**Task 1 - Das Experiment**

**Cost**: 1 points.

Create HashMap<Integer, Integer>. The first thread adds elements into the map, the other go along the given map and sum the values. Threads should work before catching ConcurrentModificationException. Try to fix the problem with ConcurrentHashMap and Collections.synchronizedMap(). What has happened after simple Map implementation exchanging? How it can be fixed in code? Try to write your custom ThreadSafeMap with synchronization and without. Run your samples with different versions of Java (6, 8, and 10, 11) and measure the performance. Provide a simple report to your mentor.

**Java Map**

**HashMapIntegers**

* Using HashMap in a multi-threading environment where is 1 thread is reading and another is writing causing **ConcurrentModificationException**
* If we use **Thread.sleep()** between the Add Thread and the Sum **Thread** the exception will not occur.

**SynchronizedMapIntegers**

* Using **synchronizedMapin** a multi-threading environment where 1 thread is reading and another is writing causing **ConcurrentModificationException** in case if you have a lot of objects without the use of **Thread.sleep()** between writing thread and reading thread.
* objects with the use of **Thread.sleep()** between writing thread and reading thread will not cause the exception if the objects are not a lot, because the reading Thread will be sleeping until the writing Thread finishes it is execution.

**ConcurrentHashMapIntegers**

* Using **ConcurrentHashMap** Solved the **ConcurrentModificationException,** but we have a wrong result each time.

**Custom Map**

**ThreadSafeMapWithSync**

* Using **ThreadSafeMapWithSync** ina multi-threading environment where 1 thread is reading and another is writing causing **ConcurrentModificationException** in case if you have a lot of objects without the use of **Thread.sleep()** between writing thread and reading thread.
* objects with the use of **Thread.sleep(100)** between writing thread and reading thread will not cause the exception if the objects are not a lot, because the reading Thread will be sleeping until the writing Thread finishes it is execution.
* It turns out that using regular for loop it works and solve the issue with **ConcurrentModificationException**, it is also fast almost 5000 MS to process 50 million records, if I increase to more record, I got Java heap space exception.

ThreadSafeMapWithoutSync

* Using **ThreadSafeMapWithoutSync** in amulti-threading environment where 1 thread is reading and another is writing causing **Infinite Loop** without the use of **Thread.sleep()** between writing thread and reading thread.
* objects without the use of **Thread.sleep(100)** between writing thread and reading in case if you have a lot of objects thread will cause **ConcurrentModificationException** and will produce a partial value.
* It turns out that using regular for loop it works and solve the issue with **ConcurrentModificationException**, it is slow almost 12000 MS to process 50 million records (Maybe that is due to the use of Lock object), if I increase to more record, I got Java heap space exception.
* **For the custom implementations the difference between time of execution not noticeable when you increase the sleep time with increasing the objects amount.**
* **Increasing the sleep time between the Add Thread and the Sum Thread will lead to correct calculation without any exception.**