Multithreading in Java

- Multithreading in Java is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing.
- We use multithreading than multiprocessing because threads use a shared memory area.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread

What is Thread in java

- A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.
- Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

Main Thread

- The execution of main thread always starts from main(String[] args) method and ends at main(String[] args) method.

Attributes of Thread

1.Name

2.Thread id

3.Priority etc

Thread Class

- In java we have class that define a thread that class is known as Thread class present in java.lang package.
- In thread class name,id,priority,state etc. of a thread is defined.
- Thread class has built in methods to perform actions on thread.
- Thread class is encapsulated class, all the attributes are made private we can access them through setter and getter methods.

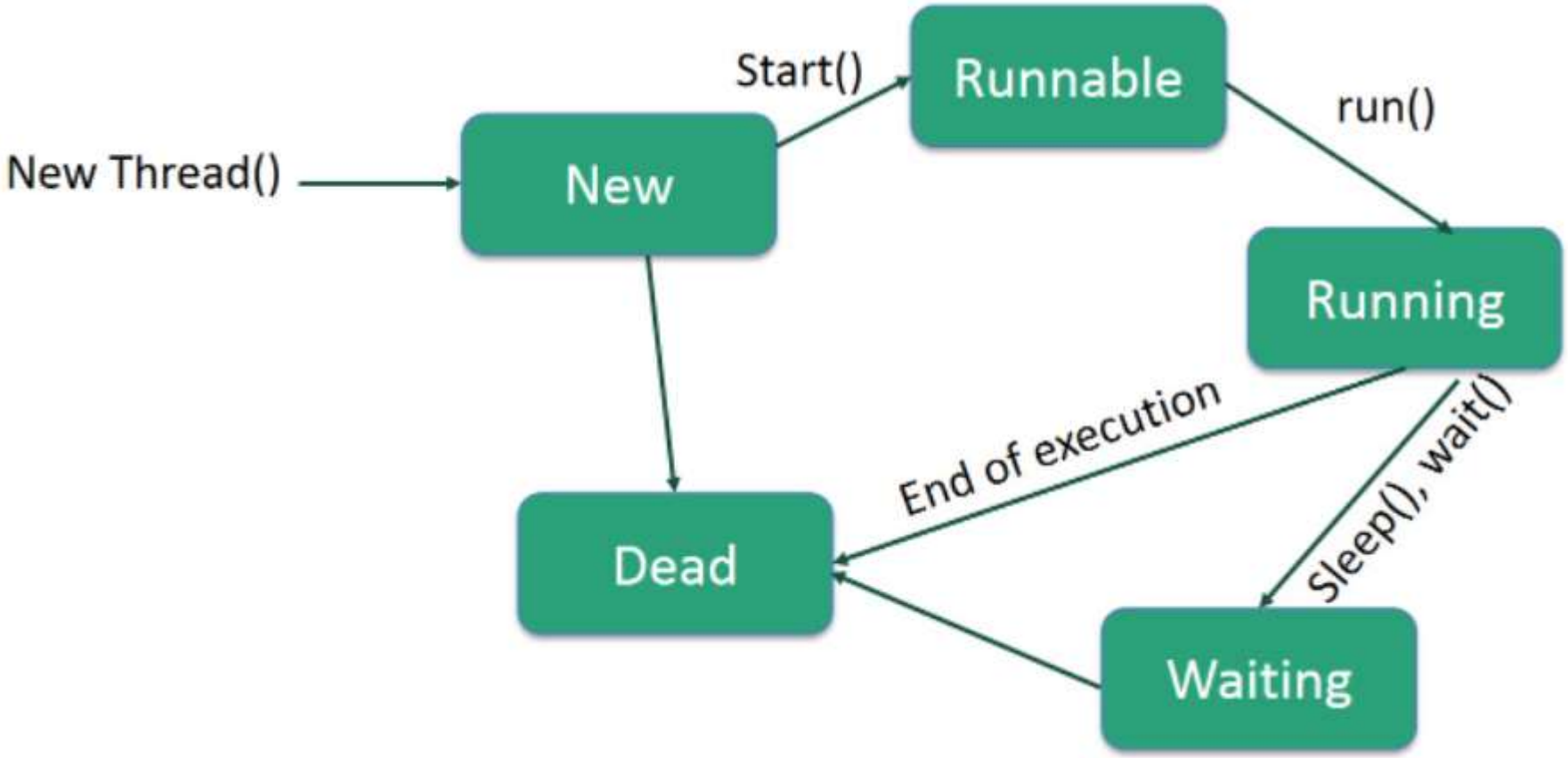
Declaration of Thread Class:

Thread class declared in java.lang package.

public class Thread extends **Object** implements **Runnable**

Life cycle of Thread

- A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Data inconsistency

- Data inconsistency is a problem we face in multithreaded System, when multiple threads try to access the same resource(Object) at the same time.

