Program Structures and Algorithms

Spring 2023(SEC – 1)

NAME: Venkat Pavan Munaganti

NUID: 002722397

Task: Assignment 3: **Benchmark**

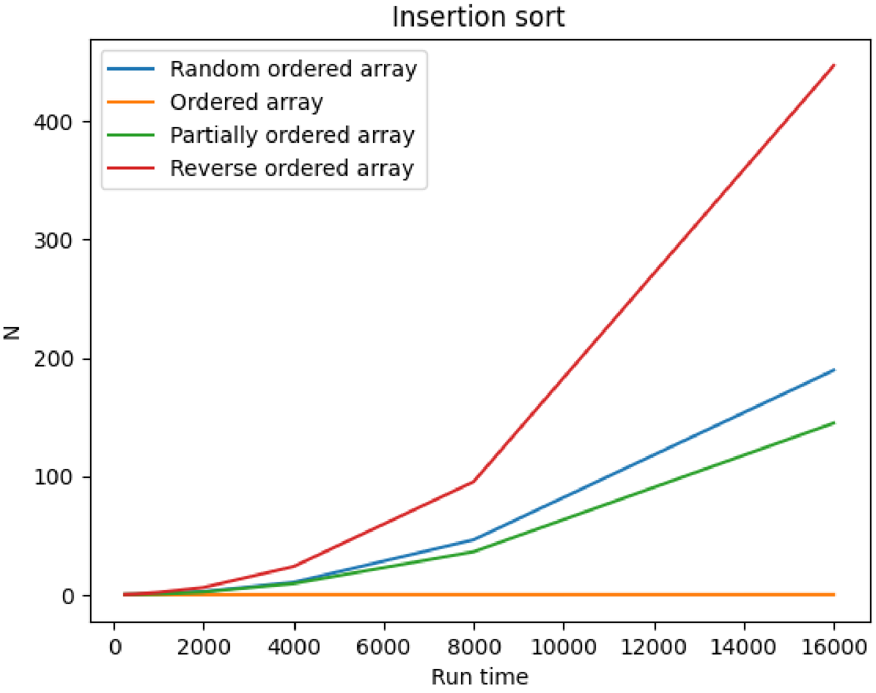
(Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*. Please see the skeleton class that I created in the repository. *Timer* is invoked from a class called *Benchmark_Timer* which implements the *Benchmark* interface.

(Part 2) Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort*. If you have the *instrument = true* setting in *test/resources/config.ini*, then you will need to use the *helper* methods for comparing and swapping (so that they properly count the number of swaps/compares). The easiest is to use the *helper.swapStableConditional* method, continuing if it returns true, otherwise breaking the loop. Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in *InsertionSortTest*.

(Part 3) Implement a main program (or you could do it via your own unit tests) to run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially ordered and reverse ordered. I suggest that your arrays to be sorted are of type *Integer*. Use the doubling method for choosing *n* and test for at least five values of *n.* Draw any conclusions from your observations regarding the order of growth.

Observations:

N	Random	Ordered	Partially ordered	Reverse ordered
250	0.08	0	0.05	0.1
500	0.21	0	0.16	0.49
1000	0.77	0	0.64	1.91
2000	2.59	0	2.22	6.04
4000	10.37	0.01	9.29	23.72
8000	46.32	0.02	36.24	95.34
16000	189.51	0.04	144.89	446.76

