**Use of ThreadLocal in Java Multithreading - 2022**

Let us see first, if we do not use ThreadLocal, what will happen.

**public class** Check1 {  
 **public void** add(List<String> list) {  
 **for** (**int** i = 0; i < 10; i++) {  
 list.add(**"Add-Value-"** + i);  
 }  
 }  
  
 **public void** remove(List<String> list) {  
 **for** (**int** i = 0; i < 2; i++) {  
 list.remove(i);  
 }  
 }  
  
 **public void** show(List<String> list) {  
 **for** (Iterator<String> itr = list.iterator(); itr.hasNext(); ) {  
 String value = itr.next();  
 System.***out***.println(**"Now Value : "** + value);  
 }  
 }  
  
 **public void** check() **throws** InterruptedException {  
 List<String> alphabets = Arrays.*asList*(**"A"**, **"B"**, **"C"**, **"D"**, **"E"**, **"F"**);  
 List<String> list = **new** ArrayList<>(alphabets);

Runnable add = () -> add(list);  
 Runnable remove = () -> remove(list);  
 Runnable show = () -> show(list);  
  
 Thread t1 = **new** Thread(add);  
 Thread t2 = **new** Thread(remove);  
 Thread t3 = **new** Thread(show);  
  
 t1.start();  
 t2.start();  
 t3.start();  
  
 t1.join();  
 t2.join();  
 t3.join();  
 System.***out***.println(**"All threads completed operations ..."**);  
  
 }

**OUTPUT**

Now Value : B

All threads completed operations ...

Exception in thread "Thread-2" java.util.ConcurrentModificationException

at java.base/java.util.ArrayList$Itr.checkForComodification(ArrayList.java:1042)

at java.base/java.util.ArrayList$Itr.next(ArrayList.java:996)

at com.ddlab.rnd.type1.Check1.show(Check1.java:24)

at com.ddlab.rnd.type1.Check1.lambda$check$2(Check1.java:34)

at java.base/java.lang.Thread.run(Thread.java:834)

Let use a modified program using ThreadLocal.

**Using ThreadLocal in new Version of Java**

**public class** TestThreadLocal {  
  
 **public void** add(ThreadLocal<List<String>> thList) {  
 List<String> list = thList.get();  
 **for** (**int** i = 0; i < 5; i++)  
 list.add("Value-" + i);  
 }  
  
 **public void** remove(ThreadLocal<List<String>> thList) {  
 List<String> list = thList.get();  
 **for** (**int** i = 0; i < 2; i++)  
 list.remove(i);  
 }  
  
 **public void** show(ThreadLocal<List<String>> thList) {  
 List<String> list = thList.get();  
 **for**(String s: list)  
 System.***out***.println("Now Value : " +s);  
 }  
  
 **public void** check() **throws** InterruptedException {  
 List<String> alphabets = Arrays.*asList*("A", "B", "C", "D", "E", "F");  
 List<String> list = **new** ArrayList<>(alphabets);  
 Supplier supplier = () -> list;  
 ThreadLocal<List<String>> threadLocal = ThreadLocal.*withInitial*(supplier);  
  
 Thread t1 = **new** Thread(() -> add(threadLocal));  
 Thread t2 = **new** Thread(() -> remove(threadLocal));  
 Thread t3 = **new** Thread(() -> show(threadLocal));  
  
 t1.start();  
 t2.start();  
 t3.start();  
  
 t1.join();  
 t2.join();  
 t3.join();  
 System.***out***.println("All threads completed operations ...");  
 System.***out***.println("Original List: "+list);  
 System.***out***.println("ThreadLocal List: "+threadLocal.get());

}  
  
 **public static void** main(String[] args) **throws** Exception {  
 **new** TestThreadLocal().check();  
 }  
}

**OUTOUT**

Now Value : B

Now Value : D

Now Value : E

Now Value : F

All threads completed operations ...

Original List: [B, D, E, F, Value-0, Value-1, Value-2, Value-3, Value-4]

ThreadLocal List: [B, D, E, F, Value-0, Value-1, Value-2, Value-3, Value-4]

In this case, it does not throw any exception as each thread gets a local copy of List from ThreadLocal.

**Do not use *ThreadLocal* with *ExecutorService***If we want to use an *ExecutorService*and submit a *Runnable*to it, using *ThreadLocal*will yield non-deterministic results – because we do not have a guarantee that every *Runnable*action for a given *userId* will be handled by the same thread every time it is executed.