Contents

[Object Oriented Programming 3](#_Toc2326868)

[Object Oriented Terminologies 3](#_Toc2326869)

[Object Oriented Concepts 3](#_Toc2326870)

[OOP Principles 3](#_Toc2326871)

[GRASP 4](#_Toc2326872)

[Software Engineering Techniques 4](#_Toc2326873)

[Java 4](#_Toc2326874)

[Interface and Abstract Class 4](#_Toc2326875)

[Why String has String Pool 4](#_Toc2326876)

[Immutability 5](#_Toc2326877)

[Singleton and Static 5](#_Toc2326878)

[Java Memory Model 5](#_Toc2326879)

[Heap 5](#_Toc2326880)

[PermGen 6](#_Toc2326881)

[Stack 6](#_Toc2326882)

[Garbage Collection 6](#_Toc2326883)

[Types of Garbage Collection 6](#_Toc2326884)

[References in Garbage collection 7](#_Toc2326885)

[Java 8 features 7](#_Toc2326886)

[Storing password in char array instead of String 8](#_Toc2326887)

[Thread-Safe 8](#_Toc2326888)

[Synchronized and locks 8](#_Toc2326889)

[Thread & Concurrency 9](#_Toc2326890)

[Process and Thread 9](#_Toc2326891)

[Difference between wait, sleep, yield and join methods 9](#_Toc2326892)

[Create deadlock in java 9](#_Toc2326893)

[Difference between Callable and Runnable 9](#_Toc2326894)

[Difference between Executor submit and execute methods 9](#_Toc2326895)

[Enum vs Constants 9](#_Toc2326896)

[Volatile and transient variables 10](#_Toc2326897)

[Web Services 10](#_Toc2326898)

[SQL 10](#_Toc2326899)

[Types of Joins 10](#_Toc2326900)

[Design Patterns 11](#_Toc2326901)

[Creational Patterns 11](#_Toc2326902)

[Structural Patterns 12](#_Toc2326903)

[Behavior Patterns 12](#_Toc2326904)

[Microservice Design Patterns 13](#_Toc2326905)

[Algorithms used by Java 14](#_Toc2326906)

[Batch & Stream Processing 14](#_Toc2326907)

[Java NIO 14](#_Toc2326908)

[Pass by Value 14](#_Toc2326909)

[HashMap 15](#_Toc2326910)

[Effective Java 15](#_Toc2326911)

[Spring 16](#_Toc2326912)

[Spring Boot 17](#_Toc2326913)

[Spring Data 18](#_Toc2326914)

[Microservices 19](#_Toc2326915)

[Scrum 19](#_Toc2326916)

[Data Structures 20](#_Toc2326917)

[Stacks and Queues 20](#_Toc2326918)

[Graphs 20](#_Toc2326919)

[AWS 20](#_Toc2326920)

[EC2 (Elastic Compute Cloud) 20](#_Toc2326921)

[Migration process 21](#_Toc2326922)

[Databases 22](#_Toc2326923)

[Hibernate 23](#_Toc2326924)

[Maven 24](#_Toc2326925)

[Different Scopes in Maven 24](#_Toc2326926)

[Python 24](#_Toc2326927)

[Installing pip and packages 24](#_Toc2326928)

[Tools 24](#_Toc2326929)

[Load Balancer 24](#_Toc2326930)

[Stream Processing 24](#_Toc2326931)

[Containers 24](#_Toc2326932)

[Scheduling 24](#_Toc2326933)

[Docker 25](#_Toc2326934)

# Object Oriented Programming

## Why OOP

* OOP resulted as part of bad coding which has below symptoms
  + Rigidity or high coupling – high interdependence between software modules.
  + Fragility – tendency of the code change to break other module which was not part of the functionality).
  + Not reusable (if want to reuse a part of the module, the module might have tighter coupling with its own database or server which is not needed)
* OO is about managing dependencies by selectively reinverting certain key dependencies in your architecture so that you can prevent rigidity, fragility and non reusability.

## Object Oriented Terminologies

* Cohesion – refers to the degree of elements inside a module belonging to each other.
* Coupling – degree of interdependence between software modules.
* Software built with high cohesion is associated with traits like robustness (capability to recover from failure), reliability, reusability,
* High cohesion correlates to loose coupling and vice versa.

## Object Oriented Concepts

* Abstraction: In Java, abstraction is hiding certain details and showing essential features of the object. It lets users to focus on capabilities of the object, not on how it does.
* Encapsulation: This deals with the state of the object. Objects encapsulate their state and users of object can access through functions or methods. Example is classes having variables with private modifiers and expose their state using getter and setter.
* Inheritance: Java allows classes to inherit state and behavior from other classes. Each class is allowed to have only one superclass and each superclass can have unlimited subclasses.
* Polymorphism:
  + We can call method overriding as polymorphism where subclasses can provide their own implementation.
  + Without polymorphism, modules will be built in a tree structure where main module has the reference of high level modules and high level modules had reference of low level modules. The flow of control will be from main to low. To simplify, M calling function F of N needs to have the reference of N (import in java) which can be called as source code dependency (M->N has flow of control and source code dependency in the same direction). In this case, flow control and source code dependency work in the same direction. This will trigger the recompilation of all source code dependencies till the main for any change in the low level module. This is the symbol of rigidity and fragility.
  + With polymorphism, we can have M call interface I which will be implemented by N. With interface coming in source code dependency (M->I, N->I) goes against flow of control (M->N).Here we can have one module calling other but source code dependency can go against flow of control. This will provide us control over our dependency structure which will help us to build non rigid and non fragile code.
* Class: blueprint or prototype of object which defines its state and behavior.
* Interface: It is a contract where anybody implementing an interface needs to provide the behavior.

## OOP Principles

SOLID is an acronym for five OO design principles to make software designs more flexible, understandable and maintainable.

