1.Maven

* Types of Repositories
  + local
  + central
  + remote
  + life cycle
  + validate, compile, test, package (jar), verify, install, deploy

2.Git

* 2 ways to push
  + IntelliJ
  + Terminal

3.Eight Basic Data Types

* primitive type
* wrapper class
* autoboxing & unboxing

4.String/StringBuilder/StringBuffer

* immutable
* String
  + String constant pool
    - ==
    - equals()
    - Integer (constant pool) limited range -127~128
  + mutable
* StringBuilder (not thread safe)
* StringBuffer (thread safe)

5. String/String constant pool

* String constant pool
  + ==
  + equals()
  + Integer (constant pool) limited range -127~128
* == vs equals()
  + reference vs content
  + internal work at hashmap (important)

6. equals / hashcode

* override equals() and hashcode()
* return the true value of equals().

7. Collection

* List
  + ArrayList
    - insert/remove takes O(n)
    - retrieve data O(1) (random access)
  + LinkedList
    - insert/remove takes O(1)
    - retrieve data O(n)
  + ~~Vector~~ (not under Collection)
    - Deprecated
    - thread safe
* Set (unique)
  + HashSet HashMap
    - doesn’t keep insertion order
    - internal work by HashMap (store key (unique), value = null)
  + LinkedHashSet
    - insertion order
  + TreeSet
    - sorted
* Deque (double-ended queue)
  + ArrayDeque
    - deque.offerFirst(), deque.pollFirst()
  + ~~Stack~~ (not under Collection)
    - push(), pop()
    - Deprecated
    - thread safe
* Map (not implements Collection)
  + HashMap
    - internal work
      * hashcode()
      * equals()
      * separate chaining (Linked List) O(n)
        + red-black tree (JAVA 8) O(logn)
  + ~~HashTable~~
    - Deprecated
    - thread safe (1 lock)
  + ConcurrentHashMap
    - thread safe (16 lock)
  + LinkedHashMap
    - Insertion Oder
  + TreeMap
    - Sorted (by key)
* Data Structure
  + Queue (FIFO) Stack(FILO)
  + Heap
    - by PriorityQueue
    - maxHeap
    - minHeap
  + array
    - int[], String[], …
    - int[][], String[][], …
* compare
  + list vs set
  + HashMap vs HashTable vs ConcurrentHashMap

8, Comparator vs Comparable

* maintain TreeSet (TreeMap)
* Comparable
  + class implements Comparable<class>
  + Override compareTo(Object o)
  + return this.attribute - o.attribute (ascending order)
* Comparator
  + create class implements Comparator<class>
  + Override compare(DataType o1, DataType o2)
  + return o1.attribute – o2.attribute

9.&10. JVM

* Class Loader (main memory)
  + Loading (parent delegation mechanism, recursive method, super class of lower level class))
    - Bootstrap Class Loader (java.util, java,lang, …)
    - Extension Class Loader (JDBC driver, ODBC driver)

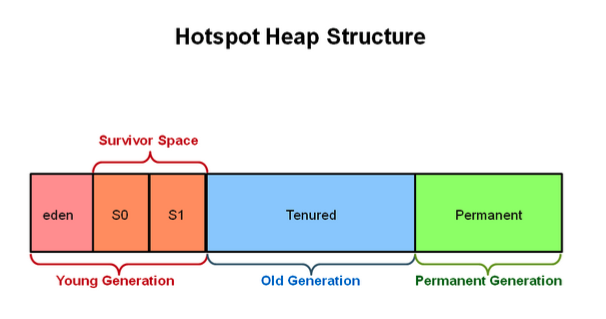
* + - Application Class Loader (User define class)

ClassLoader Sub System

* + Linking (3 steps)
    - Verify (correctness of .class file)
    - Prepare (allocated memory for static fields for class or interface)
    - Resolve (the process of dynamically determining concrete values from symbolic references in the run-time constant pool.)
  + Initialization
    - initialize (initialize static block, static class, constant pool…)
* Runtime Memory/Data Area
  + Method Area (class level data, static field)
  + Heap Area (object, new keyword)
  + Stack Area (private by single thread, call method))
    - Threads
  + PC Register (execution address of current thread)
    - Threads
  + Native Method Stack (implement by other language like c/c++)
* Execution Engine
  + Interpreter (execute byte code line by line)
  + JIT Compiler (find hotspot to improve performance of Interpreter)
  + Garbage Collector
* Native Method Interface (JNI) (bridge between execution engine and Native Method Library)
* Native Method Library (implement by other language like c/c++)