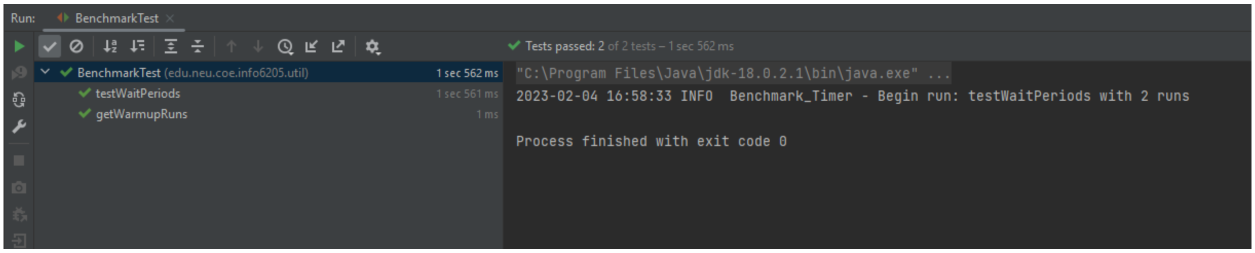


It is evident from the above graph that the best-case runtime for the insertion sort is when the array is already sorted. When the array is already sorted, insertion sort algorithm makes only $\boldsymbol{N}$ comparisons where $\boldsymbol{N}$ is no of elements. So, the best case run time will be linear i.e., $\mathbf{0}(\boldsymbol{N})$

The algorithm performs with the highest runtime when the array is sorted in reverse order. Since, it must pick each element which takes $\mathbf{0}(\boldsymbol{N})$,and for each picked element it takes $\mathbf{0}(\boldsymbol{N})$ times to search for its respective position. Hence, it makes $\mathbf{0}(\boldsymbol{N} * \boldsymbol{N})$ comparisons.

The average case or ***Big-O*** for the insertion sort algorithm is when the array is randomly or partially sorted which evident from the above runtime benchmark data and the plotted graph.

Unit tests:



