Before Class:

1. register LMS

We are gonna use LMS system to help with the training like assignments and materials.

Please follow the instructions to finish the enrollment before the training:

1. Open URL : http://lms.antra.com/

2. Register with your info.

3. Enroll courses ( core java and advanced java).

Enroll with the Key antrajava20

Invitation Code: IIRSKP

Let me know if there are any questions.

1. update name in zoom group

please change your Zoom group member name as your real name which is the same as the one on your passport. Anyone who didn’t do that will be removed from the group. @all

Format: **FirstNameOnPassport\_LastNameOnPassport\_Java\_2022\_5\_6**

Deadline: 23:59PM EST 02/22/2022

# GitHub

<https://github.com/Arthur-Shuahua-Zhang/Java5_6Batch>

# 0, Preparation

Maven

Git

Java 11 / 8 jdk

Intellij

# 1, Maven

Types of Repositories

* local
* central
* remote

local repository

* ~/.m2

remote

life cycle

* validate
* compile
* test
* package
  + jar/ war
* verify
* install
* deploy

mvn clean

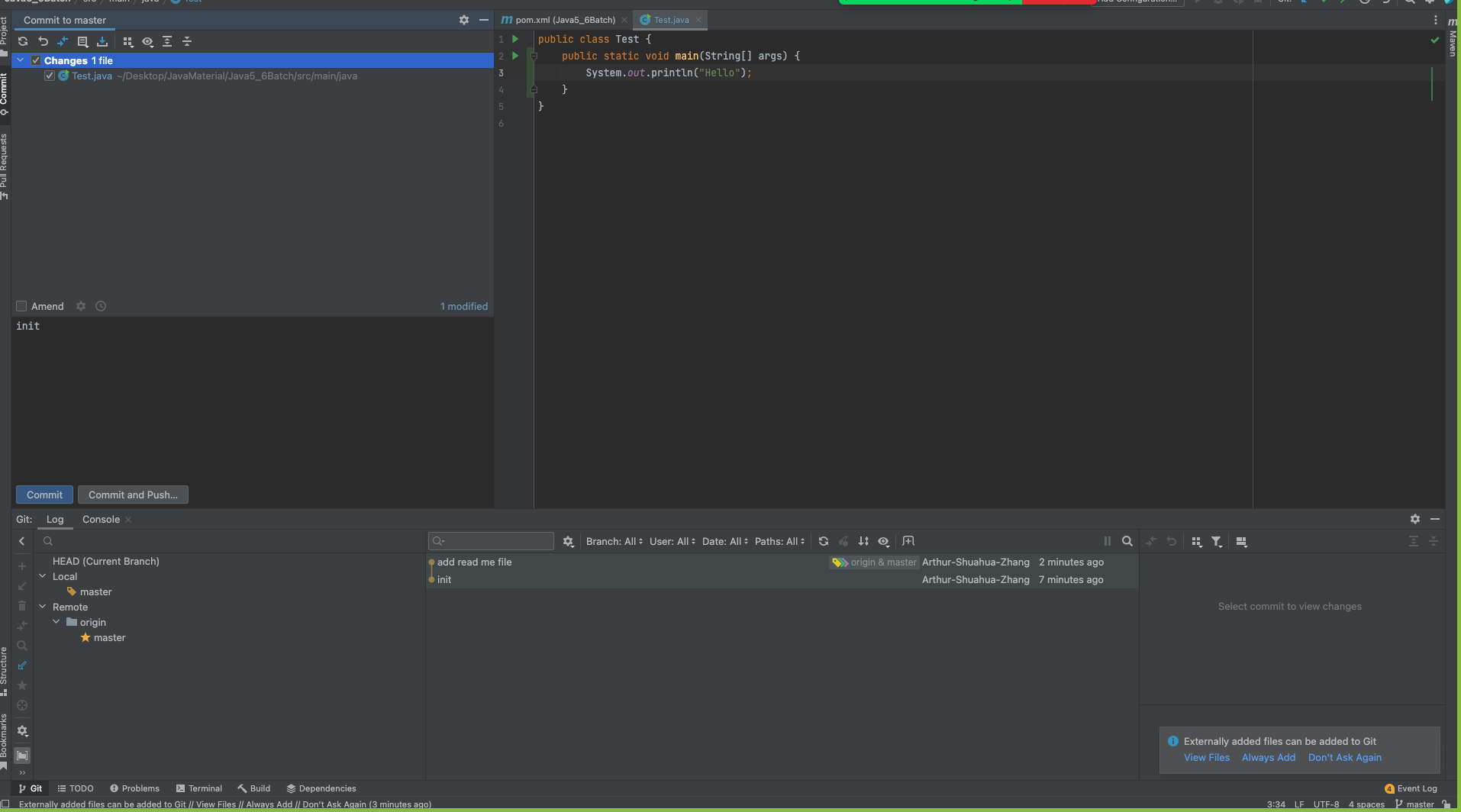
mvn test

mvn install

# 2, Git

git vs svn

intellji



terminal

271 vi ReadMe.txt

272 ls

273 git brach

274 git branch

275 git add .

276 git commit -m "add read me file"

277 git push

### Homework 1

1. download and install
   1. Maven
   2. Git
   3. Java 11 / 8 jdk
   4. Intellij
2. create a maven project and push to the github
3. send the github link to the zoom group

