Java Training

<https://meet.google.com/era-hupi-xwh>

<https://docs.google.com/spreadsheets/d/1ftUeu95G2LePlEc7XfR77sWLOpzQOYUJZZe_2r6F54o/edit?gid=0#gid=0>

Day1 note:

<https://aws.amazon.com/certification/certified-solutions-architect-associate>

Day 0:

1. AWS certificate: https://www.udemy.com/course/aws-certified-solutions-architect-associate-saa-c03/?kw=solution+associate&src=sac

AWS exam simulation:

https://www.examtopics.com/exams/amazon/aws-certified-solutions-architect-associate-saa-c03/

1. IntelliJ Licensed
2. Docker
3. Github account
4. AWS -> S3 bucket setup
5. Mic + Cam + Screen Share

Memory handler

singleton

Four access modifiers

public, private

default, protected

static scope: field, method, class, block, JVM memory.

final scope: field, method, class.

Final vs finally finalized

**Day 2**

Public private default protected

Object/instance vs template

Object/instance when use new keyword

Class vs interface vs abstract class vs enum vs annotation

(JVM) Stack, heap, PC, method area, native method stack

All object created are stored in heap, and source code are in method area,

Static stuff are stored in method area, only executed once.

final: for a field, the reference cannot be changed

for a method, cannot be override

public final Employee{

final List<Friends> friends;

}

Employee e1 = new Employee();

E1.getFriends().add(new Employee(“xxx”)); //deep copy

Override vs overload

Runtime vs compile time polymorphism

In-between classes vs within same class.

8 primitive data type,

Factory design pattern

1. Create a interface
2. Create two payment method and implement the interface, and override the pay method
3. Create a factory class and inside create a payment method, pass payment type and return new payment class

Throwable

Exception vs Error

Extends RuntimeException, extends Exception

Collection framework

Day 3

Singleton review,

Collection review,

OOP

SOLID principle

Edge condition

Overloading: same method signature, different parameters order, type.

The finalize() method in Java is a special method defined in the java.lang.Object class. It is called by the garbage collector just before an object is destroyed and its memory is reclaimed.

Exception status code,

Exception message

Day4

String-> constant String pool -> heap

String vs StringBuilder vs StringBuffer

String comparison equals vs ==

Interger i1 = Integer.valueOf(1); Interger i2 = Integer.valueOf(1); System.out.println(i1==i2);

The expression i1 == i2 will be true. This is because of the **integer cache**, a Java performance optimization for Integer wrapper objects created by the Integer.valueOf() method

Treeset is ordered (comparable vs compactor)

Collection

List – ArrayList, LinkedList

Queue – PriorityQueue, deque,

Set – hashset, treeset

Map – HashMap, HashTable, ConcurrentHashMap

**Day 5**

Difference between program, process, thread

In java, how to share memory between thread, volatile keyword

Thread Life Cycle

Different versions of JDK and their features

JDK 8:

Functional interface, Lambda expressions, Stream API, Optional, Default method, Reference Method

JDK 11:

Standardized HTTP Client,

Var in lambda parameters

String methods: isBlank(), lines()

Flight recorder

JDK 17

Sealed classes

Pattern matching for switch

Records

Instanceof pattern matching

Text blocks

JDK 21

Virtual Threads

Sequenced collections

Record patterns