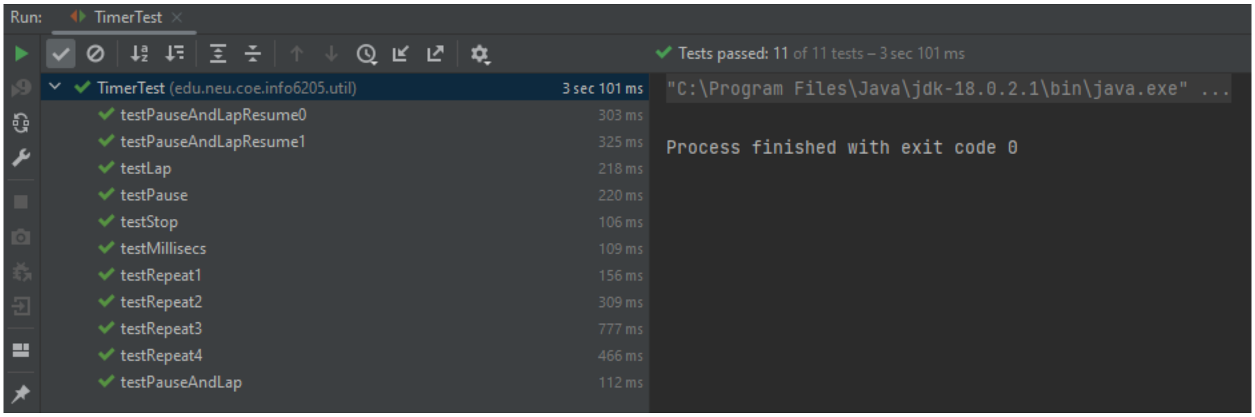
Run: BenchmarkTest

✓ Tests passed: 2 of 2 tests – 1 sec 562 ms

```
✓ BenchmarkTest (edu.neu.coe.info6205.util)   1 sec 562 ms
    ✓ testWaitPeriods                         1 sec 561 ms
    ✓ getWarmupRuns                                  1 ms
```

```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
2023-02-04 16:58:33 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

BenchmarkTest.java



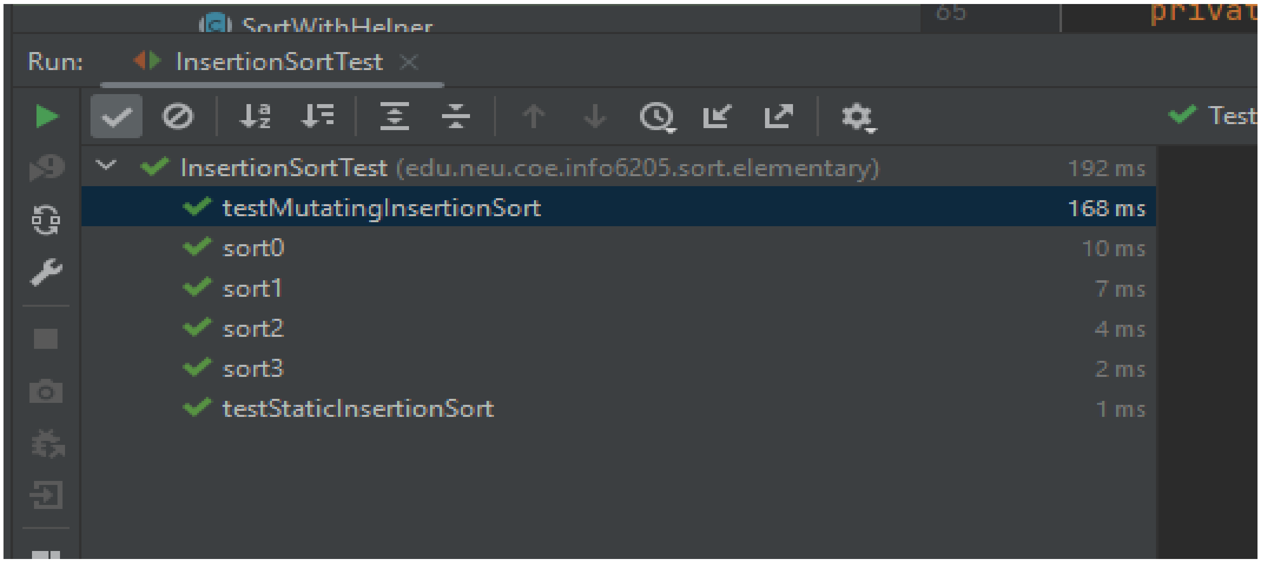
Run: TimerTest

✓ Tests passed: 11 of 11 tests – 3 sec 101 ms

```
✓ TimerTest (edu.neu.coe.info6205.util)   3 sec 101 ms
    ✓ testPauseAndLapResume0                   303 ms
    ✓ testPauseAndLapResume1                   325 ms
    ✓ testLap                                  218 ms
    ✓ testPause                                220 ms
    ✓ testStop                                 106 ms
    ✓ testMillisecs                            109 ms
    ✓ testRepeat1                              156 ms
    ✓ testRepeat2                              309 ms
    ✓ testRepeat3                              777 ms
    ✓ testRepeat4                              466 ms
    ✓ testPauseAndLap                          112 ms
```

```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...

