# Evaluation

* Monday & Tuesday,
* 30 mins
* java core (70%) and database (30%)

# 

# 0, Outline

Maven

Git/Github

JVM Runtime Memory Structure

Eight basic data type

String, Integer Object

equals/hashcode method

Collection data structure: List Set Map Queue

Set v.s. List

List: ArrayList v.s. LinkedList

Set: HashSet, TreeSet, LinkedHashSet

Map: HashMap, ConcurrentHashMap

Comparator v.s. Comparable

Keyword

class v.s. interface

primitive data type

Auto-boxing/Unboxing

access modifier

default v.s. protected

inheritance

implements v.s. extends

control flow:

break v.s. continues

instantiation:

this v.s. super

immutable class

final, static

OOP: encapsulation

abstraction

polymorphism:

1. Inheritance

2. Override and Overload.

3. Upper casting

Exception Handling

Exception v.s. Error

Exception: checked/unchecked Exception

Exception Handling: 1. try-catch-finally (try with resource) 2. throws

Customized Exception

Generic in Java

what is generic

why is generic

generic template

customized own generic type

I/O Stream:

File class

InputStream/OutputStream

FileReader/FileWriter

Byte Stream / Character Stream

try-with-resource

multiple streams

autocloseable interface

Stream resources in GC

close stream

I/O Stream:

BufferedStream

performance increment

close BufferedStream resource

Java 8 features:

Lambda Expression

Comparator

Runnable

Functional Interface

Consumer

Supplier

Predicate

Function

Method Reference

object::implemented method

class::static method

class:: implemented method

Optional:

handle NullPointerException

empty()

isPresent()

of()

ofNullable()

Stream API:

intermediate operations

terminal operations

Multithreading:

four ways to create thread:

Thread class

Runnable interface

Callable interface

ThreadPool

thread-safe block: synchroznied keyword + object locker

thread-safe Singleton design pattern

deadlock

threadPoolExecutor

ExecutorService

Executors

ThreadPoolExecutor

customized threadPool

corePoolSize

maximumPoolSize

keepAliveTime

unit

workQueue

threadFactory

handler

fork join pool

RecursiveFutureTask

completableFuture

Asynchronous

# github: <https://github.com/Arthur-Shuahua-Zhang/JavaBatch23>

# 1, Maven

Types of Repositories

* local
* central
* remote

local repository:

* Unix/Mac OS X – ~/.m2
* Windows – C:\Documents and Settings\{your-username}\.m2.

Central repository

remote:

life cycle:

* validate:
* compile
* test
* package
  + war, jar, docker….
* verify
* install:
* …

mvn clean

mvn test

mvn install

# 2, Git

git vs svn

github,

* intellij
* terminal

Homework1.1

* install maven, git, java 11/8, intellij/eclipse
* create public repository in github
* **send the github link to zoom group**
* **lms assignment**

# 3, Eight basic data types

primitive type

* byte 8 bits
* short 16 bits
* int 32 -2^31 -> (2^31) -1
* long 64
* float 32
* double 64
* char 16
* boolean 1

wrapper class

* Byte
* Short
* Integer
* Long
* Float
* Double
* Character
* Boolean

autoboxing and unboxing

# 4, String, StringBuilder, StringBuffer

String

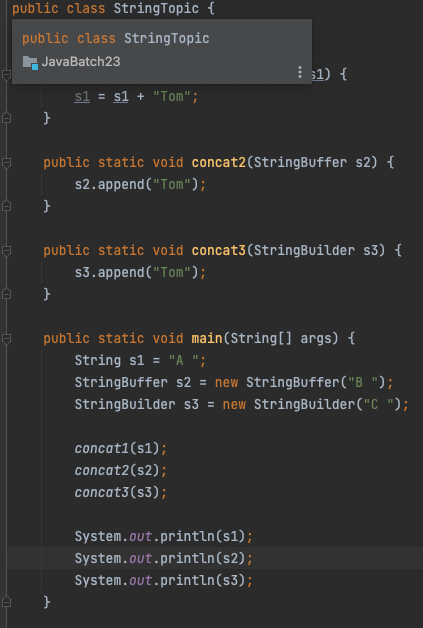
* inmutable
* thread safe

StringBuilder

* mutable
* not thread safe

StringBuffer

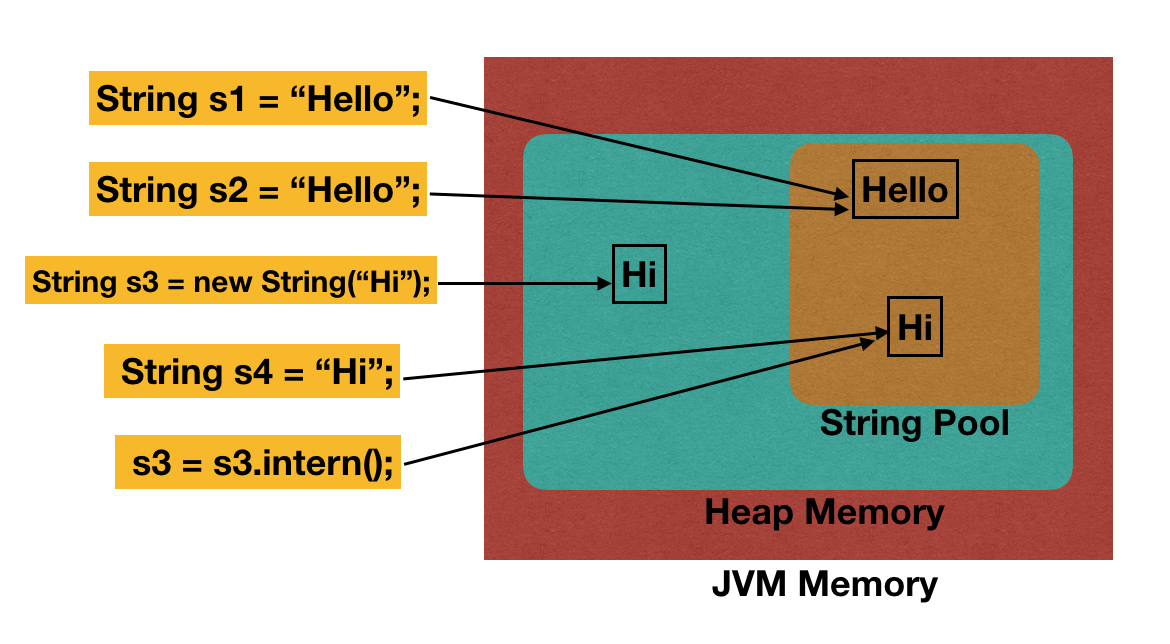
* mutable
* thread safe



# 5, String/Integer constant pool

heap: new

string pool: “abc”





# 6, Equals/ HashCode

equals() vs ==

| public class EqualsHashCode {  public static void main(String[] args) {   EHNode node1 = new EHNode(1);  EHNode node2 = new EHNode(1);   System.out.println(node1 == node2); *// false*  System.out.println(node1.equals(node2)); *// true*   Map<EHNode, Integer> map = new HashMap<>();  map.put(node1, 1);  map.put(node2, 1);   EHNode node3 = new EHNode(1);  System.out.println(map.get(node3)); *// 1*   } }  class EHNode {  int a;  EHNode(int a) {  this.a = a;  }   *//Object.equals() the same as ==*  @Override  public boolean equals(Object o) {  if (o == this) return true;  if ( !( o instanceof EHNode)) return false;  EHNode other = (EHNode) o;  return this.a == other.a;  }   @Override  public int hashCode() {  return a \* 31;  } } |
| --- |

# 7, Collection

## **untitled (20).jpg**

## List

* ArrayList

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

O(1)

* LinkedList

node1 < – > node2 <- > node3

| node1  node1\_address  next(node2) |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  | node2  node2\_address  next(node3) |  |
|  |  |  |  |  |
| node3  node3\_address  next(...) |  |  |  |  |

Vector

* thread safe

Stack

* thread safe, FILO

Deque -> ArrayDeque

* deque.offerFirst()
* deque.pollFirst()

Set

* HashSet
* TreeSet
* LinkedHashSet

unique/ don’t keep the insertion order

Map

* HashMap
* LinkedHashMap
* TreeMap
* HashTable
* ConcurrentHashMap

Queue

* FIFO

Heap

* PriorityQueue
* minheap
* maxheap

root

/ \

left right

array

* int[] String[], Object[] ..
* int[][] char[][]

Binary Tree

root

/ \

null right

list vs set

hashmap vs hashtable vs concurrentHashMap

hashmap

* key-value
* allows one null key, which will be in the first bucket(index 0)

hashtable

* key-value
* thread safe
* object level lock
* don’t allow null key

concurrentHashMap

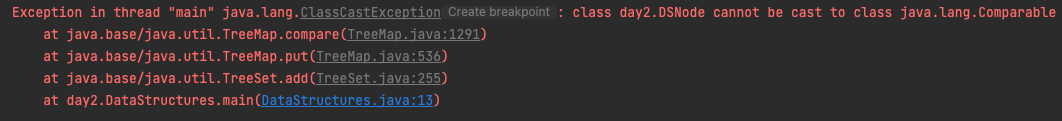
* key-value
* thread safe
* bucket level lock

HashSet <- HashMap

* HashMap : <Key, Value>
* HashSet: <Key, Dummy Object> == hashmap.keySet();

