03/09/2022 – 03/11/2022

**Concurrency vs. Parallel**

Concurrency: multiple tasks run simultaneously. You can’t tell which task comes first and which task comes after. You can’t tell the order in general.

Parallel: multiple tasks **physically** run simultaneously. This is an implementation level concept. In real time, there are at least two executors.

If there is concurrency, there does not have to have parallel.

If there is parallel, there must have concurrency.

ConcurrentModificationException example:

for (int i : myList) {

myList.remove(i);

}

**Multi-process vs. multi-thread**

Process: an independent execution of instructions with independent memory space, stack, heap, and OS resource. Each process sees a complete memory space (pretend to be the only task of a system). Different processes communicate through inter process communication (explicit IPC).

Thread: an independent execution of instructions with shared memory space. Each thread has its private stack, program counter, and register states. Thread in the same process has shared: heap, static memory segment, OS resource. Communication performed through shared memory read/writes.

Difference: **independent memory space**. If has, process; otherwise, thread.

Multi-process: higher communication overhead, better resource isolation (fault tolerance), higher creation/destroy overhead.

Multi-thread: lower communication overhead, worse resource isolation (fault tolerance), lower creation/destroy overhead.

Multi-thread example:

Public static void main () {

Thread t = new Thread () {

@Override

Public void run () {

System.out.println(“1”);

}

};

t.start();

System.out.println(“2”);

t.join();

System.out.println(“3”);

}

Main thread --- new thread --- print “2” --- join --- print “3” --- exit

| |

---------- print “1” -----

Another example:

Public static void main () {

Thread t = new Thread () {

@Override

Public void run () {

System.out.println(“1”);

}

};

t.start();

System.out.println(“2”);

}

Main thread --- new thread --- print “2” --- exit

| |

---------- print “1” -----

**When will JVM exit? When there are no alive non-daemon threads.**

To make t a daemon thread: t.setDaemon(true);

PS: for JVM, GC is a daemon thread.

**Ways of creating threads and making them run**

1. extends Thread
2. implements Runnable
3. implements Callable

**Methods of thread**

Static methods: sleep(1000), yield() // we can see yield() as sleep(0)

**Synchronization and Race**

Data race: If two “conflicting operations” are in different threads and are not properly synchronized (concurrent), they will introduce data races. In general, two operations conflict with each other if they operate on the same memory location, and at least one of them is a write. Races are mostly treated as bugs in Java programs.

Three factors of data race:

1. more than one operation work on the same memory location
2. at least one operation is a write
3. at least two of those operations are concurrent

**Locks**

**Deadlock**

Condition to form a deadlock:

1. mutual exclusion: at least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
2. hold and wait or resource holding: a process is currently holding at least one resource and requesting additional resources which are being held by other process.
3. no preemption: a resource can be released only voluntarily by the process holding.
4. circular wait

**Livelock**

**Condition Synchronization**

Producer consumer problem:

* Consumer: if the queue is empty, wait for it to be not empty, then poll one element from it.
* Producer: if the queue is full, wait for it to be not full, then offer one element to it.

03/14/2022

**Volatile keyword**

Ex:

Class SharedObj{

// changes made to sharedVar in one thread

// may not immediately reflect in other thread

static int sharedVar = 6;

}

Suppose two threads are working on the SharedObj and they are running on different processors. Each thread may have its own local copy of shared variables. If one thread modified its value, the changes may not reflect in the original one in the main memory instantly. Now the other thread is not aware of the modified value which leads to data inconsistency.

Class SharedObj{

// volatile keyword makes sure that the changes made in one thread are immediately

// reflect in other thread

static volatile int sharedVar = 6;

}

**volatile vs. synchronized**

1. Mutual exclusion: it means that only one thread or process can execute a block of code (critical section) at a time.
2. Visibility: it means that changes made by one thread to shared data are visible to other threads.

The keyword synchronized guarantees both mutual exclusion and visibility.

The keyword volatile guarantees only the visibility. The value of volatile variables will never be cached and all writes and reads will be done to and from the main memory.

!!! the use of volatile is limited to the cases as most of the times, atomicity is desired.

Ex: a++ is a compound read-modify-write sequence of operations that must execute atomically.

**Sequential consistency**

Writing of a normal variable without any synchronization actions might not be visible to any reading thread.

