

Day 17

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 achieve 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 responsible 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 priorities.

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

- 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
    }
}
```

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();
```