Nots: TreeSet, TreeMap

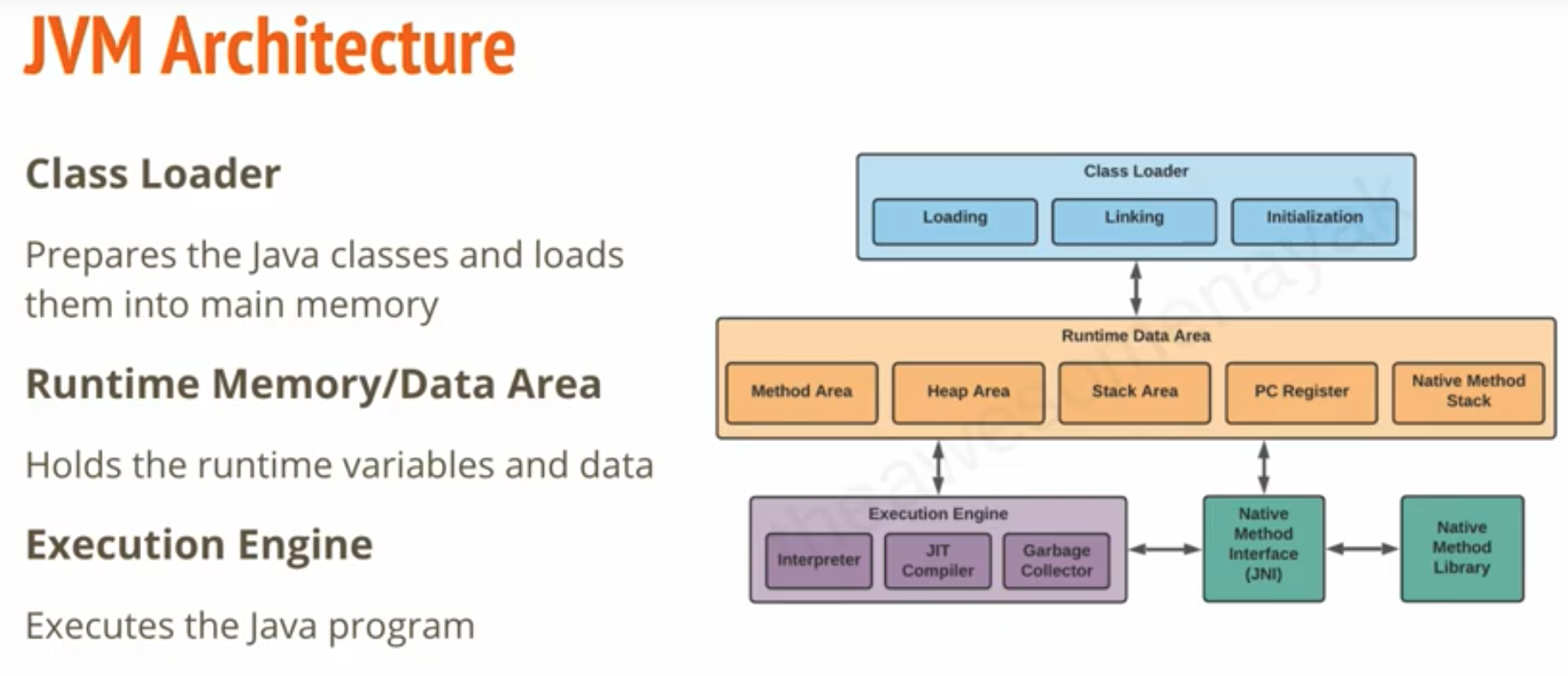


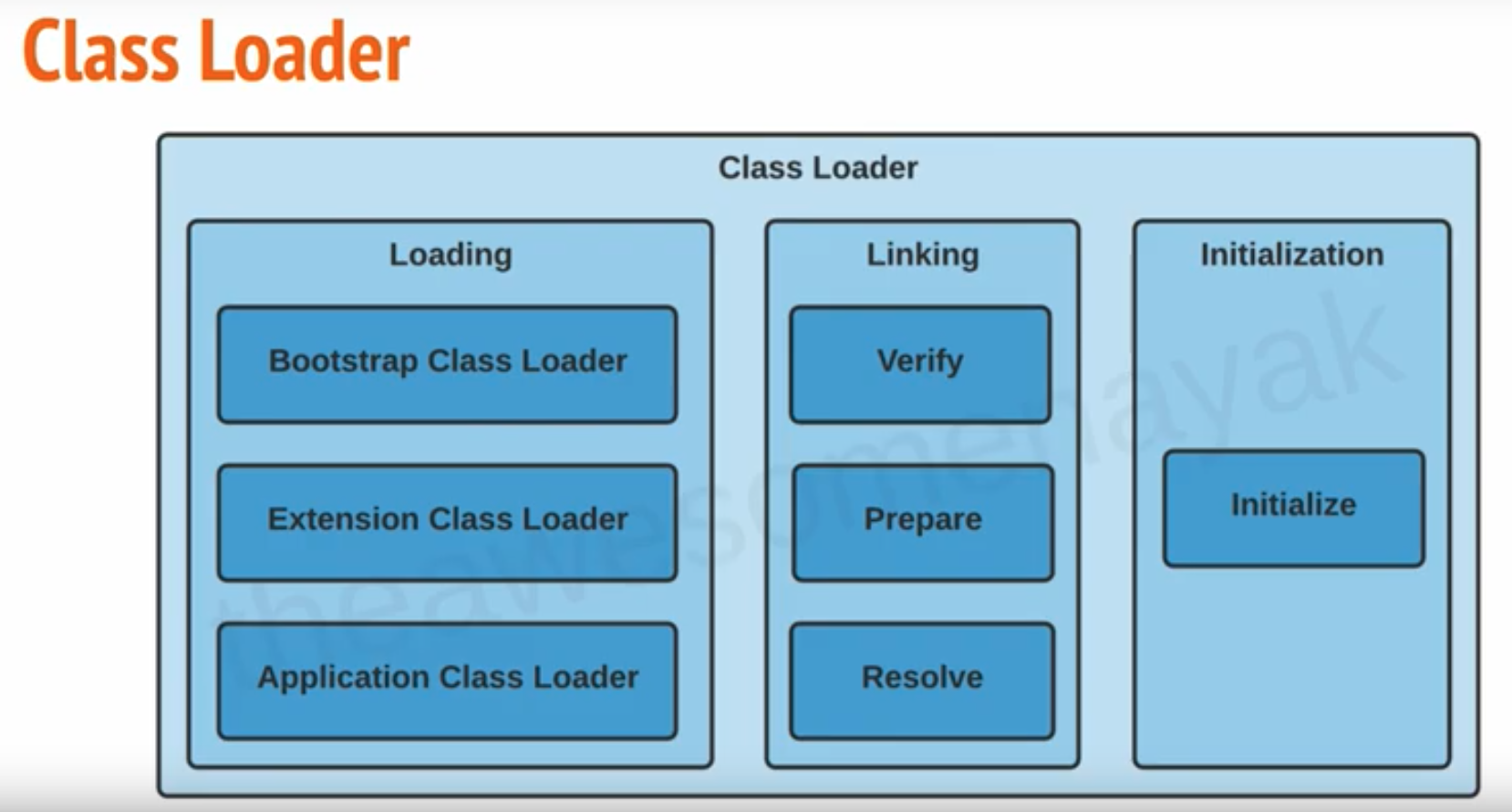


# 8, Comparator vs Comparable

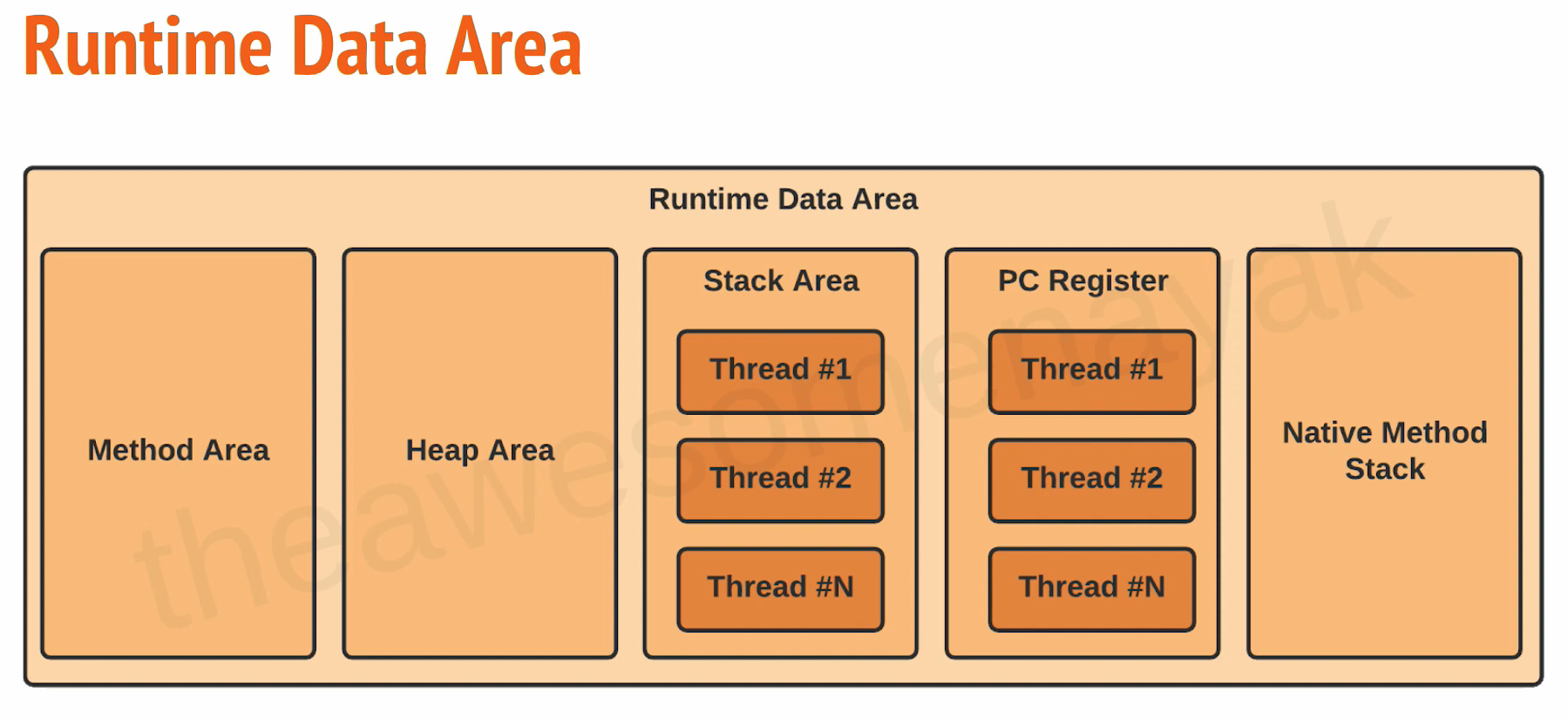
| public class ComparatorComparable {  public static void main(String[] args) { *// Queue<CCNode> heap = new PriorityQueue<>();* *// heap.offer(new CCNode(2));* *// heap.offer(new CCNode(1));* *// heap.offer(new CCNode(3));* *// System.out.println(heap.peek().value);*  *// Set<CCNode> set = new TreeSet<>(new Comparator<CCNode>() {* *// @Override* *// public int compare(CCNode o1, CCNode o2) {* *// return o2.value - o1.value;* *// }* *// });*  Set<CCNode> set = new TreeSet<>((n1, n2) -> n2.value - n1.value);  set.add(new CCNode(2));  set.add(new CCNode(1));  set.add(new CCNode(3));  System.out.println(set.stream().findFirst().get().value);   }  }  class MyComparator implements Comparator<CCNode> {  @Override  public int compare(CCNode n1, CCNode n2) {  return n2.value - n1.value;  } }  class CCNode implements Comparable<CCNode>{  int value;  CCNode(int value) {  this.value = value;  }   @Override  public int compareTo(CCNode other) {  if (this.value == other.value) {  return 0;  }  return this.value < other.value? -1: 1; *// bit wise operation*  *// return this.value - other.value; // integer overflow Integer.MIN\_VALUE - Integer.MAX\_VALUE;*  }   *// int res = node1.compareTo(node2)*  *// Node (value, key, firstName, lastName)* } |
| --- |