03/15/2022

**String/StringBuilder/StringBuffer**

* String is immutable
* StringBuilder and StringBuffer are mutable
* StringBuffer is thread safe, StringBuilder is not

Constant pool

* == compares reference address
* Equals() depends on how you implement this method, usually compare the content. Default implementation is same as ==
* new -> in heap
* String a = “aaa”; String b = “aaa”; // in String pool, so a.equals(b) true

03/16/2022

Reviewed:

**Java Collection**

!!! Collection + Map, Map is not a part of Collection

* List – use ArrayList/LinkedList to implement
  + Remove() for both are O(n) not different reason
* Stack and Queue – use Deque to implement
  + For Deque, we use offerFirst(), pollFirst(), offerLast(), pollLast()
  + There are method like pop() but it can be confusing -> prefer not to use
* HashTable vs. HashMap vs. ConcurrentHashMap
* How HashMap works? Ex: find the value of a key
  + Key -> use hashCode() to get hash code
  + Hash code % length (16 buckets) -> get index of the bucket
  + Compare each nodes in that bucket, use equals()
  + If true, return value; if false until the end, does not exist
* Heap – use PriorityQueue
  + We **don’t know** which is larger: left and right

Tomorrow plan:

Comparable and comparator

JVM architecture

Today:

**Java reflection** – Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its members and display them.

* jave.lang.reflect.\*;
* ex: Class c = Class.ForName("Student”);

Method m[] = c.getDeclaredMethods();

**Singleton design pattern**

* lazy initialization
  + object is created only if it is needed.
  + Implementation: use getInstance() method to return the instance.

public class SingletonExample1 {

private SingletonExample1() {}

private static SingletonExample1 instance = null;

public static SingletonExample1 getInstance() {

if (instance == null) {

instance = new SingletonExample1();

}

return instance;

}

}

* eager initialization
  + object of class is created when it’s loaded to the memory by JVM. It’s done by assigning the reference an instance directly.

public class SingletonExample2 {

private SingletonExample2() {}

private static SingletonExample2 instance = new SingletonExample2();

public static SingletonExample2 getInstance() {

return instance;

}

}

* thread safe
  + synchronized getInstance()

Factory design pattern

Builder design pattern

proxy design pattern

03/17/2022

Reviewed:

**Comparable vs. Comparator**

* Comparable

A comparable object can compare itself with another object.

The class implement Comparable interface to compare its instances.

Override the method compareTo().

class Movie implements Comparable<Movie> {  
 double rating;  
 String name;  
 int year;  
   
 public Movie(String name, double rating, int year) {  
 this.name = name;  
 this.rating = rating;  
 this.year = year;  
 }  
   
 @Override  
 public int compareTo(Movie m) {  
 return this.year - m.year;  
 }  
}

There is only one chance to implement the compareTo() method.

The object itself must know how it is to be ordered, and the sorting of objects needs to be based on natural order.

If a class implements Comparable interface, then collection of that object either List or Array can be sorted automatically by using Collections.sort() or Arrays.sort().

* Comparator

Comparator is external to the element type we are comparing. It’s a separate class.

We can create multiple separate classes to compare by different members.

class RatingCompare implements Comparator<Movie> {  
 @Override  
 public int compare(Movie m1, Movie m2) {  
 if (m1.getRating() < m2.getRating()) return -1;  
 if (m1.getRating() > m2.getRating()) return -1;  
 else return 0;  
 }  
}

We can write more than one custom comparator using different interpretations of what sorting means.

**JVM architecture**

* Class Loader – prepares the Java classes and loads them into main memory
  + Loading
    - Bootstrap Class Loader
    - Extension Class Loader
    - Application Class Loader
  + Linking
    - Verify
    - Prepare
    - Resolve
  + Initialization
    - Initialize
* Runtime Memory – holds the runtime variables and data
  + Method Area
  + Heap Area
  + Stack Area
    - Thread #1
    - Thread #2
    - Thread #N
  + PC Register
    - Thread #1
    - Thread #2
    - Thread #N
  + Native Method Stack
* Execution Engine – executes the Java program
  + Executions Engine
    - Interpreter
    - JIT Compiler
    - Garbage Collector
  + JNI (Native Method Interface)
  + Native Method Library

Tomorrow:

GC

Class Loader in details

Today:

Java Web

**OSI model (7 layers) and TCP/IP model (4 layers)**

|  |  |
| --- | --- |
| 7.Application layer | 1.Application layer |
| 6.Presentation layer |
| 5.Session layer |
| 4.Transport layer | 2.Transport layer |
| 3.Network layer | 3.Internet layer |
| 2.Data Link layer | 4.Network Access layer |
| 1.Physical layer |

**HTTP**

HTTP Request

* HTTP version type
* A URL
* HTTP method
  + GET, PUT, POST…
* HTTP request headers
* HTTP body (optional)

HTTP Response

* HTTP status code
  + 1XX information
  + 2XX success
    - 200 OK, get/put/post
    - 201 successfully created, post
    - 202 successfully received the request, but still processing
    - 204 no content, usually update the resources without changing the current display page, put
  + 3XX redirection
    - 307 temporary redirect
    - 308 permanent redirect
  + 4XX client error
    - 400 bad requests, the server could not understand the request due to invalid syntax
    - 401 unauthorized, the client is not authenticated
    - 403 forbidden, the client has not permission
    - 404 not found, the server cannot find the requested resources
  + 5XX server error
    - 500 internal server error
    - 501 not implemented, method not supported by the server
    - 502 bad gateways
* HTTP response header
* HTTP response body (optional)