CITI (08-Feb-2021) (2nd round 1 hour around)

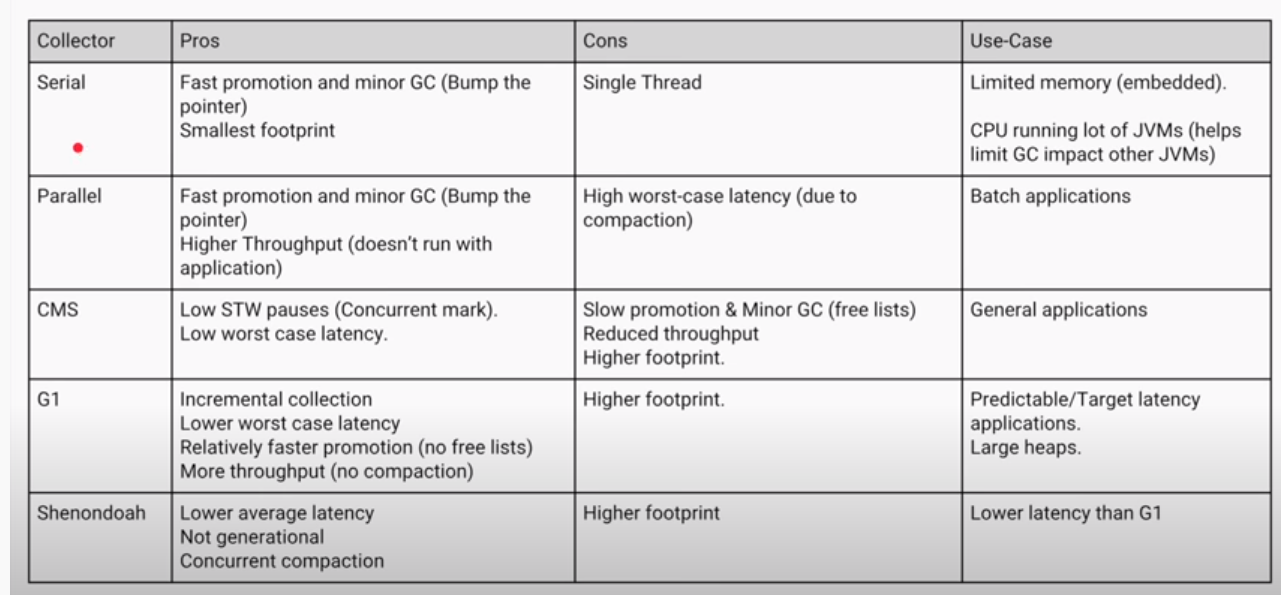
Java -

1. If I need to maintain insertion order in map how can I do that? – LinkedHashMap
2. How LinkedHashMap work internally?

<https://www.dineshonjava.com/internal-working-of-linkedhashmap-in-java/>

1. How TreeMap work internally? – it is red black tree – self balanced tree?
2. What is red-black tree?
3. Which GC you used? What is G1 and CMS GC? https://www.youtube.com/watch?v=2AZ0KKeXJSo

The heap is partitioned into a set of equal-sized heap regions, each a contiguous range of virtual memory. Certain region sets are assigned the same roles (eden, survivor, old) as in the older collectors, but there is not a fixed size for them. This provides greater flexibility in memory usage.



1. What are the OOPS concept? Do you think these are useful?
2. What is encapsulation?
3. What is aggregation, composition association? - Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.  
   In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. **Composition** and **Aggregation** are the two forms of association.

Aggregation : It is a special form of Association where:

* It represents **Has-A** relationship.
* It is a **unidirectional association** i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
* In Aggregation,**both the entries can survive individually** which means ending one entity will not effect the other entity.

Composition : It is a restricted form of Aggregation in which two entities are highly dependent on each other.

* It represents **part-of** relationship.
* In composition, both the entities are dependent on each other.
* When there is a composition between two entities, the composed object **cannot exist** without the other entity.

**Aggregation vs Composition**

1. **Dependency:** Aggregation implies a relationship where the child **can exist independently** of the parent. For example, Bank and Employee, delete the Bank and the Employee still exist. whereas Composition implies a relationship where the child **cannot exist independent** of the parent. Example: Human and heart, heart don’t exist separate to a Human
2. **Type of Relationship:** Aggregation relation is **“has-a”** and composition is **“part-of”** relation.
3. **Type of association:**Composition is a **strong** Association whereas Aggregation is a **weak** Association.
4. Print odd even in two threads but in a sequence –
5. The wait and notify on which you called? Means why these methods are in objects and not in thread?
6. Why Synchronized/locking is required?
7. Threadlocal variables when to use?

