1. What is Maven?

A build tool. Maven can manage a project's build, reporting and documentation from a central piece of information.

1. What are some common Maven Goals?

Maven’s primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal, Maven deals with several areas of concern:

* Making the build process easy
* Providing a uniform build system
* Providing quality project information
* Encouraging better development practices

1. What is the default archetype?

The model or template of projects that do the same things.

1. What does Maven do with dependencies?

Maven manages dependencies automatically. Downloads requirements from a central repository, adds required jars to project and allows the specification to different scopes of dependencies.

1. What are the dependency scopes?

Define that where the dependencies would be included.

Test

1. What is the basic folder structure?

Src/main src/test pom.xml

1. What is the POM?

Project Object Model-the xml file containing all configuration for a project.

* Contains full description of the project and dependencies used.
* Identifies the project uniquely
* Project website
* Project name

1. Which plugin must you always remember?

Maven-compiler-plugins : sets the target and source java version. Compile java source code.

1. What is TDD?

Test driven development.

1. What is the 3 key steps?

Red- write fail test

Green - write the minimum amount of code to make the test pass

Blue – generalize to work for additional cases

1. What is the advantages of TDD than TLD?

* High test coverage percentage
* Code is kept to a minimum
* Design is emergent
* Less test bias

1. Two approaches?

Inside-out: focus on the components and the collaboration would be considered later.

TDD fully steers the design of the application

Outside-in: Start with a collaborating components and mock components that are not yet created.

Once a component has been completed start writing tests for a mocked collaborator and begin the TDD process again there.

* Lends well to creating integration tests in addition to unit tests.
* Pairs well with YAGNI(You aint gonna need it)
* During the development lifecycle of a project, often tests will need to change to meet the changing nature of the application.
* It is acceptable to have to update tests for implementation details.
* Changing tests that cover external facing API can be difficult or dangerous – as external applications or services may be making use of these.
  1. This can lead to versioning of API to help differentiate between definitions
  2. In an ideal world an API test should be implemented in agnostically and not require changing unless the API/Specification changes.

**Unit testing is the process of ensuring that the smallest testable parts of an application (“units”) work as expected.**

1. What is JUnit?

Provide methods and framework to do the unit test.

1. What are the signs of a good test?
   * Simple
   * Focused
   * Independent
   * Flexible
   * Easy to read
2. What are the different Annotations and what do they do?

Test

Before

BeforeClass: once

After

AfterClass: once

1. What are the different Assertions and their behaviour?
   * **assertEquals**(expectedObject, actualObject)
   * **assertTrue**(boolean)
   * **assertNull**(object)
   * **assertFalse**(boolean)
   * **assertNotNull**(object)

**@Test(expected=CustomException.class)**

1. What are the 5 steps of TDD?

**Five Steps**:

* 1. Write Test
  2. Make the test compile
  3. Watch the test fail
  4. Do just enough to make all tests pass
  5. Refactor and generalise

1. Arrange-Act-Assert pattern:

Arrange: Set up preconditions

Act: Call the code under test

Assert: Check that expected results have occured

1. What is the most important part of a test method?

Mockito

[1.@Mock](mailto:1.@Mock)

1. @InjectMock
2. MockAnnotation.initMocks(this)
3. Wen then
4. Verify: 1. Count time 2. Check if use 3. Get captor

Exception

1. What is the purpose of exceptions?

Capture the exception and handle it without ruin the whole application

1. What are the two main types of Exception?

Checked Unchecked(RuntimeExecption)

1. What are Errors?
   * External to the program
   * Cannot be recovered from
   * Should not be caught
2. Which type of Exception should you not handle?

Unchecked exception. You should prevent them by implementing adequate if statements.

1. How do you handle exceptions?

Try catch/throws

1. How can you create your own Exception?

* To create your own exception, create a class that:
  1. Extends Exception for a checked exception
  2. Extends RuntimeException for an unchecked exception
  3. Typically includes constructors that take in a message and/or a cause

FileIO

1. What is a stream?

A flow of data with a writer at one end and a reader at the other.

1. What is an InputStream and OutputStream?

Read data from the source code. Write data to the destination.