* Single Responsibility Principle (SRP)
  + States that a class or module should have one and only reason to change. The reason could be that class should serve single business purpose so that it doesn’t change when another business team request for a change.
  + Other way of saying things – Gathering together the things that change for the same reasons and separate those that change for different reasons.
* Open Closed Principle (OCP)
  + Open closed principle states that software entities such as classes, modules, functions should be open for extension, but closed for modification.
  + A module is said to be open if it is still available for extension such as adding field to the data structure it contains or new elements to the set of functions.
* Liskov Substitution Principle
  + Derived classes must be usable through the base class interface, without the need for the user to know the difference.
  + If S is a declared subtype of T(for Java, S is a declared subtype of T if S is T or S implements or extends T or S implements or extends a type that implements or extends T and so on), objects of type S should behave as objects of type T are expected to behave, if they are treated as objects of type T.
* Interface Segregation Principle (ISP)
  + States that no client should be forced to depend on methods it does not use.
  + ISP splits large interfaces into smaller ones so that clients will only have to know the functions that interests them.
* Dependency Inversion Principle
  + Specific form of decoupling software modules where high level modules should not depend on low level modules (both should depend on abstractions) and abstractions should not depend on details (details should depend on abstractions).

## GRASP

* GRASP stands for General Responsibility Assignment Software Patterns consist of guidelines to assign responsibility to classes and objects in object oriented design. Different patterns and principles used in GRASP are
* Controller
* Creator
* Indirection
* Information Expert
* High Cohesion
* Low Coupling
* Polymorphism
* Protected Variations
* Pure Fabrication

## Software Engineering Techniques

* Dependency Injection

# Java

## Interface and Abstract Class

With Java 8, Interfaces can have default and static methods. The main difference between abstract class and interface will be abstract class can have the state whereas interface cannot. Abstract class can have constructors which interface cannot have.

Java designers had to add default and static methods in interfaces so that Collection API can be adapted to have lambda expressions and have backward compatibility.

## Why String has String Pool

* JVM actually holds all string literals in a constant pool. Any repetition of a String literal can be referenced (by all running classes) from the same constant in the pool.
* By making string pool, memory is used efficiently as String is the largest used data type across.
* Equal methods on String class is faster since it checks if the reference is same.
* When String object is created using new operator, Java will create the object inside heap space. We can then manually intern to store the reference in the pool.
* String object created using quotes will refer the value in string pool (if it doesn’t exist, it will create one in string pool).
* [Immutable](#_Immutability) objects are [thread-safe](#_Thread-Safe).
* All of number classes such as Integer, Double, Character and BigInteger are immutable (but they don’t have pools).
* String pool was in [Permgen space](#_Java_Memory_Model) until Java 7, since Permgen space was in fixed size, it can’t be extended at runtime and not eligible for garbage collection. From Java 7, string pool is stored on heap space, which is garbage collected.

## Immutability

* Immutable classes are those whose objects cannot be modified once they are created. If we try to modify the object, a new immutable object will be created.
* Immutability offers inbuilt [thread safety](#_Thread-Safe) mechanism, it doesn’t need to be synchronized.
* Creating immutable class starts by declaring the class as final and its member variables as final. You can also have non final member variable, but you have to declare it as private or not allow them to modify its value except constructor. One more possibility is that it could have mutable class as its member variable (for example java.sql.Date). While giving back mutable member variable, we should give clone of the mutable class object to preserve immutability.

## Singleton and Static

## Java Memory Model

* What JVM does basically is loads the code, verifies it, executes the code and manages the memory and provides runtime environment.
* JVM memory is primarily divided into heap, non-heap (both are created during JVM startup) and other.

Heap - contains class instances and arrays. Size of the heap can be mentioned during JVM startup using -Xms option. Heap can be fixed or variable size based on garbage collection strategy.

* Non-heap - stores class structures (fields, code for methods and constructors), interned Strings ([until Java 7](#_Why_String_has)), runtime constant pools.
* Other - contains JVM code and JVM internal structure, profiler agent code and data.
* Heap is divided into nursery (young space or young generation) and old space (old generation). Nursery is reserved for new objects, when nursery becomes full, young collection is run to move the objects from nursery to old space (minor GC). Nursery is further divided into Eden memory and 2 survivor memory.
* Most of newly created objects are located in Eden memory space. When Eden space is full, Minor GC is performed and all active objects will be moved to one of the survivor space. Minor GC also checks survivor objects in survivor space and moves them to other survivor spaces. Objects that have survived many cycles of GC (threshold) will be moved to Old generation.
* When old generation is full, old collection (major GC) happens which takes longer time as it contains which has survived multiple Minor GC.
* Recent releases include a part of nursery called keep area and it is reserved. Keep area contains the most recently allocated objects in nursery and will not be part of garbage collection till the next young generation (it prevents objects from being promoted because they were allocated just before Minor GC starts).
* OutOfMemoryError will be thrown if there is no space in heap to create objects.

PermGen (replaced by Metaspace in Java 8) contains classes, methods used in application, it also contains Java SE library classes and methods. They are garbage collected during full garbage collection (Major GC). Java allocated Metaspace in native memory. Metaspace doesn't have any size limit by default (there are options to set metaspace size limits) and it will keep increasing eating up OS memory.

* Code Cache - Java executes the code in tiered manner (Java 8 has this by default), first using client compiler to compile code with instrumentation (?), then using server compiler (?) to compile in an optimized manner. Just-in-time (JIT) compiler stores the compiled code in special heap called code cache. This area is flushed if code cache size is reached.

Stack is a part (not part of heap) of the memory where primitive values, object and method references are stored. The lifetime of variables on the stack is governed by the scope. Like, when we call a method, all declared variables will be placed on top of the stack. Calling another method will push new method’s variables onto the stack. Once the execution has left the scope, those variables declared in the stack will be removed.

* StackOverflowError comes if the method calls itself too many times, fills up the stack memory and any more method unable to allocate memory in the stack.
* Memory leak happens in Java when GC fails to recognize unused objects to be collected. It happens when an object is no longer used in the program, but it is referenced at a location which is not reachable by GC.

## Garbage Collection

* Garbage collection involves 3 main steps - Marking, Normal Deletion and Deletion with compacting.
  + Marking - GC identifies objects in use and not in use.
  + Normal Deletion - GC removes unused objects and reclaims free space.
  + Deletion with compacting - after deletion, all survived objects can be moved to increase the performance of allocation for new objects.
* JVM does mark and sweep approach consisting of mark and sweep phases. During mark phase, all objects that are reachable from Java thread, native handlers and other root sources are marked as alive as well as the objects reachable from these objects. Rest are considered as garbage. Sweep phase, JVM traverses heap to find gaps between live objects for new object allocation.

