Multi Threading

Thread:-

1) Thread is nothing but separate path of sequential execution.

2) The independent execution technical name is called thread.

3) Whenever different parts of the program executed simultaneously that each and every part is called thread.

4) The thread is light weight process because whenever we are creating thread it is not occupying the separate memory it uses the same memory. Whenever the memory is shared means it is not consuming more memory.

5) Executing more than one thread a time is called **multithreading**.

Information about main Thread

When a java program started one Thread is running immediately that thread is called **main thread** of your program.

1. It is used to create a new Thread(child Thread).

2. It must be the last thread to finish the execution because it perform various actions.

It is possible to get the current thread reference by using currentThread() method it is a static public method present in Thread class.

The main important application areas of the multithreading are

1. Developing video games

2. Implementing multimedia graphics.

3. Developing animations

A thread can be created in two ways:-

1) By **extending Thread class**.

2) By implementing **java.lang.Runnable interface**.

Flow of execution:-

1) Whenever we are calling **t.start()** method then JVM will search start() method in the MyThread

class since not available so JVM will execute parent class**(*Thread)*** start() method.

Thread class start() method responsibilities

a. User defined thread is registered into **Thread Scheduler** then only decide new Thread is created.

b. The Thread class start() automatically calls run() to execute logics of userdefined Thread.

Thread Scheduler:-

Thread scheduler is a part of the JVM. It decides thread execution.

Thread Scheduler mainly uses two algorithms to decide Thread execution.

1) Preemptive algorithm.

2) Time slicing algorithm.

First approach

important point is that when extending the **Thread** class, the sub class cannot extend any other base classes because Java allows only single inheritance.

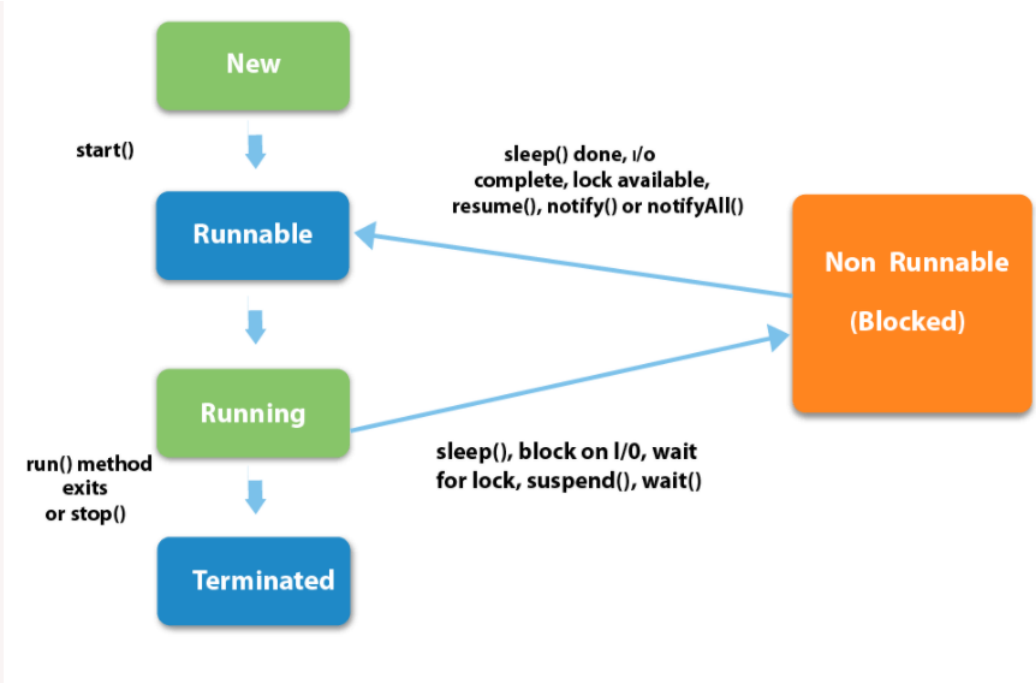
Second approach:-

1) Implementing the **Runnable** interface does not give developers any control over the thread

itself, as it simply defines the unit of work that will be executed in a thread.

2) By implementing the Runnable interface, the class can still extend other base classes if necessary.