by the end of today

# 3, Eight Basic Data Types

primitive type

* byte, short, int, long, float, double, char, boolean

wrapper class

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

autoboxing and unboxing

# 4, String/StringBuilder/StringBuffer

String

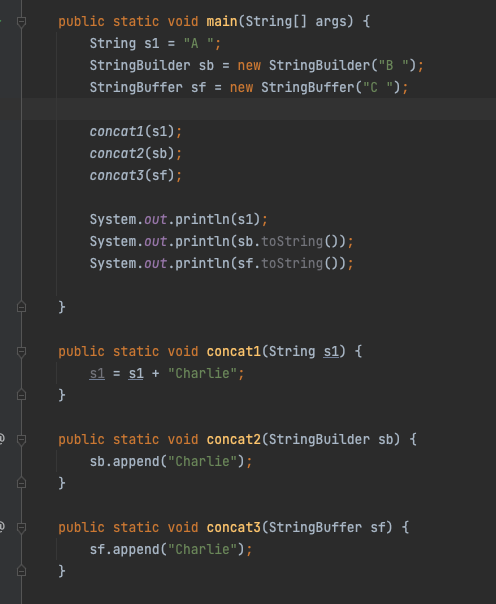
* immutable
* thread safe

StringBuilder

* mutable
* not thread safe

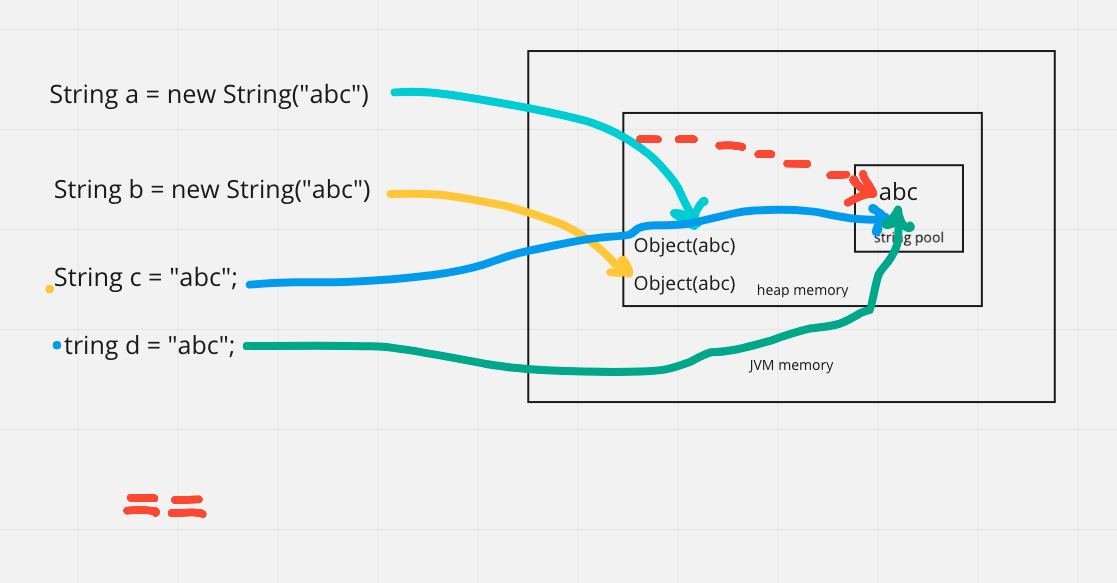
StringBuffer

* mutable
* thread safe

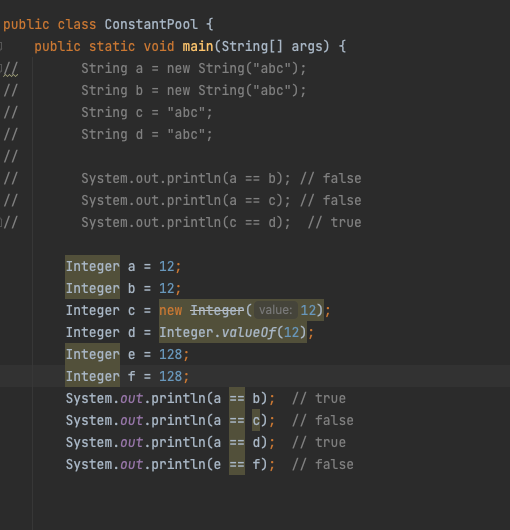


# 5, String/String constant pool

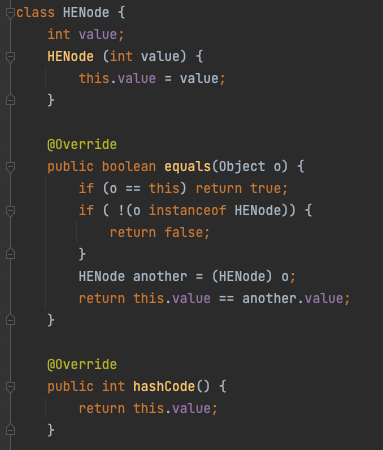
== vs equals()



-128 -> 127



# 6, equals / hashcode



# 7, Collection

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

List

* ArrayList

address 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| data0 | data1 | data2 | data4 | data5 |  |

O(1)

* LinkedList

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| node1  data  next = node2 |  |  |  | node3  data  next node4 |
|  |  |  |  |  |
|  |  | node2  data  next = node3 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

O(n)

vector:

* thread safe

Stack

* thread safe,
* FILO
* push, pop

Deque: ArrayDeque

* first [ ] last
* replace Stack: deque.offerFirst(), deque.pollFirst();

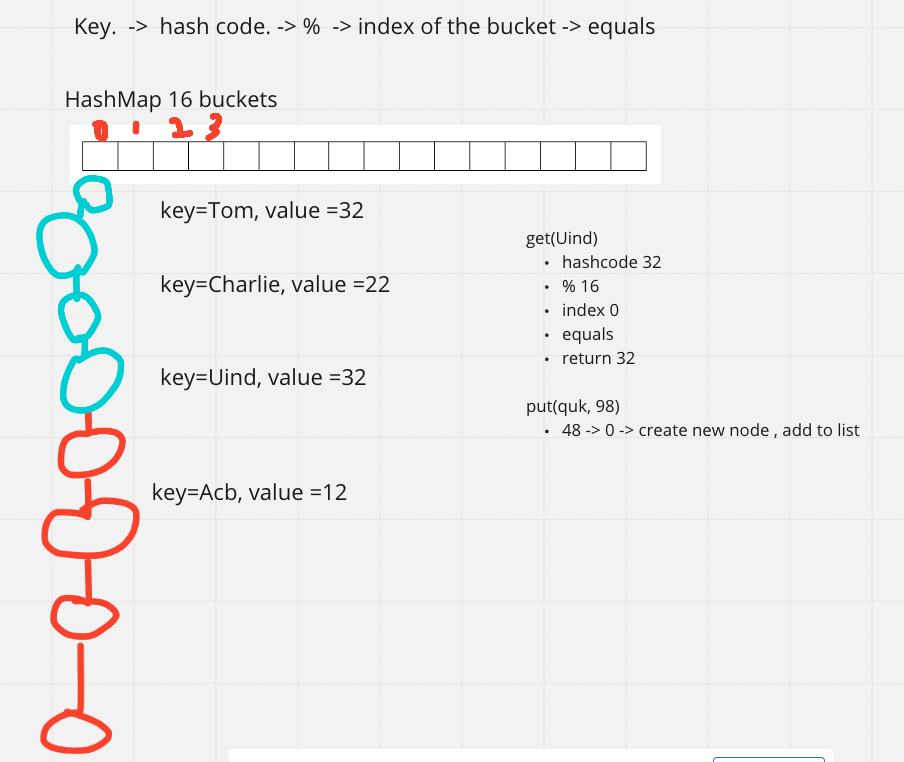
Set

* HashSet
  + unique
  + don't’ keep insertion order
* TreeSet
  + unique
  + sorted
* LinkedHashSet
  + unique
  + keep insertion order

