Singleton Design Pattern

- **Creational Design pattern** which restricts a class to instantiate its multiple objects. It is nothing but a way of defining a class. It is used where only a single instance of a class is required to control the action throughout the execution.
- It should have only one instance and Instance should be globally accessible (public).
- Eg- **java.lang.Runtime** : Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the getRuntime() method. An application cannot instantiate this class so multiple objects can't be created for this class. Hence Runtime is a singleton class.
- Sometimes we need to have only one instance of our class for eg- a single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly. Similarly, there can be a single configuration manager or error manager in an application that handles all problems instead of creating multiple managers.

Fenil Shah

Singleton Design Pattern

- **Hardware interface access**: Hardware printers where the print spooler can be made a singleton to avoid multiple concurrent accesses and creating deadlock.
- **Logger** : If there is multiple client application using this logging utility class they might create multiple instances of this class and it can potentially cause issues during concurrent access to the same logger file. We can use the logger utility class as a singleton and provide a global point of reference so that each user can use this utility and no 2 users access it at the same time.
- **Configuration File**: This has a performance benefit as it prevents multiple users to repeatedly access and read the configuration file or properties file. It creates a single instance of the configuration file which can be accessed by multiple calls concurrently as it will provide static config data loaded into in-memory objects. The application only reads from the configuration file for the first time and thereafter from second call onwards the client applications read the data from in-memory objects.
- **Cache**: We can use the cache as a singleton object as it can have a global point of reference and for all future calls to the cache object, it will use the in-memory object.

Generics

- Generics means parameterized types. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.
- The Object is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature. We will discuss that type of safety feature in later examples. Generics in Java are similar to templates in C++.
- **Code Reuse**: We can write a method/class/interface once and use it for any type we want.
- **Type Safety**: Generics make errors to appear compile time than at run time.Suppose you want to create an ArrayList that store name of students, and if by mistake the programmer adds an integer object instead of a string, the compiler allows it. But, when we retrieve this data from ArrayList, it causes problems at runtime.

Anonymous Inner Class

- It is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain “extras” such as overriding methods of a class or interface, without having to actually subclass a class.
- Can implement only one interface at a time, can extend a class or can implement an interface but not both at a time and we can’t write any constructor.
- Eg- Suppose we need an immediate thread but we don’t want to create a class that extends Thread class all the time. With the help of this type of Anonymous Inner class, we can define a ready thread.

Fenil Shah

Functional Interface

- Contains only one abstract method. They can have only one functionality to exhibit. From Java 8 onwards, lambda expressions can be used to represent the instance of a functional interface. A functional interface can have any number of default methods.
- Functional Interface is additionally recognized as Single Abstract Method Interfaces. In short, they are also known as SAM interfaces. Functional interfaces in Java are the new feature that provides users with the approach of fundamental programming.
- @FunctionalInterface is used to detect compile time error. There is no limit to a functional interface containing static and default methods. Overriding methods from the parent class do not violate the rules of a functional interface.
- By Using Functional Interface, We can use Lambda Expression to instantiate them by eliminating the boiler plate. We can directly call the method without having class/interface reference.
- Eg- Runnable (contains the run() method), Comparable (contains the compareTo() method)

Lambda

- Lambda expressions implements the only abstract function and therefore implement functional interfaces. Java lambda expression is treated as a function, so compiler does not create .class file.
- lambda expression is an object in Java. It is an instance of a functional interface. We have assigned a lambda expression to any variable and pass it like any other object.
- A function that can be created without belonging to any class. A lambda expression can be passed around as if it was an object and executed on demand. Lambda expressions are just like functions and they accept parameters just like functions.
- When there is a single statement curly brackets are not mandatory and the return type of the anonymous function is the same as that of the body expression.
- Lambda expression cannot be used where an interface with more than one abstract method is expected.
- If we are creating lambda for a Functional Interface in a class, that class doesn't need to implement the interface

Streams

- Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result. The central API class is the Stream<T>.
- A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result. It’s not a data structure instead it takes input from the Collections, Arrays or I/O channels.
- Streams don’t change the original data structure, they only provide the result as per the pipelined methods. Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result. There should be exactly one terminal operation in the stream, without a terminal operation the stream wouldn’t flow
- Stream is lazy and evaluates code only when required. The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.
- streams are wrappers around a data source, allowing us to operate with that data source and making bulk processing convenient and fast.

Intermediate Operations:

- **map**: The map method is used to returns a stream consisting of the results of applying the given function to the elements of this stream.

  ```
  List number = Arrays.asList(2,3,4,5);

  List square = number.stream().map(x->x*x).collect(Collectors.toList());
  ```

- **filter**: The filter method is used to select elements as per the Predicate passed as argument.

  ```
  List names = Arrays.asList("Reflection","Collection","Stream");

  List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
  ```

- **sorted**: The sorted method is used to sort the stream.

  ```
  List names = Arrays.asList("Reflection","Collection","Stream");

  List result = names.stream().sorted().collect(Collectors.toList());
  ```

Terminal Operations:

- **collect**: The collect method is used to return the result of the intermediate operations performed on the stream.

```
List number = Arrays.asList(2,3,4,5,3);

Set square = number.stream().map(x->x*x).collect(Collectors.toSet());
```

- **forEach**: The forEach method is used to iterate through every element of the stream.

```
List number = Arrays.asList(2,3,4,5);

number.stream().map(x->x*x).forEach(y->System.out.println(y));
```

- **reduce**: The reduce method is used to reduce the elements of a stream to a single value.