### Types of Garbage Collection

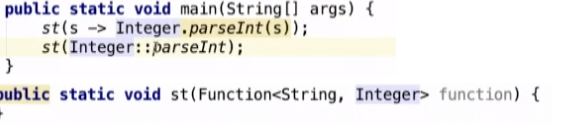
* UseSerialGC (-XX:+UseSerialGC): uses simple mark-sweep-compact approach for young (Minor) and old generations (Major) GC.
* ParallelGC (-XX:+UseParallelGC): same as SerialGC except that it spawns N (number of CPU cores) threads for Minor GC. N can also be configured using –XX:ParallelGCThreads=n JVM option.
* ParallelOldGC (-XX:+UseParallelOldGC): same as ParallelGC where multiple threads will be spawn for both Minor and Major GC.
* UseConcMarkSweepGc(-XX:+UseConcMarkSweepGC)/CMS: CMS does garbage collection for old generation objects. CMS collector tries to minimize the pauses by doing most of garbage collection work concurrently within application threads. CMS on young generation is same as that of parallel collector. This collector is most suitable for responsive applications (with shorter pause times). Can limit the number of threads using –XX:ParallelCMSThreads=n JVM option.
* G1 Garbage Collector (-XX:+UseG1GC): primarily to replace CMS collector. It just divides heap space into multiple equal-sized heap regions. The collector will collects the region with lesser live data when invoked.

### References in Garbage collection

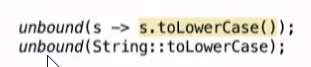
* Strong reference – default level for new objects. Any object with at least one string reference is not eligible for GC.
* Soft reference – soft reference objects can be created by wrapping the class with SoftReference using Generics. Objects with soft reference will be GC only if JVM is about to run out of memory and on the brink of throwing OutOfMemoryError. Used in memory sensitive caches.
* Weak reference – weak reference objects can be created by wrapping the class with WeakReference using Generics. Objects with weak reference will be GC whether memory is tight or not. Weak references are intended for use in Canonicalized mapping (?).
* Phantom reference - phantom reference objects can be created by wrapping the class with PhantomReference using Generics. Phantom reference is created along with reference queue object. It will be immediately eligible for GC. It is actually substitute for finalize() method.
* Soft and weak references are added to queue as soon as GC “marks” these objects. But phantom reference is added to the queue as soon as the memory objects have been finalized.

## Java 8 features

* Lambda expressions are functions which can be expressed as values. Basically, with lambda, we can represent block of code with a variable. Lambdas are type of functional interface. Prior to Java 8, we need to create a Class code or write anonymous inner class to implement the functional interface.
  + Enables functional programming.
  + Eliminates boilerplate code and supports parallel processing.
  + Java uses type inference where compiler identifies the data type of the expression automatically.
* Method references - Java 8 construct to refer functional interface.
  + Static method reference –



* + Unbounded method reference (not bounded to specific object)

* + Bounded method reference (bounded to specific object)





* + Constructor method reference





* + Array method reference



* Optional - It encapsulates optional value and is present to request developers to handle null values explicitly.
* Functional Interface - Any interface that contains one abstract method (may contain one or more default or static methods).
* Default methods
* Nashorn, JS engine
* Stream API – sequence of elements supporting sequential and parallel aggregate operations.
* Date API

## Storing password in char array instead of String

* Since String is immutable, storing clear text password in String will be available in memory until garbage collector clears it.
* Anyone who has access to memory dump may access the password easily.

## Thread-Safe

* As per definition from Java Concurrency in Practice, a class is thread safe if it behaves correctly when accessed from multiple threads, regardless of the scheduling or interleaving of the execution of those threads by the runtime environment, and with no additional synchronization or other coordination on the part of the calling code.

## Synchronized and locks

* Synchronized block should be present within a method. Lock and unlock methods can be called in different methods.
* We can achieve fairness by specifying the fair property within Lock API so that longest waiting thread is given access to lock. With synchronized block, any thread can get the access.
* In synchronized, thread moves to blocked state if access is blocked. Using tryLock() method, we can have thread acquire the lock only if its available.
* A thread which is in waiting state for the access to synchronized block cannot be interrupted. Lock API provide lockInterruptibly() method where the waiting thread can be interrupted.

## Thread & Concurrency

### Process and Thread

* Process is an independent piece of software which runs in its own memory. A process may contain multiple threads. Process is synonymous to application or program. Every process at least has a thread.
* Thread (lightweight process) is part of an application which shares a common memory.

### Difference between wait, sleep, yield and join methods

Sleep and yield methods are static and it will always operate on current thread.

* Sleep – thread will go into sleep but will not lose ownership. It will run into Runnable state after the sleep.
* Wait – thread will go into wait mode until another thread invokes notify method for this object. Thread releases ownership when it goes into wait mode. This method should be called within synchronized block and wait is mainly used for inter-thread communication. Thread needs to acquire the lock after the wait (since it lost the ownership) and move to Runnable state.
* Yield – can pause the current thread temporarily for a chance for other threads with same priority to execute. If there are no waiting threads or other threads have low priority, current thread will continue to execute.
* Why wait and notify should be called from synchronized block? IllegalMoniterStateException will be thrown if not, because thread may not have own the specified object monitor and attempting to wait or notify the waiting thread without owning the monitor will not work. Also, to keep things atomic ([help](https://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html))

### Create deadlock in java

* Deadlock occurs when 2 or more threads wait for each other to release the lock of the resource and get stuck by infinite time. ([sample](https://github.com/pradpk/tryouts/tree/master/java/deadlock))

### Difference between Callable and Runnable

* Runnable does not return a result and cannot throw checked exceptions whereas Callable will return the result and can throw an Exception

### Difference between Executor submit and execute methods

* Executor submit returns Future object while execute method doesn’t return.

### Enum vs Constants

* Enum constants are static and final by default. Enum actually represent fixed set of constants like months in a year etc. Before enum, we had to declare these by constants.
* Enums enable type safety which is helpful to constrain the arguments which must take small set of values. Consider a method which provides the error message has an argument of integer value (error code), it needs to check with all set of valid values from constants (multiple if else), and using enum here instead makes sure that calling code is sending the valid value.
* Enum can be used as singleton classes as it ensures thread safety and single instance. But it doesn’t allow lazy initialization.