11. Garbage Collector

* serial GC (single thread)
* parallel GC (multithread, default in JAVA)
* G1 GC (by rank of chunk)
* CMS GC(Concurrent Mark Sweep) G1
  + deprecated since java 9
  + completely removed in java 14



* minorGC -
  + Young Generation
  + more frequently
* majorGC -
  + Old Generation
  + sometimes

12. Keywords

* 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, strictfp, transient, volatile
* exception handling
  + (try, catch, finally), (throw, throws), assert
* class related
  + class, package, import, extends, implements, interface
* Object related keywords,
  + new, instanceof, super, this
* final finally finalize()
  + variable - constant, must be initialized
  + method - can’t be overridden
  + class - can’t be extended
  + container (like list) – can modify the content, can’t address to different reference (new container).
* Immutable class
  + final class
  + private final fields
  + no setter
  + return deep copy of the collections for getter
* static (method area)
  + block
  + variable
  + methods
  + classes

13. OOP

* Abstraction
  + abstract class
  + interface
* Encapsulation
  + private variables
  + getter & setter
* Inheritance
  + extends (single class)
  + implements (multiple interfaces)
* Polymorphism
  + override (method of parent class)
  + overload (same name with diff inputs)

14. Exception

* checked exception
  + IO or Compile time exception
* unchecked exception
  + Runtime or Null Pointers Exception
* Handle Exception
  + try catch (finally)
  + throws
  + throw vs throws
* customize exception
  + user class extends Exception
* handle multiple exceptions
  + catch(E1){}

catch(E2){}…

* + catch(E1|E2|E3…){}

15. Generics - <K, V, E> ..

* easier and less error-prone
* enforce type correctness at compile time
* without causing any extra overhead to your application

16.IO Stream

* Byte Stream
  + 1 Byte
  + InputStream, OutputStream
* Character Stream
  + 2 Bytes
  + Reader, Writer
* File
  + part of java.io
  + access to underlying file systems

17. Serialization and deserialization

* serialization
  + implements Serializable (Marker Interface)
  + transient
  + Object -> server
  + kafka
* deserialization
* server -> Object
* see data in kafka

18. Java 8 features

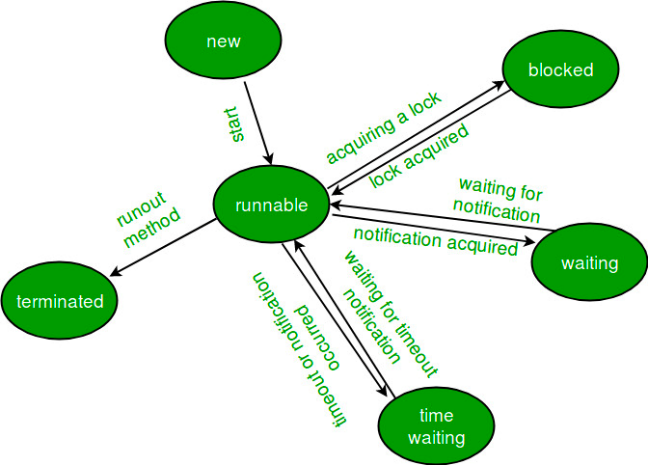
* lambda
  + (arguments) -> {body}
  + functional programming
  + less code
* Functional Interface
  + with only one abstract method
  + allow # of default (concrete) method
  + can define by lambda expression in main()
  + Predicate - test(T t), Fuction - apply(T t),

Consumer - accept(T t), Supplier - get()

* Stream API (can define by lambda)
  + intermedia - return a stream
    - map( -> ), filter( -> )...
    - map() // append 03/09/2022
      * applies function to each element
      * single value
      * transformed
    - flatmap() = map() + flatting
      * applies function to each element
      * stream of new values
      * flatten
  + terminal - return nun-stream
    - forEach( -> ), collect( -> )…
* Optional
  + to handle NullPointerExcception
* CompletableFuture // append 03/09/2022
  + Future
    - Interface for return val of Callable<T>
    - Difficult to coordinate multiple futures
  + CompletableFuture
    - new in JAVA 8
    - deal with multiple Future