Map

* HashMap
* LinkedHashMap
* TreeMap
* HashTable
* ConcurrentHashMap

HashMap



Queue

* FIFO

Heap

* PriorityQueue
* minHeap
* maxHeap

array

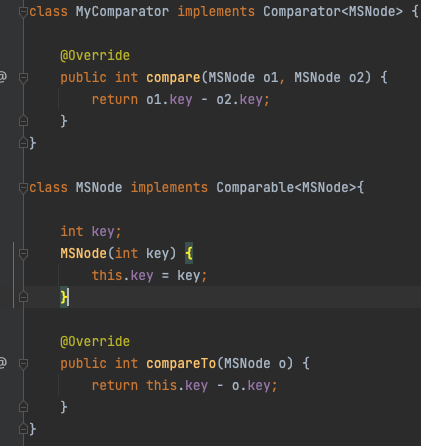
* int[] String[] Object[]
* int[][], char[][]

list vs set

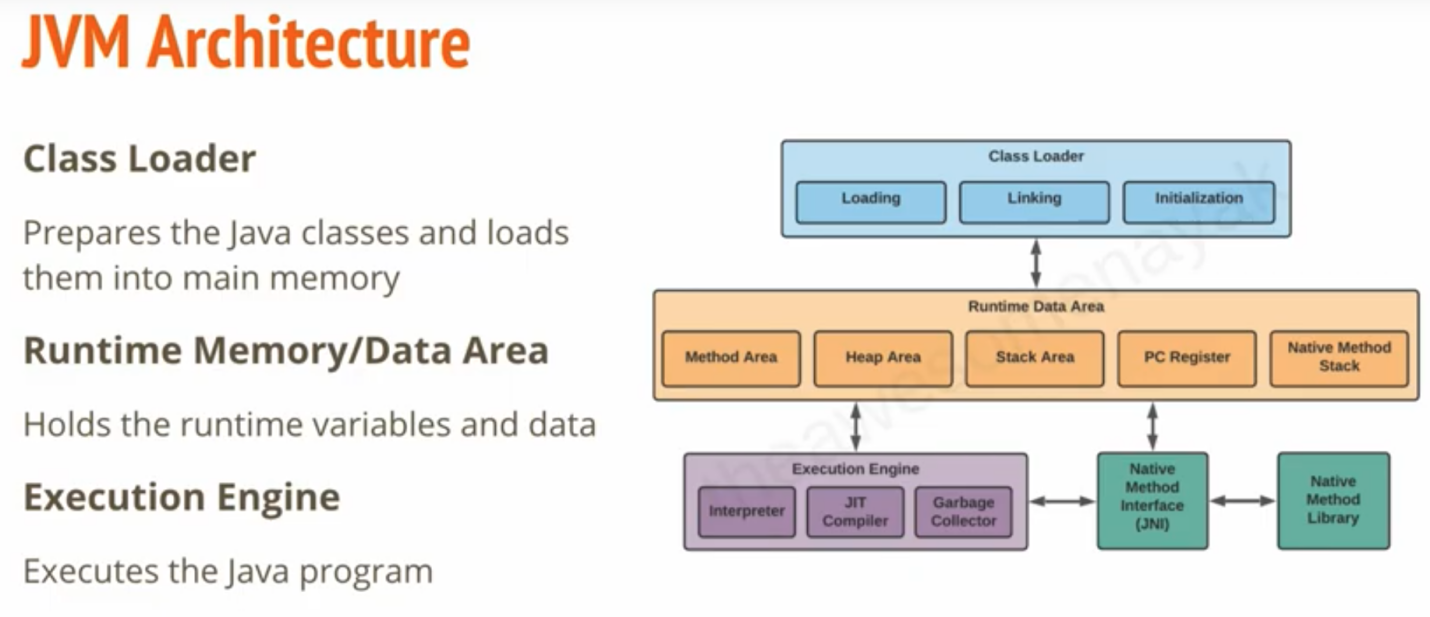
HashMap vs HashTable vs ConcurrentHashMap