### Volatile and transient variables

* When a field is declared as volatile, compiler and runtime are put on notice that this variable is shared. They will not be cached in registers or caches. Read of a volatile variable always returns the most recent write by any thread.
* Accessing the volatile performs no locking or thread blocking, which is why it is light weight synchronization mechanism. But volatile reads are costlier than nonvolatile.
* Locking can guarantee both visibility and atomicity, while volatile guarantees only visibility.
* Transient is used during serialization, whenever we don’t want to save a value during serialization, we can declare the variable as transient.

## Web Services

* Top Down Approach – WSDL document is generated based on the design and passed on to the client. Client and server can implement the service in parallel.
* Bottom Up Approach – Service implementation is done first, then WSDL is generated based on the code.
* We can generate Stubs from WSDL using commonly JAX-WS libraries like Apache Axis2, Apache CXF with tools like Eclipse.

## SQL

* CHAR strictly allows only strings with the length mentioned, whereas VARCHAR allows any strings within the length mentioned.
* DDL refers to queries like CREATE, ALTER and DROP which define data, DML refers to queries like insert, delete, retrieve and update which access or manipulate data.
* Primary key can be only one and cannot be null, while there can be multiple unique constraints and can carry null values.
* View are virtual tables selected from one or more tables by having certain conditions.
* SQL is declarative, while PLSQL is procedural. SQL is used to write DDL and DML statements, while PLSQL is used to program blocks, functions, triggers and procedures.
* Database index is a data structure that improves the speed of retrieval of data from the database at the cost of additional writes and storage space to maintain the index data structure. Indexes can be created using one or more columns of a database table.

### Types of Joins

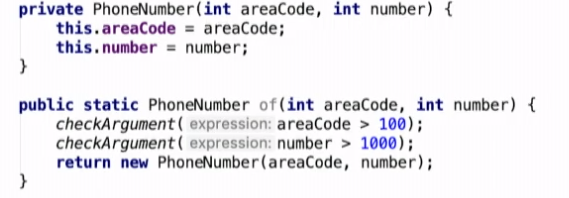
* INNER Join – default join, it selects all rows from both the tables.
* LEFT join – also called as LEFT OUTER JOIN, selects all rows from the table on the left side of the join and matching rows from the right side of the join. Unmatched rows from the right side of the join will carry NULL values.
* RIGHT join – also called as RIGHT OUTER JOIN, selects all rows from the table on the right side of the join and matching rows from the left side of the join. Unmatched rows from the left side of the join will carry NULL values.
* FULL join – creates the result set by combining both LEFT JOIN and RIGHT JOIN, unmatched rows on the left and right side of the join will carry NULL values.

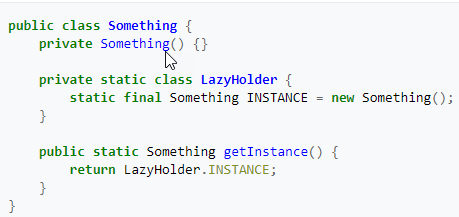
## Design Patterns

* Helps in providing the correct level of abstraction for loose coupling. It also capture solutions which have been successfully applied to problems. It provides a platform for software developers to move towards efficient applications.

### Creational Patterns

* These are patterns talking about instantiation process of objects.
* Abstract Factory
* Builder
* Factory Method



* + Advantages
    - Capability to provide a meaningful name.
    - Internally can use flyweight design pattern to return cached instances.
    - Can create different object flavors with logic inside factory methods.
* Prototype
  + Concept is to copy an existing object rather than creating a new instance from scratch. The existing object acts as a prototype and contains the state of the object. The newly copied object (using clone method) may change some properties if required.
  + In Spring, prototype scope will create new (like Java’s new constructor) object rather than copying existing object.
* Singleton
  + Initialization on demand holder –
    - It’s an idiom for lazy loaded singleton which enables safe, highly concurrent lazy initialization with good performance.
    - 
  + When class Something is loaded by JVM, class goes through initialization. Since the class doesn’t have any static variables, initialization completes trivially. Static class LazyHolder is not initialized until JVM determines that LazyHolder must be executed (i.e when static method getInstance is called).
  + Java Language Specification guarantees that class initialization phase is sequential (non-concurrent), no synchronization is required.

### Structural Patterns

* Concerned with how classes and objects are composed to form larger structures.
* Adapter
* Bridge
* Composite
* Decorator
* Façade
* Flyweight
  + Used to reduce memory footprint for improving application performance. It recycles created objects by caching them and every time an object is requested, existing object is returned if present.
  + Flyweight objects are Immutable. Java libraries used flyweight pattern where Integer object from -128 till 127 is cached in a static block when Integer is referenced first time by JVM. Anytime, when our program calls valueOf(int i), it will return the cached object for the range (that’s why it is recommended to create Integer object from valueOf instead of new Integer)
* Proxy

### Behavior Patterns

* Concerned with algorithms and assignment of responsibilities between objects.
* Chain of Responsibility
* Command
* Interpreter
* Iterator
  + Intent is to provide a way to access the elements of aggregate object sequentially without exposing its underlying representation.
* Mediator
* Memento
* Observer
* State
* Strategy
* Template Method
* Visitor
* Front Controller Pattern