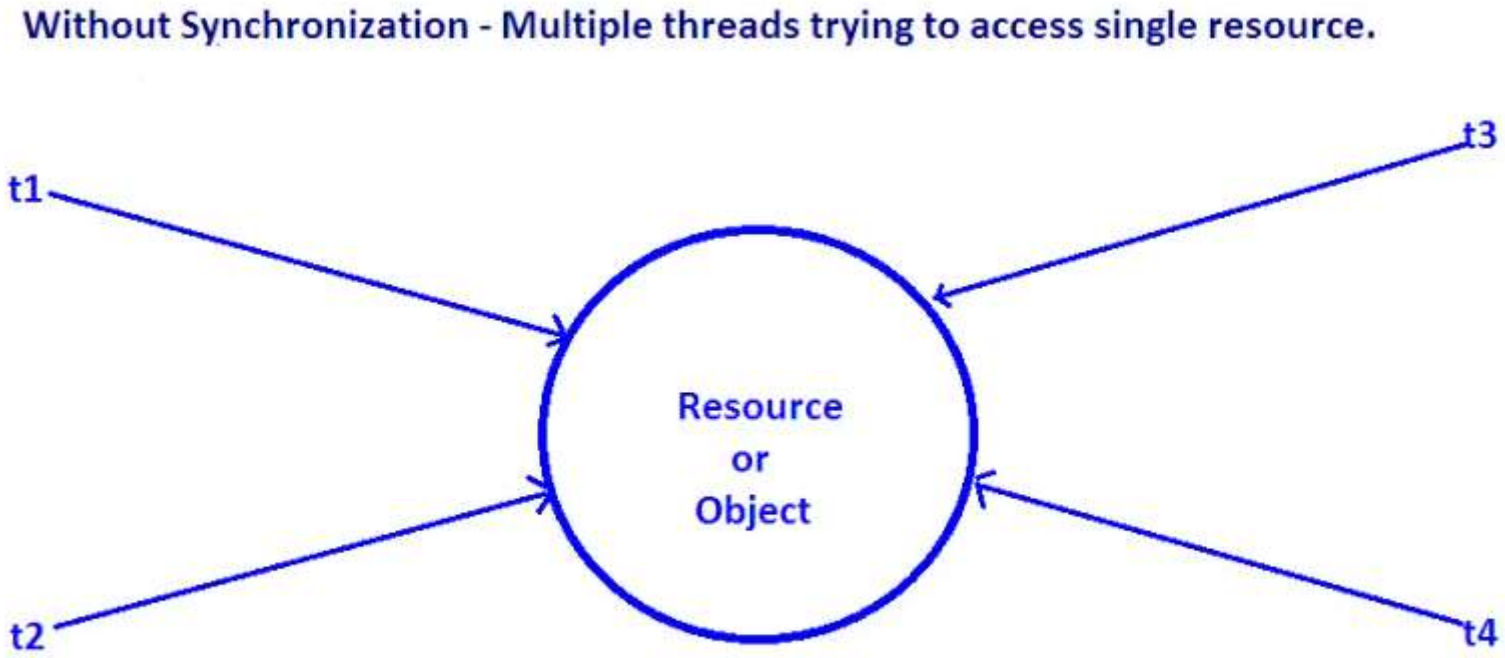
Example:

Let us assume we have one reader thread and one writer thread to perform one task on the same Object. There is possibility of reader thread reading an invalid data(neither old data nor new data)

Synchronization:

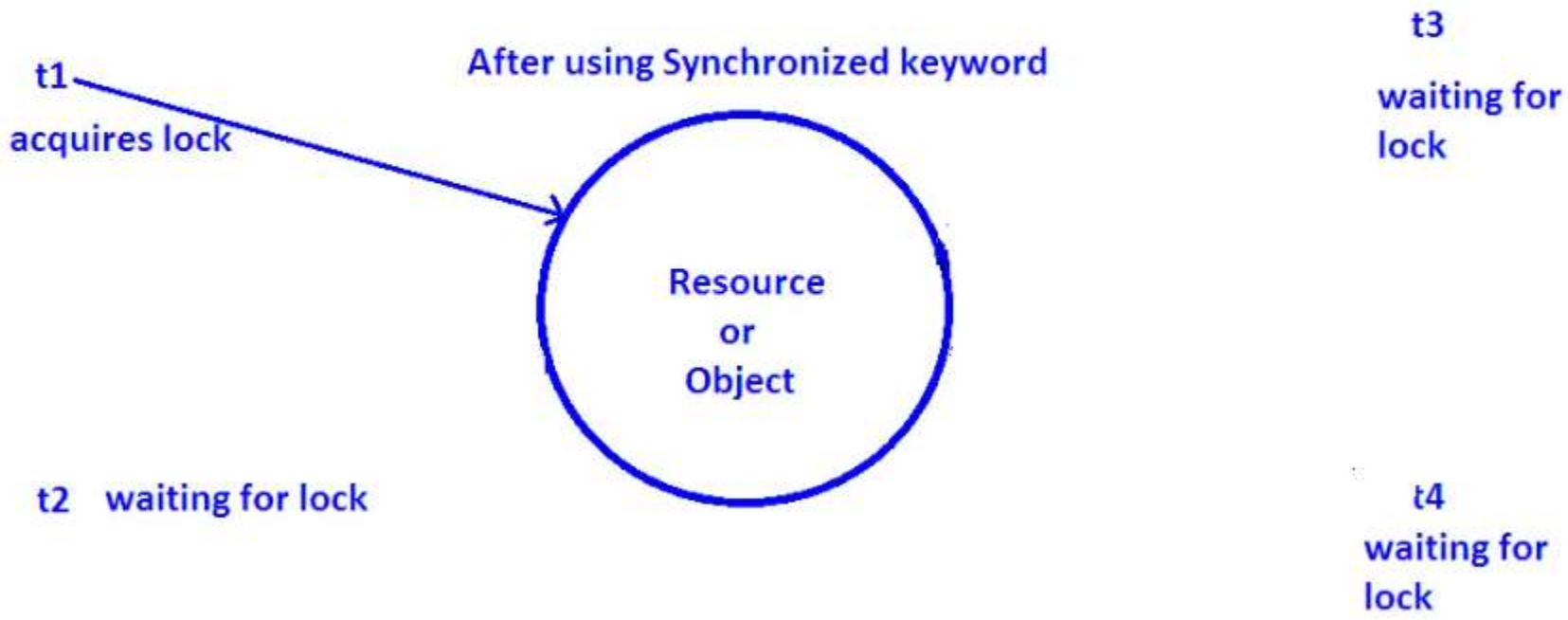
Synchronization is used to avoid data inconsistency occurred by interference of two threads in a same Object.