ThreadLocal in Java is another way to achieve thread-safety apart from writing immutable classes.

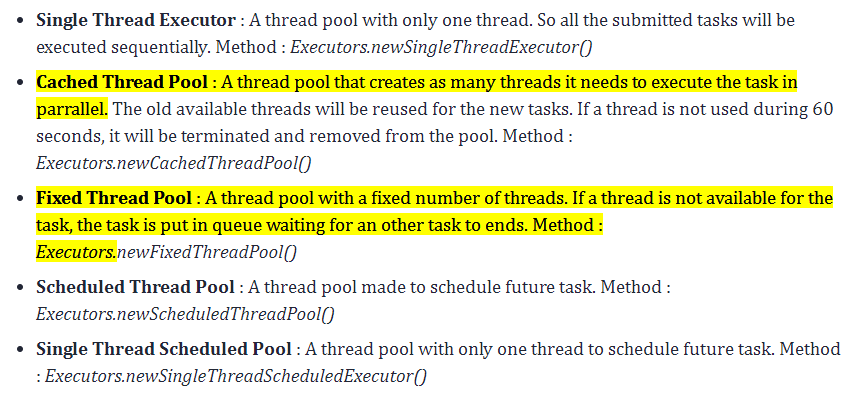
Java ThreadLocal class provides thread-local variables. It enables you to create variables that can only be read and write by the same thread. If two threads are executing the same code and that code has a reference to a ThreadLocal variable then the two threads can't see the local variable of each other.

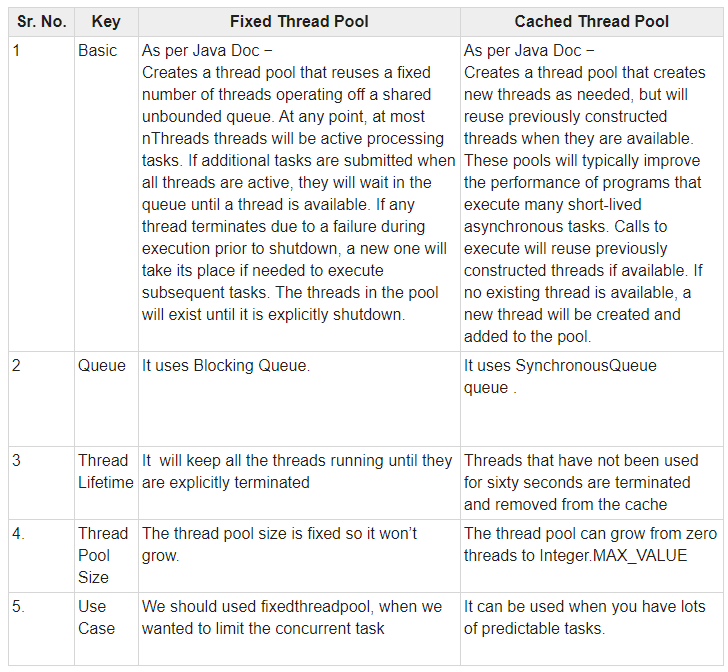
**private** ThreadLocal<Integer> threadLocal = **new** ThreadLocal<Integer>();

threadLocal.set(100);

Integer threadLocalValue = threadLocal.get();

1. Cache thread pool vs Fixed thread pool?



1. 
2. How many ways threads can be created? 2 ways Extends Thread class and implement Runnable
3. How many threads one can create? How threads are created internally?
4. What is RejectedExecutionException? When do we get and how to handle?

New tasks submitted in method execute(java.lang.Runnable) will be rejected when the Executor has been shut down, and also when the Executor uses finite bounds for both maximum threads and work queue capacity, and is saturated. In either case, the execute method invokes the RejectedExecutionHandler.rejectedExecution(java.lang.Runnable, java.util.concurrent.ThreadPoolExecutor) method of its RejectedExecutionHandler. Four predefined handler policies are provided:

In the default ThreadPoolExecutor.AbortPolicy, the handler throws a runtime RejectedExecutionException upon rejection.

In ThreadPoolExecutor.CallerRunsPolicy, the thread that invokes execute itself runs the task. This provides a simple feedback control mechanism that will slow down the rate that new tasks are submitted.

In ThreadPoolExecutor.DiscardPolicy, a task that cannot be executed is simply dropped.

