Summary

Maven

- local, central, remote repository
- life cycle
- command line for maven

Git

Java

- primitive type
  - byte, short
- wrapper class
  - Byte, Short..
- autoboxing and unboxing
- String/StringBuilder/StringBuffer
  - what different between them?
    - immutable
    - threadsafe
    - which situation prefer which one
- String, Integer constant pool
  - what situation will use pool?
- equals/hashCode
  - why need to override equals() and hashCode() (logical backhand)

- Collection
  - List, Set, Queue, Map
  - list vs set
    - list can set order, and access by index
    - set elements can not allow duplicated values
  - arraylist vs linkedlist
    - dynamic array
      - `Collections.synchronizedList(new ArrayList<YourClassNameHere>())` - Thread Safe
    - Linked nodes?
  - heap (**completed tree**)-> PriorityQueue
  - deque -> ArrayDeque
    - arrayDeque implement deque
  - HashMap vs HashTable vs ConcurrentHashMap
    - concurrentHashMap
    - In ConcurrentHashMap, the Object is divided into a number of segments according to the concurrency level. The default concurrency-level of ConcurrentHashMap is 16.
  - HastSet, TreeSet, LinkedHashSet
    - TreeSet
      - ordered by redblackTree
    - LinkedHashSet
      - keep insertion order of set
    - HashSet → based on HashMap?
      - unorder
  - TreeMap, LinkedHashMap

        - TreeMap → Keys are ordered,
        - linkedHashMap - > first unique character in a String
            - an additional feature of maintaining the order of the inserted element.
    - Stack, Queue
        - Stack → FILO
        - Queue → FIFO
    - array
        - Integer array, 1D array
        - 2D array
    - binary tree, balanced binary tree
        - Balance binary tree: is commonly defined as a binary tree in which the height of the left and right subtrees of every node differ by 1 or less

- Comparator vs Comparable
    - Comparable internal order, natural order
    - comparator, third party judge
- **JVM**
    - classloader
        - Loading
            - bootstrap/extension/application classloader
                - application class loader
                    - classes in application classpath (like classes written by ourselves)
        - Linking
        - Initialization

        -
    - runtime data area/Memory Area
        - What’s the functionality for each of them?
        - method areas, heap, stack, PC register, native method stack
        - what’s the data store in method area
        - what’s in the heap?
        - what’s in the stack?
        - what’s is PC register?
        - what’s native method stack?
    - Execution Engineer
        - interpreter, JIT compiler, Garbage Collector
    - Native method interface, Native method Library
        - native is a keyword in Java, what’s native means?
- **Garbage Collector**
    - types
        - serial GC
        - parallel GC
        - G1 GC
        - note: CMS GC
            - deprecated since java 9, completely removed in java 14
    - GC Process
        - young generation
            - eden, S0, S1
        - old generation
        - Permanent Generation
- Keywords

- 53 keywords
  - reserved literals: true, false, null
  - unused keywords: goto, const
  - used keywords: 48
    - (**data types**)byte,short,int,long,float,double,char,boolean
    - (**flow control**)if ,else, switch ,case ,default ,for ,do ,while ,break ,continue, return
      -
    - (**modifiers**)public private protected, static, final, abstract, synchronized, native, stripfp, transient, volatile
      - strictfp
        - used in java for restricting floating-point calculations and ensuring the same result on every platform while performing operations in the floating-point variable.
      - volatile
        - volatile must be used **whenever you want a mutable variable to be accessed by multiple threads**
    - (**exception handling**)try, catch, finally ,throw, throws, assert(1.4 version)
    - (**Class** related keywords) class, package, import, extends, implements, interface
    - (**Object** related keywords)new, instanceof, super ,this
      - this() vs. super()
        - super() as well as this() both are used to make **constructor calls**
        - super() is used to call(run/运行) **Base (i.e, Parent's class)**class's constructor
        - this() is used to call **current class's constructor**

        - i.e. this(10), this(10, “abc”)
    - **Attention: constructor can have multiple ones (with different signature)**

Most frequent interview question:

- final vs. finally vs. finalized
    - finalized: used **to perform cleanup operations on unmanaged resources held by the current object before the object is destroyed**
- what’s is volatile
- what’s static
- implements vs extends
- immutable class
- **throw vs throws**
    -
    - **Throws** is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code.