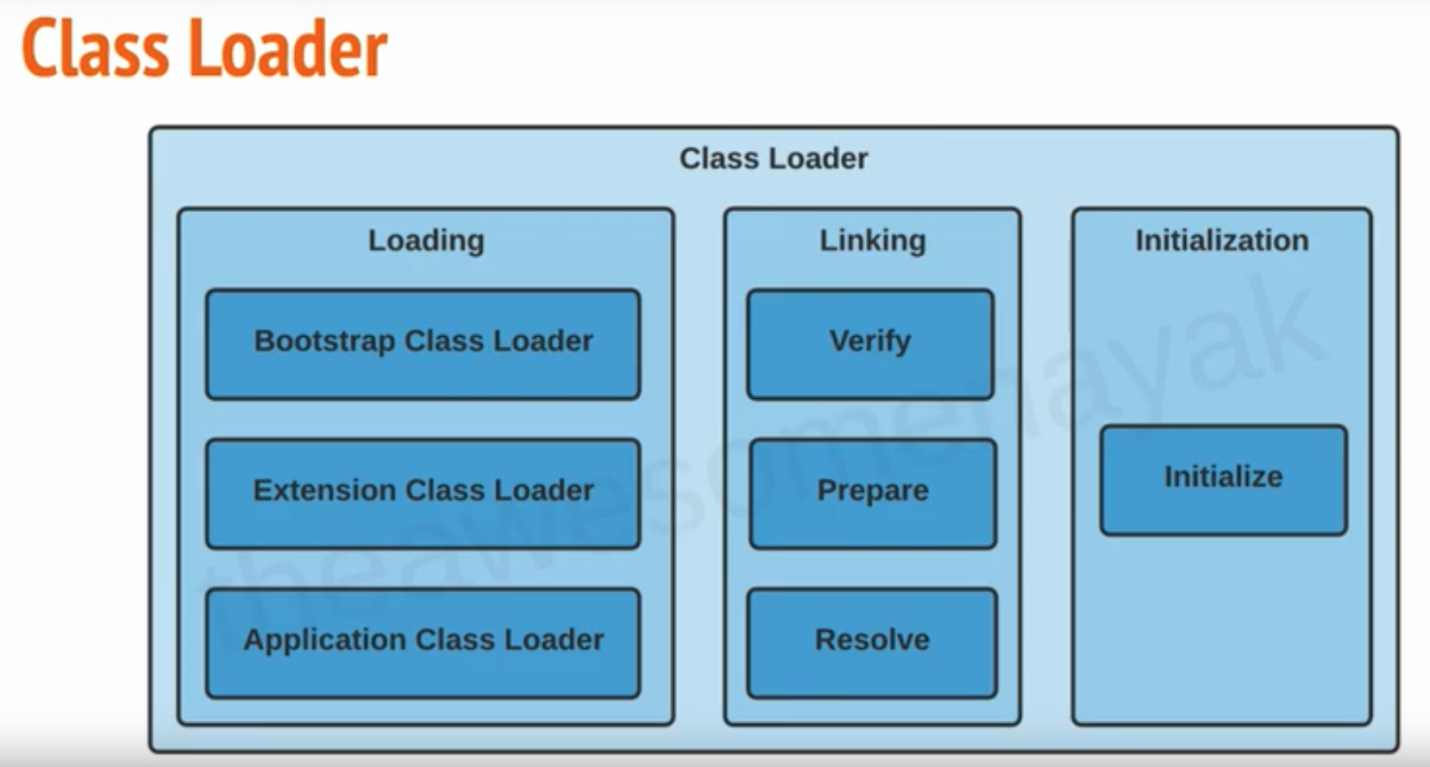
HashSet <- HashMap

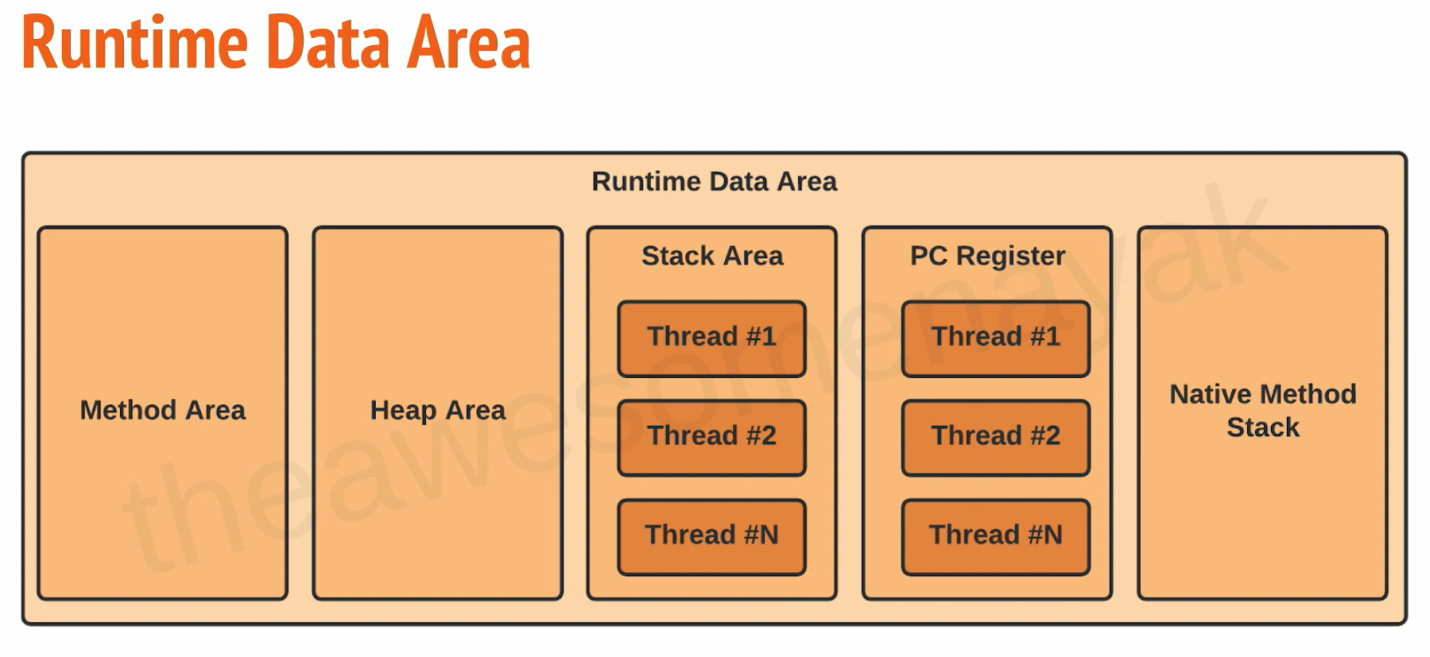
# 8, Comparator vs Comparable

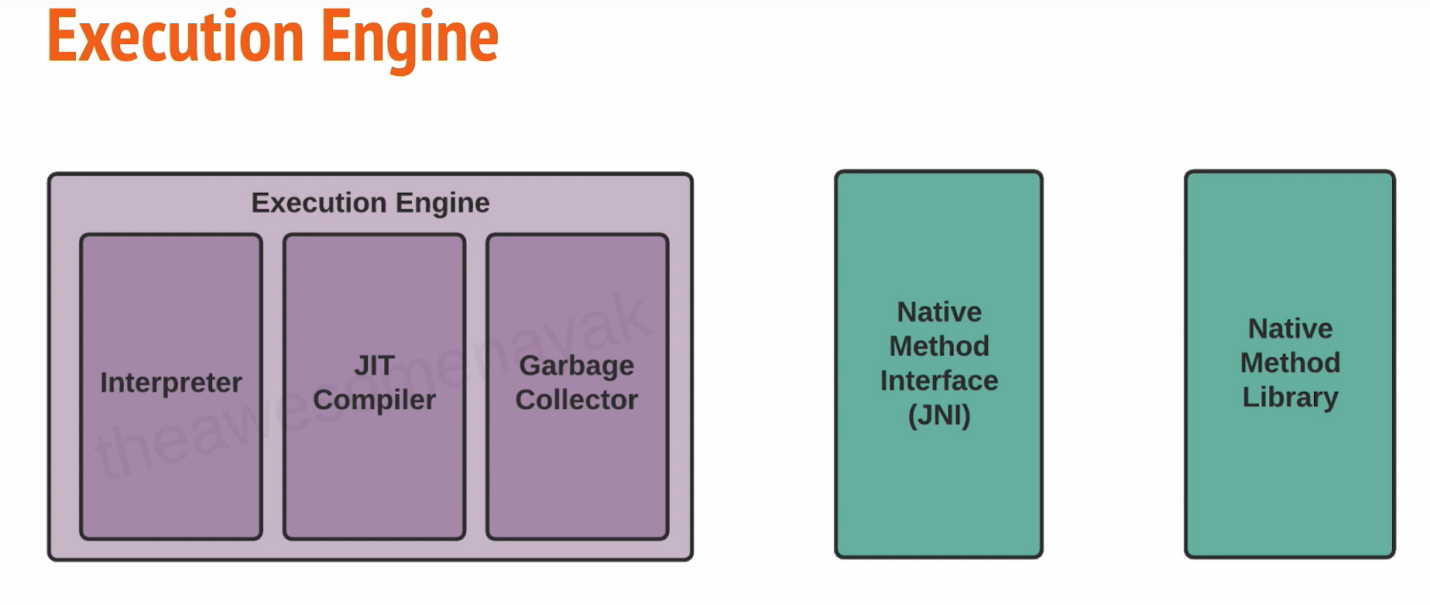


# 9, JVM

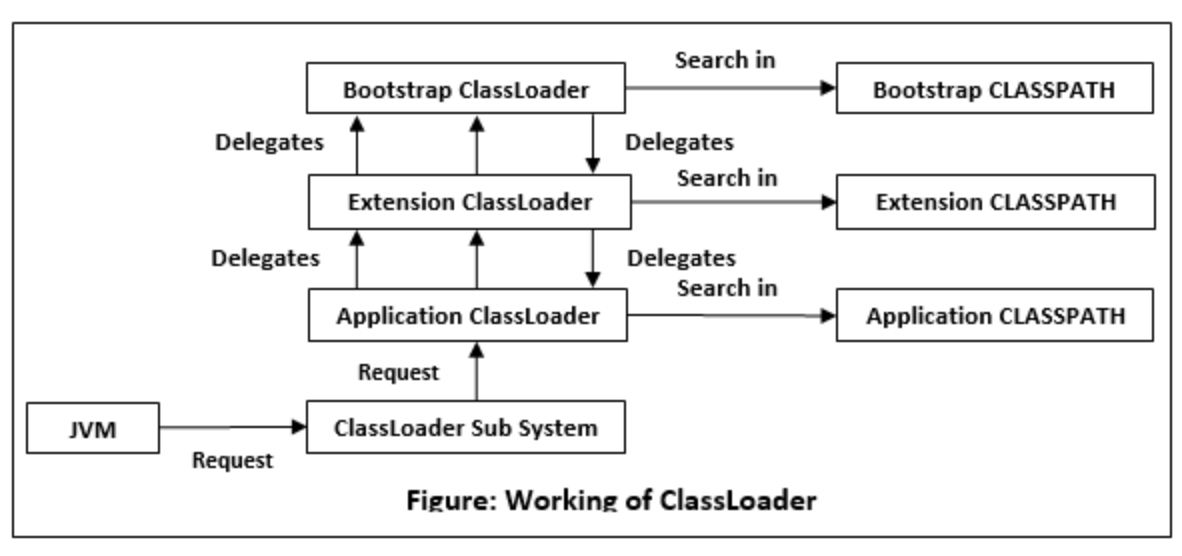








# 10, class Loader



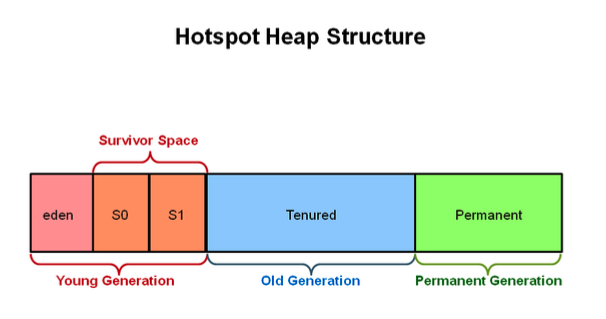
# 11, Garbage Collector

* serial GC
* parallel GC
* G1 GC

|  |  |  |  |
| --- | --- | --- | --- |