The reduce method takes a Identity(initial value), Accumulator/BinaryOperator (logic of aggregation) & combiner(In parallel mode) as a parameter.

```
List number = Arrays.asList(2,3,4,5);

int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);
```

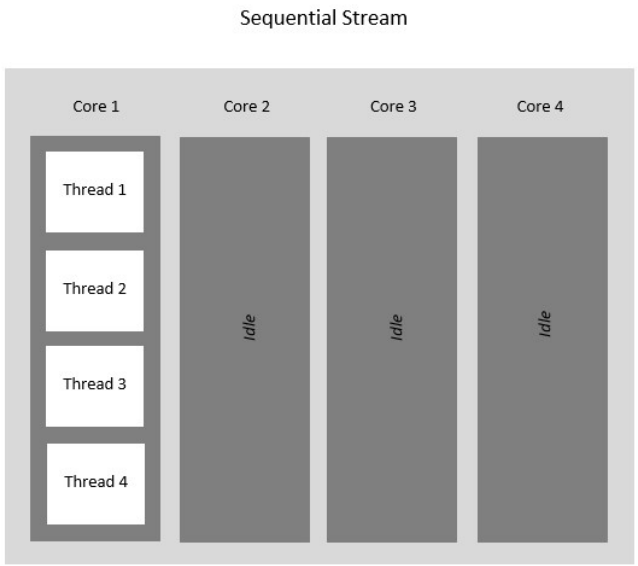
Here ans variable is assigned 0 as the initial value and i is added to it .

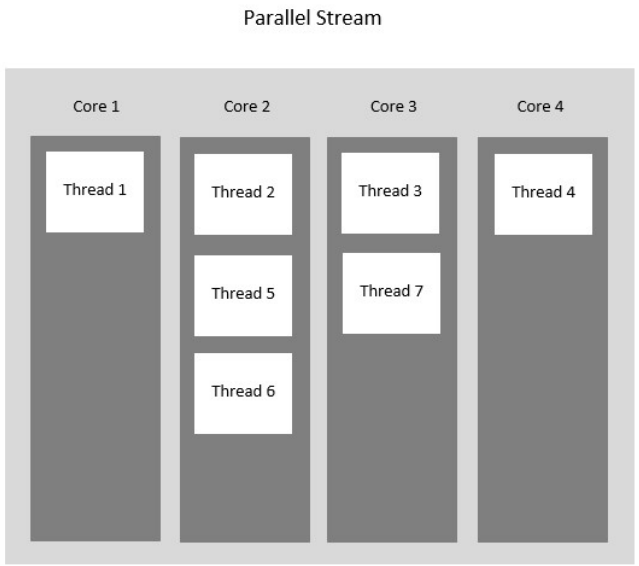
Primitive Streams

- To work with the three most used primitive types – int, long and double – the standard library includes three primitive-specialized implementations: IntStream (sequence of primitive int-valued elements), LongStream, and DoubleStream.
- Boxing and unboxing does take some time, but it's not a lots. A lot of temporary boxed objects also triggers Garbage Collection a lot more often, and that's a performance drain too. It all adds up, so if the stream processes a lot of integer values in a tight "loop", the difference can be relevant.
- The overhead of Integer is quite large. An int is 4 bytes for the value, while an Integer is 4 bytes for the reference plus 16 bytes for the object, so Integer uses 20 bytes per value, i.e. 5 times the memory.
- Methods: Stream.of(1,2,3), Stream.range(1, 10), Arrays.stream(array), stream().mapToInt(i -> i), .average() .getAsDouble(), getMax(), boxed();

Parallel Streams

- Parallel stream leverage multi-core processors, which increases its performance. our code gets divide into multiple streams which can be executed parallelly on separate cores of the system and the final result is shown as the combination of all the individual core's outcomes.
- Order is not maintained as the list.parallelStream() works parallelly on multiple threads.