Without Synchronisation

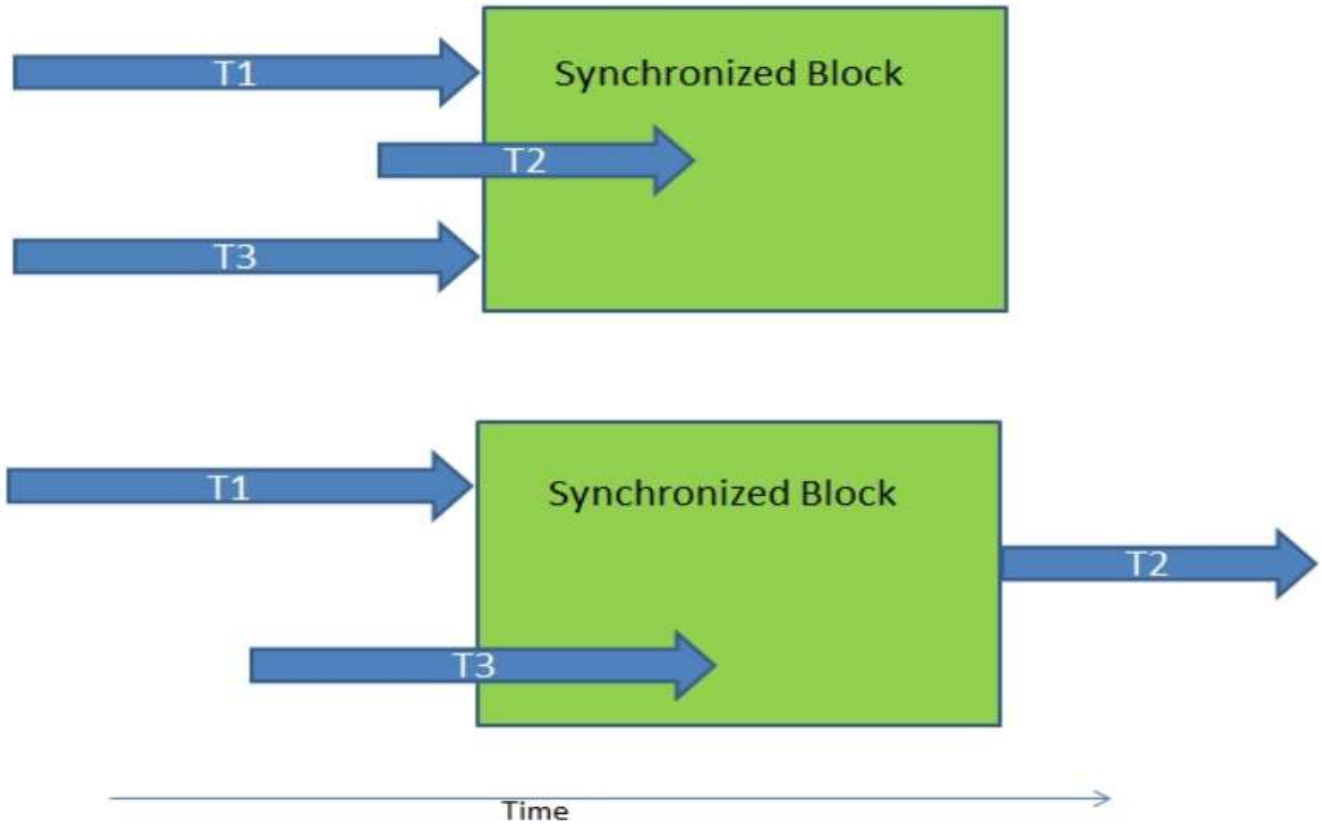


With Synchronized

With Synchronized - Suppose thread t1 will get lock then other thread will wait untill t1 will release lock



Synchronised block



In Synchronized Block, other threads will have to wait when one thread is in

Disadvantages of Synchronization

- Only one thread can access Synchronized block of an Object, therefore if there are many other threads who has to execute same block of the same object they go to wait for locks(this happens only in synchronised block).
- We generally come across two situations

1.Starvation

2.DeadLock

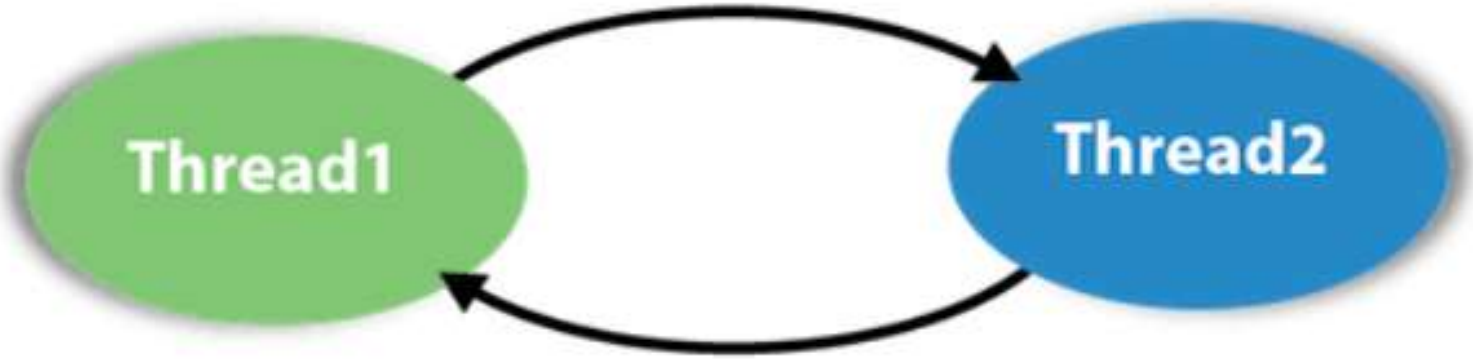
Starvation:

Starvation describes the situation where a thread is unable to join regular access to shared resource(Object)

And unable to access any progress. This happens when shared resources are made unavailable for very long time by greedy threads(High priority). This state is called as Starvation.

Deadlock

- Deadlock describes the situation where two or more threads are blocked forever, waiting for each other.(A deadlock will occur due to **Synchronization**)
- Deadlock in Java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread.
- Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



Differences between Starvation and Deadlock

Deadlock	Starvation
All processes keep waiting for each other to complete and none get executed	High priority processes keep executing and low priority processes are blocked
Resources are blocked by the processes	Resources are continuously utilized by high priority processes
Also known as Circular wait	Also know as lived lock
Deadlock happens when every process holds a resource and waits for another process to hold another resource.	Starvation happens when a low priority program requests a system resource but cannot run because a higher priority program has been employing that resource for a long time.