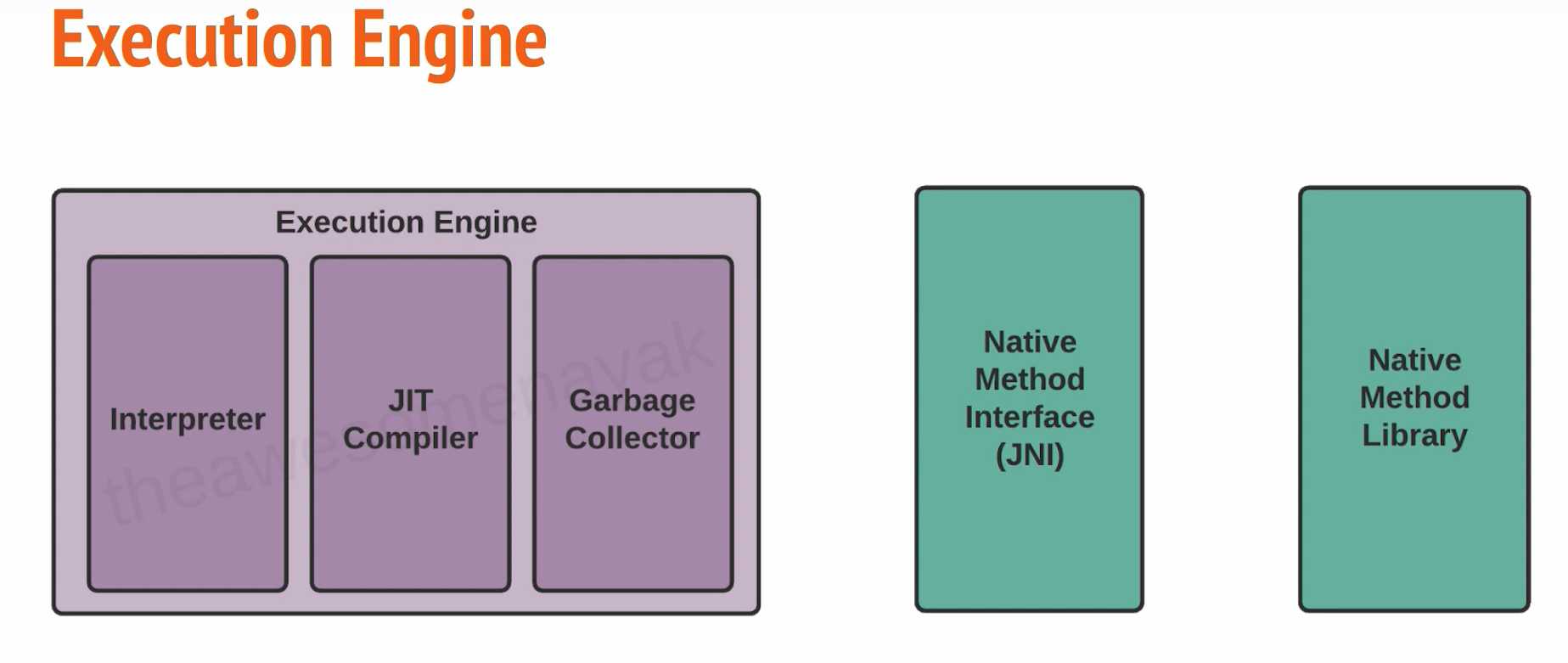
# 9, JVM





java.lang, java.net





Homework 2.1

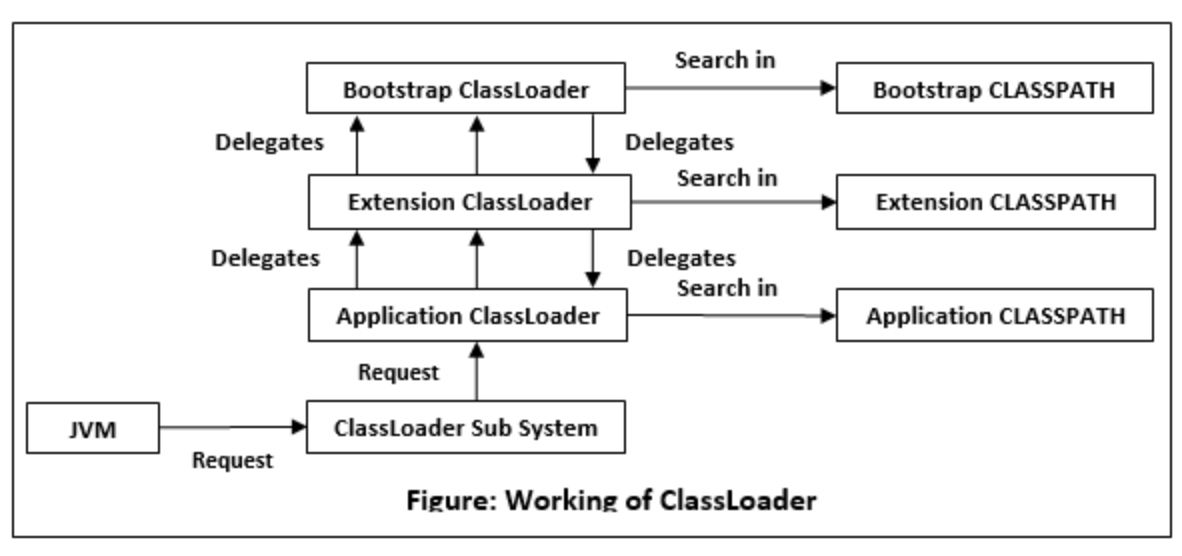
* what is JVM
* play around comparator, comparable, data structures
* put into your github repo

# 10, Class Loader

BootStrap classloader

extension classloader

application classloader



# 11, Garbage Collector

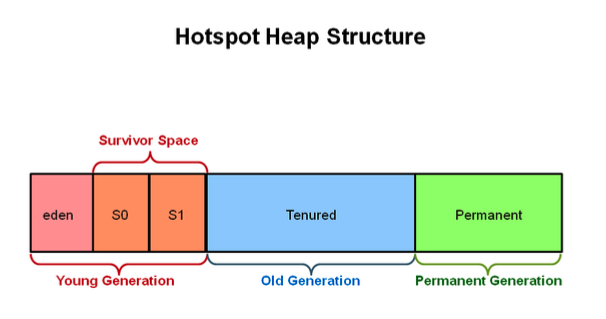
* serial GC
  + -XX:+UseSerialGC
* parallel GC
  + -XX:UseParallelGC
* G1 GC
  + divide heap into equal sized chunks
  + -XX:++G1GC

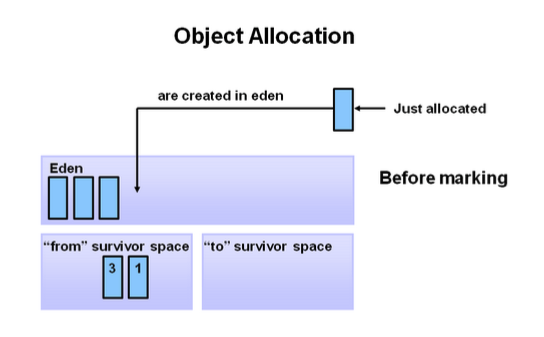
| chunk1  rank1 | chunk2 | …rank3 |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | rank2 |  |
|  |  |  |  |

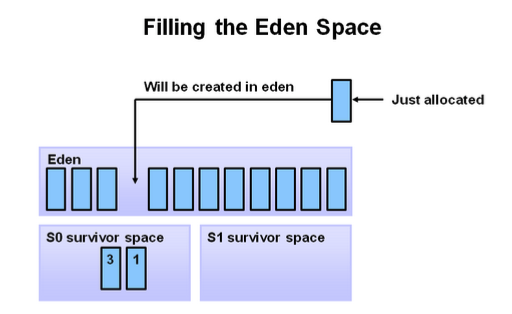
CMS(Concurrent Mark Sweep)

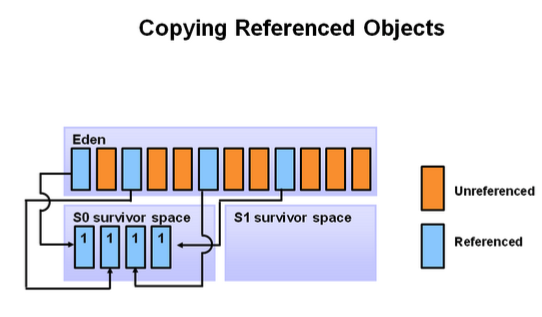
* deprecated since java 9
* completely removed in java 14

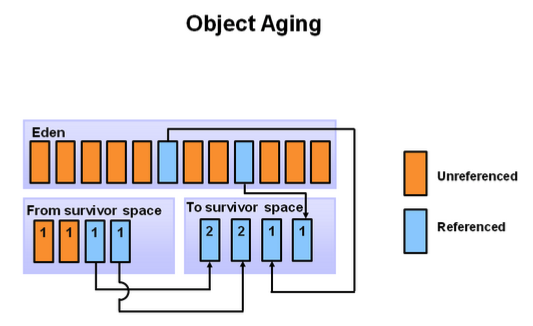
GC process

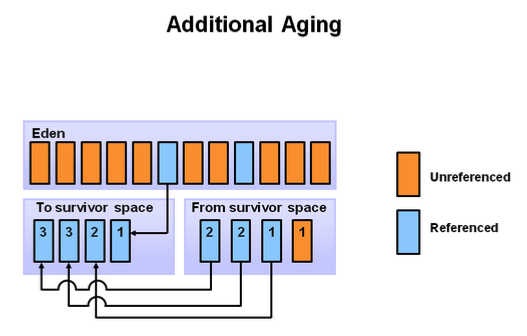


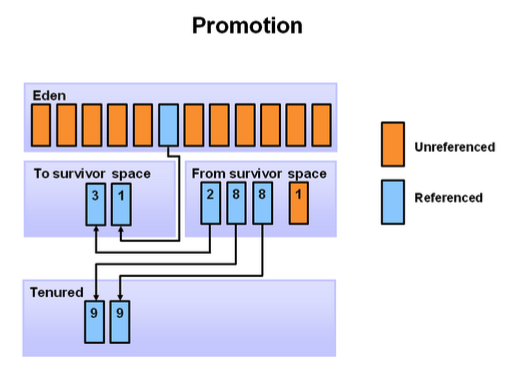






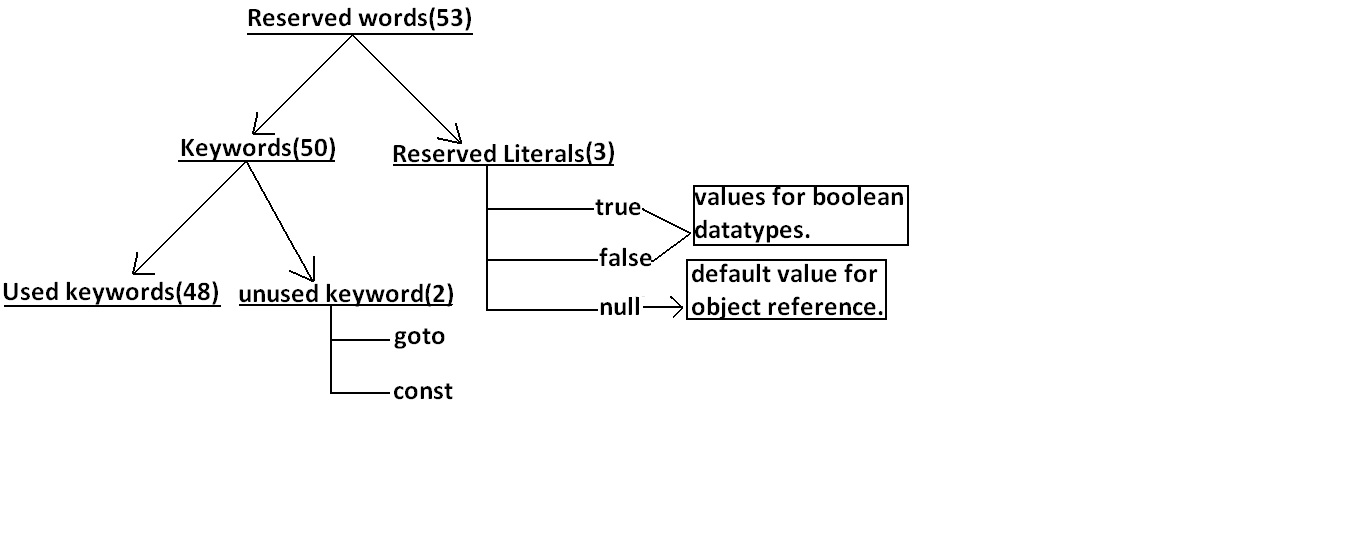








# 12, Keywords



* Reserved words for data types:
  + byte,short,int,long,float,double,char,boolean
* Reserved words for flow control:
  + if ,else,switch ,case ,default ,for ,do ,while ,break ,continue,return
* Keywords for modifiers:
  + public ,private ,protected,static,final ,abstract ,synchronized ,native ,strictfp(1.2 version)

transient,volatile

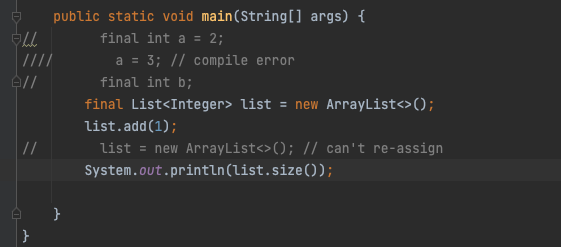
* keywords for exception handling:
  + try,catch,finally ,throw,throws,assert(1.4 version)
* Class related keywords:
  + class,package,import,extends,implements,interface
* Object related keywords:
  + new,instanceof ,super ,this,

## Final

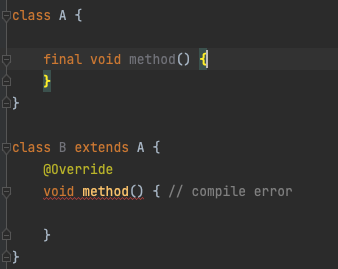
final variable

* to create constant variable
* must be initialized

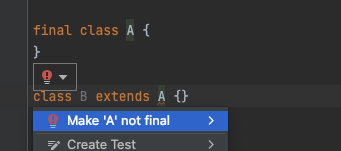




final method -> can’t be overridden



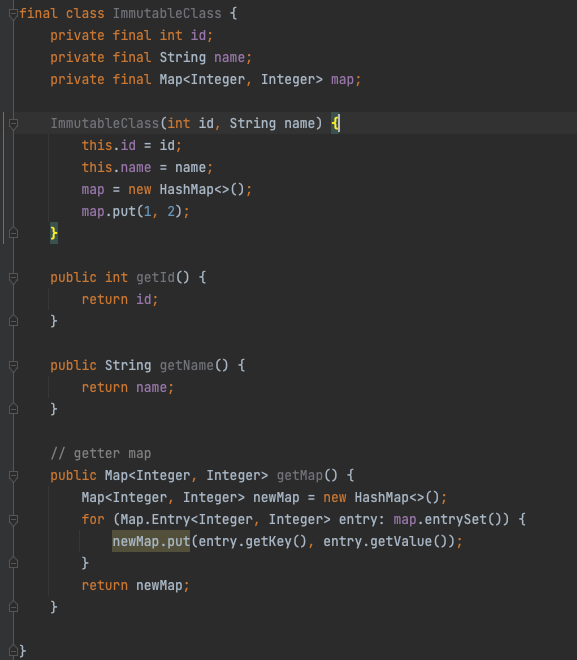
final class -> cannot be inherited



## immutable class

String Integer….