Process vs Threads

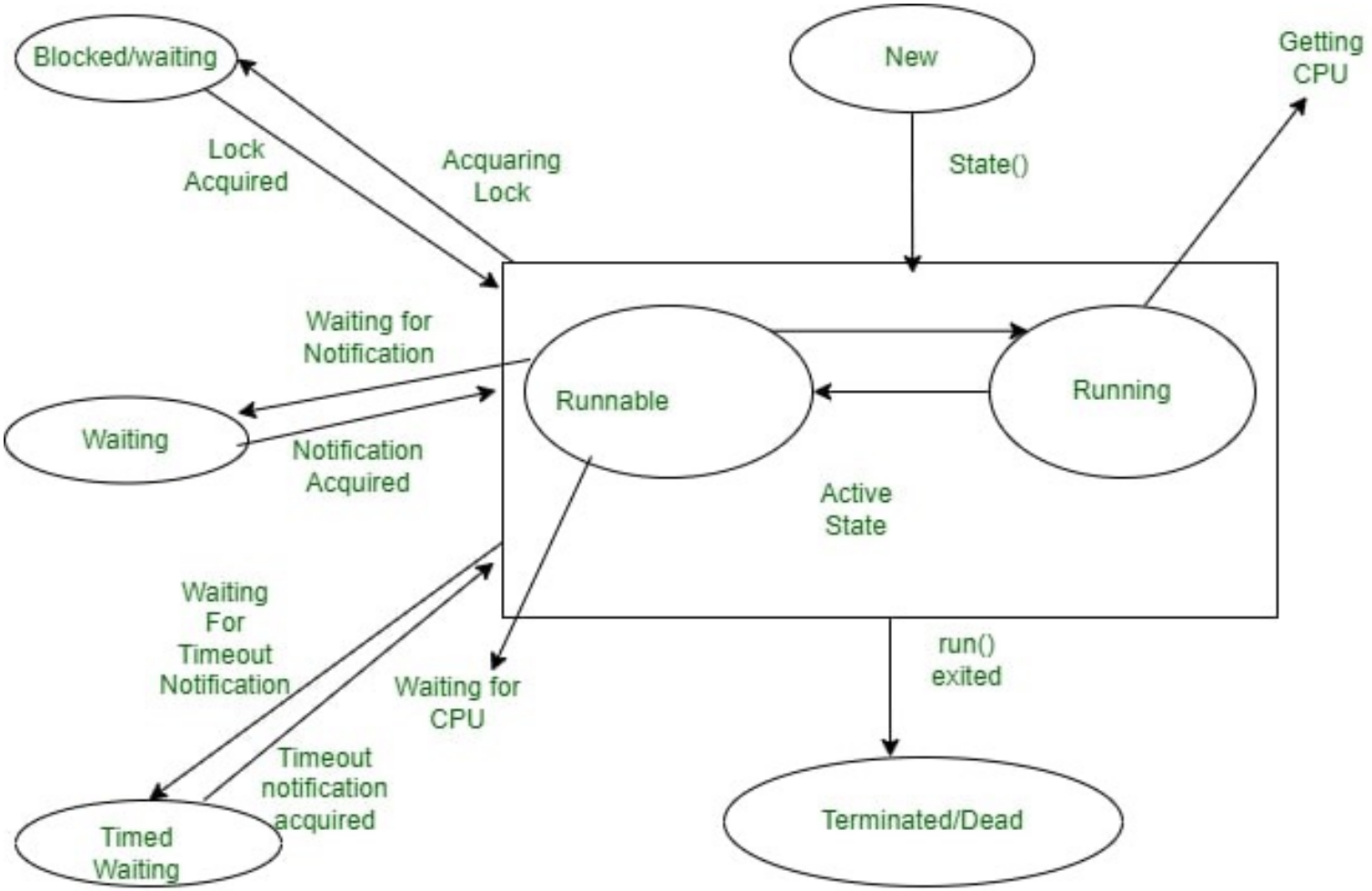
PROCESS

- An executing program is called a process. Process provides environment for execution of Thread.
- Every process has its separate address space.
- Process-based multitasking allows a computer to run two or more than two programs concurrently.
- Communication and Context-Switching between two processes is expensive and limited.
- Process are also called heavyweight task.

THREAD

- A thread is a small part of a process.
- All the threads of a process share the same address space cooperatively as that of a process.
- Thread-based multitasking allows a single program to run two or more threads concurrently.
- Communication and Context-Switching between two threads is less expensive as compared to process.
- Thread are also called lightweight task.

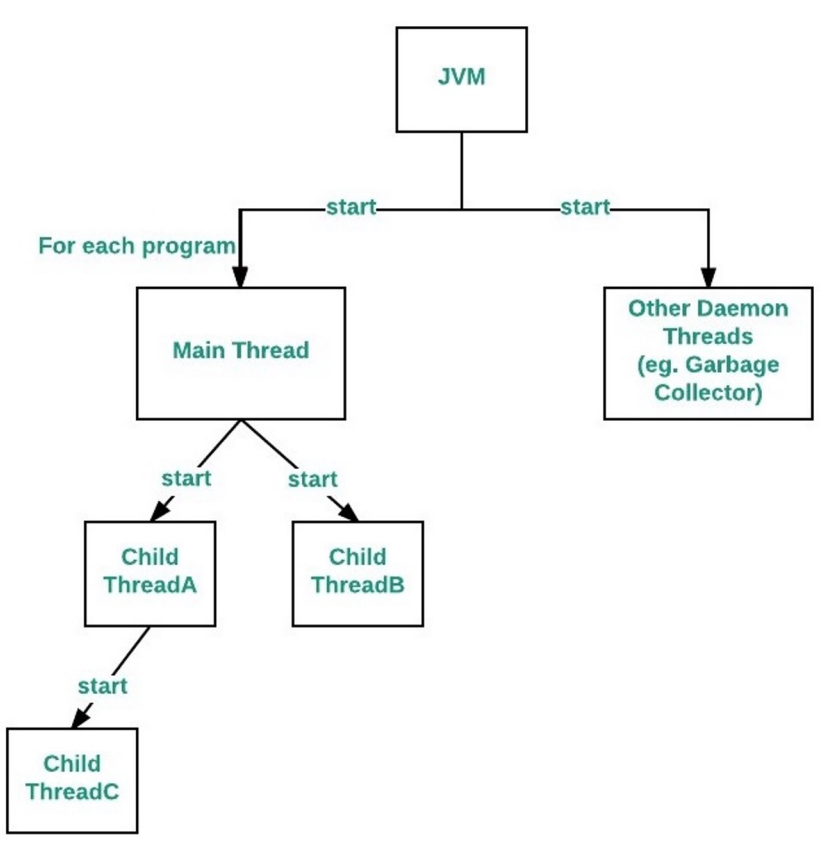
Fenil Shah



Life Cycle
Of Thread

Threads

- **start()**: a new thread is created and then the run() method is executed. Can't be invoked more than one time otherwise throws java.lang.IllegalStateException. Defined in java.lang.Thread class.
- **run()**: No new thread will be created and run() method will be executed as a normal method call on the current calling thread itself and no multi-threading will take place. Multiple invocation is possible. Defined in java.lang.Runnable interface and must be overriden in the implementing class.
- **Main Thread**: Java program starts up, it begins running immediately. Main thread is created by JVM. It is the thread from which other "child" threads will be spawned. Often, it must be the last thread to finish execution because it performs various shutdown actions.



Threads

- The significant differences between extending Thread class and implementing Runnable interface:

When we extend Thread class, we can't extend any other class even we require and When we implement Runnable, we can save a space for our class to extend any other class in future or now.

- When we extend Thread class, each of our thread creates unique object and associate with it. When we implements Runnable, it shares the same object to multiple threads.
- **Daemon** thread in Java is a low-priority thread that runs in the background to perform tasks such as garbage collection. Daemon thread in Java is also a service provider thread that provides services to the user thread. Its life depends on the mercy of user threads i.e. when all the user threads die, JVM terminates this thread automatically. Eg- Garbage collection in Java (gc), finalizer, etc.
- They can not prevent the JVM from exiting when all the user threads finish their execution. JVM terminates itself when all user threads finish their execution. By default, the main thread is always non-daemon but for all the remaining threads, daemon nature will be inherited from parent to child.
- Methods: setDaemon(boolean status), isDaemon(), etc

Multithreading

- Java provides built-in support for multithreaded programming. A multi-threaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.
- we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- You can perform many operations together, so it saves time. Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

currentThread() - returns a reference to the currently executing thread object. **Native** Method -method is implemented in native code using JNI (Java Native Interface). The native modifier indicates that a method is implemented in platform-dependent code, often seen in C language. improve the performance of the system, achieve machine level/memory level communication, use already existing legacy non-java code, we are not responsible for providing an implementation. it breaks the platform-independent nature of java.