1. What type of data InputStream and OutputStream deal with?

character

1. Name a type of InputStream and a type of OutputStream.

FileInputStream

FileOutputStream

DataInputStream(Read any primitive type except char)

DataOutputStream

1. What is a Reader and a Writer?

Provide basic functionality for reading and writing characters

1. What type of data Reader and Writer deal with?

Character

1. Name a type of Reader and a type of Writer.

InputStreamReader

OutputStreamWriter

FileReader/FileWriter

1. How can reading or writing of character be more efficient?

BufferedReader BufferedWriter

JSON & Jackson

1. What is JSON?

JavaScript Object Notation

It’s a lightweight human and computer-parsable file format

1. How do you write an object in JSON?

Using curty brackets{}(brace)

Name: value

1. What is the purpose of the Jackson API?

Provide standard API for java to read and write JSON.

1. What is data-binding?

Defines how java objects will be represented in JSON.

1. What is streaming?

The reading/writing of JSON from/to various location

1. How does Jackson determine the default names of JSON properties?

The variable name.

1. ObjectMapper

The ObjectMapper class in Jackson provides a customizable entry point for reading / writing JSON from a Java application

readValue( … ) can be used to deserialize JSON

writeValue( … ) can be used to serialize Java objects

JSON can be written from / read to several locations via overloaded readValue( … ) and writeValue( … ) methods

Files

Network locations

Strings

Etc.

1. What are 5 Jackson annotations?

@JsonProperty("property")

* @JsonInclude
* @JsonIgnoreProperties({"property1", "property2"})
* @JsonIgnore
* @JsonSerialize / @JsonDeserialize

1. What does lambda do?

Represent shorthand syntax for the implementation of a functional interface.

1. What is a functional interface?

A regular interface in Java with some constraints. Only one abstract method and should be using @FuntionalInterface above the class definition.

Enables flexibility with in the system through the dependency inversion principle.

1. What is method reference?

Like: String::isEmpty

Just pass the method reference into the function interface and then return the result using the method.

If you want to use parameter, you need to use the Lambda.

Stream has not storage(not a object) just an API to pull the data from the source and do the jobs.

1. What is a stream?

A Stream is an interface in Java which enables us to act on a stream of data such as a collection or an IO Stream.

It provides methods to assist in filtering, mapping and collecting a stream of data into another data type.

It was introduced to the Java programming language in Java 8 and has many dependencies on Functional Interfaces to support the use of lambda functions.

1. **What can be treated as a Stream?**

Collection/primitive streams/arrays.stream/

Convert input into a stream: BufferedReader

1. Why used?

The Stream API provides simple to use functions, which require less code to achieve commonly occurring tasks with streams of data in Java.

Like lambda functions they streamline our existing code by reducing lines required to do certain tasks.

Developers can however write code that does the same thing without the use of streams.

Enable the use of parallel stream manipulation to improve efficiency.

1. What is intermediate methods?

The methods that return a stream. We can chain 0 or many calls to intermediate methods.

1. Why used?

There are many places where functional interfaces are used within Java.

This enables flexibility with in the system through the dependency inversion principle.

Wherever a functional interface is being used as a data type we can replace it using a lambda expression.

A lambda expression often represent small amounts of functionality that does not constitute the addition of a full class into the code base.

1. What is lambda?

They represent shorthand syntax for the implementation of a functional interface.