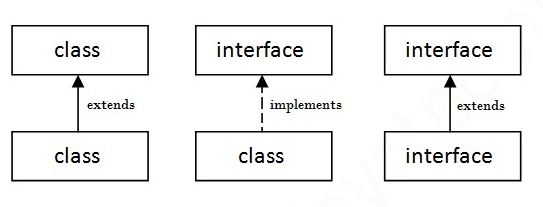
* final class
* private final field
* no setter
* return deep copy of the collections for getter



## static

* blocks
* variable
* methods
* classes

## implements vs extends



this() vs super()

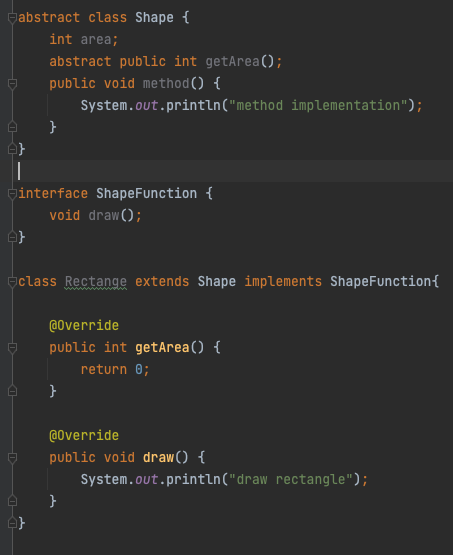


## final vs finally vs finalize()

# 13, OOP

## Abstraction

* + abstract class
  + interface



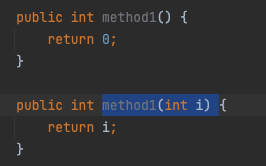
## Encapsulation

* + declare the variable as private
  + declare setter and getter

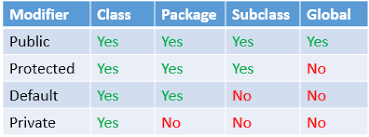
## Inheritance

* + extends,
  + implements

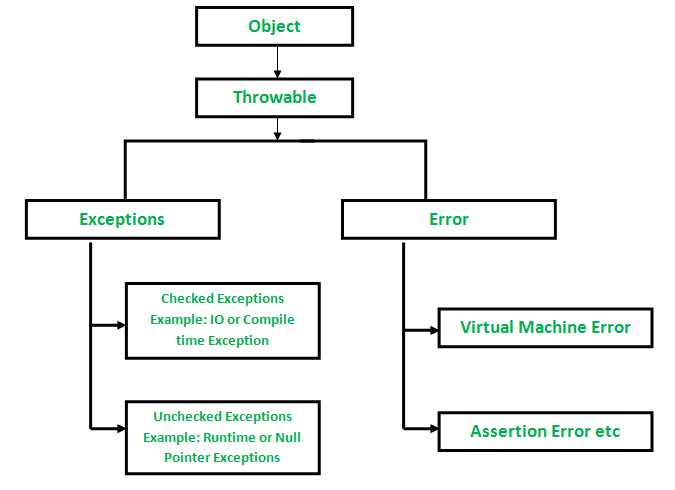
## Polymorphism

* + override
  + overloading
  + 

Access Modifier



# 14, Exception

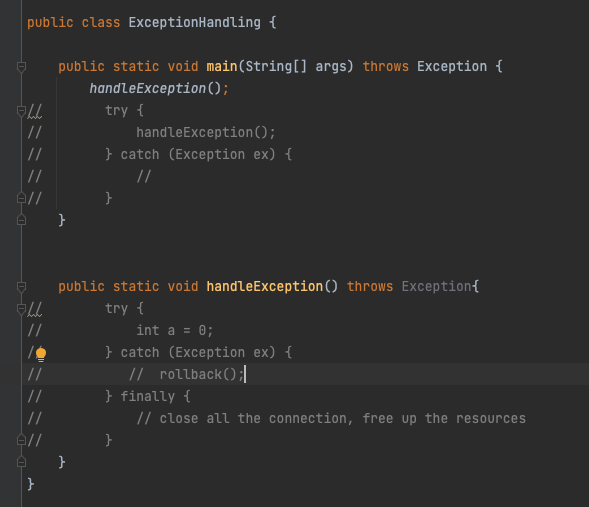


## (unchecked)runtime exception:

* ArrayStoreException, NullPointException, ArrayIndexOutOfBoundException…

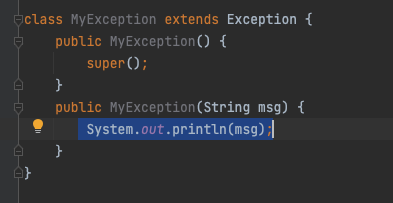
## checked Exception

* classNotFountException, IOException, SQLException….
* handle
  + try catch
  + throws



## throw vs throws

## Customized Exception



## Multiple catch

try {

// business logic

} catch (IOException ex) {

} catch (Exception ex) {

} catch () {}

catch () {}

catch () {}

catch () {}

catch () {}

try {

Connection con = DataDriver.getConnect();

// some logics

// some logics

// some logics

// some logics

} catch (IOException | SQLException ex) {

// exception handling

// rollback

} finally {

if (con != null) con.close();

}

## try with resources

try (

Connection con = DataDriver.getConnect();

Statement stm = con.createStatement();

PreparedStatement ps = con.createPrepareStatement();

) {

// some logics

// some logics

// some logics

// some logics

} catch (ex ) {

}

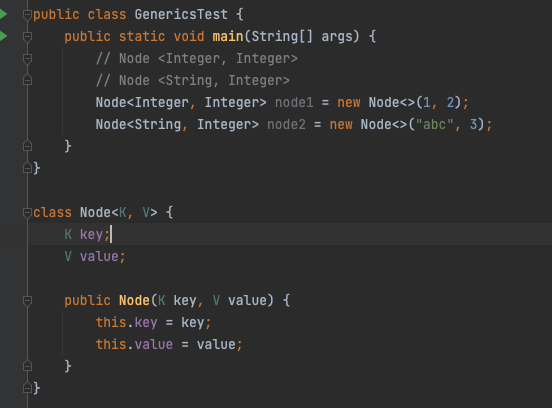
AutoClosable interface

Override close()

# 15, Generics

## generics basic

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





### Homework4.1

## type erasure

## <T extends E>

## <? extends E>

## <? super T>

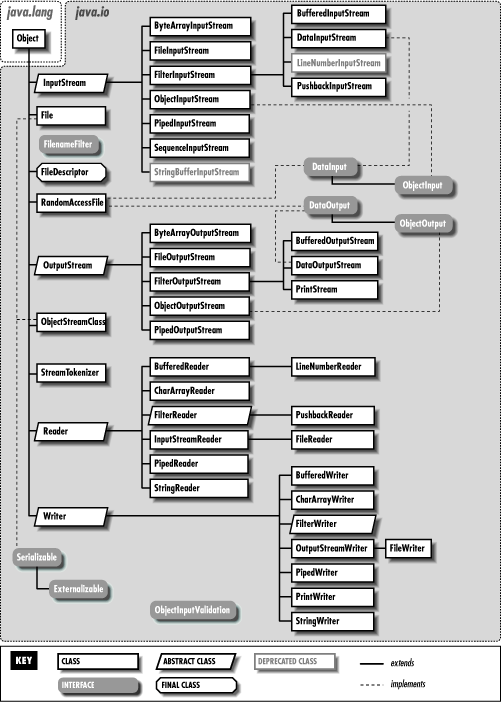
# 16, IO stream

## ByteStream

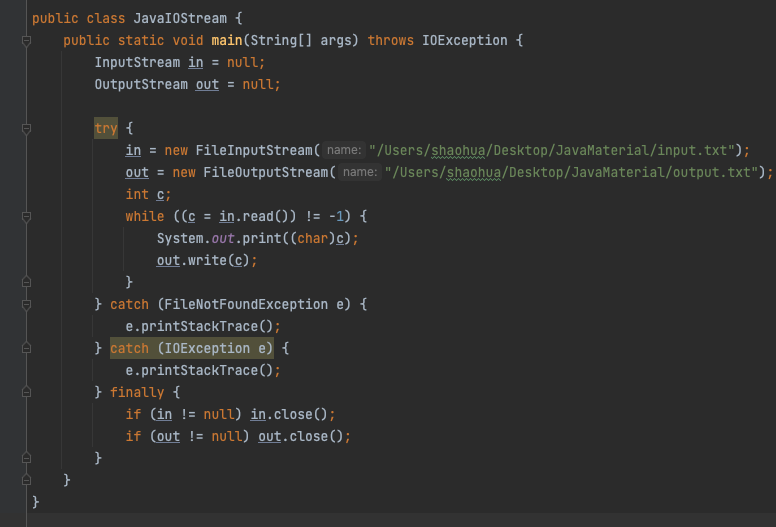
## CharaterStream

## 

* A stream is just a continuous flow of data.
* The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination
* The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java.



* Byte stream performs input and output of 8-bit bytes.
* All byte stream classes are descended from InputStream and OutputStream.
* Character stream is 2 bytes stream used for character transfer.
* All character stream classes are descended from Reader and Writer.



