String, String Builder, String Buffer

☐ They are all String type classes

☐ compare: String and StringBuffer are both thread safe class, because string is immutable, StringBuffer is Synchronized

StringBuilder is non-sychronized, so it's not thread safe

☐ use: nomally 2 ways to create a String obj

- ☐ call constructor: the reference points to obj on heap
- ☐ use double quote: the reference points to obj on String pool
- ☐ ~~use~~ often use equals to compare obj, rather than "=="

  reason is equals compare content. "==" compares reference

Collection

☐ Collection is an Interface of ~~some~~ in java, and it has many sub-Interf

~~☐ List~~

☐ compare: Map Interface stores ~~a~~ key value pair

- ☐ List:
  - ☐ Arraylist: occupies consecutive memory. O(1) → random access
  - ☐ LinkedList: occupies fragment memory. O(1) → add/delete at the head
  - ☐ Vector: thread safe arraylist
  - ☐ Stack: thread safe stack → use Deque
- ☐ Queue: FIFO / heap
  - ☐ priority Queue
  - ☐ Deque

- ☐ Set : has no duplicate value, ~~fo~~ not follow insert order
  - ☐ hashset
  - ☐ TreeSet : ~~be~~ predifine an order rule
- ☐ Map
  - ☐ HashMap
    - ☐ key value pair, key unique
    - ☐ implemented by array and Linked List
    - ☐ hash collision & equals, hashcode method
    - ☐ hashmap vs ~~set~~ Hashset
    - ☐ hashmap vs hashtable
    - ☐ hashmap vs Concurrent HashMap
  - ☐ TreeMap

Comparator vs Comparable

- ☐ both interface to ~~do~~ set compare rules
- ☐ when comparator , there is an existing class canot modified

JVM

- ☐ class loader : load java classes
  - ☐ loading : to load { boot strap | extension | application
  - ☐ linking : to validation
  - ☐ initialization : initialize classes
- ☐ Runtime data area
  - ☐ method area : static method , constant pool
  - ☐ heap
  - ☐ stack
  - ☐ Pc register : thread position
  - ☐ Native Method
- ☐ Execution Engine
  - ☐ interpreter
  - ☐ JIT compiler
  - ☐ Garbage Collector

Garbage Collector

- ☐ Serial GC
- ☐ parallel GC
- ☐ G1 GC
- ☐ minor GC major GC Full GC

Abstract vs Interface

- ☐ syntax

  abstract class means "is a"

  Interface means "has an ability"

- ☐ logic

  abstract can have instance variable, constructor, implemented method non-public field

  Interface only has public abstract method

What is Thread

- ☐ Thread is an ~~[illegible]~~ independent execution of instructions
- ☐ compare: both thread and process are to implement concurrency
  - ☐ Thread share memory in the same process
    - ☐ like heap, static memory segments, os recource
    - ☐ private memory space, stack, program counter, register stats
  - ☐ process has independant memory space
    - ☐ like stack, heap, os resource
- ☐ data race solution: lock
  - ☐ synchronized
    - ☐ block: synchronized (this){ }, synchronized (Demo. class){ }
    - ☐ static method
    - ☐ method
  - ☐ lock Interface

- ☐ how to use
  - ☐ extends Thread (run())
  - ☐ implements Runnable (run())
  - ☐ implements Callable (call())
    - ☐ compare Runnable: has exception / has return value
  - ☐ Thread pool
    - ☐ customized thread pool : Thread Pool Executor
      It sets max queue, max cores, max total threads, handler
    - ☐ in-built thread pool ExecutorService es1 = Executor. newFixedThread Pool
      ① thread size fixed
      ② new Single Thread Executor ③ newCached Thread Pool ④ new Scheduled Thread Pool

[illegible]
[illegible]

What is Database

The space to store data

File sys	vs	DBMS
① manage files		manage databases
② redundant data		no redundant
③ slow query		efficient query
④ less consis		more consis
⑤ less secure		more secure
⑥ less exp		⑦ higer cost

→ Relational Database (RDBMS)

↓ what is (features)

① predefined schema →

id	name	age

② vertical scaling → add more records (increase the power of ~~each~~ one node

③ ACID

client ~~bank~~ retrieve money from bank ←why— A: atomicity : in transaction, all the operations to data base must be totally completed or none changes are made

regulates the data coming in and reject data ←why— C: consistency: gurantee all data is valid according to defined rules

To make system secure ←why— I: isolation : all transaction can't be affected by others

less error prone ← D= durability : once transaction has been submitted, always in the system

① mapping of obj is difficult ← ④ not suited for hierarchical data store

② this model is not suitable for huge databases

↓ how to query

SQL: structrued query language
(some difference betwen different company)

how to design (database normalization)

1NF : each column of one record only one value
id is unique

id	name	age
1	T	18, 29

2NF : single column primary key $\xrightarrow{[illegible]}$ no composite key

3NF : no transitive functional dependencies

id	age	age+10
1		

no sql $\xrightarrow{\text{what}}$ is ① non-relational database

② dynamic schema $\xrightarrow{\text{how}}$ we can add or delete attributes

③ horizontal scaling $\xrightarrow{\text{how}}$ { sharding : distribute a single database on a cluster servers
| replica : duplicate the database to back up

