* + Java Development
* + Collections
  + Differences between Linked List and Array List. Best in:
    - Array List get (*Recuperar)* data. Because no need to walk through list, getting it directly.
    - Linked List add/remove (*Añadir Borrar)* data. No need to do bit shifting since there is two side link.
    - If there will be many add/remove data, the performance is better in Linked List. But Array List is better when there are more get operations.
  + Differences between Lists and Sets
    - A Set has unique items, but a List can have many same records.
  + Differences between Queues and Stacks
    - Queue – FIFO, Stack – LIFO
  + How HashMaps / Hashtables work (hashing, collissions, Object#hashcode)
    - HashTable is synchronized and if no need to be synchronized, HashMap is a better choice for performance, and null can be used.
    - It is better to create the hashmap with an initial size to avoid re hashing in each size exceed for a better performance.
* + JVM pooled objects
  + Strings
  + primitive data types
* + Concurrency
* + Concepts
  + Deadlock
  + Livelock
  + Starvation
  + Race Condition
* + Abstractions
* + SYNCHRONIZED

*See BasicThreadCreation on Eclipse.*

*Synchronization is used to solve data inconsistency problem.*

* + at class level

What happens when a static synchronized method is invoked, is; since a static method is associated with a class, not an object. In this case, the thread acquires the intrinsic lock for the Class object associated with the class. Thus access to class's static fields is controlled by a lock that's distinct from the lock for any instance of the class

* + at instance level

Synchronized keyword can be applied to a method or block. It creates a critical section, means that only one thread can access this block at a time, uses, then releases it.

When there are two or more threads are changing the same instance's value, there may be same time processes and this same time process will cause the two threads read same value , but not aware of each other's change.

*Every object in java has monitor log (mutex) - intrinsic lock. This lock will be able to taken by one thread in a time that it will release the lock after finishes its work.*

*Volatile is making an object to be seen to all of the threads.*

*Synchronized, makes this available also.*

* + Object#wait, Object#notify, Object#notifyAll
  + Thread#join

Join is used to make current thread to wait the joining thread to finish it's job.

* + + Concurrency API
  + + java.util.concurrent.locks
  + Lock
  + Condition
    - + java.util.concurrent
  + CyclicBarrier

*await()*

It is a number watch to make all threads wait in a specific point of code(cyb.await()) until the specified amount of threads come until this point, and then make all continue (in a random priority as JVM acted in other thread queues)

* + CountDownLatch

*countdown(), await(), getCount()*

It is a synchronizer. The thread where the .await() is positioned, will be waiting for that Latch to complete the amount that specified when it instantiated like :

*CountDownLatch ctl = new CountDownLatch(5);*

* + Java 8
  + Lambdas
  + Method References

+ Interfaces

* + Default Methods
  + Static Methods
  + Interfaces vs. Abstract Classes

+ API

* + Streams
  + Optional
  + CompletableFuture
* + Garbage Collections
* + Generations
  + + Young
  + Eden
  + Survivor
  + Virtual
  + + Tenured
  + Virtual
* + GC Algorithms
  + Serial Collector
  + Parallel Collector
  + Concurrent Mark Sweep (CMS) Collector
  + Garbage-First Garbage Collector
* + Profiling
  + visualmv
  + + load testing
  + jmeter
  + locust io
* + Frameworks / Specs
* + Spring
  + IOC
  + MVC
  + Transactions
  + Cache
  + Boot
* + ORM
  + Hibernate
  + JPA
* + Servlets / JSP
* + 3.0+
  + Async Requests
  + JSTL
* + Architecture
  + SOLID principles
* [**Single responsibility principle**](https://en.wikipedia.org/wiki/Single_responsibility_principle)[[4]](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)#cite_note-4)
* a [class](https://en.wikipedia.org/wiki/Class_(computer_science)) should have only a single responsibility (i.e. changes to only one part of the software's specification should be able to affect the specification of the class).
* [**Open/closed principle**](https://en.wikipedia.org/wiki/Open/closed_principle)[[5]](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)#cite_note-5)
* "software entities … should be open for extension, but closed for modification."
* [**Liskov substitution principle**](https://en.wikipedia.org/wiki/Liskov_substitution_principle)[[6]](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)#cite_note-6)
* "objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program." See also [design by contract](https://en.wikipedia.org/wiki/Design_by_contract).
* [**Interface segregation principle**](https://en.wikipedia.org/wiki/Interface_segregation_principle)[[7]](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)#cite_note-7)
* "many client-specific interfaces are better than one general-purpose interface."[[8]](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)#cite_note-martin-design-principles-8)
* [**Dependency inversion principle**](https://en.wikipedia.org/wiki/Dependency_inversion_principle)[[9]](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)#cite_note-9)
* one should "depend upon abstractions, [not] concretions
  + GoF patterns
  + EAI patterns
  + SOA patterns
  + Java design patterns
    - Singleton Pattern
    - Factory Pattern (define the type of the class that will be created)
    - Strategy Pattern
  + Microservices patterns
  + Reactive patterns
* + Distributed Apps
  + ACID transactional properties
  + BASE and Eventual Consistency
  + CAP Theorem
  + FLP Impossibility
  + + Consensus
  + Paxos
  + Raft
* + Tools / Frameworks
  + Spring Cloud
  + Hystrix (Netflix)
  + Apache Zookeper
* + DevOps
  + Docker
  + Kubernetes
  + Chef / Puppet
  + Bash
* + Methodoligies
  + Agile
  + Scrum