Contents

[Java 2](#_Toc534641853)

[Object Oriented Concepts 2](#_Toc534641854)

[Interface and Abstract Class 2](#_Toc534641855)

[Why String has String Pool 2](#_Toc534641856)

[Immutability 2](#_Toc534641857)

[Java Memory Model 3](#_Toc534641858)

[Heap 3](#_Toc534641859)

[PermGen 3](#_Toc534641860)

[Stack 3](#_Toc534641861)

[Garbage Collection 4](#_Toc534641862)

[Storing password in char array instead of String 4](#_Toc534641863)

[Thread-Safe 4](#_Toc534641864)

[Synchronized and locks 4](#_Toc534641865)

[Thread & Concurrency 5](#_Toc534641866)

[Process and Thread 5](#_Toc534641867)

[Difference between wait, sleep, yield and join methods 5](#_Toc534641868)

[Create deadlock in java 5](#_Toc534641869)

[Difference between Callable and Runnable 5](#_Toc534641870)

[Difference between Executor submit and execute methods 5](#_Toc534641871)

[Volatile and transient variables 5](#_Toc534641872)

[Design Patterns 6](#_Toc534641873)

[Flyweight Pattern 6](#_Toc534641874)

[Java NIO 6](#_Toc534641875)

[Pass by Value 6](#_Toc534641876)

[HashMap 6](#_Toc534641877)

[Maven 7](#_Toc534641878)

[Different Scopes in Maven 7](#_Toc534641879)

[Python 7](#_Toc534641880)

[Installing pip and packages 7](#_Toc534641881)

# Java

## 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: expressing in different forms or behaviors. Subclasses of a class can define their own unique behaviors by overriding the method with their own implementation.
* 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.

## 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 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.

## 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).

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 part 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.

## 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 due to garbage collection 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.

## 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.

### 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.

## Design Patterns

### Flyweight Pattern

* Used to reduce memory footprint for improving application performance. It recycles created objects by storing 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)

## 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.
* 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).

# 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.

# 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}