- OOP
    - abstraction

      how do we achieve abstraction?
        - Abstract class
        - interface
    - encapsulation
        - declare variable as private
        - declare setter and getter
            - not access variable directly
    - inheritance
        - extends,

        - implements
    - **polymorphism**
        - override
        - overload
    - public protected, default, private scope
- Exception
    - checked exception vs unchecked exception and **name some examples**
    - **exception vs error**
    - how to handle exception
    - how to **customize** exception
    - how to handle multiple exception
- try with resources
    - AutoCloseable interface
    - override close()
- **Generics Basics**
    - what ? how? why?
    - E vs ?(question mark)
        - what's the different between E and ? in Generics
    - what's type erasure?
        - https://www.cnblogs.com/flydean/p/java-type-erasure.html
        - List<int> wrong?
        - type erasure will change to type to object, primitive type extend object
- **IO Stream**
    - Byte/ Character Stream (category)
    - InputStream/ OutputStream,(serialization)
    - Reader/ Writer (upper level class?)

    - File
- Serialization and deserialization

  how do we do the Serialization and deserialization?

  use input/output stream

  serialVersionVID
    - transient keyword
    - Serializable interface
- Java 8 features
    - lambda (video YT)
        - what? why? how?
        - lambda is used to do the functional programming
        - how do we create lambda?
        - All the variable created by lambda is immutable, so thread safe
    - **Functional Interface**
        - @FuncationalInterface
        - pre-define functional Interface
            - Predicate -> test
            - Function -> apply
            - Consumer -> accept
            - Supplier -> get
    - Optional
        - what? why? how?
        - optional used to prevent what's kinds of problems
            - advantage of optional is don't need to if else to do the null-check every time
            - prevent NPE

        - of, ofNullable, orElse, orElseThrow
        - orElseThrow(): if input is null, it can throw exception
    - Stream API
        - intermediate operation → return Stream
        - terminal operation → return null Stream
        - API:
            - **map vs. flatmap**
            - **how to transfer list to map**
    - Method reference *
- **multi thread**
    - process vs thread
    - Thread state
        - new, runnable, waiting, timed_waiting, blocked, terminated
        - how does one state transfer to another state?
    - how to create a thread? When should I use which one?
        - 4 ways to create
        - when use run(), When use call()
    - runnable vs callable
- Thread Pool
    - customized thread pool
        - threadPoolExecutor
            - corePoolSize
            - maximumPoolSize
            - keepAliveTime
            - unit
            - workQueue

        - BlockingQueue - threadsafe
        - extends Queue → FIFL
    - threadFactory
    - handler (how do we reject)
        - **AbortPolicy**
        - **CallerRunPolicy**
        - **DiscardPolicy**
        - **DiscardOldestPolicy**
  - inbuild(Predefined) thread pool (4)

    what different between them? when should we use them?
    - newFixedThreadPool
    - newSingleThreadExecutor
    - newCachedThreadPool
    - newScheduledThreadPool
  - **completableFuture vs Future**
    - (YT)
- **Lock**
  - synchronized
  - Lock interface
    - reentrant lock
  - ReadWriteLock interface
    - ReentrantReadWriteLock
  - synchronized keyword can be used on method, block, object, static method
    - synchronized static method vs. synchronized non-static method?
  - Fair lock / unfair lock
    - Fair lock: first come, first served.

-

```
public ReentrantLock() { sync = new NonfairSync(); }
```

```
public ReentrantReadWriteLock(boolean fair) {
    sync = fair ? new FairSync() : new NonfairSync();
    readerLock = new ReadLock(this);
    writerLock = new WriteLock(this);
}
```

Other:

> A ReentrantLock is *unstructured*, unlike `synchronized` constructs -- i.e. you don't need to use a block structure for locking and can even hold a lock across methods. An example:
>
> ```
> private ReentrantLock lock;
>
> public void foo() {
>   ...
>   lock.lock();
>   ...
> }
>
> public void bar() {
>   ...
>   lock.unlock();
>   ...
> }
> ```
>
> Such flow is impossible to represent via a single monitor in a `synchronized` construct.

- Enum
  - why we should use Enum?
    - Enum used to create some constant value
    - some method in Enum,
      - like Enum.values() → get all values
      - ordinal() → get index