④ CAP

MongoDB is consistency bc, read and write also on its replica set

C : consistency ~~[illegible]~~ All clients always have the same view of the data

P : partition tolerance : if nodes are partitioned in different distributed system, still work

A : each client can always read and write

4 types : ~~[illegible]~~ column family →

id	identity
1	age: name:

| graph
| document
| key-value

MongoDB what is MongoDB

☐ non-sqL database, document based database { suitable to changable data / don't need join

☐ reason we use : ① dynamic schema : clients can change data schema ezly

② ~~secon~~ supports secondary indexes (non-cluster index)
efficiently locate the target data

③ replica set :
~~help~~ automated failover

④ built in horizontal scaling (sharding)

⑤ follow CP
increase the consistency and partition tolerance

☐ how consistency
when the primary node failed, MongodB will stop write temporily. until Mongo pick one primary node ~~again~~ from secondary nodes

☐ how MongodB work

☐ when client send requst ______ $\boxed{\text{config servers}}$

$\boxed{\text{Mongos}}$: sharding processes, interface between client & sha
~~see~~ can be seen as db router

☐ process the request according to info store on config servers

☐ decide how many which ~~to~~ Mongod receive the query

$\boxed{\text{Mongod}}$ primary
database instance

$\boxed{\text{mongod}}$ secondary

What is Redis (distributed lock)

☐ non-sql database, key-value based database, supports different data structure: String, List, Sets, Sorted Sets, hashes

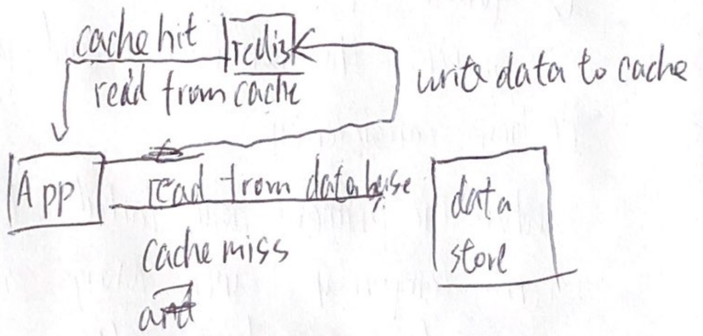
☐ reason: ① in memory store: decrease time between client and database

② persistence mechanisms:

RDB: make snapshots point in time at specific intervals

AOF (append only file): logs every write operation received by the server. It will play again when you need

☐ how:



Index

☐ index is a data structrue, ~~its~~ like index of an Array, used to minimiza ~~the~~ the times to access disk

☐ ~~cluster index~~ 2 types of index

- ☐ cluster index
  - ☐ data is sorted according to the column of cluster index phsica normally the column is primery key
  - ☐ only one per table
- ☐ non - cluster index
  - ☐ compare: it has as many as non cluster comparing to cluster
  - ☐ data structure: like b+ tree , bit map, hashtable

SQL Tuning

☐ check the execution plan
optimizer will show the process, we can see the reason cause slowness

☐ reduce joins ~~and~~ unused joins

☐ index

☐ union all rather than union

☐ Limit. we don't need fetch all data from db

☐ View & stored procedure : All reduce duplicate operations
↓
reduce compile time

Application tuning

- ☐ db → sql tuning
- ☐ db connection → connection pool
- ☐ jvM → Jstack, JMap, Jconsole
- ☐ cpu memory
- ☐ code
- ☐ network

Transaction

- ☐ is a series actions
- ☐ follow ACID

A: atomicity, can't execute partially

C: keep consistent state. For ex. A+B alway same

I: isolation, can't visible to others transaction

D: ~~[illegible]~~ durability, once transaction complete, the changes permanent

Concurrency

- ☐ multiple tasks run simultaneously
- ☐ Three types in transaction
  - ☐ Dirty Data : one transaction read uncommited data from another transact
  - ☐ non-repeatable read : data read 2 times are different in one transactio
    because another transaction's update query
  - ☐ phantom read : the results of read query are different . because insert/delete
    query from another
- ☐ to deal with
  - ☐ set isolation level in MySQL engine
    - ☐ read uncommited
    - ☐ read commited
    - ☐ repeatable read
    - ☐ serializable
  - ☐ lock . it is a variable to show the status of resouce
    - ☐ binary lock : use 0, 1 to show the status
    - ☐ shared lock : allow multiple threads to read , but not write
    - ☐ exclusive lock : only allow one thread to write or read
  - ☐ be careful of dead lock
    a transaction holds one lock and ask for another lock
    - ☐ detect
      wait for graph

SQL

DDL (data defination) create, drop, alter, truncate

DQL (data query) select

DML (data manipulation) insert . update, delete

DCL (data control) grant revoke

DTL (data transaction) commit, rollback

aggregation function : max() . count()