Threads

- A multithreaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources.
- Multitasking is when multiple processes share common processing resources such as a CPU.
- Multithreading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel.

Advantage of Java Multithreading
1) It doesn't block the user because threads are independent and you can perform multiple operations at 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 exception occur in a single thread.

What is a thread?
- A thread is a lightweight sub process, a 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 shares a common memory area.
- Thread is executed inside the process.

Life cycle of a Thread (Thread States):
The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:
- **New** - The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
- **Runnable** - The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- **Running** - The thread is in running state if the thread scheduler has selected it.
- **Non-Runnable (Blocked)** - This is the state when the thread is still alive, but is currently not eligible to run.
- **Terminated** - A thread is in terminated or dead state when its run() method exits.

Thread Creation:
There are two ways to create a thread:
- By extending Thread class
- By implementing Runnable interface.

Thread class:
- Thread class provide constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.

Constructors of Thread class:
- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Eg:
```
class Multi extends Thread {
        public void run() {
                System.out.println("thread is running...");
        }

        public static void main(String args[]) {
                Multi t1 = new Multi();
```

Eg:
```
ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of 5 threads
        for (int i = 0; i < 10; i++) {
                Runnable worker = new WorkerThread("" + i);
                executor.execute(worker);//calling execute method of ExecutorService
        }
        executor.shutdown();
}
```

Shutdown Hook:

- If you want to execute some code before JVM shuts down, use shutdown hook.
- The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly.
- This can be achieved with addShutdownHook(Runnable r) method
- The addShutdownHook() method of Runtime class is used to register the thread with the Virtual Machine.

Eg:
```
                Runtime r=Runtime.getRuntime();
                r.addShutdownHook(new MyThread());
```

or

```
        Runtime r = Runtime.getRuntime();

                r.addShutdownHook(new Runnable() {
                        public void run() {
                                System.out.println("shut down hook task completed..");
                        }
        });
```

Deadlock

- 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.
- To avoid this problem thread synchronization is used.

Thread Synchronization

There are two types of thread synchronization: **mutual exclusive and inter-thread communication.**

- Mutual Exclusive
  - Synchronized method.
  - Synchronized block.
  - static synchronization.
- Cooperation (Inter-thread communication in java)

Synchronized method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Eg:
```
                synchronized void printTable(int n){}
```

Synchronized block

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the