Thread Life Cycle



Life cycle stages are:-

1) New

2) Ready

3) Running state

4) Blocked / waiting / non-running mode

5) Dead state

New :- MyThread t=new MyThread();

Ready :- t.start()

Running state:- If thread scheduler allocates CPU for particular thread. Thread goes to running state

The Thread is running state means the **run()** is executed.

Blocked State:-

If the running thread got interrupted or goes to sleeping state at that moment it goes to the blocked state.

Dead State:-If the business logic of the project is completed means run() over thread goes dead state.

Sleep method in java

The sleep() method of Thread class is used to **sleep a thread for the specified amount of time.**

Syntax of sleep() method in java

The Thread class provides two methods for sleeping a thread:

- *public static void sleep(long miliseconds)throws InterruptedException*

- *public static void sleep(long miliseconds, int nanos)throws InterruptedException*

The join() method

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

Syntax:

public void join()throws InterruptedException

public void join(long milliseconds)throws InterruptedException

Naming Thread

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using setName() method. The syntax of setName() and getName() methods are given below:

1. **public String getName():** is used to return the name of a thread.
2. **public void setName(String name):** is used to change the name of a thread.

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, th
schedules the threads according to their priority (known as preemptive scheduling). But it is not guarant
depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY).

The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Java Thread Pool

Java Thread pool represents a group of **worker threads** that are waiting for the job and reuse many times.

