MULTITHREADING

TOPICS

- Basics of Multithreading
- How to Create a Thread
- Getting And Setting Name of Thread
- Priority of Thread
- Methods to prevent Thread Execution
- Synchronization

BASICS OF MULTITHREADING

- **Multithreading** :- Execution of more than one thread at a time.
- **Multitasking** :- Performing more than one task at a time.
- Two types of Multitasking :-

1. Process Based Multi-tasking :- Executing multiple task simultaneously where each task is a separate Independent process.
2. Thread Based Multi-Tasking :- Executing multiple tasks simultaneously where each task is separate and independent part of the process

HOW TO CREATE A NEW THREAD

- Two ways to Create a new Thread :-
  - By Extending Thread Class
  - By Implementing Runnable Interface

1) BY EXTENDING THREAD CLASS

Case 1 - Thread Schedular :- which thread will execute first is decided by Thread(CPU) Schedular.

* If both thread priority are same than we cannot predict the output.

Case 2 – start() & run() method :-

1. start() :- new thread will be created which is responsible for execute run() of Thread class.

2. run() :- no new thread will be created.

- Case 3 : Overriding run() method :- Highly recommend to override run() method.
- If we cannot override it then Thread class run method execute which

  is having empty implementation. Which does not make any sense.

- Case 4 : Overriding start() method :- if we are overriding start() method it will behave like normal method.
- Highly recommended not to override start() method.

- Case 5 : Life Cycle of Thread :-

threadName.start()   (if run() method completed)

New state   --->   Ready/Runnable   ------>   Running   ------>   Dead

(Create new thread)   (if Thread Schedular allocated CPU)

- Case 6 : Cannot Start again Same Thread :-
- After starting a Thread you will not restart the same thread.

2) BY IMPLEMENTING RUNNABLE INTERFACE

- We have to implement Runnable Interface in our class and override run method in our class.

- Case 1 : start() method :- New Thread will be created which is responsible for execute run() method.

- Case 2 : run() method :- No new Thread will be created.

GETTING & SETTING NAME OF THREAD

- Every thread in java has some name it may be provided by the programmer or automatically by the JVM.
- Two methods:-
  - public final void setName(Argument);
  - public final String getName();

Example : - Setting name of main thread

Thread.currentThread().setName("anyN
ame");

PRIORITIES OF THREAD

- Thread priority vary from 1 to 10.
- Thread.MIN_PRIORITY ------> 1
- Thread.MAX_PRIORITY -------> 10
- Thread.NORM_PRIORITY -------> 5
- **Default Priority only for the main Thread.**
- **Default priority for all other threads depends on the priority of parent thread.**
- Methods :-
  - 1) public final void setPriority(int new Priority);
  - 2) public final int getPriority();

METHODS TO PREVENT THREAD EXECUTION

- **sleep() method** **: -** Thread doesn't want to perform any operation for amount of time.
  - Two type of sleep method :-
  1. public static native void sleep(long ms) throws InterruptedException
  2. public static void sleep(long ms,int ns)throws InterruptedException

- **Join() method** **: -** * If a thread wants to wait until completing some other thread.
  - Type of join method :-
  1. public final void join()throws InterruptedException
  2. public final void join(long ms) throws InterruptedException
  3. public final void join(long ms,int ns) throws InterruptedException

yield() method : -

- "to pause current executing Thread for giving the chance of remaining waiting Threads of same priority".
- If all waiting Threads have the low priority or if there is no waiting Threads then the same Thread will be continued its execution.
- If several waiting Threads with same priority available then we can't expect exact which Thread will get chance for execution.
- The Thread which is yielded when it get chance once again for execution is depends on mercy of the Thread scheduler.
- public static native void yield();

❑ Note : Some operating systems may not provide proper support for yield() method.

INTERRUPT() METHOD

- If a Thread can interrupt a sleeping or waiting Thread by using interrupt()(break off) method of Thread class.
- We only use with sleep or wait method.
- **Syntax :- ThreadName.interrupt();**
- It is called after starting of a thread.

Note:

1. Whenever we are calling interrupt() method we may not see the effect immediately, if the target Thread is in sleeping or waiting state it will be interrupted immediately.
2. If the target Thread is not in sleeping or waiting state then interrupt call will wait until target Thread will enter into sleeping or waiting state. Once target Thread entered into sleeping or waiting state it will effect immediately.
3. In its lifetime if the target Thread never entered into sleeping or waiting state then there is no impact of interrupt call simply interrupt call will be wasted.

SYNCHRONIZATION

1. Synchronized is the keyword applicable for methods and blocks but not for classes and variables.
2. If a method or block declared as the synchronized then at a time only one Thread is allow to execute that method or block on the given object.
3. The main advantage of synchronized keyword is we can resolve date inconsistency problems.
4. But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system.
5. Hence if there is no specific requirement then never recommended to use synchronized keyword.
6. Internally synchronization concept is implemented by using lock concept.
7. Every object in java has a unique lock. Whenever we are using synchronized keyword then only lock concept will come into the picture.
8. If a Thread wants to execute any synchronized method on the given object 1st it has to get the lock of that object. Once a Thread got the lock of that object then it's allow to execute any synchronized method on that object. If the synchronized method execution completes then automatically Thread releases lock.
9. While a Thread executing any synchronized method the remaining Threads are not allowed execute any synchronized method on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method simultaneously. [lock concept is implemented based on object but not based on method].

SYNCHRONIZE – BLOCK

- It is used whenever we have to perform some operation on a portion of method.
- Syntax : - Synchronized (Object reference){   }
- Every object have two area synchronized & non-synchronized area and having Lock Also.
- Lock is mostly used for synchronization.
- Thread acquire the lock and don't leave until the execution is completed.

STATIC SYNCHRONIZE

- If dealing with two objects or more than two objects and possibility of having conflicts, then use "static synchronization".
- Static synchronization allows only one Thread of an object at a time to execute.
- By using Static Synchronized. We are attaining a class-level lock such that only one thread will operate on the method.

WAIT() METHOD

- Whenever any Thread call wait() method, then current thread lock will be released and invoke the waiting thread.
- If any Thread calls wait() method, it calls the current thread to release its lock and wait until another thread invokes the notify() & notify All() method.
- It is used inside synchronized block.

NOTIFY() & NOTIFY ALL() METHOD

- Notify() method : - wake up the single thread & release object lock.
- notify() wakes up the first thread that called wait() on the same object.
- Notify All() method : - waked up the multiple thread and release object lock.
- Notify All sends notification to all waiting threads hence it is not clear which of the thread is going to receive the lock.

THANK YOU