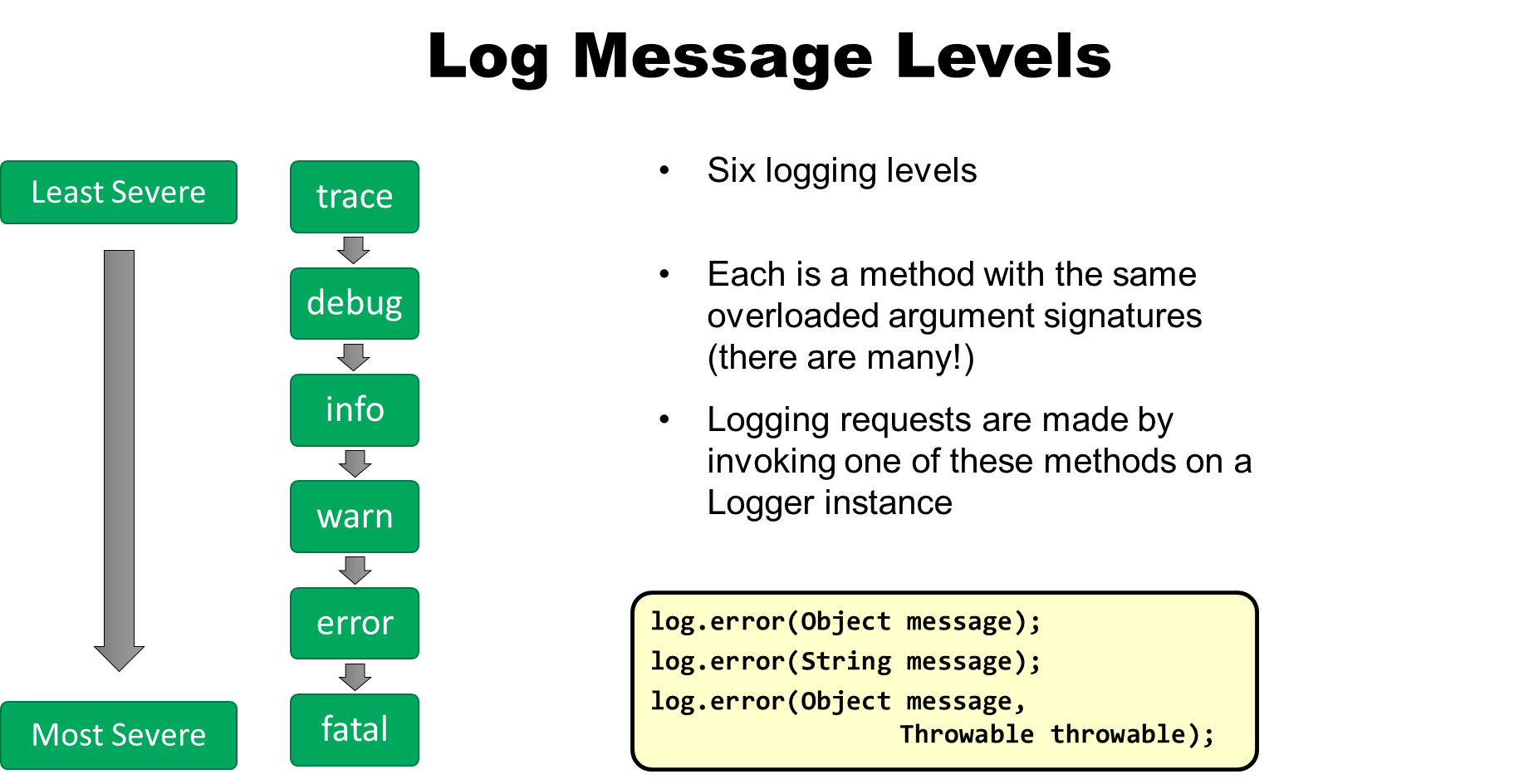
1. What is logging?

When something goes wrong the first point of call is the system logs.

