Day14: Multithreading, Executor Framework

Callable and Future:

In the case of Runnable tasks ,Threads won't return anything after completing the task.

If a thread is required to return some result after execution, then we should use the Callable Interface instead of Runnable.

Callable interface belongs from **java.util.concurrent** package.

Callable interface also contains only one method:

```
public Object call()throws Exception
```

Note:- the object of Callable we can't pass to the normal Thread class constructor unlike Runnable object, here we need to use the ExecutorService class help.

if we submit a Callable object to **ExecutorService** object, then after completing the task, thread returns an object of the type **Future**.

The **Future** object can be used to retrieved the result from the Callable tasks.

Example:

```
import java.util.concurrent.*;
  class MyCallable implements Callable{

    int num;

    public MyCallable(int num) {
        this.num = num;
    }

    @Override
    public Object call() throws Exception {

        System.out.println(Thread.currentThread().getName()+" .. is responsible to find the sum of first "+num+" numbers"); int sum=0;

        for(int i=0;i<=num;i++){
            sum = sum+i;
        }
        return sum;
    }

}


class Main{

    public static void main(String[] args)throws Exception {

        MyCallable[] jobs = {
```

```
                new MyCallable(10),
                new MyCallable(20),
                new MyCallable(30),
                new MyCallable(40),
                new MyCallable(50),
                new MyCallable(60),

        };

        ExecutorService service=Executors.newFixedThreadPool(3);

        for(MyCallable job:jobs){
```



```
            Future f= service.submit(job);
            System.out.println(f.get());
        }

        service.shutdown();
    }
}
```

Difference Between Runnable and Callable:

Runnable	**Callable**
If a thread won't returns anything.	If a Thread returns anything
only one method **public void run()**	only one method **public Object call() throws Exception**
return type void	return type is Object
if any exception raise compulsory we need to handle within try catch.	not required to use try-catch
Belongs to java.lang package	Belongs to java.util.concurrent package
from java 1.0 version	from java 1.5 version

Suspending a thread unconditionally:

yield() method:

This yield() method is a static method defined inside the Thread class, it is used to pause the current executing thread for giving the chance to remaining waiting thread of same priority, if there is no any waiting thread or all waiting threads have low priority then same thread will get the chance once again for the execution.

suspend() method:

In order to suspend a thread unconditionally, we make use of non-static method suspend() present in a Thread class.

resume() method:

In order to resume a thread which has been suspended long back we use resume() method, it is also a non-static method defined inside the Thread class.

Note: suspend() and resume() method are deprecated methods. we should not use those methods. we should not use any deprecated methods.

The alternate methods are **wait()** and **notify()**. these methods are the non-static method defined inside the **Object** class.

Difference between yield() and wait() method:

The wait() method will suspend a thread till notify() method is called, whereas yield() method says right now it is not important please give the chance to another thread of same priority.