19.Multi-threading

* process - independent memory space, heap, OS resources
* thread - shared memory space, private stack, program counter, register
* states (understands with pic)
  + new - create, not start
  + runnable - execute in JVM
  + blocked - wait for a monitor lock to enter synchronized block or method
  + waiting
    - Object.wait with no timeout
    - Thread.join() with no timeout
    - park()
  + time\_waiting
    - thread sleep
    - Object.wait() with timeout
    - Thread.join() with timeout
    - park
  + terminated – thread has completed



* thread creation
  + extends Thread
  + implements Runnable
    - no return
    - no exception
    - run();
  + implements Callable
    - has return
    - has exception
    - call();
* thread pool
  + corePoolSize
  + maximumPoolSize
  + KeepAliveTime
  + Time unit
  + work queue
  + thread factory
  + handler
    - occurred when # of thread > max + queue(#)
    - abortPolicy
    - callerRunPolicy
    - discardPolicy
    - discardOldestPolicy
  + in-built thread pool (not often use)
    - newFixedThreadPool(nThreads)
      * core = max = nThreads
    - newSingleThreadExecutor()
      * core = max = 1
    - newCachedThreadPool()
      * core = 0, max = MAX\_VALUE
    - SchedualThreadPoolExecutor(corePoolSize)
      * corePoolSize, max = MAX\_VALUE
    - Problem: OutOfMemoryError
      * LinkedBlockQueue
      * max = MAX\_VALUE
* Lock
  + synchronized
    - block
    - method
    - static method
    - class
  + Lock interface
    - method:
      * lock(), unlock(), newCondition(), tryLock(), lockInterruptibly()
    - implements class:
      * ReentrantLock
  + ReadWriteLock interface
    - method
      * Lock readLock();
      * Lock writeLock();
    - class
      * reentrantReadWriteLock

---------------------------------

20. Database

* DBMS - Database Management System
  + MySQL, SQL Server, …
* SQL - Structured Query Language
  + different DBMS has minor different SQL
* file system and database
  + DBMS
    - manage database
    - advantages:
      * no redundant data
      * efficient query processing
      * more data consistency
      * more security
    - disadvantages
      * higher cost
  + File system
    - manage files in storage medium
    - advantages:
      * less expensive
    - disadvantages:
      * redundant data
      * no efficient query processing
      * less data consistency
      * less security

21. Database Normalization

* eliminate redundant data and ensure data is stored logically
* 1NF
  + single value in each table cell (atomic)
  + each record (row) needs to be unique
* 2NF
  + be in 1NF
  + single column primary key
* 3NF
  + be in 2NF
  + has no transitive functional dependencies
* normalization denormalization

22. Non-relational Database and no sql

* key/value data store: redis, riak
* graph data store: Neo4j, GraphDB
* document data store: mongoDB, CouchDB
* columnar data store : cassandra, Hbase
* Follow CAP
  + Consistency: all clients always have the same view of data
  + Availability: each client can always read and write
  + Partition Tolerance: the system works despite physical network partitions
  + CP: redis, MongoDB
  + AP: Cassandra
  + CAP: none

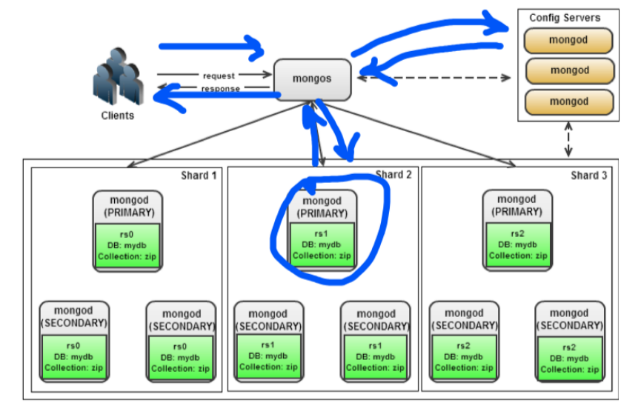
23. Sharding and replica:

* Sharding - distribute a single logical database across a cluster of machine.
* replica - secondary node, redundancy, failover