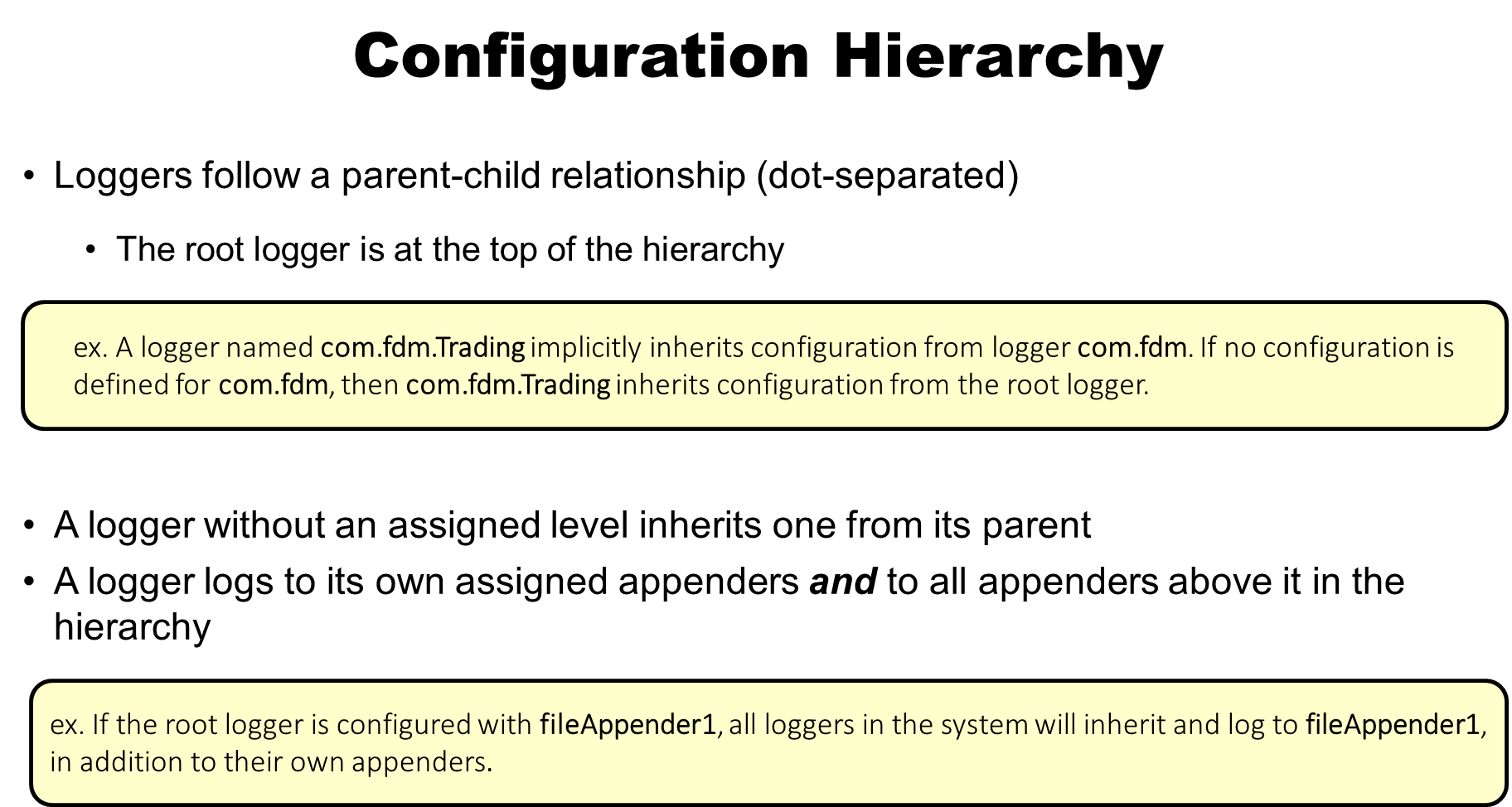
Logs provide a trace to figure out circumstances that cause issues, both at runtime and during debugging

Logs are the main piece of information available to Support consultants and Developers

Almost every large application devotes development efforts to logging, tracing, and/or auditing



Configuration : make in names



1. What is Optional?

* Represent a value that might be present.

1. What are some issues that occur with using null?

NullPointerException

Using too much if condition statements.

1. What should be returned if a collection of items is empty?

An empty collection

1. What should be returned if an item might be present?

Null

1. How do you construct an Optional instance?

Optional.of()

1. What are some common methods of Optional?

Optional.empty()

Optional.ofNullable()

isPresent() : boolean – Returns true if the optional has a value

get() : T – Returns the value if it is present

Note that this method throws a NoSuchElementException if there is no value in the optional. Always check with .isPresent() before calling get()

orElse(T other) : T – Returns default if no value is present

ifPresent(Consumer<? super T> consumer) : void - If a value is present, use it via a Consumer implementation

1. Where is the most appropriate place to use Optional?

* Optionals should only be used as a return value
* Optionals should be used when one piece of data "might" be returned, such as from a Data Access Object
  + Optional data from Collections should still return the Collection type and not populate it
* Optionals should not be used as parameters
* In particular, it is fine to have nullable data in objects as private state
  + This may or may not be represented by an Optional
  + Annotations are a cleaner option for fields to represent nullability