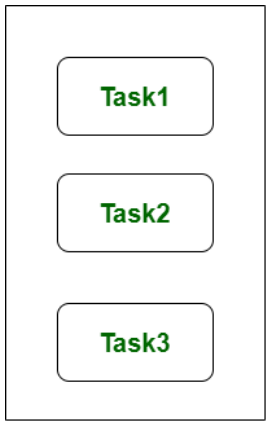
join() method of thread class waits for a thread to die. It is used when you want one thread to wait for completion of another. Execution cant go ahead until the thread on which join is called dies

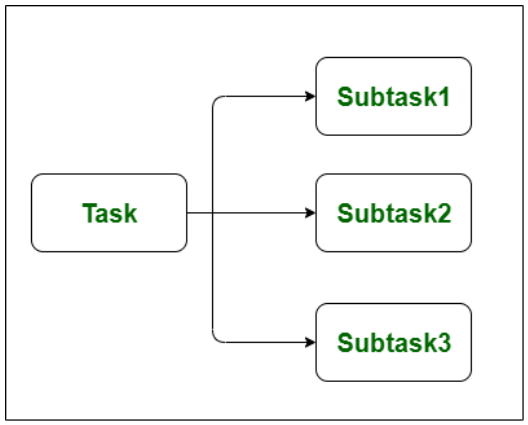
setPriority(): changes the priority of the thread.

Fenil Shah

Concurrency Vs Parallelism

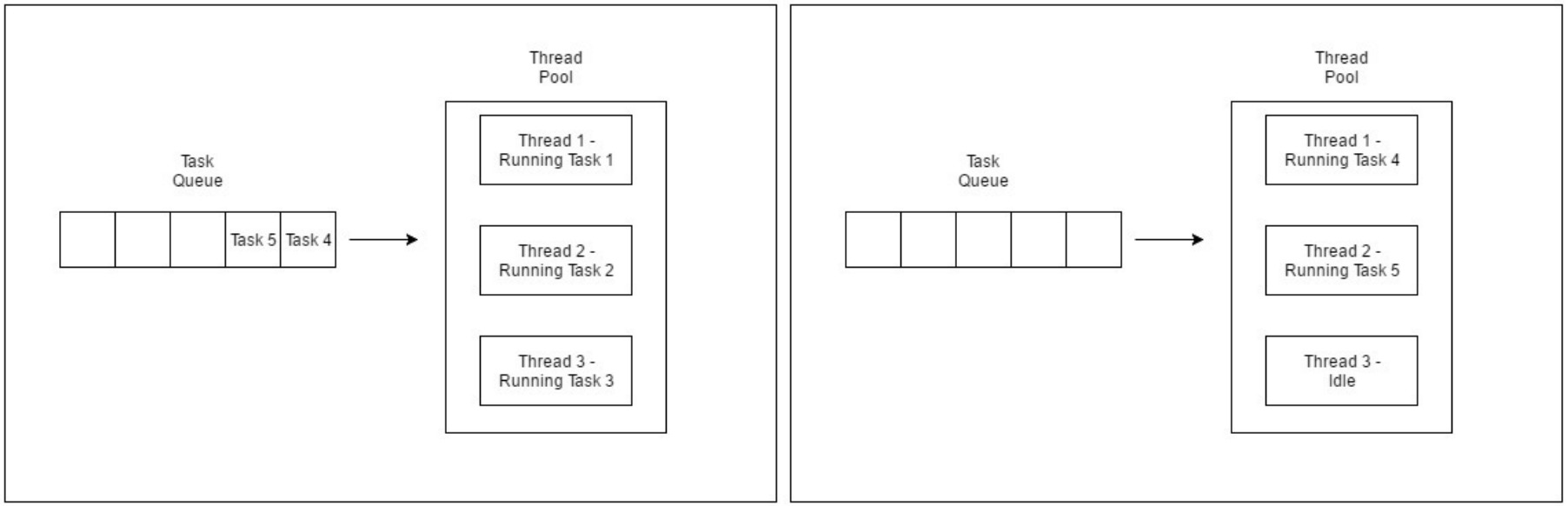
- Concurrency is an approach that is used for decreasing the response time of the system by using the single processing unit. In concurrency the speed is increased by overlapping the input-output activities of one process with CPU process of another process.
- Parallelism is related to an application where tasks are divided into smaller sub-tasks that are processed seemingly simultaneously or parallel. It is used to increase the throughput and computational speed of the system by using multiple processors.Parallelism leads to overlapping of central processing units and input-output tasks in one process with the central processing unit and input-output tasks of another process.





ThreadPool

- Java Thread pool represents a group of worker threads that are waiting for the job and reused many times. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, the thread is contained in the thread pool again.
- Thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing. Since the thread is already existing when the request arrives, the delay introduced by thread creation is eliminated, making the application more responsive.
- Eg- Server Programs such as database and web servers repeatedly execute requests from multiple clients. JVM creating too many threads at the same time can cause the system to run out of memory.
- newFixedThreadPool(int) - Creates a fixed size thread pool.
- newCachedThreadPool() - Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available
- newSingleThreadExecutor()- Creates a single thread.
- Thread Pool has to be ended explicitly at the end. If this is not done, then the program goes on executing and never ends. Call shutdown() on the pool to end the executor.