### Microservice Design Patterns

* Event Sourcing
  + Event sourcing is a state of data not stored in a database row, instead represented by a stream of events. Event sourcing is for faster writes, but not good for reads as you have to traverse from initial data state to its current state to get the value.
  + As per old norm, we design as domain model e.g., Customer having attributes like name, age etc. Basically, we think about structure of information system before the behavior. For event sourcing, there is a shift in thinking where we think of events or phases in the system and the structure of the system.
* CQS (Command Query Separation)
  + CQS means system should have different abstraction for writing and reading operations. Writing operation should modify the state, whereas reading operation should read the current state, but should not change the state. (Asking question should not change the answer)
* CQRS (Command Query Responsibility Segregation)
  + Typically CRUD (Create, read, Update and Delete) have same entity for read and write. Simpler form of CQRS will have entity object for write and data transfer object for read which will mimic the screen. So, we will have 2 different domains for read and write operation. Extending in grand scale, we even can have 2 different technologies or 2 different data stores for read and write.
  + Having 2 different data stores as CQRS will bring efficiency in performance as the read and write data store can be optimized to its capabilities, one simple example will be like aggregation can be done within the database instead of the application memory during read.
  + With latest micro service architecture, we can get more advantages like distribution, availability, integration and analytics. Having multiple read and write will ensure consistent availability with replica data and also security where we can have separate data store with public data instead of same database having public and sensitive data.
  + Event sourcing comes in here when you want to update multiple data store from write data store (also can be called as canonical). We can pass them as events and there are multiple techniques to pass these events based on requirements or factors like latency (how much delay can be accepted from write to read), size (how much dataset can be passed), staleness (how much data is added to write), ownership (who owns write vs read) and security.
  + Some of the technologies used for passing the data are
    - In memory data grid – basically, they are used for caching, session storage. In memory data grid has continuous queries which can be indexed to have a call back to generate events.
    - Distributed processing – The logic here is to push the code near to data clusters and process it and stream the data. It can be thought as instead of transporting the data to the code, code is transported to the data.
    - Data virtualization – We can have virtual views of data store which will be enabled with cache.

## Algorithms used by Java

|  |  |  |
| --- | --- | --- |
| Operation | Algorithm | Performance |
| Arrays.sort() method for int, float, long, double | Dual-Pivot Quicksort | O(nlog(n)) |
| Arrays.sort() method for objects | Adaptive, iterative Mergesort | O(nlog(n)) |
| TreeMap | Red Black Tree Algorithm | O(log(n)) for containsKey, get, put and remove |
| ArrayList |  | Size, isEmpty, get, set, iterator and listIterator run in constant time.  Adding n elements in O(n) time. |
| HashMap | Binned hash table transformed to TreeNodes when bins get too large |  |

## Batch & Stream Processing

* Batch processing is processing blocks of data which has been stored over a period of time. It works well when you don’t want real time analytics result, but used for getting detailed insights. Apache Hadoop MapReduce, Spring Batch, Apache Spark are good frameworks for batch processing.
* Stream processing is useful when you want real time analytics result. There are open source stream processing platforms like Apache Kafka, Apache Flink, Apache Storm, Apache Samza, WSO2 stream etc.

## Java NIO

* Java already have file access related classes in java.io packages (File, InputStream etc). Java NIO also provides file access related functionalities. But there are basic differences between IO and NIO
* IO is stream oriented and NIO is buffer oriented. IO is blocking and NIO is non-blocking IO.
* Stream oriented means we read one or more bytes from a stream. We cannot move back and forth in the data in a stream, if we have to move, we need to cache it in a buffer first.
* Buffer oriented means data is read into a buffer which will be processed. We can move back and forth and adds flexibility while processing. But we have to be careful with whether all data has been copied to buffer or data is not overwritten while reading more data.

## Pass by Value

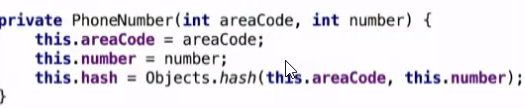
* Java always passes the arguments by value. When you send primitive data as an argument to a method, any changes in the value of the parameter will exist only within the scope of the method. When the method returns, any changes to them are lost.
* When you send an object to a method, JVM passes the object reference to the method. Values of the object’s fields can be changed if they have proper access level and it will still be reflected outside of the method. If you create a new reference for the object within the method, object reference will not be changed outside the method.

## HashMap

* Hashmap is a binned (bucketed) hash table. We create HashMap with initial capacity and load factor. When the load factor reaches 75% (12), the size of the hashmap is doubled by recomputing its hashcode of existing data structure elements.
* It is similar to Hashtable except that HashMap is unsynchronized and permits null key (once) and (multiple) null values. Hashtable will allow non-null object as key or value. Hashmap actually stores key and value in a nested object called Entry within the buckets or bins.
* Hashmap has 2 methods mainly - put and get. Get method will calculate the hash of key to find the bucket location. If there is only one Entry object in the bucket, value from Entry object will be returned. If there are multiple Entry objects, key.equals method will be compared with Entry object’s key and appropriate object will be returned. Put method actually links the Entry object to one another within the bucket using LinkedList (which got changed in Java 8). [Link](http://www.java67.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html)
* Java 8 uses tree nodes instead of LinkedList to optimize within the bucket after a certain threshold is reached to improve the performance boost from O(n) to O(log n). Hashtable, WeakHashMap and IdentityHashMap will use only linked list within the buckets.
* Keys in the hashmap should be [immutable](#_Immutability) else we can change the value of the key in HashMap and it could be unreliable.
* Types: ConcurrentHashMap, ConcurrentSkipListHashMap, EnumMap, Hashtable, IdentityHashMap, LinkedHashMap, TreeMap, WeakHashMap.
* ConcurrentHashMap is a version of synchronized HashMap and recommended than Hashtable or Collections.synchronizedMap(HashMap). ConcurrentHashMap get and put are not synchronized and synchronizes only necessary portion which provides better performance. Hashtable provides synchronized get and put methods.
* EnumMap for collections of enum types. All of the keys in EnumMap should come from single enum type. IdentityHashMap uses reference equality (k1==k2) instead of object equality (k1.equals(k2)) while comparing keys. LinkedHashMap uses linked list implementation with hash table.
* TreepMap is sorted (based on natural ordering and provides log(n) for get, put, remove and containsKey operations) Map implementation. ConcurrentSkipListMap provides concurrent implementation with SkipList type of ordering. It can be used for faster in-order traversal (but TreeMap is recommended for overall sorted map operations).

## Effective Java

* While creating the object, we can check the constraints during object creation instead of using it so that it will save time and avoid object to reach inconsistent state.
* While adding equals and hashcode method, they both need to have the same set of parameters in the same order. It would be a better idea to add the hash when the object is created specifically for immutable classes which can go as a key in HashMap.