## File

* the file class is part of java.io
* give you access to underlying file system

| String dirName = "/Users/shaohua/Desktop/JavaMaterial"; File d = new File(dirName); String[] paths = d.list(); for (String path: paths) {  System.out.println(path); } |
| --- |

# 17, Serialization and Deserialization

Entity

| import java.io.Serializable;  public class Employee implements Serializable {  public String name;  public int age;  public transient int SSN;   public String getName() {  return name;  }   public int getAge() {  return age;  }   public int getSSN() {  return SSN;  }   public void setName(String name) {  this.name = name;  }   public void setAge(int age) {  this.age = age;  }   public void setSSN(int SSN) {  this.SSN = SSN;  } } |
| --- |

## Serialization

| import java.io.\*;  public class JavaSer{  public static void main(String[] args) {  Employee emp = new Employee();  emp.setAge(123);  emp.setName("Kaidong");  emp.setSSN(123456);   try {  OutputStream fileout = new FileOutputStream("/Users/shaohua/Desktop/JavaMaterial/employee.ser");  ObjectOutputStream out = new ObjectOutputStream(fileout);  out.writeObject(emp);  out.close();  fileout.close();  } catch (FileNotFoundException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  }  }   }  *// employee.ser* |
| --- |

## deserialization

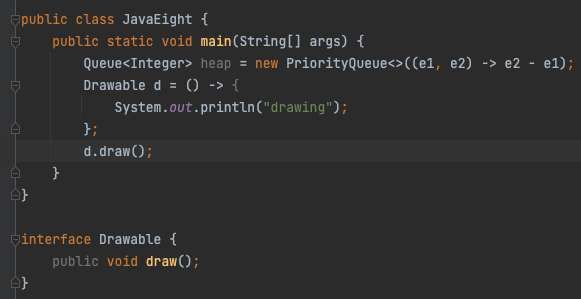
| public class JavaDes {  public static void main(String[] args) {  Employee e = null;  try {  InputStream fileIn = new FileInputStream("/Users/shaohua/Desktop/JavaMaterial/employee.ser");  ObjectInputStream in = new ObjectInputStream(fileIn);  e = (Employee) in.readObject();  in.close();  fileIn.close();  } catch (FileNotFoundException ex) {  ex.printStackTrace();  } catch (IOException ex) {  ex.printStackTrace();  } catch (ClassNotFoundException ex) {  ex.printStackTrace();  }    System.out.println(e.getAge());  System.out.println(e.getName());  System.out.println(e.getSSN());  } } |
| --- |

# 18, Java 8 features

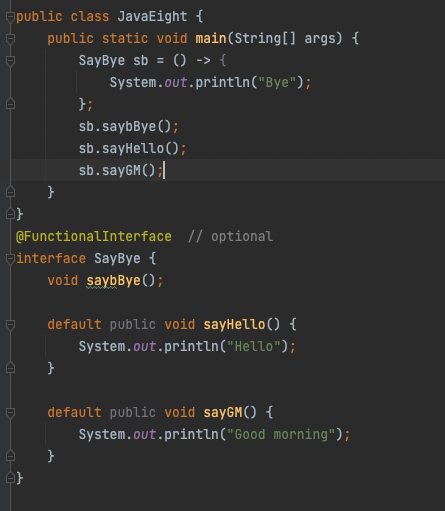
## Lambda

* functional programming
* less code

(arguments) -> {body}



## Functional Interface



Predicate

* public Boolean test(T t);

Function

* public R apply(T t);

Consumer

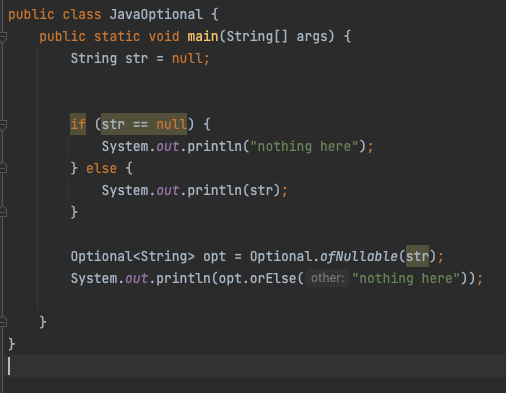
* public void accept(T t);

Supplier

* public R get();

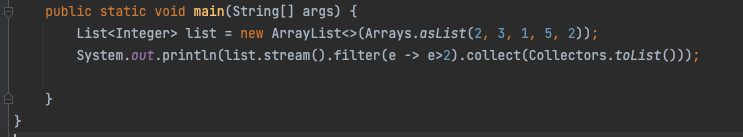
| Supplier<Double> generateRandomNumber = () -> Math.random(); System.out.println(generateRandomNumber.get()); |
| --- |

## Optional



## Stream API

* intermediate operation: return a stream as result
  + map, flatmap, filter, ….
* terminal operation: return non-stream,
  + forEach , collect…



Homework 4.2

* explore other stream API, (10 API)
  + map, flatmap, distinct, limit…..
* Method reference
* …

# 

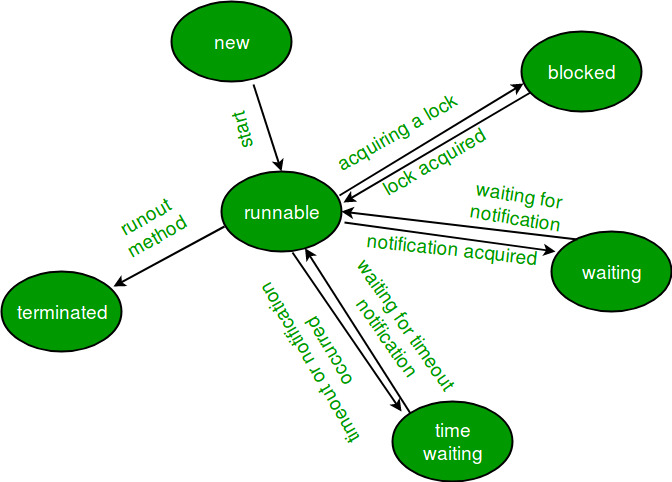
# 19, Multithread

## process vs thread

* process
  + independent memory space, stack, heap, and OS resources
* Thread
  + shared memory space
  + private stack, program counter, register

## Thread state

* new
  + thread created, not yet start
* runnable
  + executing in JVM
* blocked
  + wait for a monitor lock to entry synchronized block or method
* waiting
  + Object.wait with no timeout
  + Thread.join with no timeout
  + park();
* timed\_waiting
  + thread sleep
  + Object.wait with timeout
  + Thread.join with timeout
  + park
* terminated
  + thread has completed



## Thread creation

create method:

1. extends Thread
2. implements runnable
3. implement callable
4. thread pool

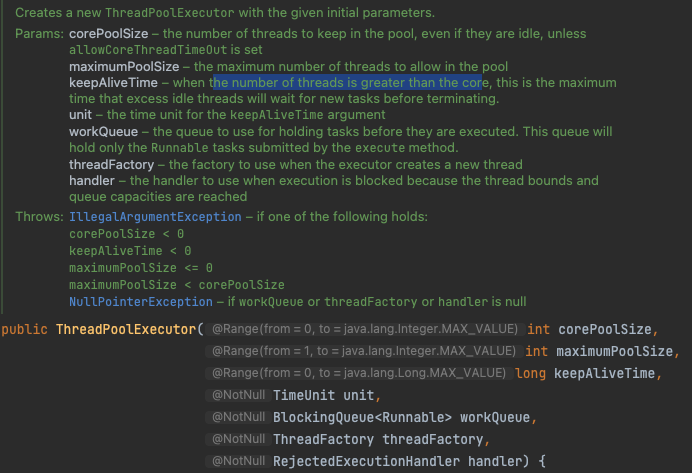
runnable vs callable

* no return / has
* no exception / has
* run() / call();

| import java.util.concurrent.Callable; import java.util.concurrent.ExecutionException; import java.util.concurrent.FutureTask;  public class ThreadCreation {  public static void main(String[] args) throws ExecutionException, InterruptedException {  Thread t1 = new FromThread();  t1.start();   Thread t2 = new Thread(new FromRunnable());  t2.start();   FutureTask ft = new FutureTask(new FromCallable());  Thread t3 = new Thread(ft);  t3.start();  System.out.println(ft.get());  } }  class FromThread extends Thread {  public void run() {  System.out.println("extends Thread class, the current thread is " + Thread.currentThread().getName());  } }  class FromRunnable implements Runnable {   @Override  public void run() {  System.out.println("Implements Runnable interface, the current thread is " + Thread.currentThread().getName());  } }  class FromCallable implements Callable {   @Override  public Integer call() throws Exception {  System.out.println("Implements Callable interface, the current thread is " + Thread.currentThread().getName());  return 200;  } } |
| --- |

## Thread pool

### customized thread pool

****

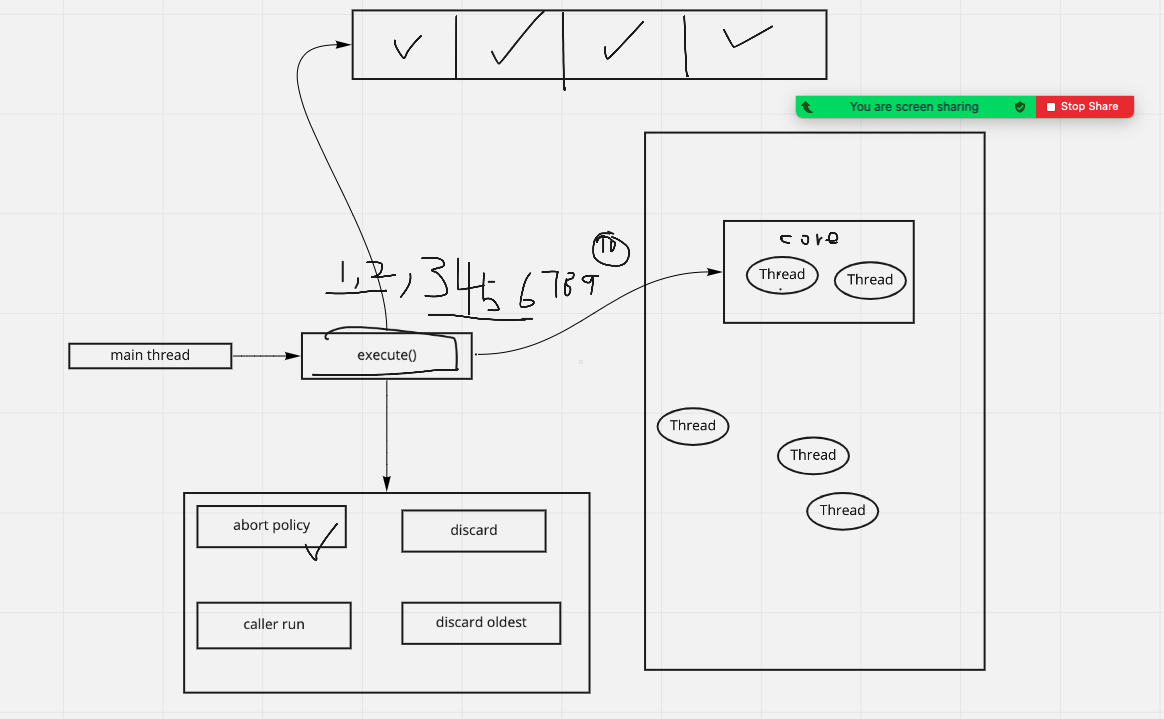
**ThreadPoolExecutor**

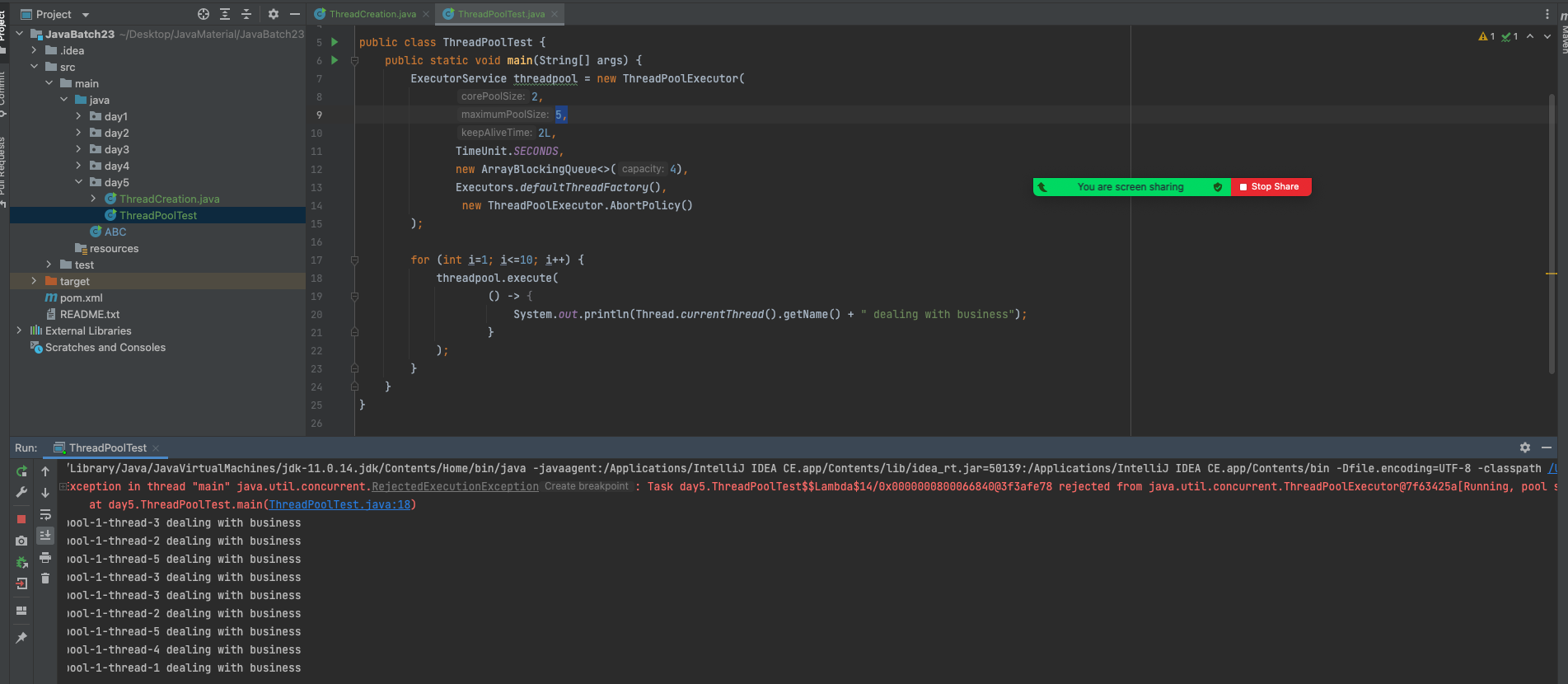
* **corePoolSize**
* **maximumPoolSize**
* **keepAliveTime**
* **unit**
* **workQueue**
* **threadFactory**
* **handler**
  + **AbortPolicy**
  + **CallerRunPolicy**
  + **DiscardPolicy**
  + **DiscardOldestPolicy**