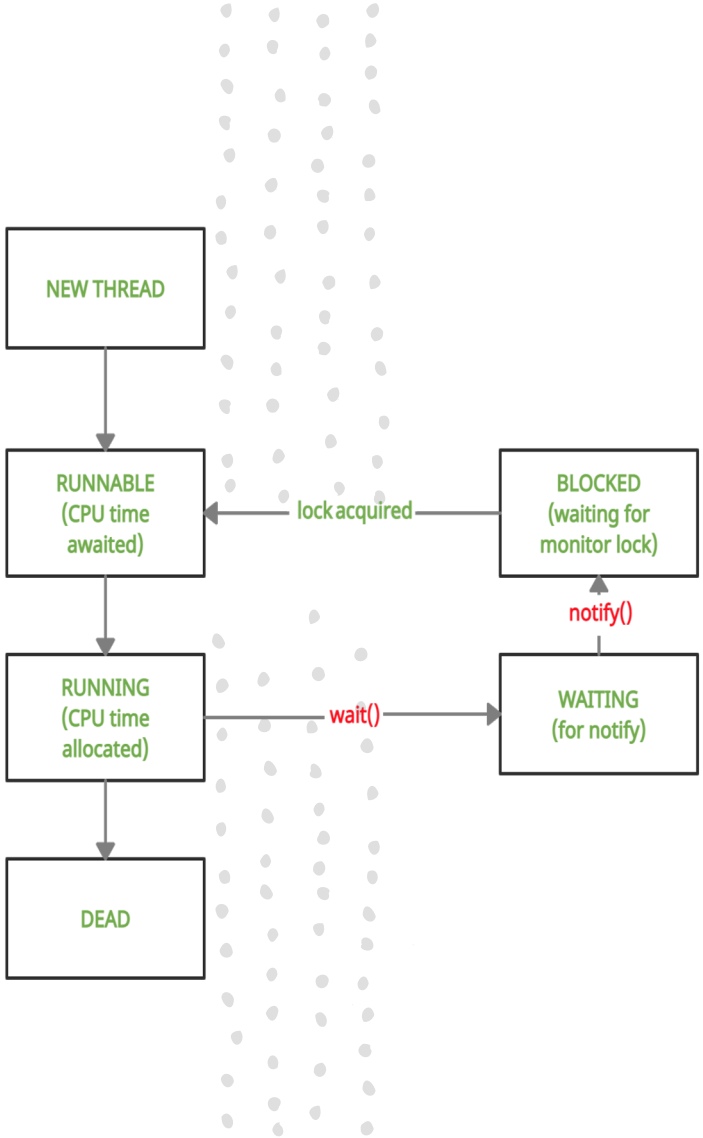
The task 4 or task 5 are executed only when a thread in the pool becomes idle. Until then, the extra tasks are placed in a queue.

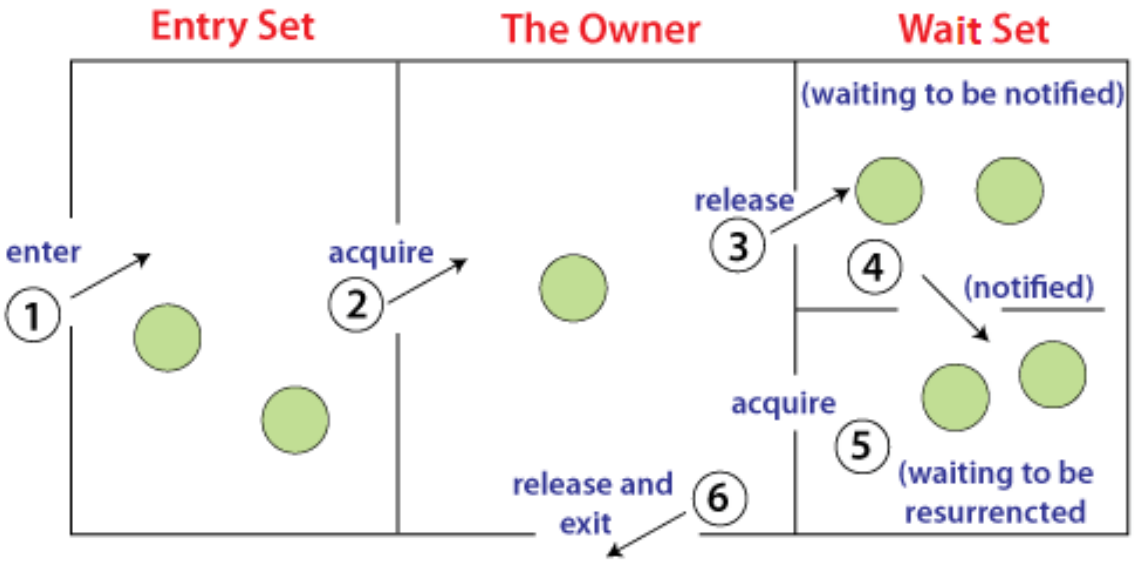
- **Deadlock**: all the executing threads are waiting for the results from the blocked threads waiting in the queue due to the unavailability of threads for execution. Sol - Don't queue tasks that concurrently wait for results from other tasks
- **Thread Leakage** :Thread Leakage occurs if a thread is removed from the pool to execute a task but not returned to it when the task completed. As an example, if the thread throws an exception and pool class does not catch this exception, then the thread will simply exit, reducing the size of the thread pool by one. If this repeats many times, then the pool would eventually become empty and no threads would be available to execute other requests.
- **Resource Thrashing** :If the thread pool size is very large then time is wasted in context switching between threads. Having more threads than the optimal number may cause starvation problem leading to resource thrashing as explained.

Synchronized

- Synchronization in Java is the capability to control the access of multiple threads to any shared resource. To prevent thread interference & prevent consistency problem.
- only one thread can access the resource at a given point in time. This synchronization is implemented in Java with a concept called monitors. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class(because they are related to lock and object has a lock.)

- Wait(): the calling thread stops its execution until notify() or notifyAll() method is invoked by some other Thread. When wait() is called on a thread holding the monitor lock, it surrenders the monitor lock and enters the waiting state. if a thread present in the waiting state gets interrupted, then it will throw InterruptedException.
- Notify(): used to wake up only one thread that's waiting for an object, and that thread then begins execution. The thread class notify() method is used to wake up a single thread. When the notify() is called on a thread holding the monitor lock, it symbolizes that the thread is soon going to surrender the lock. One of the waiting threads is randomly selected and notified about the same. The notified thread then exits the waiting state and enters the blocked state where it waits till the previous thread has given up the lock and this thread has acquired it. Once it acquires the lock, it enters the runnable state where it waits for CPU time and then it starts running.

