- Click on the Course Event Link
- Press Check-In

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

World Class Practitioners

Note About Virtual Trainings

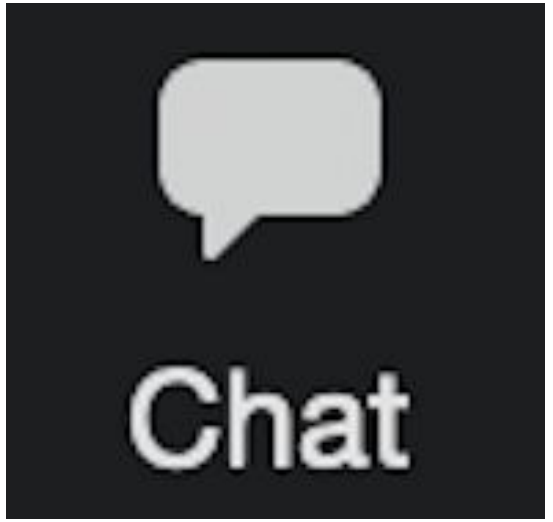
What we want

...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

- You should have written code in Java and be familiar with the majority of syntax that exists in Java 7.

- This course will introduce some of the features that are less well understood from Java 7 onwards.

Objectives

Develop Intelligence

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

- Use try with resources and multi-catch constructs to handle exceptions.
- Use inner and nested classes to implement appropriate design patterns.
- Make good use of collections and streams apis.
- Use the java.time api.
- Create generic classes and methods
- Use Java's functional features including writing and using lambda expressions.
- Create annotations and write code that identifies and uses annotations at runtime.
- Use key features of the java.util.concurrent apis.

About you

In 90 seconds!

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

Agenda

- Investigating the try-with-resources mechanism
- Reviewing Java syntax, design patterns, and using inner classes.
- Investigating collections and introducing functional ideas
- Building more functional syntax
- Using streams
- Building generic types and methods
- Working with annotations
- Using the concurrency api

THANK YOU

Develop
Intelligence


Develop
Intelligence

Exception handling in Java 7

Develop Intelligence

Java 7 added:

- Try-with-resources
- multi-catch

Try-with-resources

Develop Intelligence

Prior to this feature Java's try/finally had significant issues when trying to ensure reliable closure of O/S resources (such as files).

- Variable scope rules made access to the resource in the finally block cumbersome
- The definite initialization rule requires null-checking every resource
- Many close() methods throw a checked exception, mandating try/catch in the finally block
- Perhaps worse only one live exception is possible per-thread in the JVM, which means significant exceptions can be lost if a close throws an exception.

Try-with-resources

Develop Intelligence

Java 7 try-with-resources addresses all of these problems:

```
try ( // declare+initialize AutoCloseable resources
  BufferedReader in =
    Files.newBufferedReader(Path.of("input.txt"));
  BufferedWriter out = Files.newBufferedWriter(
    Path.of("out.txt", StandardOpenOption.CREATE));
) {
  // regular try body.
} // No "finally" needed, resources automatically closed
```

Try-with-resources

- Multiple resources can be listed in the resources block
- Separate with semicolon
- "Terminating" semicolon permitted but not required
- Since Java 9, a final variable that is declared above the try may be listed, allowing greater scope, but still guaranteeing closure

Design Concepts

Develop Intelligence

If a change from design A to design B is easy, and both satisfy the functional requirement, but a change from B to A is relatively hard, prefer design A unless there are other reasons to choose between them

- Make classes final until there is a determined need for inheritance, and the consequences are understood
- Prefer factories or builders over constructors
  - Constructors can only produce a new object of the exact type, or an exception
  - Any method can do this, but can do much more too
  - Removing constructors breaks clients, changing the implementation details of a method does not.

Design concepts

Develop Intelligence

Consider preferring immutable data and structures

- This can avoid "that can't happen" moments where other users of data altered them unexpectedly
- Immutable data can be shared, perhaps reducing the need for copying and compensating for the need to make new data to describe variations on old data
- Sometimes an immutable proxy (such as Collections.unmodifiableXxx) serves well to prevent clients changing something while allowing the "owner" to change it.

Design concepts

Develop Intelligence

Ensure that a business domain object is always "valid"

- Avoids unexpected states breaking client code
- Ensure any/all mutable fields are private
- Ensure that validity is verified prior to completing construction and before all changes
- Represent adjunct concepts (such as wire transfer and database storage) in secondary classes (e.g. a Data Transfer Object). Ensure the secondary class is dependent on the primary but not the other way round.

Design concepts

Develop Intelligence

Keep in mind that a Set is intended to reject duplicates, and might be preferable to a List in some situations. However:

- Sets reject duplicates simply by returning false from their add methods. This is easy to overlook.
- Sets require objects that implement equals/hashcode or ordering, and it's not always clear which is required if you simply have a java.util.Set.
- Most (but not all) core Java objects do not implement equals/hashcode unless they're immutable (String, primitive wrappers, java.time classes, for example)

Separation of concerns

Develop Intelligence

Take care to separate unrelated concerns, and also concerns that change independently, for example:

To print a subset of a list involves 3 independent concerns:

- processing the list
- distinguishing which items are wanted
- printing the result

Command pattern

Develop Intelligence

Defining the selection mechanism is done (in object oriented languages) by defining a behavior (method) in an object, and passing that object as an argument for the purpose of the behavior it contains

Passing the behavior as in argument object is known (in the Gang of Four pattern catalog) as the "command" pattern

The concept is also also known as a higher order function in "functional programming"

Lambda expressions

Develop Intelligence

A lambda expression in Java defines an object in a context.

- The context must require an implementation of an interface
- The interface must declare EXACTLY ONE abstract method
- We must only want to implement that one abstract method
- We provide a modified method argument list and body with an "arrow" between them:

```
(Student s) -> {
  // function body
}
```

- The argument list and return type must conform to the abstract method's signature

Lambda syntax variations

- Argument types can be omitted if they're unambiguous
  - This is "all or nothing"
- Since Java 10, argument types can be replaced with "var" if they're unambiguous
  - This is also "all or nothing"
- If a single argument carries zero type information the parentheses can be omitted
- If the method body consists of a single return statement, the entire body can be replaced with the expression that is to be returned.

FunctionalInterface annotation

Develop Intelligence

If an interface is intended for use with lambdas, it must define exactly one abstract method.

The @FunctionalInterface annotation asks the compiler to verify this and create an error if this is not the case.

Predefined interfaces

Develop Intelligence

Due to Java's strong static typing, and the restrictions preventing generics being used with primitives, different interfaces must be provided for a variety of different situations.

Key interface categories are:

Function - takes argument, produces result

Supplier - zero argument, produces result

Consumer - takes argument, returns void

Predicate - takes argument, returns boolean

Unary/Binary Operator - Function variants that lock args and returns to identical type

Predifined interface variations

For two arguments, expect a "Bi" prefix

For primitives:

- Int, Long, Double prefix usually means "primitive argument"
  - Note for Supplier, this prefix refers to return type (there are zero arguments.)
- ToInt, ToLong, ToDouble prefix means "primitive return"