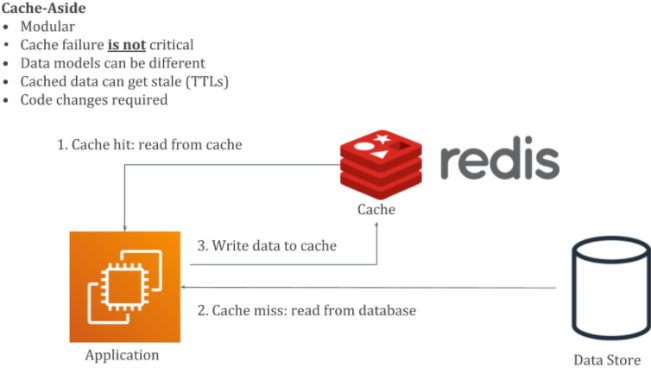
24. sql vs no-sql

* sql
  + relational database
  + pre-defined schema
  + vertical scaling
  + ACID
  + not suited for hierarchical data store
* no-sql
  + non-relational database
  + dynamic schema
  + horizontal scaling
  + CAP
  + suited for hierarchical data store

25. mongoDB

* no sql & dynamic schema & horizontal scaling
* document datastore
* architecture
  + Mongod
    - database instance
  + Mongos
    - sharding process (like a router)
    - based on the info from config servers
  + config Server
    - info of Mongods
* primary-second node with automated failover (replica)
* follow CP (consistency, partition tolerance)

26. Redis (remote directory server)

* in memory
* key/value data store
* supports different kinds of data structure
  + String
  + List
  + sets
  + sorted sets
  + hashes
* deal with JWT token store in section with different server problem.
* cache hit - find out data in cache
* cache miss – not found in cache

28. index

* optimize the performance of query (min # of I/O)
* as data structure
  + B+ tree (default)
    - O(n) -> O(logn)
  + bitmap
  + hashtable
  + r tree …
* clustered index - primary index
  + only one clustered index per table
  + defines the order in which data is physically stored
* non-clustered index - secondary index
  + as many as non-cluster index
  + doesn’t sort the physical data inside the table

29. SQL/Application tuning

* SQL
  + using execution plan to identify the cause of slowness
  + try to reduce joins, remove unused join and join conditions
  + use the index to improve the performance
  + union all instead of union
  + limit
  + view or stored procedure
* application tuning
* check the db query - do the sql tuning
* DB connection usage -> connection pool
* do JVM tuning -> Jstack, JMap, JConsole
* server side: CPU, Memory usage by using commands like top, ps
* code review
* check networking, firewall, load balancer

30. Transaction

* an action, or a series of actions, carried out by a single user or an application program
* Follow ACID
  + Atomicity
    - all transactions are atomic
    - can’t be executed partially
    - commit or rollback
  + Consistency
    - transactions take the database from one consistent state to another state
  + Isolation
    - a transaction is not visible to other transactions until it completes
  + Durability
    - once a transaction has completed, its changes are made permanent

31. Concurrency

* dirty read: read uncommitted data from another transaction
* non-repeatable read: read committed data from an update query from another transaction
* phantom read: read committed data from an insert or delete query from another
* Isolation level

一張含有 桌 的圖片

自動產生的描述

* default in MySQL - Repeatable read
* implements by: Lock + Log (store order of transactions’’ steps)

32. Lock

* manually
  + Read Lock: shared lock
    - for example: read same record of transactions simultaneously
  + Write Lock: exclusive lock
    - lock whole table
* handle by MySQL itself
* Intension Lock (IS or IX)
* DeadLock
  + might happened waiting for Lock finish with different order in transactions
  + solve: using top or ps command in Linux sever find key 3306 (default) of CPU time or resources or RAM. check Locks

34. SQL

* DDL (data definition language)
  + create, drop, alter, truncate,
* DQL(data query language) (R)
  + select … CRUD (create, read, update, delete)
* DML (data manipulation language) (CUD)
  + insert, update, delete
* DCL (data control language)
  + grant, revoke
* DTL (data transaction language)
  + commit, rollback
* aggregation function
  + max(), count(), min(), avg, sum….
* where vs having
  + having (group by),
  + where -> all rows
    - select \* from table where …
* Join vs Union
  + Join - column wise
  + Union - row wise
    - Union all - faster, duplicated rows
* common - intersection
* T1 minus T2 = T1 - common (T1, T2)