# Spring

* Inversion of Control – principle of software engineering where the control of objects is transferred to the container or framework. IoC enables framework to take control of the flow and call our custom code. Advantages are decoupling the execution of task from its implementation and modularity of the program. They can be achieved using Strategy design pattern, Service Locator pattern, Factory pattern and Dependency Injection.
* Dependency Injection – process where objects define their dependencies only through constructor arguments, arguments to a factory method or properties that are set on object instance. The container injects those dependencies when it creates the bean.
* How Spring works
  + Spring MVC is designed around [front controller pattern](#_Behavior_Patterns) where DispatcherServlet provides shared algorithm for request processing. DispatcherServlet uses Spring configuration to discover the delegate components for request mapping, view resolution and exception handling.
  + DispatcherServlet uses HandlerMapping class to map the request to handler objects. HandlerMapping also uses list of interceptors for pre and post processing. Examples of HandlerMapping are RequestMappingHandlerMapping which supports @RequestMethod annotated methods and SimpleUrlHandlerMapping maintains explicitly URI patterns to handler.
  + HandlerAdapter helps DispatcherServlet to invoke the handler mapped to the request.
  + HandlerExceptionResolver will help in resolving the exceptions by matching them to handlers or views.
  + ViewResolver resolves the logical view names from the handler to actual view. MultipartResolver abstracts the parsing the multi-part request.
* Difference between BeanFactory and ApplicationContext
  + BeanFactory interface provides advanced configuration mechanism capable of managing any type of object.
  + ApplicationContext is a sub interface of BeanFactory which adds easier integration with AOP, even publication, application layer context (such as WebApplicationContext) and message resource handling (for use in internationalization).
* Scopes in spring:
  + Singleton (default) – one per Ioc container
  + Prototype – creates a bean instance every time a request is made.
  + Request – IoC container creates a new object for every HTTP request
  + Session - IoC container manages the object at HTTP session level.
  + Global-session – same as session scope but applicable in portlet web applications.
* Spring annotations
  + @Bean - @Bean annotation is used over a method which Spring registers as bean in applicationContext.
  + @Component – Classes annotated with @Component are considered to be auto-detected by Spring when using annotation based configuration and classpath scanning.
  + @Repository – This can be used to annotate class created for encapsulating storage, retrieval, search behavior like DAO (Data Access Object) layer classes. It is a stereotype and it is eligible for Spring DataAccessException translation.
  + @Service – Can be used to annotate class which offers an operation as interface. It is a stereotype and can be used for Facade classes.
  + @Controller – These are used in combination with annotated handler methods based on RequestMapping annotation.
  + @RestController – This annotated class is processed when an appropriate handler mapping is configured with handler adapter class.
  + @SpringBootApplication – used in the base class of a Spring Boot application. It enables component scanning and Spring boot auto configuration.
  + @ServletComponentScan
  + @ComponentScan
  + @ImportResource
  + @Primary – indicates that a bean should be given preference when multiple candidates are qualified to auto wire a single valued dependency. Example is Interface Repo is implemented by ARepo and BRepo. When class X uses reference of Repo, if ARepo is declared as Primary, it will be auto wired to class X. If we want to have BRepo instead of ARepo, we can use @Qualifier annotation during auto wiring.
  + @Conditional – Bean will be created if the condition class mentioned in the annotation class passes. Class mentioned in the annotation has to implement Condition interface which has matches method and returns Boolean.
  + @Configuration – this annotation indicates to Spring that this is a configuration class that will provide beans. Methods in theis class are annotated with @Bean annotation.
  + @ContrrolerAdvice – Specialization of @Component class that have methods annotated with @ExceptionHandler, @initBody or @ModelAttribute/
  + @ExcepttionHandler – Annotation to handle exceptions in handler classes or methods. It can have arguments in arbitrary order – exception argument, request/response object, session object, input/outputstream etc.

## Spring Boot

* + Spring Boot makes it easy to build a stand-alone, production grade Spring based applications.
  + Spring Boot is started by annotating the base class as @SpringBootApplication and calling SpringApplication.run (with the base class reference). Spring Boot will searches the class annotated with @Configuration, initializes all the beans, stores them in IoC container within JVM. After the beans are created, it automatically configures dispatcher servlet and registers default handler mappings, message converts etc.
  + Advantages
    - Automatic Configuration – Spring Boot auto-configuration automatically configures the application based on the jar dependencies that we have added. These auto-configurations will ease developers from explicitly writing the configuration and focus on application functionality.
      * Examples :
        + If Spring’s jdbc depenency is in the classpath and if there is a DataSource bean, Spring Boot will automatically configure JdbcTemplate bean.
        + If Spring security is on the classpath, then configure a default web security setup.
      * How auto-configuration works?
        + Spring boot has a JAR named spring-boot-autoconfigure which contains several configuration classes, which is available on the classpath and contribute to the configuration.
        + Spring-boot-autoconfigure uses Spring 4’s conditional configuration which allows these configurations to be available to application until certain conditions are met.
        + We can override these conditions by providing our own implementation for Condition interface.
    - Starter Dependencies –
      * Without starter dependencies, we need to add search all needed dependencies with the right version, groupId and artifactId.
      * Spring boot offers several “starter” dependencies to address project dependency complexity. A starter dependency is a maven POM that defines transitive dependencies on other libraries which together provide a functionality. Some of the starter dependencies used are spring-boot-starter-web, spring-boot-starter-jdbc and we normally use spring-boot-starter-parent for a Spring Boot application.
    - Actuators – Provides the ability to inspect the internals of the application like beans, environment variables, system properties, trace of HTTP requests etc.
  + Servlet within Spring boot – Add @ServletComponentScan annotation with servlet package path in main SpringBootApplication class.
  + Filters within Spring boot – Add @WebFilter annotation in the filter class.

## Spring Data

* Java developers can work with relational database with JDBC or JPA. Spring supports JDBC using JDBCTemplate class using which developers can run SQL against a relational database. Spring JDBC comes up with spring-boot-starter-jdbc dependency.
* Spring JPA comes with spring-boot-starter-data-jpa dependency, which has Hibernate. If we want to have any other dependency, we need to exclude Hibernate.
* Spring Data provides automatic readymade repositories that can be used with custom repositories. Example, CrudRepository (Create, Read, Update, Delete) has readymade methods that can be used directly. Our custom repository interface will extend CrudRepository and we don’t need to write any implementations.
* @Transactional annotation defines the scope of a single database transaction. The database transaction happens inside the scope of a persistence context. The persistence context in JPA is EntityManager implemented internally using Hibernate session. The persistence context is a synchronizer object that tracks the limited set of Java object and ensures that updates of the object are persisted to database.

