Day 12

Multithreading

- Process of executing multiple task simultaneously is called multitasking.
- In context of oops, it called concurrency. It is minor pillar of oops.
- We can achive multitasking using process as well as thread.

> Why process based multitasking is called heavy weight multitasking?

- If CPU want to execute multiple processes simultaneously then it must first save state of current process into PCB then it can transfer control of itself to another process. It is called context switching.
- context switching is heavy task hence Process based multitasking is called heavy weight multitasking.

> Why thread based multitasking is called light weight multitasking?

- Thread always resides within process.
- To access resource assigned to the process, thread do not require context switching hence thread based multitasking is called light weight multitasking.

> Why java is multithreaded programming language?

- Thread is Non java / OS resource. To access OS thread, java application developer need not to use OS API. To access OS thread SUN/ORACLE developer has already developed ready framework. In other words java has given built in support to access OS thread hence every java application is multithread.
- When starts execution of java application, it also starts execution of main thread and garbage collector hence every java application is multithreaded.

Types of Thread

1. User Thread / Non Damemon Thread
   - User thread gets terminated immediatly after execution is over.
2. Daemon Thread / Background Thread
   - Daemon thread gets terminated when all its child thread gets terminated.

- If we create thread in java then by default it is considered as user thread. using setDaemon( ) method we can convert user thread into daemon thread.

```
Thread th = new Thread( ... );
th.setDaemon( true );
```

Thread Framework

- If we to use thread in java then we must import java.lang package.
- Interface
  1. Runnable
- Class(es)

    1. Thread
    2. ThreadLocal
    3. ThreadGroup
- Enum
    1. Thread.State
- Exception(s)
    1. InterruptedException
    2. IllegalMonitoStateException
    3. IllegalThreadStateException

Runnable

- If interface contains only one abstract method then it is called functional interface.
- Runnable is functional interface.
- "void run( )" is abstract method of Runnable interface.
- run() method is called business logic method.

Thread

- java.lang.Thread is managed version of unmanaged thread.
- In other words instance of java.lang.Thread is not a thread rather it presents operating system thread.
- JVM is resposible for mapping Java Thread instance to OS thread.

Nested Type of Thread

- Thread.State is enum declared inside Thread class.
- Enum constants of Thread state represents JVM thread states.
- Following are Thread States:
    1. NEW 0
    2. RUNNABLE 1
    3. BLOCKED 2
    4. WAITING 3
    5. TIMED_WAITING 4
    6. TERMINATED 5

```
State[] states = State.values();
for (State state : states)
{
    String name = state.name();
    int ordinal = state.ordinal();
    System.out.println(name+" "+ordinal);
}
```

Fields of Thread

1. public static final int MIN_PRIORITY = 1
2. public static final int NORM_PRIORITY = 5

3. public static final int MAX_PRIORITY = 10

- Thread fields represent thread priorites.

Constructors of Thread

1. public Thread()

```
Thread th1 = new Thread();
```

2. public Thread(String name)

```
Thread th1 = new Thread("UserThread#1");
```

3. public Thread(Runnable target)

```
Runnable target = new CThread(); //Upcasting
Thread th1 = new Thread( target );
```

4. public Thread(Runnable target, String name)

```
Runnable target = new CThread(); //Upcasting
Thread th1 = new Thread( target,"Abc" );
```

5.public Thread(ThreadGroup group, Runnable target)

```
ThreadGroup grp = new ThreadGroup("MyGrp");
Runnable target = new CThread(); //Upcasting
Thread th1 = new Thread( grp, target );
```

Methods of Thread Class

1. public static Thread currentThread()
2. public final String getName()
3. public final void setName(String name)
4. public final int getPriority()
5. public final void setPriority(int newPriority)
6. public Thread.State getState()
7. public final ThreadGroup getThreadGroup()
8. public void interrupt()
9. public final boolean isAlive()

10. public final boolean isDaemon()
11. public final void setDaemon(boolean on)
12. public final void join() throws InterruptedException
13. public static void sleep(long millis) throws InterruptedException
14. public void start()
15. public static void yield()

- If we want to use any hardware resource(CPU) efficiently then we should use thread.
- If we want to improve performance of application then we should void use of blocking calls in multithreaded application.
- Following calls are blocking calls:

1. sleep()
2. suspend()
3. join()
4. wait()
5. performing input operation

> What is the difference between sleep() and suspend( ) ?