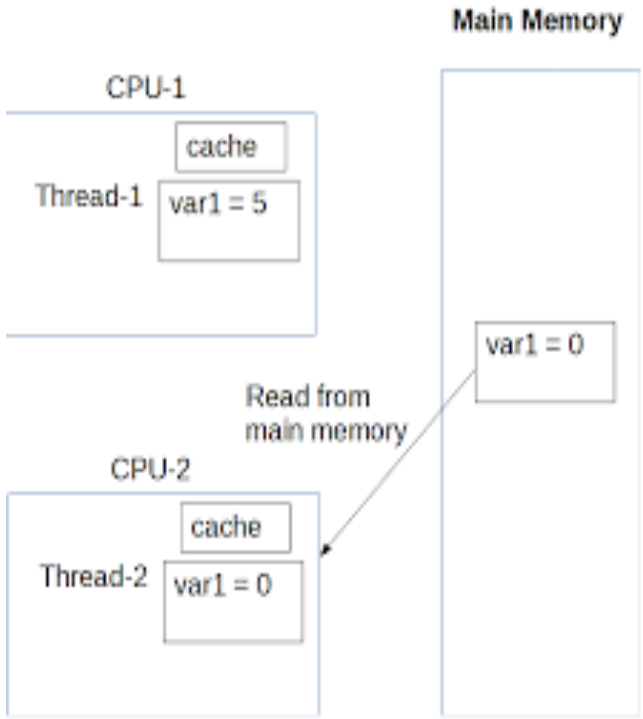




1.Threads enter to acquire lock.
2.Lock is acquired by on thread.
3.Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4.If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5.Now thread is available to acquire lock.
6.After completion of the task, thread releases the lock and exits the monitor state of the object.

Volatile

- Volatile keyword is used to modify the value of a variable by different threads. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem.
- The volatile modifier is used to let the JVM know that a thread accessing the variable must always merge its own private copy of the variable with the master copy in the memory. Accessing a volatile variable synchronizes all the cached copied of the variables in the main memory. The volatile keyword does not cache the value of the variable and always read the variable from the main memory. It ensures visibility but not mutual exclusion.
- Eg- two threads are working on the same class. Both threads run on different processors where each thread has its local copy of var. If any thread modifies its value, the change will not reflect in the original one in the main memory. It leads to data inconsistency because the other thread is not aware of the modified value.
- Alternative way of achieving synchronization in Java. Used to inform the compiler that multiple threads will access a particular statement. It prevents the compiler from doing any reordering or any optimization. Can use the volatile keyword with variables. Improves thread performance.



Maven(Build Automation Tool)

- Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation.
- Maven is a tool that can be used for building and managing any Java-based project. maven make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.
- We can easily and quickly build a project using maven.
- We can add jars and other dependencies of the project easily using the help of maven.
- Maven provides project information (log document, dependency list, unit test reports etc.)
- Maven is very helpful for a project while updating central repository of JARs and other dependencies.
- With the help of Maven we can build any number of projects into output types like the JAR, WAR etc without doing any scripting.
- Using maven we can easily integrate our project with source control system (such as Subversion or Git).

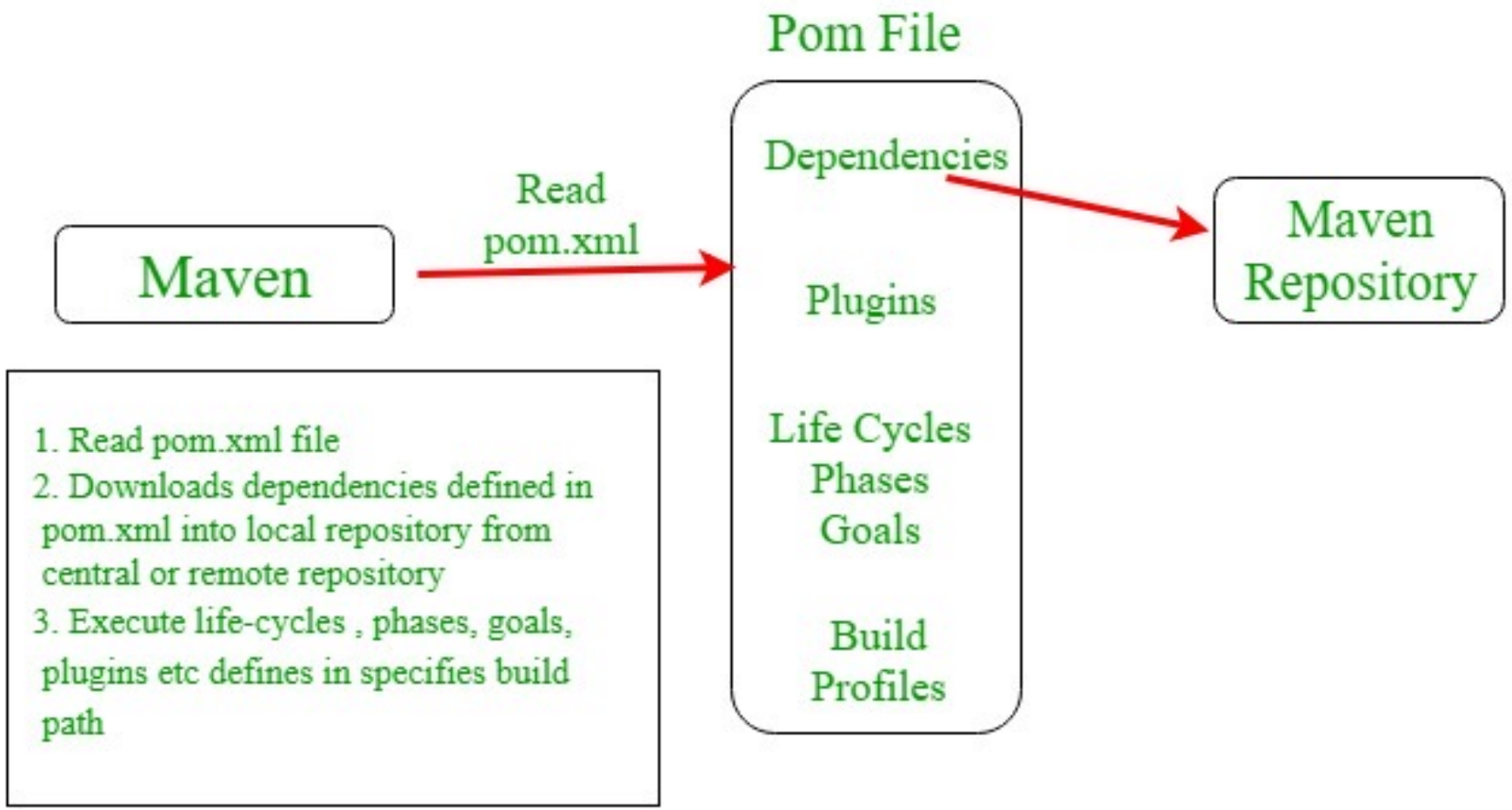
Fenil Shah

Maven(Build Automation Tool)

- Without Maven, We need to externally Add the Jar Files.
- Maven Repo: https://repo.maven.apache.org/maven2/mysql/mysql-connector-java/8.0.29/
- Build path -> configure build path -> libraries -> add external jars
- With Maven: From Archetype (Combination of GroupId + ArtifactId + Version)
- https://mvnrepository.com/ - Search – Add Dependency
- We can see the parent/child dependency by clicking on artifactid of any dependency.
- **Project Inheritance**: If you have several Maven projects, and they all have similar configurations, you can refactor your projects by pulling out those similar configurations and making a parent project. Thus, all you have to do is to let your Maven projects inherit that parent project, and those configurations would then be applied to all of them.
- <scope>: used to limit the transitivity of a dependency, and also to affect the classpath used for various build tasks. can take 6 values: compile, provided, runtime, test, system and import.