# Microservices

* Microservices is an architectural approach to develop single purpose loosely coupled services (by decomposing applications) bounded by some context (could be domain) managed by cross functional team, for delivering complex software systems with the velocity and quality required by today’s digital business.
* Characteristics
  + Should handle fault tolerant
  + Should be independent in terms of technology (polyglot – able to use multiple languages)
  + Should be automated (CI/CD pipeline)
  + Should communicate sync (REST) or async (publish subscribe using queue)
* Strategies for decompositions
  + Verb or use case
  + Noun or functional
  + Bounded context
  + Single responsibility principle – every service should be narrowly responsible for the single part of the functionality
* Service discovery
* Immutable Virtual Machine

# Scrum

* A sprint is a basic unit of Scrum. It is restricted to a specific duration. Each sprint starts with sprint planning, proceeds with daily stand up, ends with sprint review and retrospective.
* Sprint Planning
  + At the beginning of the sprint, scrum master, scrum team and product owner mutually discuss and agree on the scope of work. Scrum master will set the story point that is expected from the team. This is based on the team’s delivery performance with recent sprints.
  + The recommended time frame for spring planning is 4 hours for 2 weeks sprint.
  + Team will go through the backlog items (stories) based on priority. Discussions will happen between the team on each story and story points will be voted.
  + Once the story points reach the expected point, stories will be assigned to team members and they will be asked to fill in tasks for each story.
  + We used “planning poker” to come up with story points. Story points are estimated based on T-shirt sizes (small, medium and big) and we need to select a number from Fibonacci sequence.
* Daily stand up call/scrum
  + This is not a status call, daily call will happen to discuss on the task they worked on and they are planning to work.
  + Any dependencies or limitations will be pointed out and will be raised to Scrum master or product owner based on the issue.
  + We worked with Business Analyst instead of Product Owners, so, it will be Business Analyst who will correspond our team’s business dependencies.
* Sprint Review
  + Sprint review will happen on the last day of the sprint where a demo will happen to product owners of what was developed in the sprint.
  + Only the stories which are in completed status will be eligible for the demo.
* Spring Retrospective
  + This will happen in the last half of last day. Here, scrum master or scrum coach can work with the team to identify the area of improvements.
  + This activity will be fun as it will be mostly based on games or puzzles. It will also help the team to open up to each other

# Data Structures

## Stacks and Queues

* Stacks and queues are dynamic sets in which the element removed from the set by DELETE operation is prespecified. Basically, stack is LIFO (last in first out) and queue is FIFO (first in first out). Examples are plates stacked for stack, people standing in a bank for queue.
* Insert operation on Stack is called as PUSH, delete operation is called POP,
* Insert operation on a queue is called as ENQUEUE and delete operation is called as DEQUEUE. Queue has a head (first element which got added) and tail.
* Stack allows insertion and deletion as one end, queue allows insertion at one end and deletion at other end. Deque allows insertion and deletion at both ends.

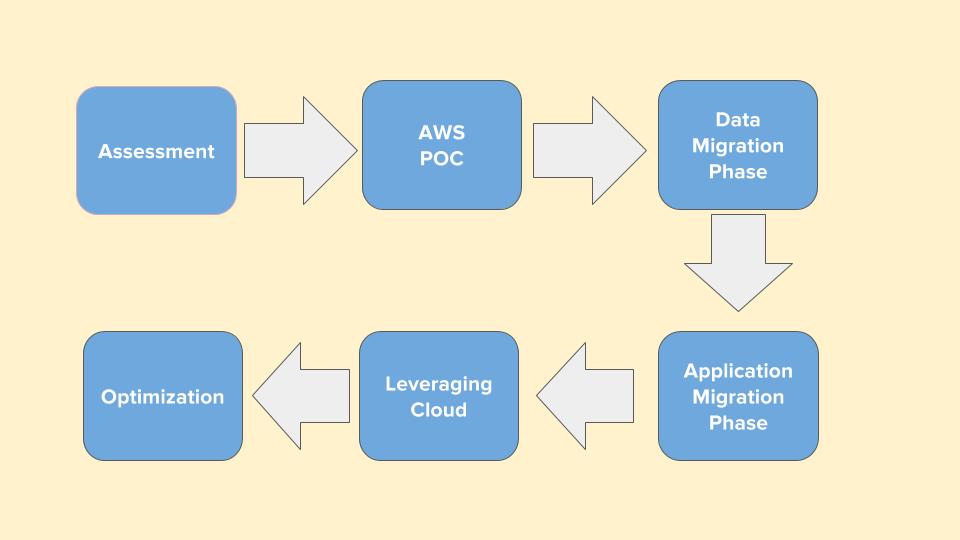
## Graphs

# AWS

### EC2 (Elastic Compute Cloud)

* EC2 is a service that provides resizable compute capacity by AWS. It enables compute capacity in the cloud.
* AWS bills EC2 on 4 ways
  + On Demand - Pay by hour or second.
  + Spot Instances – Bid on computing, for flexible start and end times.
  + Reserved Instances – We can reserve the number of computing instances.
  + Dedicated Hosts – dedicated physical EC2 server will be provided.