Process finished with exit code 0
```

TimerTest.java



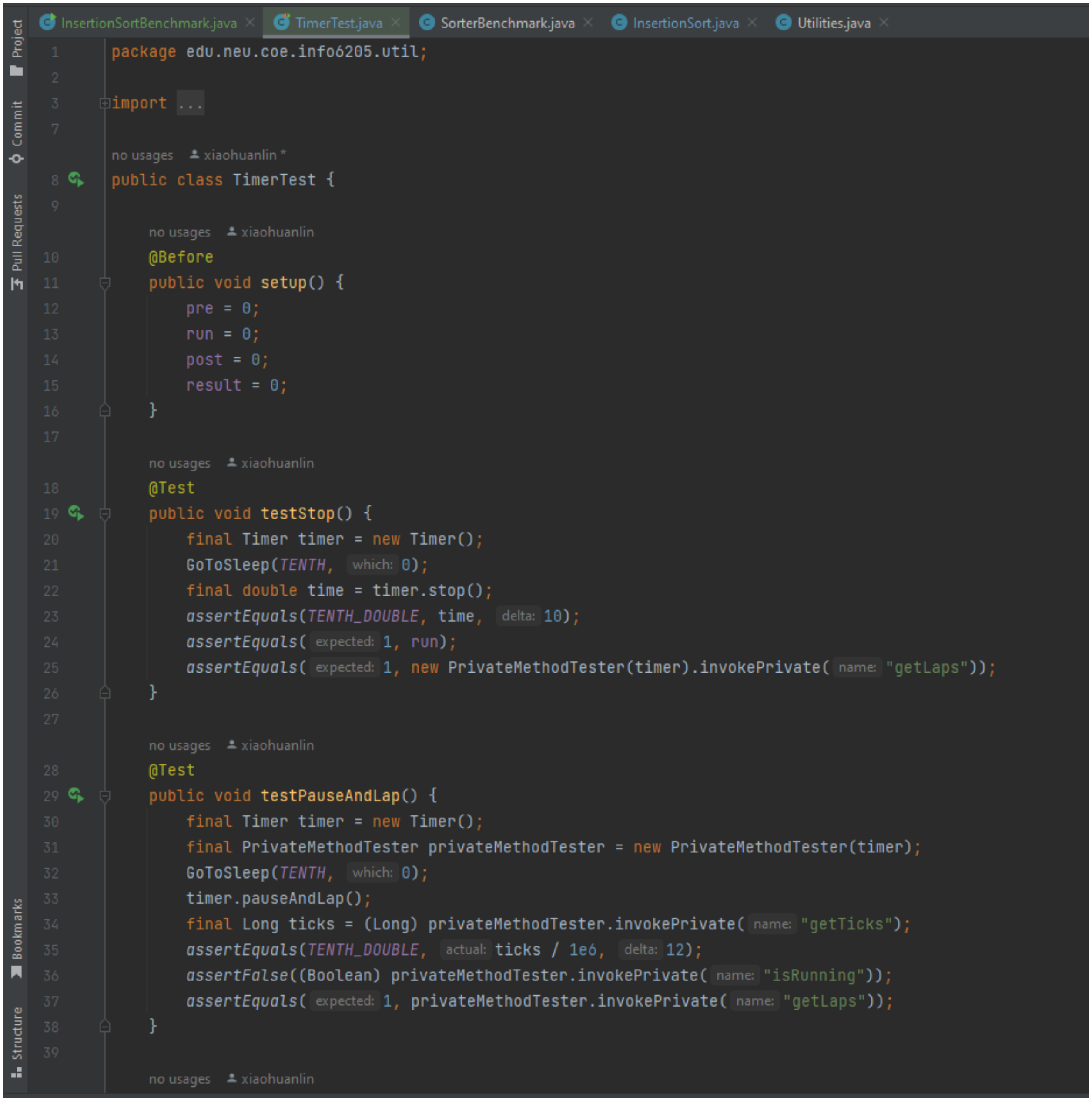
Run: InsertionSortTest

✓ Test

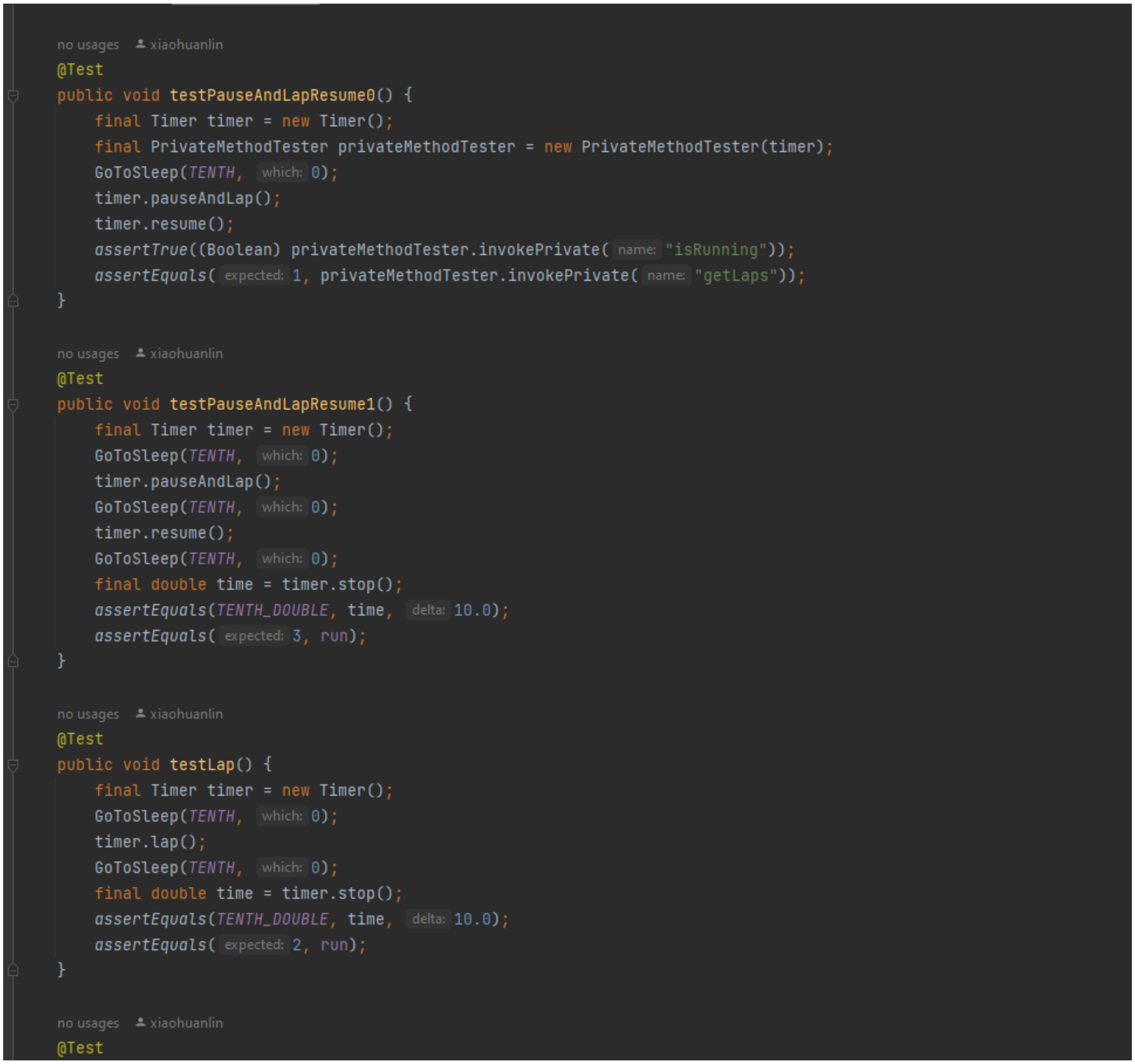
```
✓ InsertionSortTest (edu.neu.coe.info6205.sort.elementary)   192 ms
    ✓ testMutatingInsertionSort                              168 ms
    ✓ sort0                                                   10 ms
    ✓ sort1                                                    7 ms
    ✓ sort2                                                    4 ms