  E.g.- test: only available for the test compilation and execution phases.

Showing How maven works



Maven(Build Automation Tool)

- **POM Files**: Project Object Model(POM) Files are XML file that contains information related to the project and configuration information such as dependencies, source directory, plugin, goals etc. used by Maven to build the project. When you should execute a maven command you give maven a POM file to execute the commands. Maven reads pom.xml file to accomplish its configuration and operations.
- **Dependencies and Repositories**: Dependencies are external Java libraries required for Project and repositories are directories of packaged JAR files. The local repository is just a directory on your machine hard drive. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository.
- **Build Life Cycles, Phases and Goals**: A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. Maven command is the name of a build lifecycle, phase or goal. If a lifecycle is requested executed by giving maven command, all build phases in that life cycle are executed also. If a build phase is requested executed, all build phases before it in the defined sequence are executed too.
- **Build Profiles**: Build profiles a set of configuration values which allows you to build your project using different configurations. For example, you may need to build your project for your local computer, for development and test. To enable different builds you can add different build profiles to your POM files using its profiles elements and are triggered in the variety of ways.

Maven Core

- **Build Plugins**: Build plugins are used to perform specific goal. you can add a plugin to the POM file. Maven has some standard plugins you can use, and you can also implement your own in Java.
- **Installation of Maven** includes following Steps:

1. Verify that your system has java installed or not. if not then install java (Link for Java Installation)
2. Check java Environmental variable is set or not. if not then set java environmental variable.(link to install java and setting environmental variable)
3. Download maven (Link)
4. Unpack your maven zip at any place in your system.
5. Add the bin directory of the created directory apache-maven-3.5.3(it depends upon your installation version) to the PATH environment variable and system variable.
6. open cmd and run mvn -v command. If it print following lines of code then installation completed.

Fenil Shah

Maven pom.xml file

- POM means Project Object Model is key to operate Maven. Maven reads pom.xml file to accomplish its configuration and operations. It is an XML file that contains information related to the project and configuration information such as dependencies, source directory, plugin, goals etc. used by Maven to build the project.

1. **project-** It is the root element of the pom.xml file.
2. **modelVersion**- modelversion means what version of the POM model you are using. Use version 4.0.0 for maven 2 and maven 3.
3. **groupId**- groupId means the id for the project group which uniquely identifies your project across all projects. It is unique and Most often you will use a group ID which is similar to the root Java package name of the project like we used the groupId org.apache.loggerapi, com.geeksforgeeks.project, etc. It doesn't checks unique until we push to central repository in public cloud.
4. **artifactId**- artifactId used to give name of the project you are building i.e. the name of the jar without version. in our example name of our project is LoggerApi.
5. **version**- version element contains the version number of the project. If your project has been released in different versions then it is useful to give version of your project.

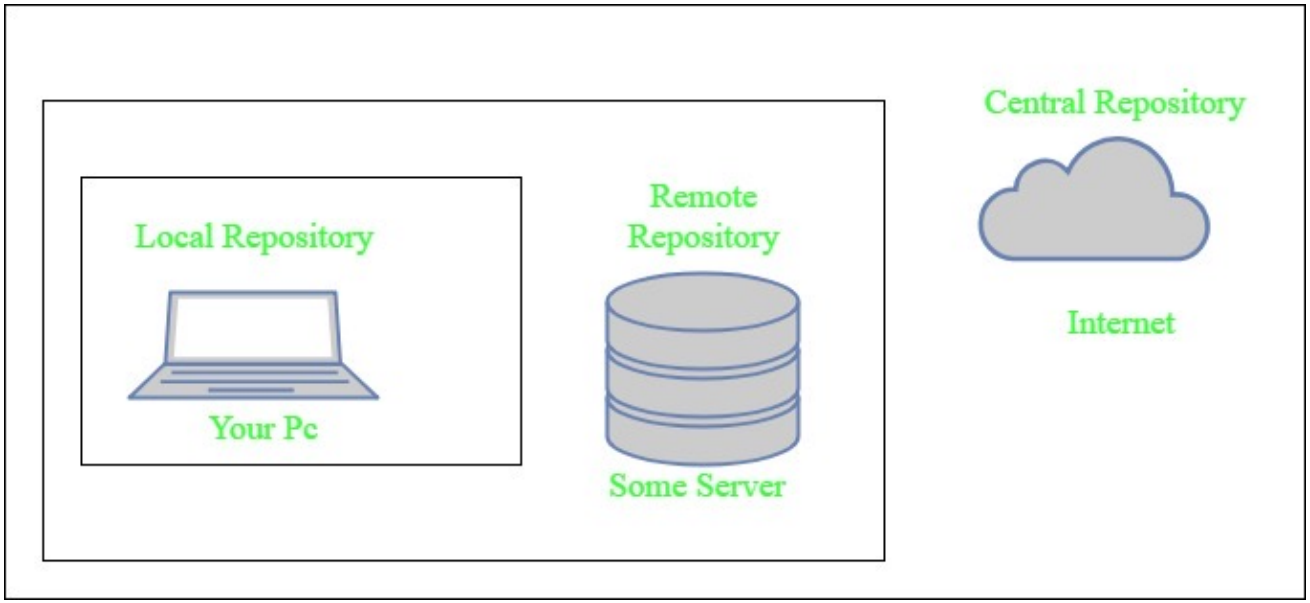
Fenil Shah

Maven pom.xml file

- **dependencies-** dependencies element is used to defines a list of dependency of project.
- **dependency**- dependency defines a dependency and used inside dependencies tag. Each dependency is described by its groupId, artifactId and version.
- **name**- this element is used to give name to our maven project.
- **scope**- this element used to define scope for this maven project that can be compile, runtime, test, provided system etc.
- **packaging**- packaging element is used to packaging our project to output types like JAR, WAR etc.