In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

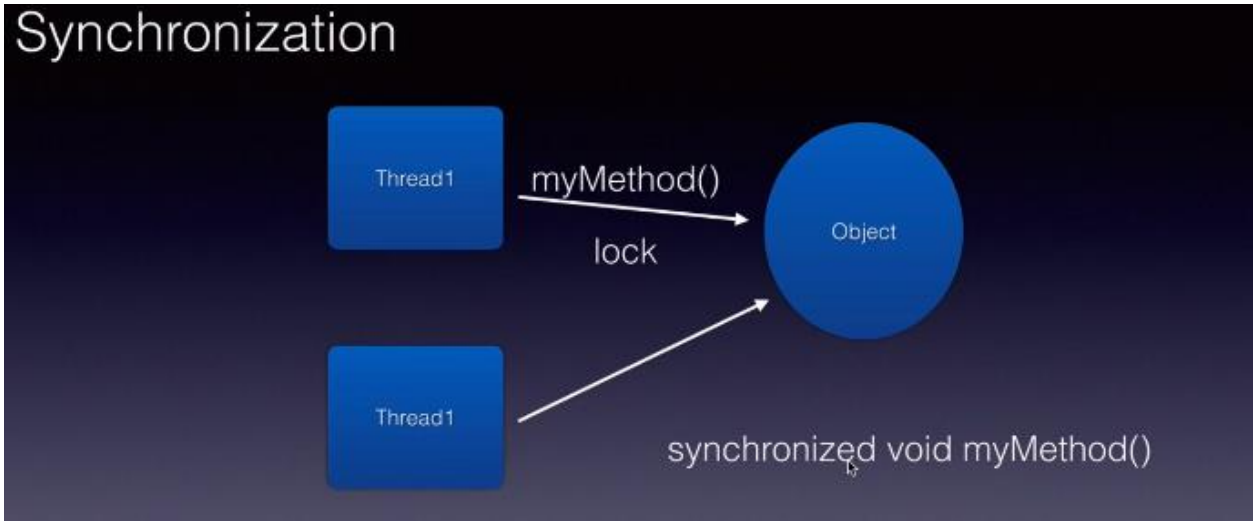
Synchronization in Java

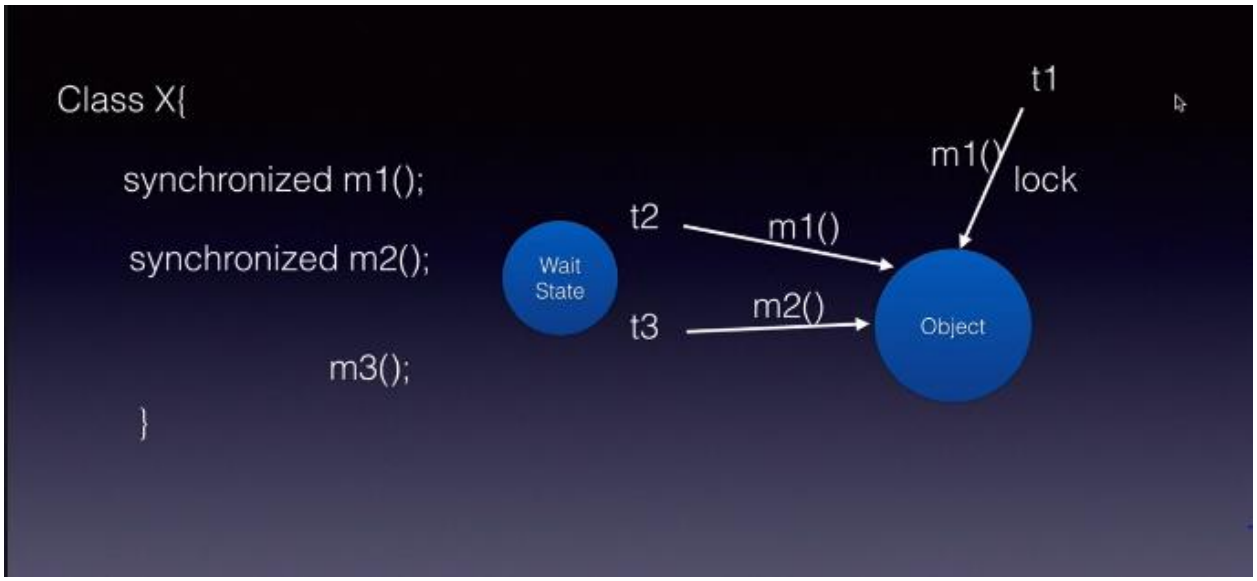
Synchronization in java is the capability *to* ***control the access of multiple threads to any shared resource***.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