In ThreadPoolExecutor.DiscardOldestPolicy, if the executor is not shut down, the task at the head of the work queue is dropped, and then execution is retried (which can fail again, causing this to be repeated.)

It is possible to define and use other kinds of RejectedExecutionHandler classes. Doing so requires some care especially when policies are designed to work only under particular capacity or queuing policies.

1. Design pattern which one you have used? – Observer, Façade, Factory, singleton
2. If I have a complex service and I want to simplify then which design pattern should I use? – Façade.
3. How to find cycle in a linked list? – using two pointer
4. What is the space complexity in merge and quick sort?
5. How to create queue like functionality from stack? – using two stack, but the time complexity will be O(n) for get

*enQueue(q, x): While stack1 is not empty, push everything from stack1 to stack2.*

* *Push x to stack1 (assuming size of stacks is unlimited).*
* *Push everything back to stack1.*

*Here time complexity will be O(n)*

*deQueue(q):*

* *If stack1 is empty then error*
* *Pop an item from stack1 and return it*

*Here time complexity will be O(1)*

1. What are the ACID properties in DB? Atomicity, consistency, Isolation and Durability **Atomicity**  
   By this, we mean that either the entire transaction takes place at once or doesn’t happen at all.

**Consistency :**  
This means that integrity constraints must be maintained so that the database is consistent before and after the transaction.

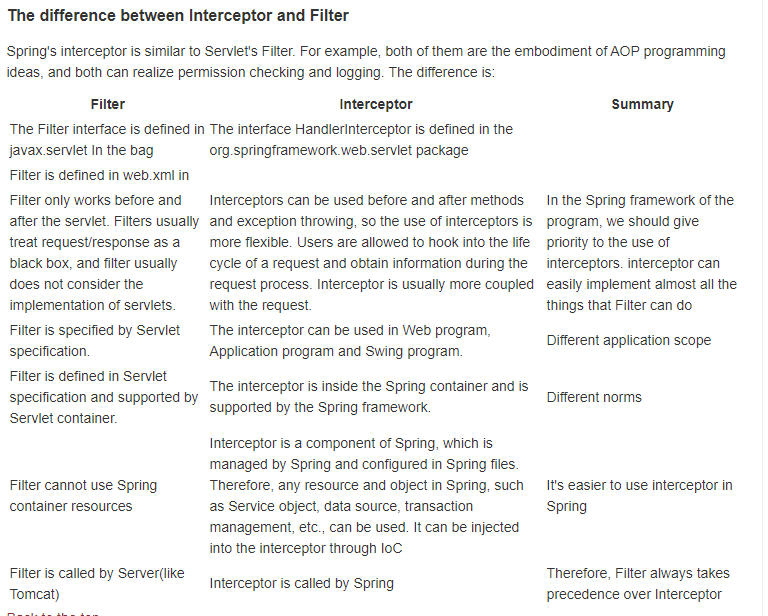
**Isolation**  
This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state.

**Durability:**  
This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.

1. Spring version? V 5.0.13 ,SPARK 2.11 ,CDH 6.3.2
2. Spring Interceptor and filter?

Spring Interceptor are used to intercept client requests and process them. Sometimes we want to intercept the HTTP Request and do some processing before handing it over to the controller handler methods. That’s where Spring MVC Interceptor come handy.

<https://programmer.help/blogs/the-difference-between-filters-and-interceptors-in-java.html>



**Difference**

**1. Different scope and specifications**

* filter is defined by servlet specification and can only be used in web program
* Interceptors can be used in web programs, application programs and swing programs. They are within the Spring container and supported by the Spring framework

**2. Different trigger times**

Order: Filter > servlet > interceptor > controller

* The filter preprocesses the request after it enters the container, but before it enters the servlet. The end of the request return is also the filter processing after the servlet processing and before returning to the front end.
* The interceptor takes effect before the method reaches the Controller layer

**3. The implementation of filter is based on callback function. The implementation of interceptor (agent mode) is based on reflection. The agent is divided into static agent and dynamic agent. Dynamic agent is a simple implementation of interceptor.**

When to use interceptors? When to use filters?

* If it's a non spring project, the interceptor can't be used, only filters can be used.
* Before and after handling the controller, you can use both interceptors and filters.
* Only filters can be used before and after the dispatcher servlet is processed.

