Notes - ThreadLocal

ThreadLocal -

***ThreadLocal is considered an anti-pattern please refrain from using it directly*.**

ThreadLocal allows us to associate an object with the Thread. It is normally used to hold the information which should be accessible anywhere during the thread execution.

Important Methods -

get() - Returns the value associated with the thread.

set(T value) - Associates the value with the thread.

remove() - Removes the value associated with the thread.

SimulatedExample -

Assume we have two requests, and we assigned two WorkerThreads for servicing them, think that this is a server application assigning a client request to a thread. Now inside the WorkerThread assume that the request is delegated to the framework. And Framework does the pre-processing. During the pre-processing step think that it extracts the user information from the

request and sets up the UserContext. The setUserName operation sets the userName in the ThreadLocal object. And then it forwards the request to the API. Assume API calls the services Service1 and Service2, and in between with a delay.Now assume Service1 want to know the user information then it can use the UserContext.getUserName() which retrieves the userName from the thread local object.

And the most important thing is that once the task is done, we need to clear this value. The postProcess step will clear the context i.e. It removes the value associated with the current Thread. May not be important for this example, but in normal applications the same thread is used to service multiple client request, if not cleared the existing value associated with the thread might interfere with some other client request.

```
.   public class Main {
.
.       public static void main(String[] args) {
.        String request1 = "a";
.        String request2 = "b";
.
.        new WorkerThread(request1).start();
.        new WorkerThread(request2).start();
.       }
.   }
.
.   class UserContext {
.       private static ThreadLocal<String>
    userInfoThreadLocal =
.               new ThreadLocal<String>() {
.                   public String initialValue() {
.                    return null;
```

```
                }
            };

    public static String getUserName() {
        return userInfoThreadLocal.get();
    }

    public static void setUserName(String userName) {
     userInfoThreadLocal.set(userName);
    }

    public static void clear() {
     userInfoThreadLocal.remove();
    }
}

class WorkerThread extends Thread {

    String request;
    Framework framework = new Framework();

    public WorkerThread(String request) {
     this.request = request;
    }

    @Override
    public void run() {
     framework.delegate(request);
    }
}

class Framework {

    API api = new API();

    public void delegate(String request) {
     preProcess(request);
     try {
         api.process(request);
```

```
        } finally {
            postProcess();
        }
    }

    private void preProcess(String request) {
        String userName = getUserName(request);
        UserContext.setUserName(userName);
    }

    private void postProcess() {
        UserContext.clear();
    }

    private String getUserName(String request) {
        return request;
    }
}

class API {

    Service1 service1 = new Service1();
    Service2 service2 = new Service2();

    public void process(String request) {
        service1.doService1();
        sleep();
        service2.doService2();
    }

    private void sleep() {
        try {
            Thread.sleep(3000);
        } catch(InterruptedException e) {}
    }
}

class Service1 {

```

```
.       public void doService1() {
.        System.out.println(
.            "Performing service1 for user - "
.            + UserContext.getUserName()
.            + " "  + Thread.currentThread());
.       }
.   }
.
.   class Service2 {
.
.       public void doService2() {
.        System.out.println(
.            "Performing service2 for user - "
.            + UserContext.getUserName()
.            + " "  + Thread.currentThread());
.       }
.   }
```

Output -

You can see in the output that Thread-0 and Thread-1 has different usernames as each thread has its own copy of the userName.

```
.   Performing service1 for user - b Thread[Thread-1,5,main]
.   Performing service1 for user - a Thread[Thread-0,5,main]
.   Performing service2 for user - b Thread[Thread-1,5,main]
.   Performing service2 for user - a Thread[Thread-0,5,main]
```

Summary -

ThreadLocal could be used to supply some vital information to all the components involved in processing the request, where this could not be passed directly. But it considered an anti-pattern and creates trouble during asynchronous processing if not handled properly.