### Migration process



* Assessment
  + Doing a financial, security/compliance, technical assessment of the applications. This can be done by enterprise architects or third party vendors who are familiar with AWS.
  + Prioritize the list of applications for the migration based on
  + Services or components having minimum dependencies.
  + Applications with under-utilized assets or flexible architecture.
  + Immediate need to scale applications running out of capacity.
  + Map the application architecture to cloud architecture by exploring aspects of cloud components like compute, storage, database etc.
  + As part of this step, goal is to setup cloud migration roadmap.
* Proof Of Concept
  + Next step is to build a ‘Proof of Concept’ to create a pilot project to learn AWS.
  + Goal of this step is to have visibility on AWS platform and hands-on experience on AWS components.
* Data Migration Strategy
  + Determining list of storage options (RDS/S3/EBS/EFS) in the cloud platform based on availability, cost, performance, relational capability, size of objects, update frequency, read-high vs write-high.
  + Exploring options to use tool such as AWS Schema Conversion Tool which can be installed using “At Your Service” (link).
* Application Migration
  + Application migration can be adopted in 2 strategies.
    - Forklift Migration Strategy
      * Logic is “pick it all up at once” and move it to cloud.
      * Applicable for serverless, self-contained or tightly coupled applications which will be migrated along with the data.
    - Hybrid Migration Strategy
      * Logic is to take some parts of the application and moving it to cloud. Low risk approach as single part can be optimized and deployed to the cloud.
      * This strategy can also be used to integrate cloud applications with other cloud-incompatible applications.
* Using Cloud Platform
  + As the migration is being done, next level is to get to the advantages of cloud platform.
  + Some of the advantages to look at
    - Auto-scaling capability
    - Harden security and adopt recovery strategy
    - With scriptable infrastructure, automatic application upgrade process is recommended to ease the production release activities.
    - Create Business Continuity Plan with availability zone options.
* Optimization
  + After the migration is successful, the focus is on optimize the cloud-based application to reduce cost, increase efficiency.
  + Some of the options are
    - Understanding usage patterns and align/scale the traffic accordingly.
    - Removing idle instances/experiment with instance types.
    - Re-engineer application based on cloud platform.
* Migration Steps
  + Jenkin job will be created and run to
    - Onboard the application by creating service policies (different roles will be created with access to resources) and KMS keys.
    - Create S3 bucket which will hold artifacts for the application.
    - Create SSL certs for the environment and domain of the application (optional).
    - Create and verify AMI using baseAMI with plugins, encrypt it with KMS key and install softwares like cloud-init (for userdata execution),

# Databases

* ACID – Atomicity, Consistency, Isolation and Durability.
  + Atomicity is a notion that all operations in a transaction executes as a single atomic unit.
  + Transactions allowing multiple users to work concurrently with the same data should not compromise the consistency.
  + A particular transaction should not be visible to other concurrently running transactions, they should be in isolation.
  + Changes made in a transaction should be durable (able to withstand pressure) even if the systems fails after the transaction has completed successfully.

# Hibernate

* Hibernate is an ORM tool and based on implementation of JPA.
* Problem statement of Hibernate – Earlier, we have to manually map each entity object variables to a database column and we need to frame SQL for all transactions with values from the objects. We had to manually handle the relationships between tables like primary –foreign key in entity side), mapping data types.
* SessionFactory – will be created once per application. Will be used to create session objects.Session – Will be used for any database operation.
* Hibernate configuration
  + Hbm2ddl.auto – valid values are validate (validates the schema), update (updates the schema if needed), and create (creates the schema destroying previous data), create-drop (drop the schema when SessionFactory is closed explicitly). This configuration automatically validates or updates DDL to the database when SessionFactory is created. When SessionFactory is closed, database schema is closed explicitly.
* Annotations
  + These are JPA annotations that will be considered by Hibernate.
  + @Entity, @Id, @Table, @Column, @Basic (to indicate lazy loading or mandatory properties), @Transient (marking field as transient which will not be persisted), @Temporal (Can be used to persist in Date/Time/Timestamp type), @Lob (can be used to annotate larger objects like CLOB and BLOB), @GeneratedValue (for surrogate key to auto generate sequence)
* Caching – Hibernate provides first level cache by default. First level cache is within the scope of session. But the session is created and maintained only within the transaction. For caching across sessions, Hibernate provides optional second level cache. Second level cache can be used across sessions, applications and clusters using third party system for caching.
* Object states
  + When an entity object is created and yet to be persisted using Hibernate’s persisting mechanism (eg. session.save()), it is called as transient object.
  + When an object is persisted using Hibernate’s persisting mechanism (eg. session.save()), it is called persisted object. Any change made in the persisted object will reflect in the database. Persistent object can move to transient state when you delete the object (like using session.delete()).
  + Persisted object becomes detached object once the session is closed. Detached object can move to persistent state if we open a new session and use session.update() method.
* Difference between save, persist and saveOrUpdate method
  + Save – can be used to persist entity without transaction.
  + Persist – can only be saved
* How Hibernate works with Spring Boot using auto configuration
  + You can add either spring-boot-jdbc (provides JDBCTemplate object) or spring-boot-data-jpa starter dependency. And, need to add DB driver dependency as the application need driver classes. DB driver and dialect configuration should be added in application.properties. Dialect is based upon database and will be used by Hibernate to convert our request to respective SQL.

# Maven

## Different Scopes in Maven

* Compile – default scope, these dependencies are propagated to dependent projects.
* Provided – used during compilation, but during runtime, JDK or container should provide the classes. Examples are Servlet API declared in pom xml.
* Runtime – dependency used only during runtime, not during compilation.
* Test – Used only during test compilation and execution.
* System – Similar to provided, but you have to mention JAR path explicitly.
* Import – used on dependency of type pom, indicates you can refer these dependencies in your pom without mentioning any version.

Javascript

# Python

## Installing pip and packages

* Follow the instructions in python documentation <https://packaging.python.org/tutorials/installing-packages/>
* To get pip, run python get-pip.py –proxy {server:port} (proxy always needed if your connection is behind firewall).
* To get new packages, python –m pip install {package name}

# Tools

## Load Balancer

With load balancer, we get high availability and improved performance. We have hardware (dedicated appliance), cloud and software (we need to install these by ourselves on a server and configure them) load balancer.

* Seesaw (used by Google)
* Nginx – this provides content caching, web server, monitoring, and firewall with load balancing.
* Ribbon – designed and used by Netflix

## Stream Processing

## Containers

## Scheduling

* We can have a bean with @Scheduled annotation which can be configured to run the job. @EnableScheduling annotation needs to be added with boot application class.
* If the application is running in a clustered environment, to ensure the job runs once, we can use ShedLock project and annotate the bean with @SchedulerLock. We can add the condition in database or cache (redis).

## Docker

* Docker allows applications to be isolated into containers with instructions for exactly what they need to survive that can be easily ported from machine to machine.
* Docker disrupts in resource efficiency while compared to traditional hypervisor virtual machines (like VMWare Workstation).
  + Multiple Docker containers can run within a single kernel whereas virtual machines need to have their OS and kernel.
  + Docker containers will be running like a process within OS.