- If we want to suspend execution of thread for specified milliseconds then we should use sleep() method.
- If we want to suspend execution of thread for infinite time then we should use suspend() method. In this case, by calling resume() method, suspended thred can come back to runnable state.
- Note : suspend() and resume() are deprecated.

```
Thread thread = Thread.currentThread();
System.out.println(thread.toString());
//Thread[main,5,main]
```

- thread.toString() returns a string representation of current thread, including
    - thread's name
    - priority
    - and thread group.

finalize() method

- It is non final method of java.lang.Object class
- Syntax: protected void finalize( ) throws Throwable
- If we want to release class scope resources then we should use finalize() method.
- If reference count of any instance is zero then it is eligible for garbage collection.
- Garbage Collector invoke finalize() method on instance whose reference count is zero.

Thread Creation

- In java, we can create thread using 2 ways

1. Using Runnable interface
2. Using Thread class

Thread Creation using Runnable

```
class CThread implements Runnable
{
        private Thread thread;
        public CThread( String name )
        {
                this.thread = new Thread( this );
                this.thread.setName( name );
                this.thread.start();
        }
        @Override
        public void run()
        {
                //TODO : Business Logic
        }
}
```

Thread Creation using Thread class

```
class CThread extends Thread
{
        public CThread( String name )
        {
                this.setName( name );
                this.start();
        }
        @Override
        public void run()
        {
                //TODO : Business Logic
        }
}
```

Realtion Between start() and run() method

- start() method do not call run() method
- If we call start() method on Thread instance (instance of java.lang.Thread class) then it is considered as request to JVM to register OS thread for Thread instance.
- When CPU schedular assign CPU to the OS thread then JVM invoke run() method on Runnable instance(argument pass to the Thread constuctor (this)).

Starting Thread

- If we want to start execution of thread then we should start() method.
- If we call start() method on thread instance multiple times then start() method throws IllegalThreadStateException.

```
Runnable target = new CThread();
Thread th = new Thread( target );
th.start();
```

Thread States

- If we create Thread instance then it is considered in NEW state.
- If we call start() method on Thread instance then it is considered in RUNNABLE state.
- If supend thread by calling Thread.sleep() then Thread is considered in TIMED_WAITING state.
- If control come out of run() method then Thread gets terminated.
- If thread gets terminated then it is considered in TERMINATED state.

> What is the difference between creating thread using Runnable and Thread?

- If class is sub class then we should create thread by implementing Runnable interface.
- If we create thread by implementing Runnable interface then every sub class must participate in threading behavior.
- If class is not a sub class then we should create thread by extending Thread class.
- If we create thread by extending Thread class then it is not mandatory for every sub class to participate in threading behavior. By overriding start() method, sub class can come out of threading behavior.

Thread Priority

- Managing thread is a job of CPU schedular.
- On the basis of Thread Priority, schedular assign CPU to the Thread.
- setPriority() method is used to set priority for Thread and getPriority() method is used to get priority of thread.

```
Thread thread = Thread.currentThread();
//thread.setPriority(thread.getPriority() + 3 ); //OK
thread.setPriority(Thread.NORM_PRIORITY + 3 ); //OK
System.out.println(thread.getName()+" : "+thread.getPriority());
```

- If the priority is not in the range MIN_PRIORITY to MAX_PRIORITY then setPriority method throws IllegalArgumentException
- Default priority of child thread depends on priority of parent Thread.
- If we set priority of a thread in java, then it gets mapped differently on Different operating system. Hence Same java application producess different behavior on different operating system
- Thread priorites makes java application platform dependant.

> Which feature of java make application platform dependant?

1. Thread Priorities
2. Abstract Window Toolkit(AWT) components

Thread Joining

- If we call join() method of thread instance then it blocks execution of all other threads.
- It is a blocking call.

```
Runnable target = new CThread();
Thread th = new Thread( target );
th.start();
th.join();
```

Resource Locking

- Without co-oridination, if multiple threads try to access shared resource then it is called as race condition.
- If we want to avoid race condition then we should lock the resource.
- synchronized is keyword in java which is used to lock the resource.
- If we use synchronized keyword with shared resource then it gets minitor object.
- Maintaining monitor object is expensive hence we should use it for minimum time otherwise waiting threads need to wait for long time which may degrade performance of application.
- Code written inside synchronized block or synchronized method is called critical section.
- If thread is waiting to get monitor object associated with shared resource then it is considered in Blocked state.

Inter Thread Communication

- Inter thread communication represents synchronization.
- With co-oridination, if multiple thread try to communicate with each other then it is called synchronization.
- Using same minitor object, if thread try to communicate with each other then it is called synchronization.
- If we want to achive synchronization then we should use wait, notify/notifyAll() methods.
- It is mandatory to call above methods from synchronized block / synchronized method.
- If we try to call these methods from unsynchronized block/method then these methods throws IllegalMonitoStateException.