**core: 2**

**max: 5**

**queue 4**

****

****

### predefined Thread pool

| ExecutorService threadPool1 = Executors.newFixedThreadPool(5); *// core == 5, max == 5* ExecutorService threadPool2 = Executors.newSingleThreadExecutor(); *// core 1, max 1* ExecutorService threadPool3 = Executors.newCachedThreadPool(); *// core 0, max = Integer.MAX\_VALUE;* ExecutorService threadPool4 = Executors.newScheduledThreadPool(3); *// Creates a thread pool that can schedule commands to run after a given delay, or to execute periodically* |
| --- |

## Lock

Synchronized

* block
* method
* static method
* class

class ClassName {

public void method () {

synchronized(ClassName.class) {

// todo

}

}

public synchronized void method() {

}

public synchronized static void method() {

}

public void method () {

synchronized(this) {

}

}

}

Lock Interface

* lock(), unlock(), newCondition(), tryLock(), lockInterruptibly()...
* ReentrantLock
  + the only class implements Lock interface

ReadWriteLock interface

* method
  + Lock readLock();
  + Lock writeLock();
* class
  + ReentrantReadWriteLock

Future /CompletableFuture

## 

## Homework 5.1

* understand the concepts about lock, future and completableFuture
* write demo code
* push to github repo

# 20, Enum

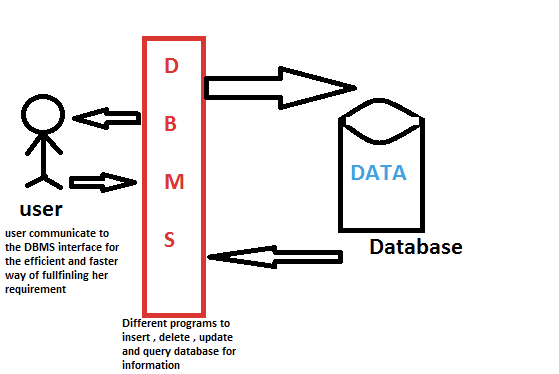
| package day5;  public class EnumTest {  public static void main(String[] args) {  *// System.out.println(EColor.BLUE.ordinal());* *// for (EColor val: EColor.values()) {* *// System.out.println(val);* *// }*  System.out.println(EColor.RED.getNum());  }  }  enum EColor {  RED(3), BLUE(2),BLACK(1);   int value;  EColor(int value) {  this.value = value;  }   public int getNum() {  return value;  } }  class Color {  public static final int Red = 3;  public static final int BLUE = 2;  public static final int BLACK = 1; } |
| --- |

# 21, Database intro

database:

database management system (DBMS):

* MySQL, Oracle, PostgreSQL, SQL server



SQL: Structured Query Language

To get the data from 21-30

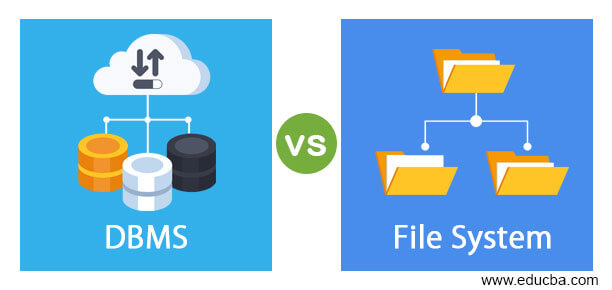
MySql:

| select \* from table order by name limit 20, 10; |
| --- |

Oracle:

| select \* from table order by name offset 20 rows fetch next 10 rows only; |
| --- |

File system



| file system | DBMS |
| --- | --- |
| manage or organize the files in storage medium | manage the database |
| redundant data | no redundant |
| no efficient query processing | efficient |
| less data consistency | consistency |
| less security | more security |
| less expensive | cost high |

# 22, Database normalization

Normalization:

* to eliminate redundant data and ensure data is stored logically

1NF

* each table cell should contain a single value
* each record needs to be unique

| id | phones | address |
| --- | --- | --- |
| 1 | 1234567, 7654321 |  |

2NF

* be in 1NF
* single column primary key

| id | firstName | lastName | phone | deparment |
| --- | --- | --- | --- | --- |
| 1 | A | B | 123 | Java |
| 2 | A | C | 123 | Java |
| 3 | H | C |  |  |

3NF

* be in 2NF
* has no transitive functional dependencies

| id | Birth date |
| --- | --- |
| 1 | 01/01/2001 |
| 2 | 01/01/1995 |
| 3 | 01/01/1990 |

| id | age |
| --- | --- |
| 1 | 21 |
| 2 | 27 |
| 3 | 32 |

de-normalization

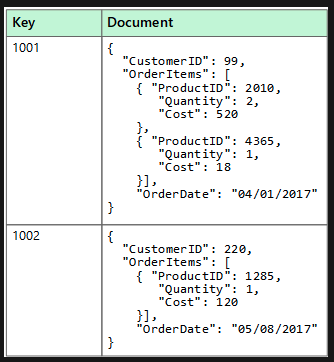
Homework 6.1

* finish all the assignments about Java on LMS
* deadline: Friday before class (1:30pm EST)

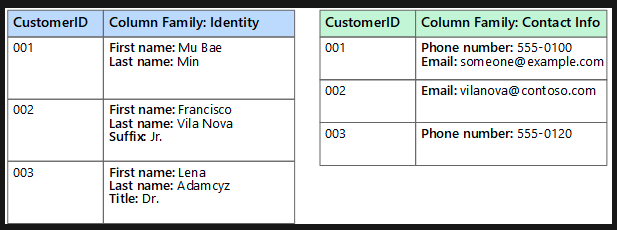
# 23, Non-relational data and NoSql

## major categories of non-relational database

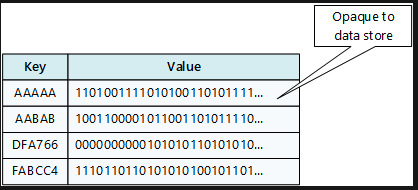
* document data store



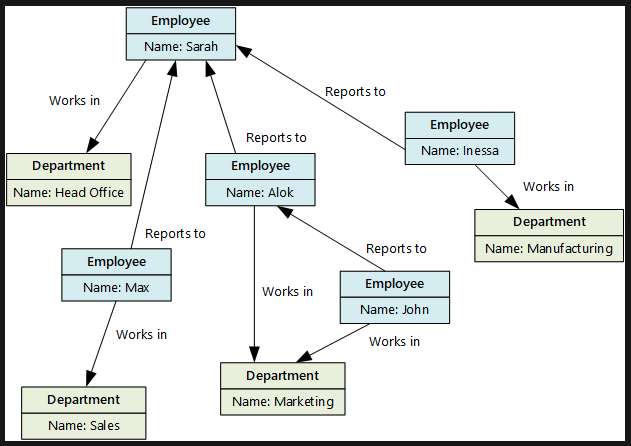
* Columnar data stores (column family)



* Key/Value data store



* Graph Data Store



## Examples for each kind of No-SQL

* key/value: **redis,** riak
* graph database: Neo4j, GraphDB
* document : **mongoDB**, couchDB
* columnar: **cassandra**, Hbase

## CAP: pick two

C: Consistency

* all clients always have the same view of the data

A: Availability

* each client can always read and write

P: Partition Tolerance

* the system works well despite physical network partitions

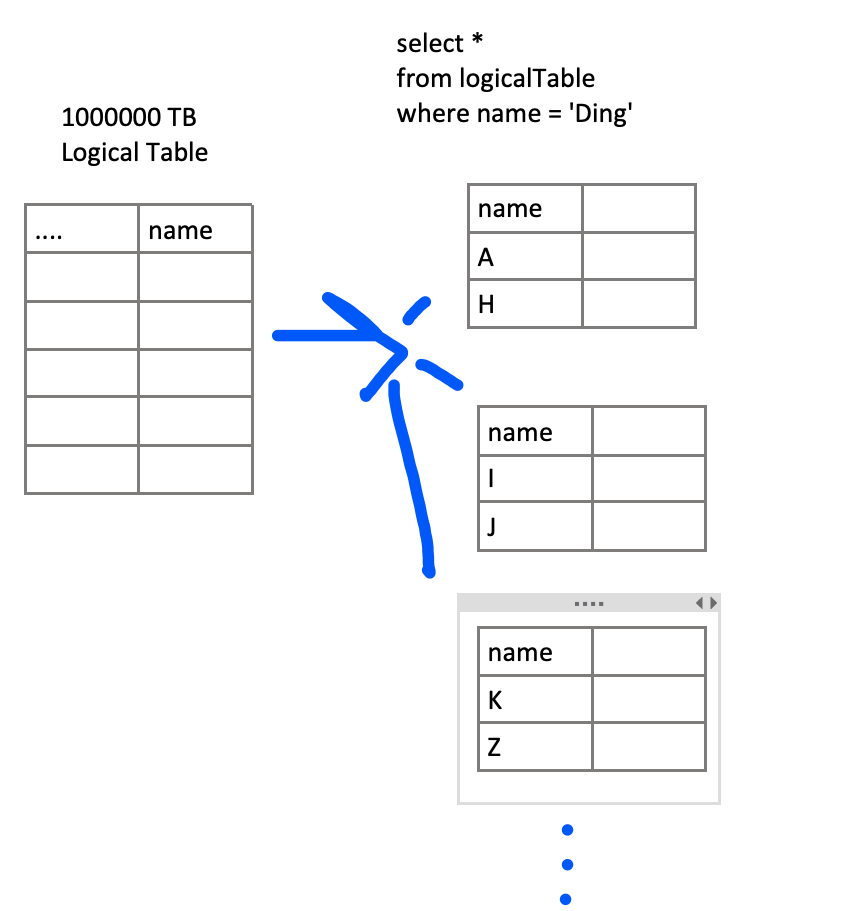
CP: MongoDB, Hbase, BigTable, Redis …

AP: Dynamo, Cassandra, SimpleDB, CouchDB

# 24. Sharding and Replica

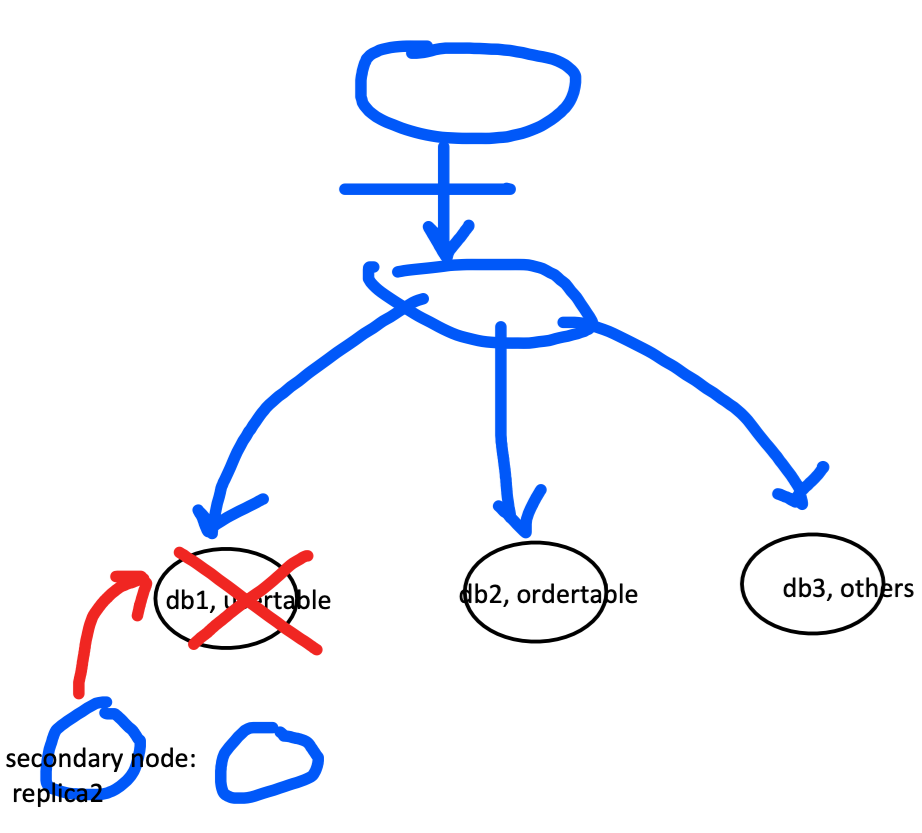
Sharding:

* distribute a single logical database across a cluster of machine



Replica

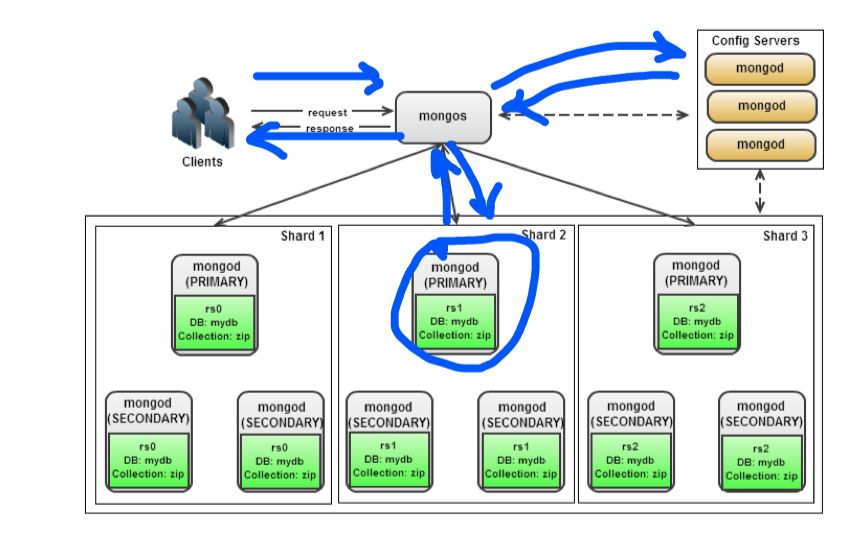
* redundancy
* failover



# 25, MongoDB

## mongodb

* no sql
* document datastore
* C++, supports APIs in many computer languages: java, python, ruby, perl….



## mongodb architure

Mongod: database instance

Mongos: sharding processes

* analogous to a database router
* process all the request based on the info from config servers
* decide how many which mongods should receive the query

Mongo: interactive shell

## Functionality of MongoDB

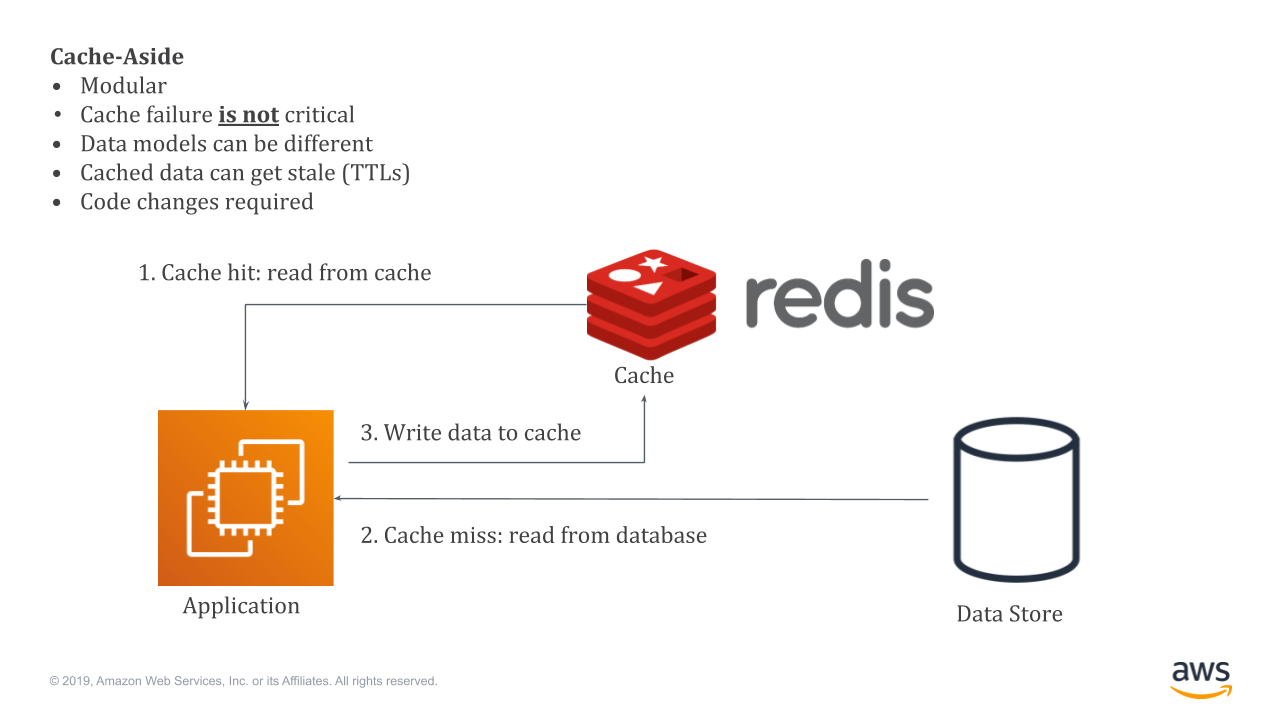
* dynamic schema
* document based database:
* secondary indexes
* primary-second node with automated failover
* built in horizontal scaling via automated ranged based partitioning of data(sharding)
* follows CP

## MongoDB Atlas

# 26, Redis

## redis (remote directory server)

* in memory
* key/value data store
* supports different kinds of data structure
  + String
  + List
  + sets
  + sorted sets
  + hashes



cache hit

cache miss

Homework 7.1

memcached vs redis

AWS: Elastic Cache

## persistence mechanisms

* RDB(redis database): the RDB persistence performs point-in-time snapshots of the database at specific intervals
* AOF (append only file): the AOF persistence logs every write operation received by the server, what will be played again at server startup, reconstructing the original dataset.

OOP Coding: LRU: least recently used cache

* LinkedHashMap
* linkedlist + hashmap

## Other usage

* cache
* distributed lock
* message queue (not recommended)
* store configuration information

# 27, sql vs no-sql

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

# 28, index

indexing is a way to optimize the performance of database by minimizing the number of the disk access required when a query is processed

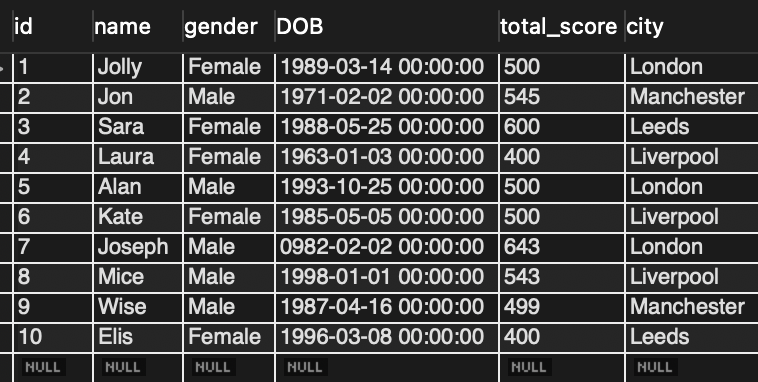
* clustered index - primary index
* non-clustered index - secondary index

## cluster index

* defines the order in which data is physically stored
* only one clustered index per table

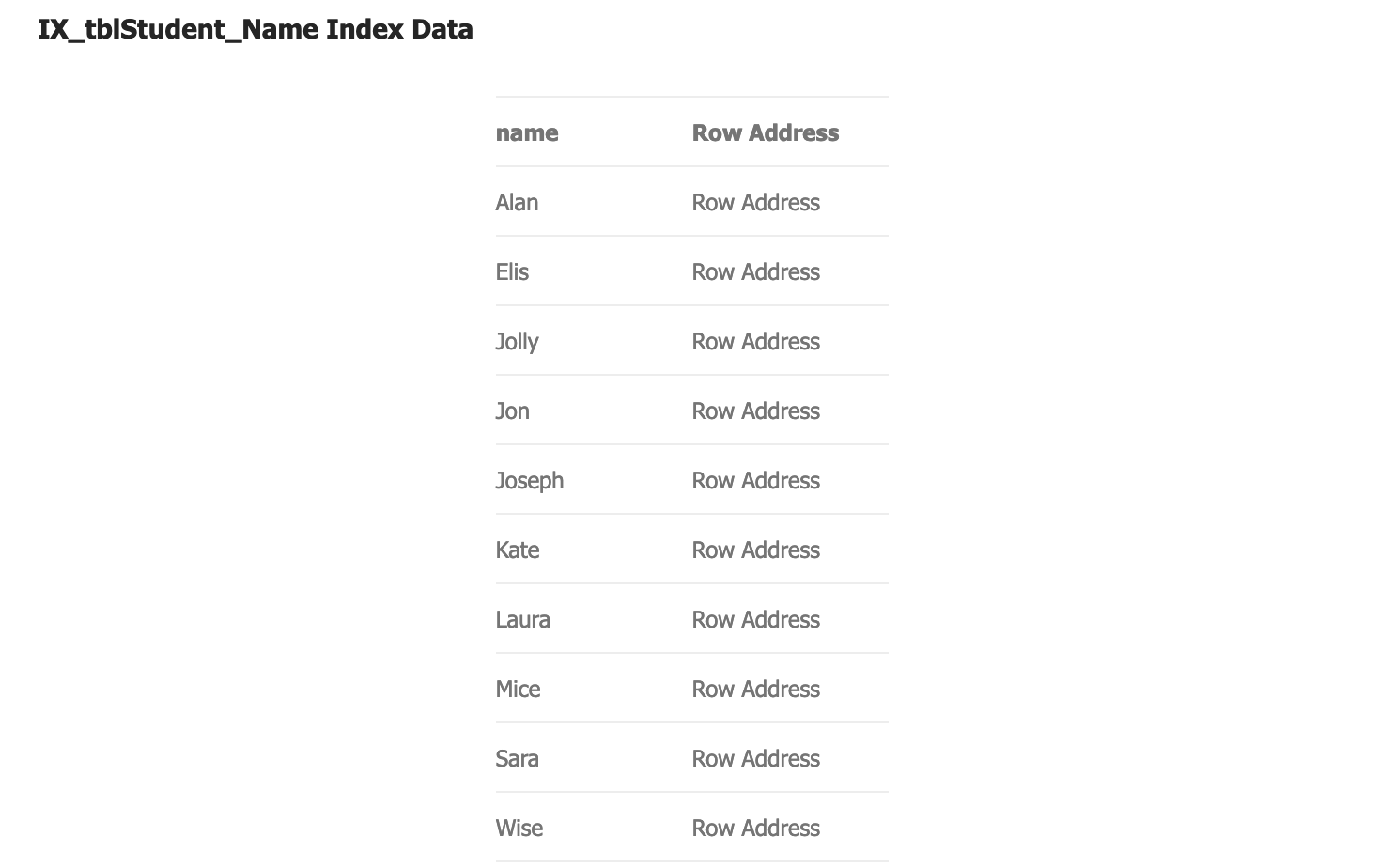
CREATE TABLE student  
(  
 id INT PRIMARY KEY,  
 name VARCHAR(50) NOT NULL,  
 gender VARCHAR(50) NOT NULL,  
 DOB datetime NOT NULL,  
 total\_score INT NOT NULL,  
 city VARCHAR(50) NOT NULL  
)