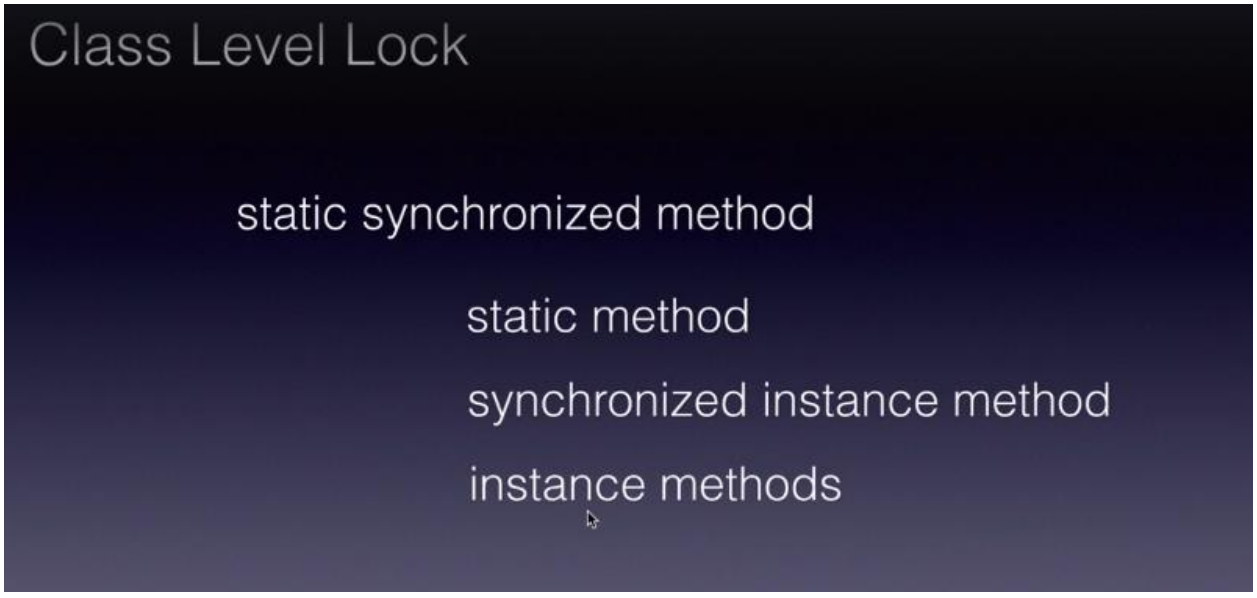
Why use Synchronization

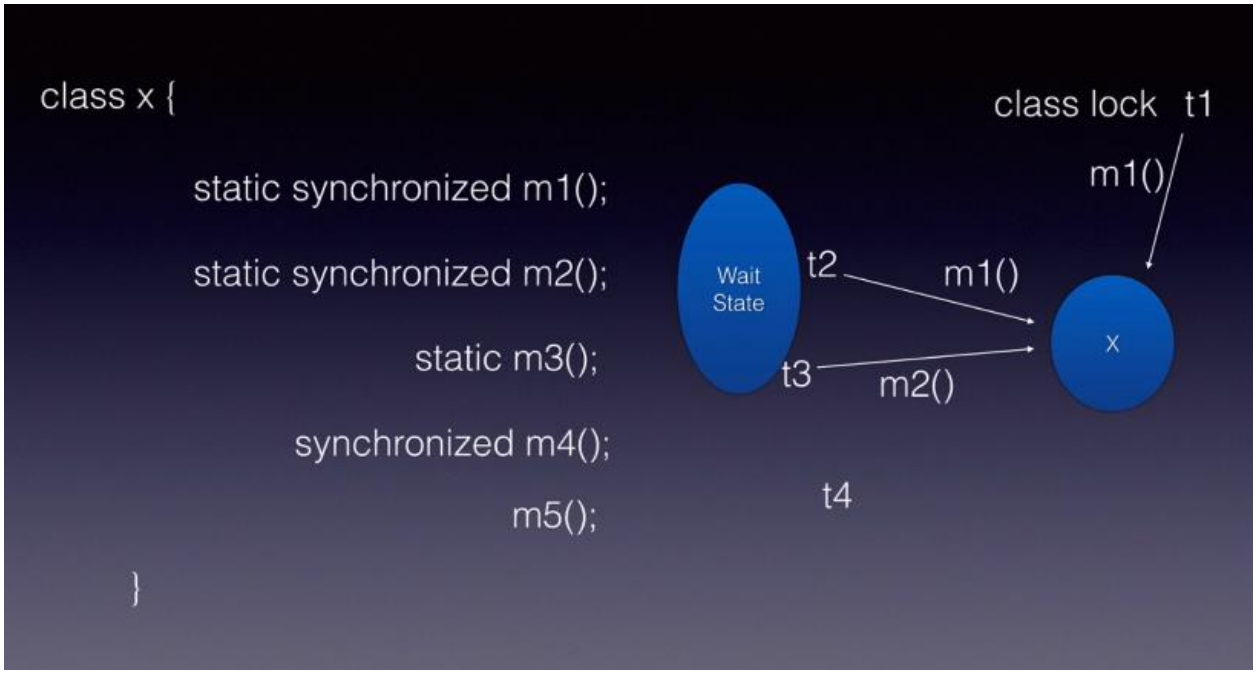
The synchronization is mainly used to concurrent modification of any resources.





Read operations non synchronized , write , updates will be synchronized.





Static blocks

Instance blocks

Garbage Collection in Java

- **In C/C++, programmer is responsible for both creation and destruction of objects. Usually programmer neglects destruction of useless objects. Due to this negligence, at certain point, for creation of new objects, sufficient memory may not be available and entire program will terminate abnormally causing OutOfMemoryErrors.**
- **But in Java, the programmer need not to care for all those objects which are no longer in use. Garbage collector destroys these objects.**
- **Garbage collector is best example as it is always running in background.**
- **Main objective of Garbage Collector is to free heap memory by destroying unreachable objects.**

Eligibility for garbage collection : An object is said to be eligible for GC(garbage collection) if it is unreachable.

We can also request JVM to run Garbage Collector. There are two ways to do it :

- **Using *System.gc()* method** : System class contain static method *gc()* for requesting JVM to run Garbage Collector.
- **Using *Runtime.getRuntime().gc()* method** : Runtime class allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

Note :

- There is no guarantee that any one of above two methods will definitely run Garbage Collector.
- The call *System.gc()* is effectively equivalent to the call : *Runtime.getRuntime().gc()*

Finalization

- Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform cleanup activities. Once *finalize()* method completes, Garbage Collector destroys that object.
- *finalize()* method is present in Object class with following prototype.
- protected void finalize() throws Throwable

Based on our requirement, we can override ***finalize()*** method for perform our cleanup activities like closing connection from database.

Note :

1. The ***finalize()*** method called by Garbage Collector not JVM. Although Garbage Collector is one of the module of JVM.
2. *finalize()* method has empty implementation, thus it is recommended to override *finalize()* method to dispose of system resources or to perform other cleanup.
3. The *finalize()* method is never invoked more than once for any given object.