Example on

Multithreading

Callable

AtomicReference

BigInteger

Factorial

StringJoiner

CompletableFuture

FutureTask

IntStream

*public class* AtomicBigInteger {  
  
 *private final* AtomicReference<BigInteger> valueHolder = *new* AtomicReference<>();  
  
 *public* AtomicBigInteger(BigInteger bigInteger) {  
 valueHolder.set(bigInteger);  
 }  
  
 *public* AtomicBigInteger multiplyAndGet(BigInteger bigInteger) {  
 *for* (; ; ) {  
 BigInteger current = valueHolder.get();  
 BigInteger next = current.multiply(bigInteger);  
 *if* (valueHolder.compareAndSet(current, next)) {  
 *return this*;  
 }  
 }  
 }  
  
 *public* BigInteger get() {  
 *return* valueHolder.get();  
 }  
}

*// Simplied with CompletableFuture*

*public class* MyThread2 {  
  
 @Getter  
 *private* AtomicBigInteger factorial = *new* AtomicBigInteger(BigInteger.*ONE*);  
 *private* StringJoiner sj = *new* StringJoiner(" x ", "", " = ");  
  
 *public* BigInteger process(Integer number) {  
 factorial = factorial.multiplyAndGet( BigInteger.*valueOf*(number));  
*// synchronized(sj) {* sj.add(number.toString());  
*// }* System.*out*.println(sj.toString() + factorial.get());  
  
 *return* factorial.get();  
 }  
  
 *public static void* main(String[] args) {  
  
  
 MyThread2 myThread2 = *new* MyThread2();  
 *List*<Integer> range = *IntStream*.*rangeClosed*(1, 10).boxed().toList();  
  
*// range.stream().parallel().forEachOrdered( i -> myThread2.process(i));  
  
  
 List*<BigInteger> listInts = range.stream().parallel().map(item ->  
 CompletableFuture.*supplyAsync*(() -> myThread2.process(item)))  
 .map(CompletableFuture::join).toList();  
  
  
 System.*out*.println( myThread2.getFactorial().get());  
  
  
  
 }  
  
  
}

*// Lousy method*

*public class* MyThread *implements Callable*<BigInteger> {  
  
 *private int* number;  
*// BigInteger factorial = BigInteger.ONE;  
 private* AtomicBigInteger factorial = *new* AtomicBigInteger(BigInteger.*ONE*);  
  
 *public* MyThread(*int* number) {  
 *this*.number = number;  
 }  
  
 *public* BigInteger call() *throws* Exception {  
 StringJoiner sj = *new* StringJoiner(" x ", "", " = ");  
 *for* (Integer counter = number; counter > 0; counter--) {  
 factorial = factorial.multiplyAndGet( BigInteger.*valueOf*(counter));  
 sj.add( counter.toString());  
 System.*out*.println(sj.toString() + factorial.get());  
 }  
  
 *return* factorial.get();  
 }  
  
 *public static void* main(String[] args) {  
 MyThread myThread = *new* MyThread(100);  
 FutureTask<BigInteger> futureTask = *new* FutureTask<>(myThread);  
 futureTask.run();  
  
 *try* {  
 System.*out*.println(futureTask.get());  
 } *catch* (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

1 x 3 x 5 x 4 x 9 x 10 x 6 x 7 = 453600

1 x 3 = 6

1 x 3 x 5 x 4 x 9 = 1080

1 x 3 = 6

1 x 3 x 5 x 4 x 9 x 10 x 6 = 64800

1 x 3 x 5 x 4 = 120

1 x 3 x 5 x 4 x 9 x 10 = 10800

1 x 3 x 5 x 4 x 9 x 10 x 6 x 7 x 8 = 3628800

1 x 3 x 5 = 30

1 x 3 = 6

3628800