| chunk1  rank2 | chunk2  rank1 | chunk3  rank3 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

CMS GC(Concurrent Mark Sweep) G1

* deprecated since java 9
* completely removed in java 14



minor GC

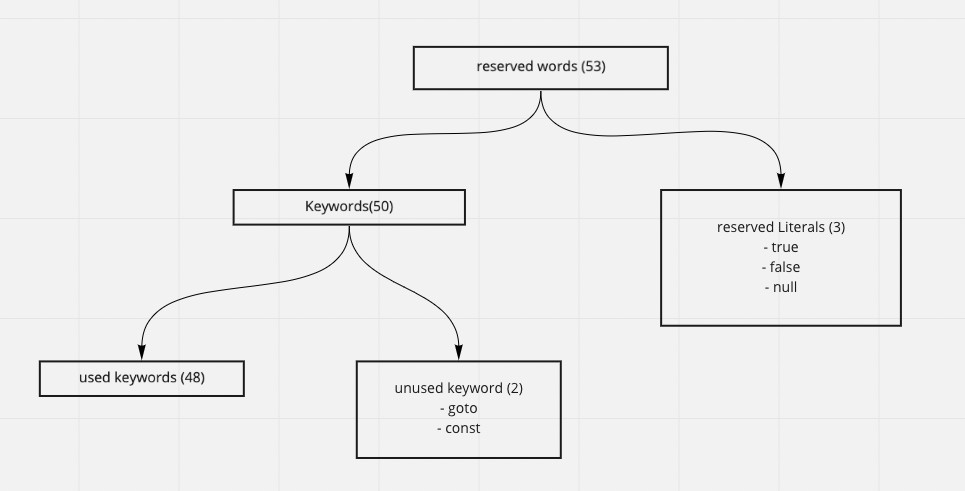
major GC

Homework

* do the assignment 1, 2 on LMS
* push it to the github
* pdf, doc,md

# 12, Keywords

## keywords overview



## Homework 3

* check all the keywords below
* push to github
* pdf, doc, md, .java….