INSERT INTO student  
   
VALUES   
(6, 'Kate', 'Female', '1985-05-05', 500, 'Liverpool'),  
(2, 'Jon', 'Male', '1971-02-02', 545, 'Manchester'),  
(9, 'Wise', 'Male', '1987-04-16', 499, 'Manchester'),  
(3, 'Sara', 'Female', '1988-05-25', 600, 'Leeds'),  
(1, 'Jolly', 'Female', '1989-03-14', 500, 'London'),  
(4, 'Laura', 'Female', '1963-01-03', 400, 'Liverpool'),  
(7, 'Joseph', 'Male', '00982-02-02', 643, 'London'),   
(5, 'Alan', 'Male', '1993-10-25', 500, 'London'),  
(8, 'Mice', 'Male', '1998-01-01', 543, 'Liverpool'),  
(10, 'Elis', 'Female', '1996-03-08', 400, 'Leeds');



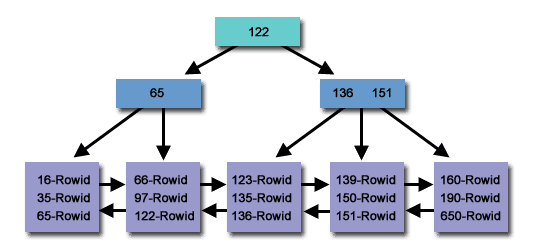
## non-cluster index

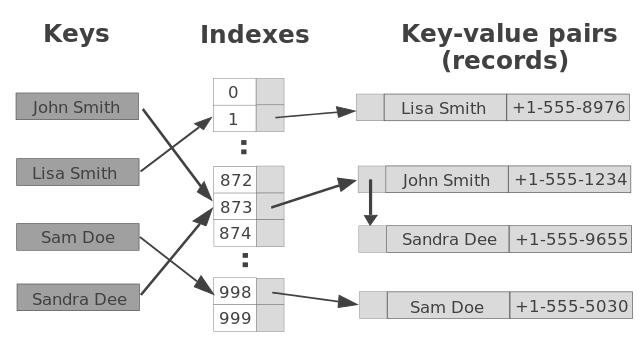
* as many as non-cluster index
* doesn’t sort the physical data inside the table



data structure

* B+ tree (default)
* bitmap
* hashtable
* r tree
* ….





select \*

from table

where id <100 and id > 50

50 -100 O(n)

n - > 2n O(n)

rwo id of 50

row id of 100

O(logn)

bit map

* gender
  + male
  + female

| CREATE BITMAP INDEX index\_name ON table\_name(columns); |
| --- |

# 29, SQL/Application tuning

SQL tuning

1. using **execution plan** to identify the cause of slowness
2. try to reduce joins, remove unused join and join conditions
3. use the index to improve the performance
4. union all instead of union
5. limit
6. view or stored procedure
7. ….

Homework 7.2

* view vs stored procedure
* view vs material view

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

A transaction is an action, or a series of actions, carried out by a single user or an application program

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

100

Transaction: account A -> accountB

read A

A = A - 100;

write A

read B

B = B +100

write B

A: shouldn’t take money from A without giving to B

C: A + B is the same, money isn’t lost or gained

I: other queries shouldn’t see A or B change until completion

D: the money doesn’t go back to A

# 31, Concurrency

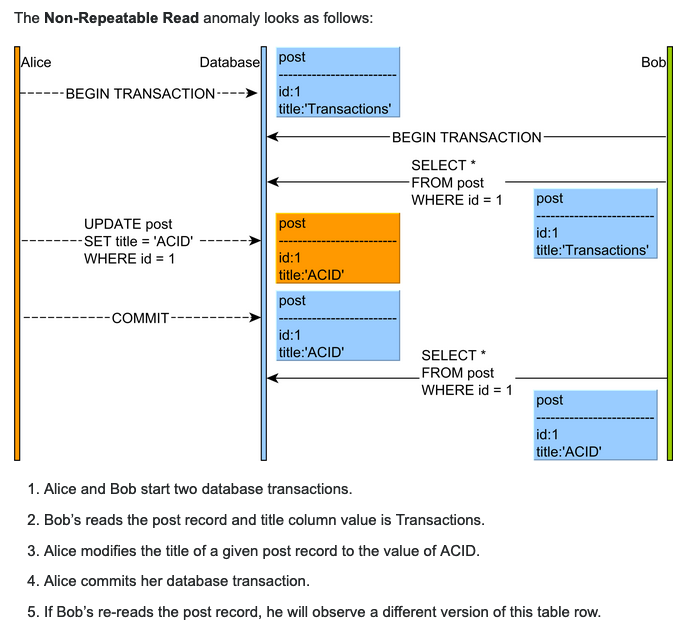
## 

## 

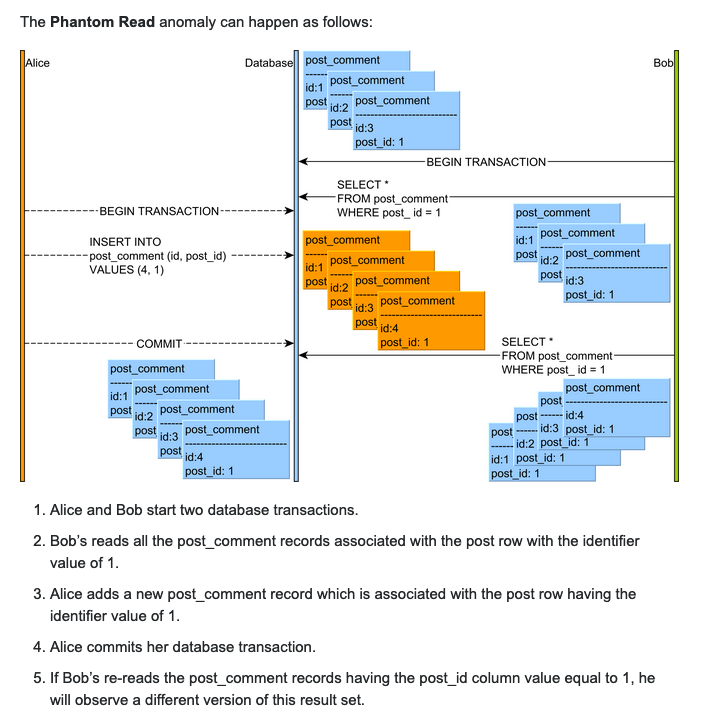
## Dirty Data

| time | T1 x=100 | T2 |
| --- | --- | --- |
| 1 | read X |  |
| 2 | X -= N N=10 |  |
| 3 | write X x=90 |  |
|  |  | read X x=90 |
|  |  | X = X + M M=20 |
|  |  | write X x=110 |
|  |  | commit |
|  |  |  |
|  |  |  |
|  | rollback |  |

## Non-repeatable read



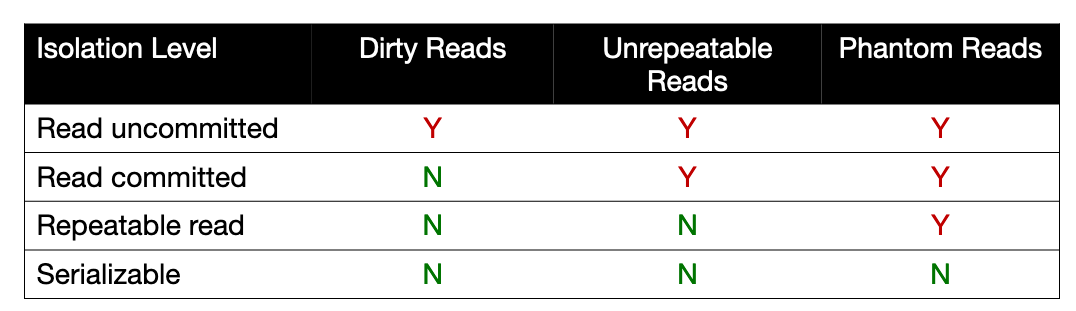
## Phantom Read



Summary:

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

## Isolation Level



# 32, Lock

## binary lock

* 1, 0
* locked or unlocked

## shared and exclusive locks

* shared lock: read lock
* exclusive lock: write lock

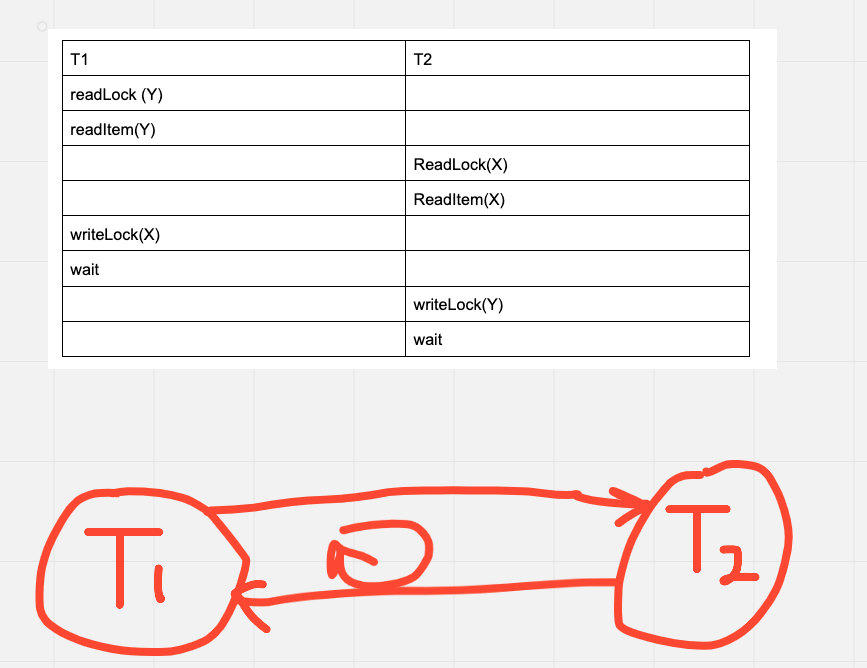
### Homework 8.1

## optimistic lock and pessimistic lock

## DeadLock

| T1 | T2 |
| --- | --- |
| readLock (Y) |  |
| readItem(Y) |  |
|  | ReadLock(X) |
|  | ReadItem(X) |
| writeLock(X) |  |
| wait |  |
|  | writeLock(Y) |
|  | wait |

How to detect the deadLock -> wait for graph



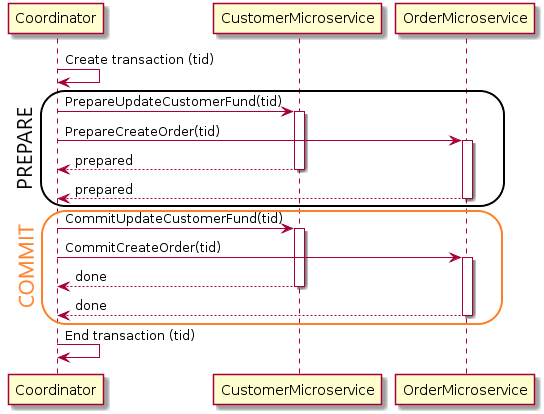
### Homework 8.2

* tracscation
* how to solve the deadlock
* live lock (optional)

# 33, Distributed Transaction

## 2PC (two phase commit)

* prepare phase
* commit phase



Homework 8.3

## Saga

SQL

Summary java and database