Maven Repository

- Maven repositories are directories of packaged JAR files with some metadata. The metadata are POM files related to the projects each packaged JAR file belongs to, including what external dependencies each packaged JAR has. This metadata enables Maven to download dependencies of your dependencies recursively until all dependencies are download and put into your local machine.
- Maven searches for dependencies in this repositories. First maven searches in Local repository then Central repository then Remote repository if Remote repository specified in the POM.



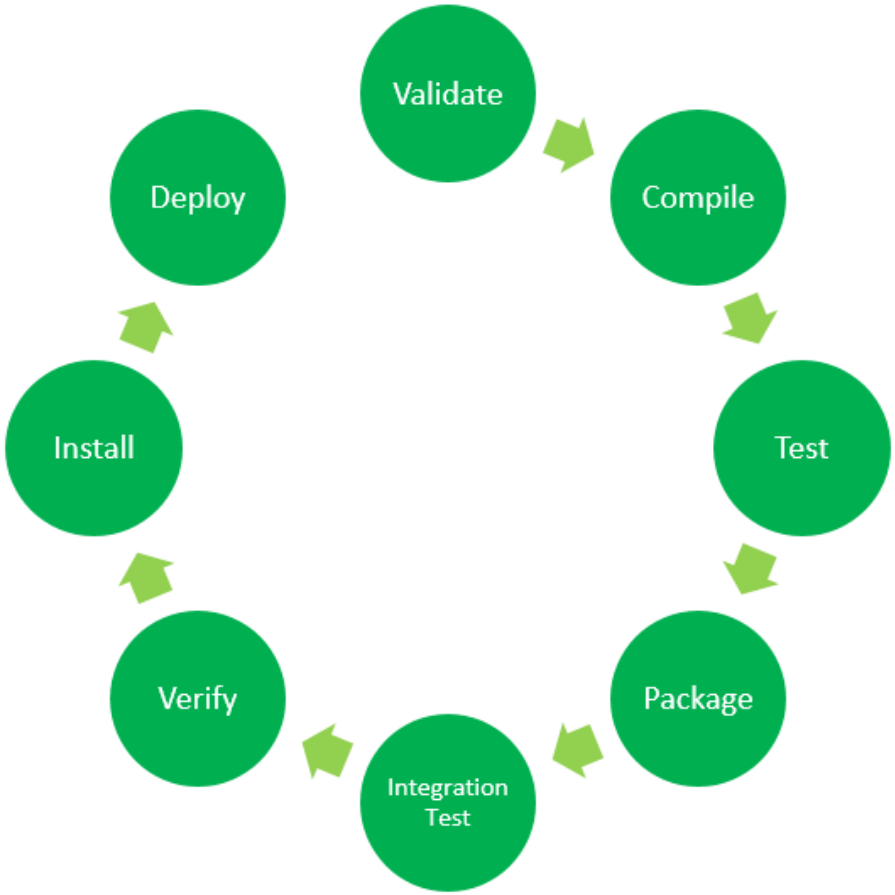
Maven Repository

- **Local repository**- A local repository is a directory on the machine of developer. This repository contains all the dependencies Maven downloads. Maven only needs to download the dependencies once, even if multiple projects depends on them (e.g. ODBC).
- By default, maven local repository is user_home/m2 directory.
- example – C:\Users\asingh\.m2
- **Central repository**- The central Maven repository is created Maven community. Maven looks in this central repository for any dependencies needed but not found in your local repository. Maven then downloads these dependencies into your local repository. You can view central repository by this link.
- **Remote repository**- remote repository is a repository on a web server from which Maven can download dependencies.it often used for hosting projects internal to organization. Maven then downloads these dependencies into your local repository.

Maven

- When working on a java project and that project contains a lot of dependencies, builds, requirement, then handling all those things manually is very difficult and tiresome. Thus using some tool which can do these works is very helpful.
- Maven is such a build management tool which can do all the things like adding dependencies, managing the classpath to project, generating war and jar file automatically and many other things.
- When dependency version update frequently. Then one has to only update version ID in pom file to update dependencies.
- Continuous builds, integration, and testing can be easily handled by using maven. When one needs an easy way to Generating documentation from the source code, Compiling source code, Packaging compiled code into JAR files or ZIP files.
- Generating Spring Project: https://start.spring.io/

Maven Lifecycle



8 Phases of the Default Maven Lifecycle

Maven Lifecycle

- **Validate**: This step validates if the project structure is correct. For example – It checks if all the dependencies have been downloaded and are available in the local repository.
- **Compile**: It compiles the source code, converts the .java files to .class and stores the classes in target/classes folder.
- **Test**: It runs unit tests for the project.
- **Package**: This step packages the compiled code in distributable format like JAR or WAR.
- **Integration test**: It runs the integration tests for the project.
- **Verify**: This step runs checks to verify that the project is valid and meets the quality standards.
- **Install**: This step installs the packaged code to the local Maven repository.
- **Deploy**: It copies the packaged code to the remote repository for sharing it with other developers.
- Maven follows a sequential order to execute the commands where if you run step n, all steps preceding it (Step 1 to n-1) are also executed. For example – if we run the Installation step (Step 7), it will validate, compile, package and verify the project along with running unit and integration tests (Step 1 to 6) before installing the built package to the local repository.

Maven Commands

- **mvn clean**: Cleans the project and removes all files generated by the previous build.
- **mvn compile**: Compiles source code of the project.
- **mvn test-compile**: Compiles the test source code.
- **mvn test**: Runs tests for the project.
- **mvn package**: Creates JAR or WAR file for the project to convert it into a distributable format.
- **mvn install**: Deploys the packaged JAR/ WAR file to the local repository. This will make it so other projects can refer to it and grab it from your local repository.
- **mvn deploy**: Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.

Fenil Shah