for 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

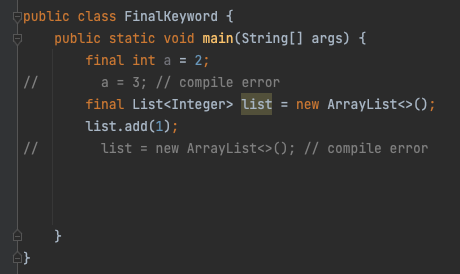
Object related keywords,

* new, instanceof, super, this

## Final

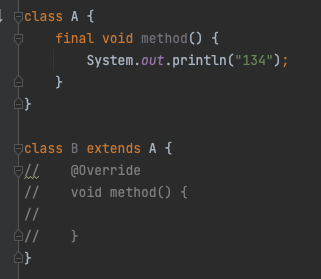
final variable

* create constant variable
* must be initialized



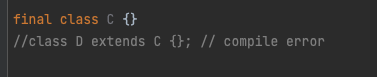
final method

* can’t be overridden



final class

* can’t be extended



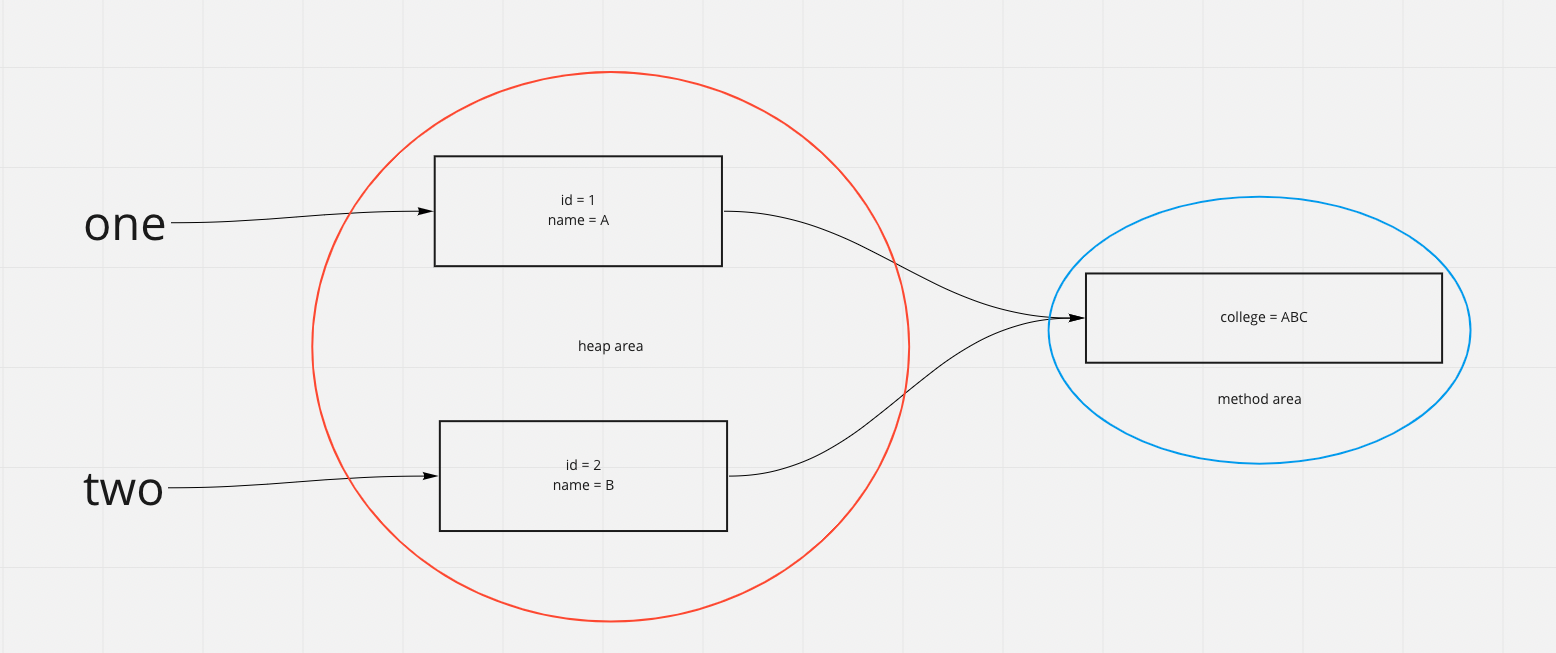
## immutable class

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

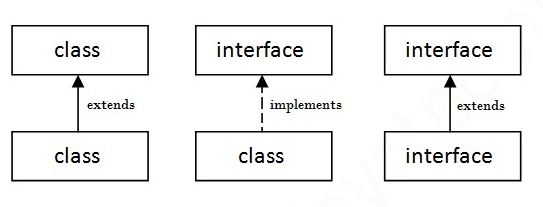


## static

* block
* variable
* methods
* classes



implements vs extends



# 13, OOP

## Abstraction

* Abstract class
* interface

## Encapsulation

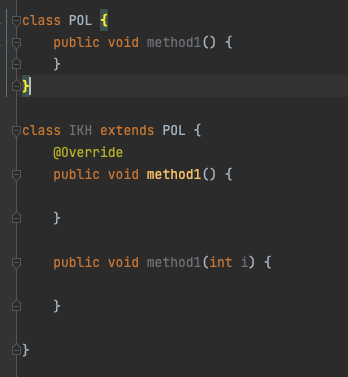
* declare all variables be private
* declare setter and getter

## Inheritance

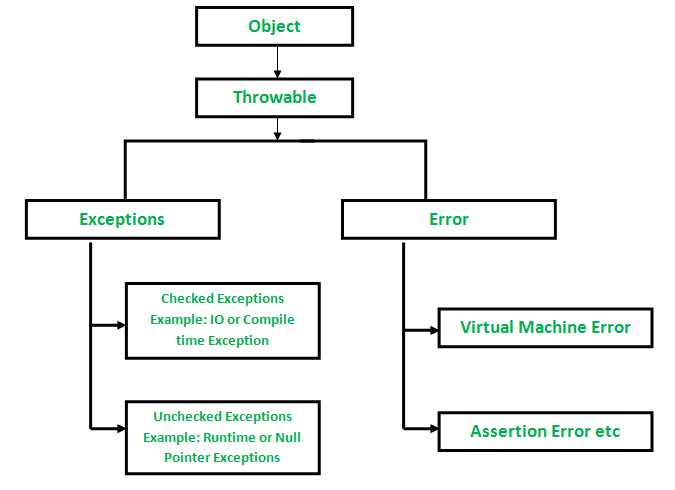
* extends
* implements

## Polymorphism

* override
* overload



# 14, Exception

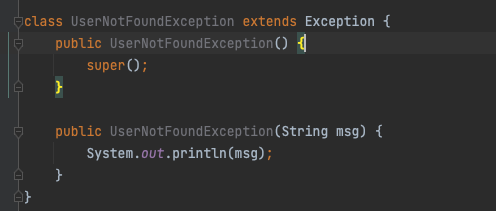


## checked exception vs unchecked exception

how to handle the exception

* try catch
* throws

## how to customize exception



## how to handle multiple exception

try {

// business log

} catch (IOException ioe) {

} catch (SQLException sqle) {

} catch () {}

catch () {}

catch () {}

—------------------------------------------

try {

} catch (IOException | SQLException | …. ex) {

}

try {

Connection con = DataDreiver.getConnection();

} catch(IOException ioe) {

} catch (Exception ex) {

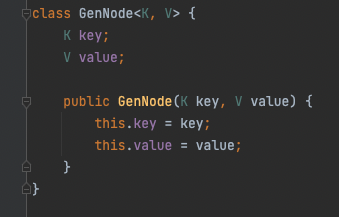
} finally {

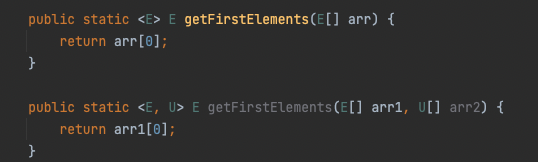
if (con != null) con.close();