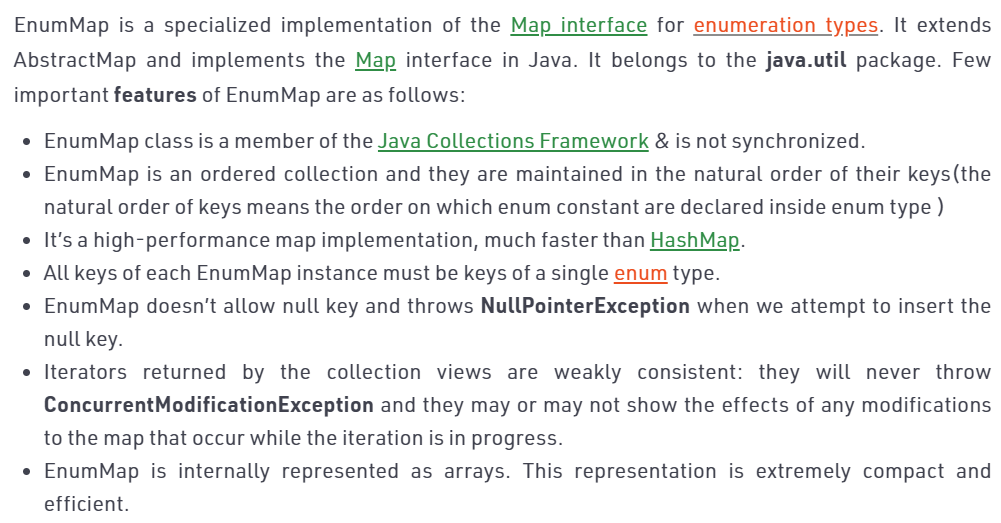
4. In the life cycle of an action, an interceptor can be called multiple times, while a filter can only be called once during container initialization.

5. The interceptor can access the objects in the action context and value stack, but the filter cannot.

6. Interceptors can only work on action requests, while filters can work on almost all requests.

7. The interceptor can obtain the bean s in the IOC container, but the filter cannot. Inject a service into the interceptor and call the business logic

**26 What is Enum MAP .**



### 27.Recursive ThreadPool, and ForkJoinPool

It solves a common problem of **spawning multiple tasks in recursive algorithms**. Using a simple *ThreadPoolExecutor*, you will run out of threads quickly, as every task or subtask requires its own thread to run.

In a *fork/join* framework, any task can spawn (*fork*) a number of subtasks and wait for their completion using the *join* method. The benefit of the *fork/join* framework is that it **does not create a new thread for each task or subtask**, implementing the Work Stealing algorithm instead

### https://www.baeldung.com/thread-pool-java-and-guava

**28.How Map.Entry storage changed in 1.8**

In Java 8, HashMap replaces linked list with a binary tree when the number of elements in a bucket reaches certain threshold. While converting the list to binary tree, hashcode is used as a branching variable if there are two different hashcodes in the same bucket, one is considered bigger and goes to the right of the tree and other one to the left. But when both the hashcodes are equal, HashMap assumes that the keys are comparable, and compares the key to determine the direction so that some order can be maintained. It is a good practice to make the keys of HashMap comparable.

This JDK 8 change applies only to **HashMap, LinkedHashMap** and **ConcurrentHashMap**

**29 ConcurrentHashMap does** not **allow null keys** and **null** values

30.which Queue is used in Executor Service by default. :- LinkedBlockingQueue

The executor service creates and maintains a reusable pool of threads for executing submitted tasks. The service also manages a queue, which is used when there are more tasks than the number of threads in the pool and there is a need to queue up tasks until there is a free thread available to execute the task.

Directly instantiating a ThreadPoolExecutor with 10 threads, a keepAliveTime of 0 milliseconds, and a LinkedBlockingQueue:

***ExecutorService executorService =***

***new ThreadPoolExecutor(10, 10, 0L, TimeUnit.MILLISECONDS,***

***new LinkedBlockingQueue<Runnable>()); //Here we are passing Queue Explicitly***

These parameters are:

* corePool size
* maxPool size
* workQueue
* keepAliveTime
* threadFactory
* rejectedExecutionHandler

**Instantiating a ThreadPoolExecutor with 10 threads using an Executors factory method:**

***ExecutorService executor = Executors.newFixedThreadPool(10);***

Meanwhile, Executors.newFixedThreadPool(10) creates a pool with a core size of 10, max size of 10, a keepAliveTime of 0ms, an unbounded LinkedBlockingQueue, the default threadFactory, and the default rejectedExecutionHandler.

[**https://dzone.com/articles/a-deep-dive-into-the-java-executor-service**](https://dzone.com/articles/a-deep-dive-into-the-java-executor-service)