}

# 15, Generics

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





<? extends E>

<? super T>

<T extends E>

# 16, IO stream

Stream

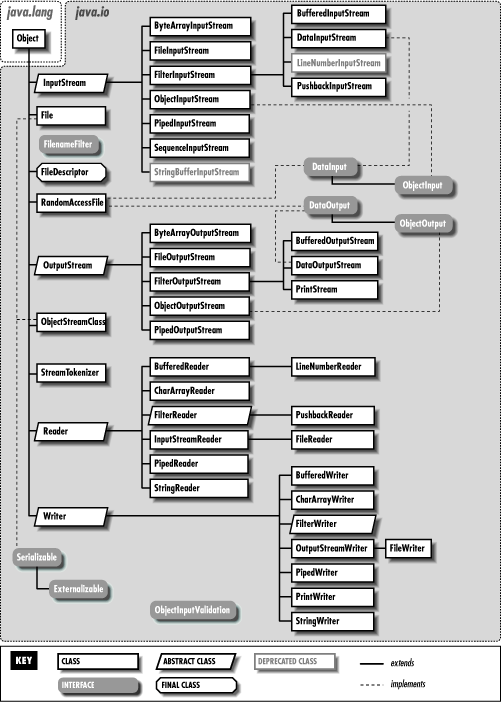
* a continuous flow of data

Byte Stream

* InputSteam, OutputStream
* 1 byte = 8 bits

CharaterStream

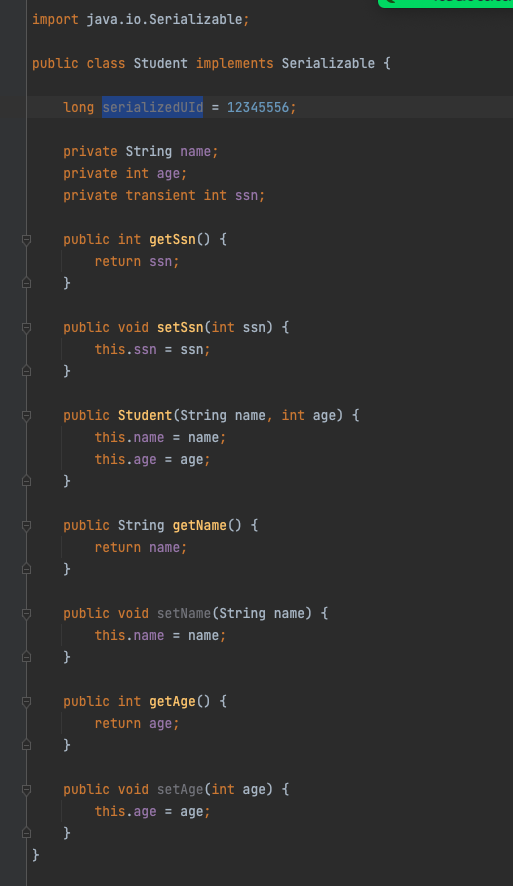
* Reader, Writer
* 2 byte = 16 bits



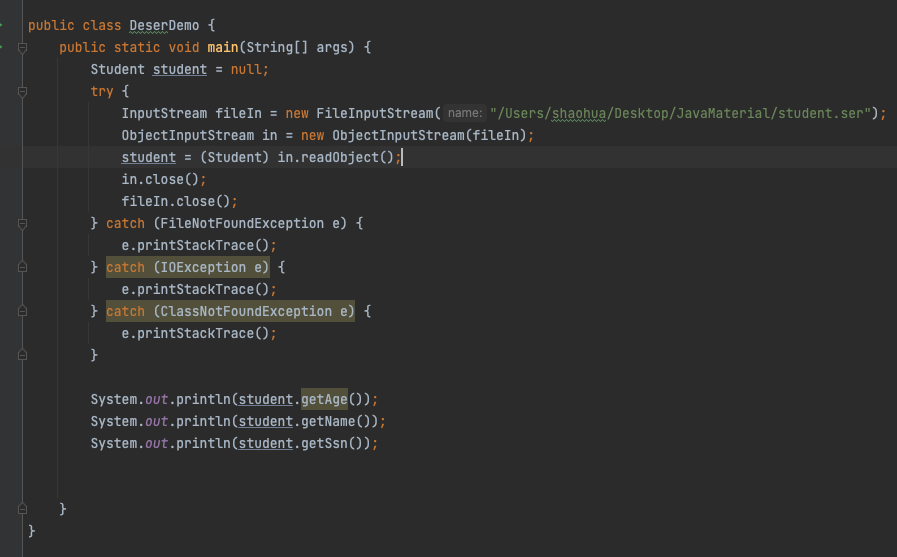
File

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

# 17, Serialization and deserialization





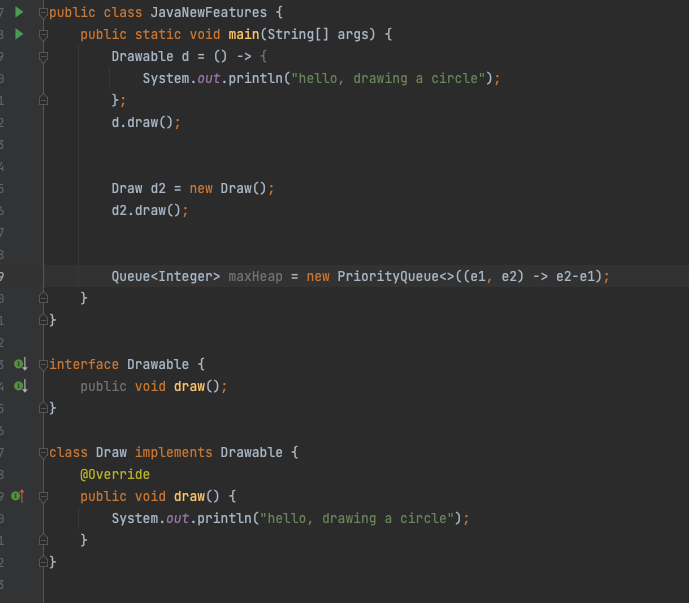


# 18, Java 8 features

## lambda

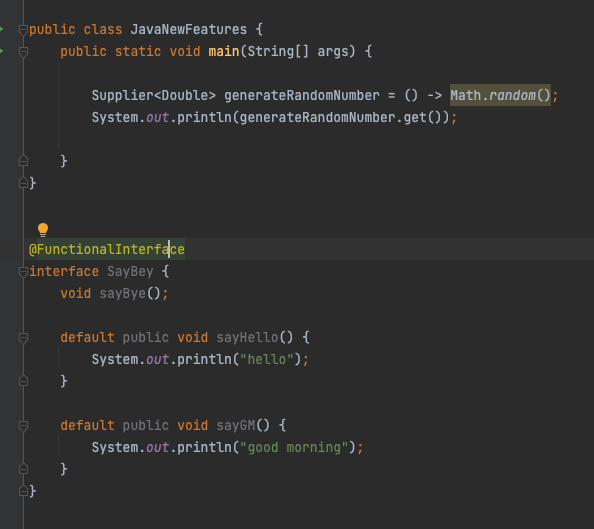
* functional programming
* less code

（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()



## Optional

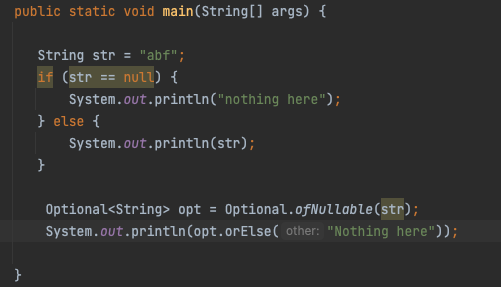
if (obj == null) {

…

} else {

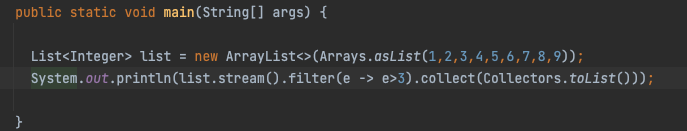
…

}



## Stream API

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



# 

Homework:

* finish assignment 3, 4 on LMS

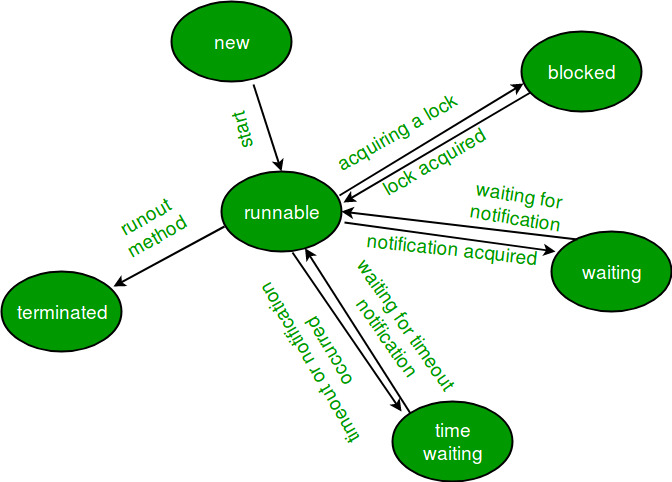
# 19, Mutli-threading

## thread vs process

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

## Thread states

* new
  + thread create, not yet start
* runnable
  + executing 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()
* timed\_waiting
  + thread sleep
  + Object.wait() with timeout
  + thread.join with timeout
  + park
* terminated
  + thread has completed



## thread creation

* extends Thread
* implements Runnable
* implements Callable
* thread pool

runnable vs callable

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



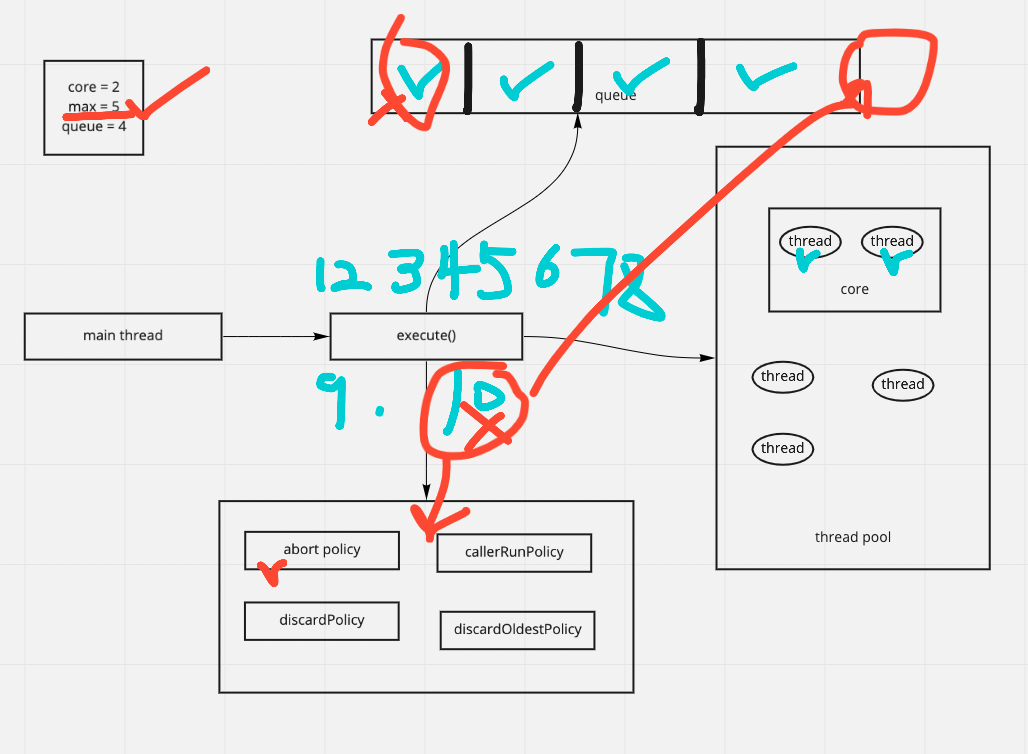
# Thread pool

## customized thread pool

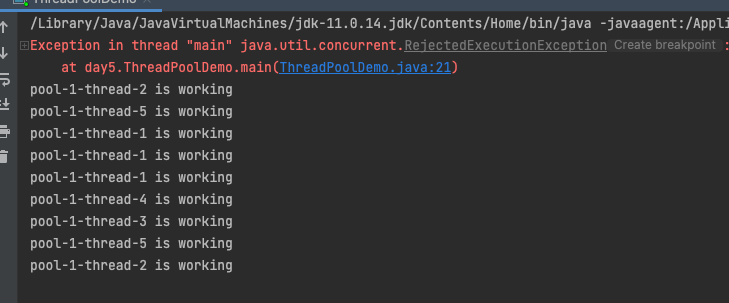


ThreadPoolExecutor

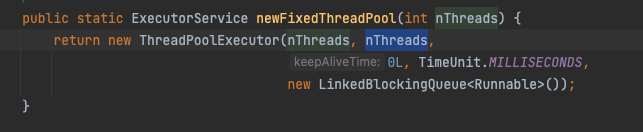
* corePoolSize
* maximumPoolSize
* KeepAliveTime
* Time unit
* work queue
* thread factory
* handler
  + abortPolicy
  + callerRunPolicy
  + discardPolicy
  + discardOldestPolicy

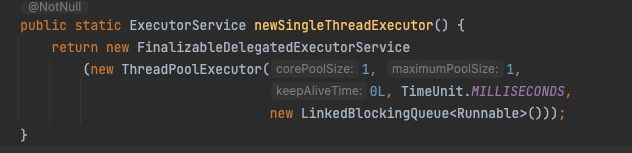




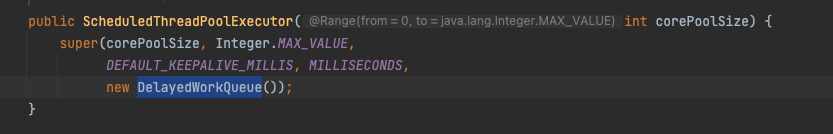


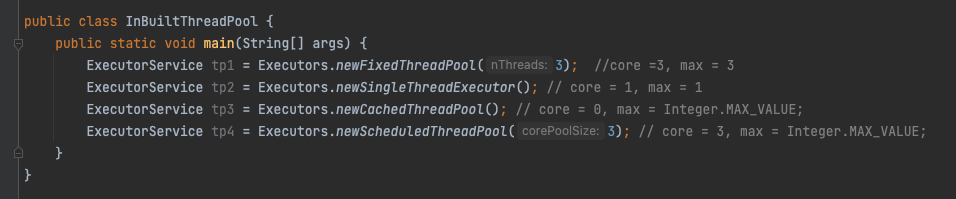
in-built thread pool











OutOfMemoryError

Lock

* synchronized
* Lock interface

synchronized

* block
* method
* static method
* class

class Demo {

public void method() {

synchronized(Demo.class) {

}

}

public synchronized void method() {

}

public synchronized static void method() {

}

public void method () {

synchronized(this) {

}

}

}

Lock interface

* lock(), unlock(), newCondition(), tryLock(), lockInterruptibly()
* ReentrantLock class

ReadWriteLock interface

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

Future vs CompletableFuture

CompletableFuture is an extension to Java’s Future API which was introduced in Java 5.

A Future is used as a reference to the result of an asynchronous computation. It provides an isDone() method to check whether the computation is done or not, and a get() method to retrieve the result of the computation when it is done.

Future API was a good step towards asynchronous programming in Java but it lacked some important and useful features.

# 

# 

# 