    ✓ sort3                                                    2 ms
    ✓ testStaticInsertionSort                                  1 ms
```

InsertionSortTest.java

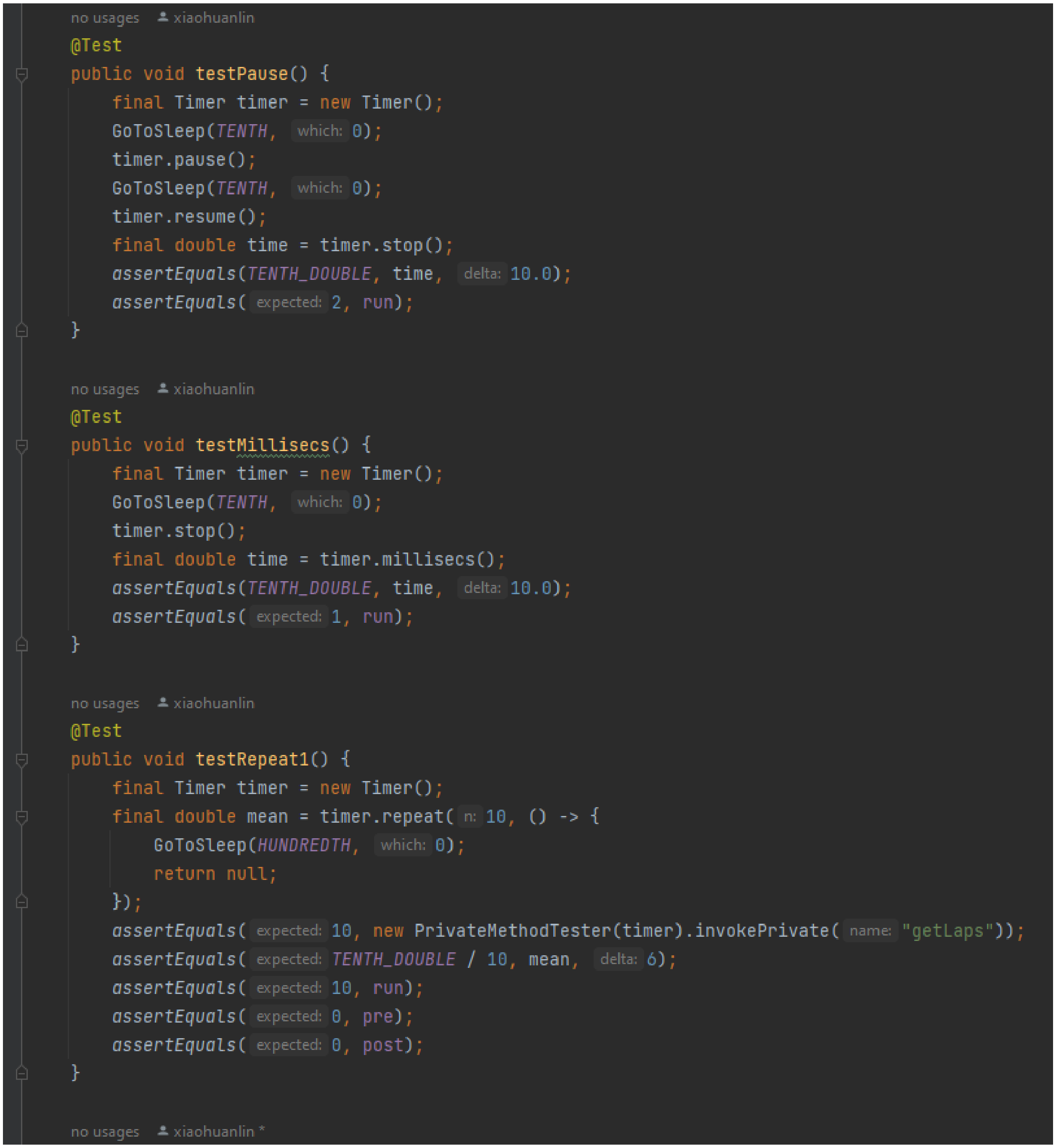
TimerTest.java



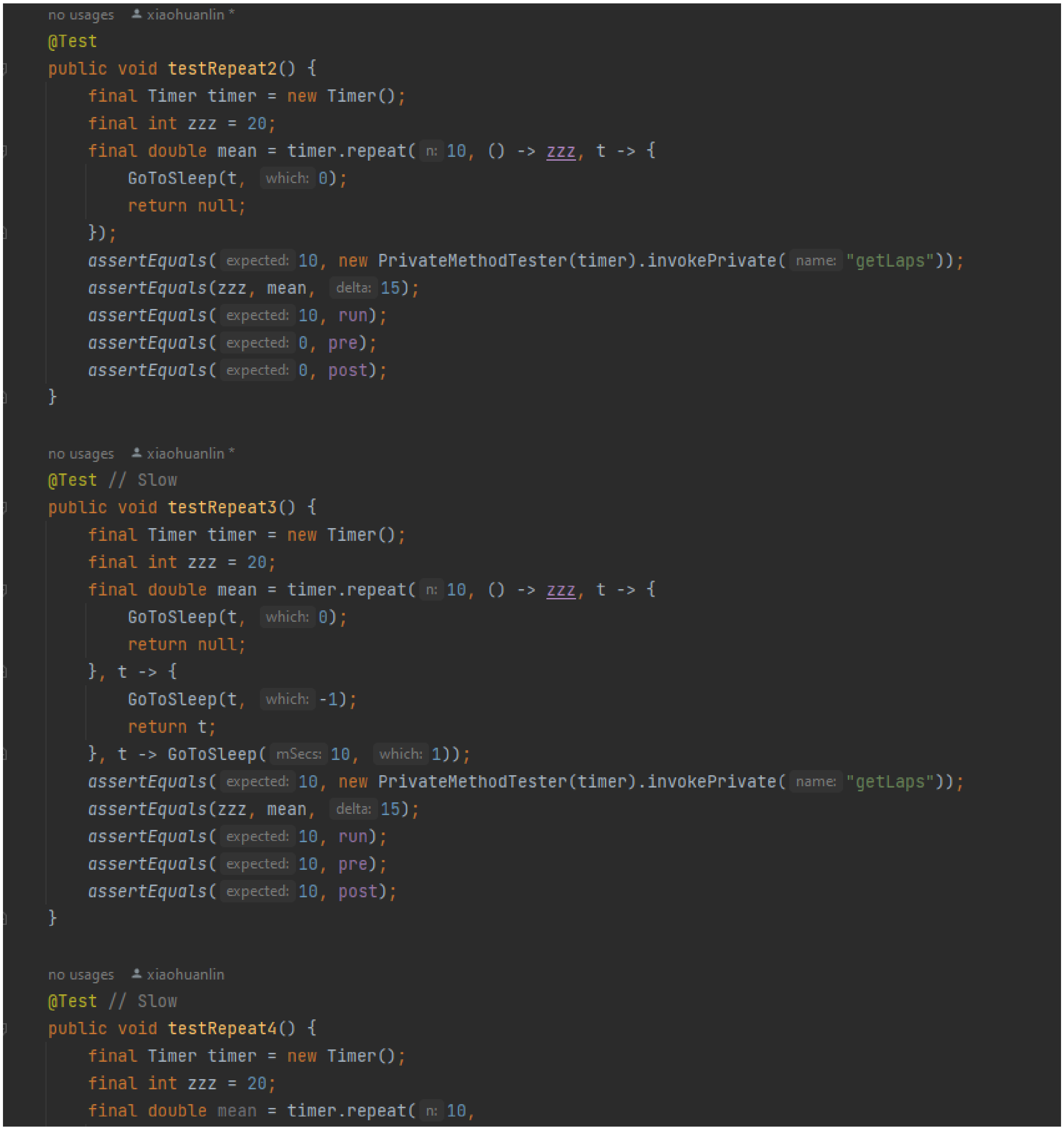
```
package edu.neu.coe.info6205.util;

import ...

no usages  xiaohuanlin *
public class TimerTest {

    no usages  xiaohuanlin
    @Before
    public void setup() {
        pre = 0;
        run = 0;
        post = 0;
        result = 0;
    }

    no usages  xiaohuanlin
    @Test
    public void testStop() {
        final Timer timer = new Timer();
        GoToSleep(TENTH,  which: 0);
        final double time = timer.stop();
        assertEquals(TENTH_DOUBLE, time,  delta: 10);
        assertEquals( expected: 1, run);
        assertEquals( expected: 1, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    }

    no usages  xiaohuanlin
    @Test
    public void testPauseAndLap() {
        final Timer timer = new Timer();
        final PrivateMethodTester privateMethodTester = new PrivateMethodTester(timer);
        GoToSleep(TENTH,  which: 0);
        timer.pauseAndLap();
        final Long ticks = (Long) privateMethodTester.invokePrivate( name: "getTicks");
        assertEquals(TENTH_DOUBLE,  actual: ticks / 1e6,  delta: 12);
        assertFalse((Boolean) privateMethodTester.invokePrivate( name: "isRunning"));
        assertEquals( expected: 1, privateMethodTester.invokePrivate( name: "getLaps"));
    }

    no usages  xiaohuanlin
```



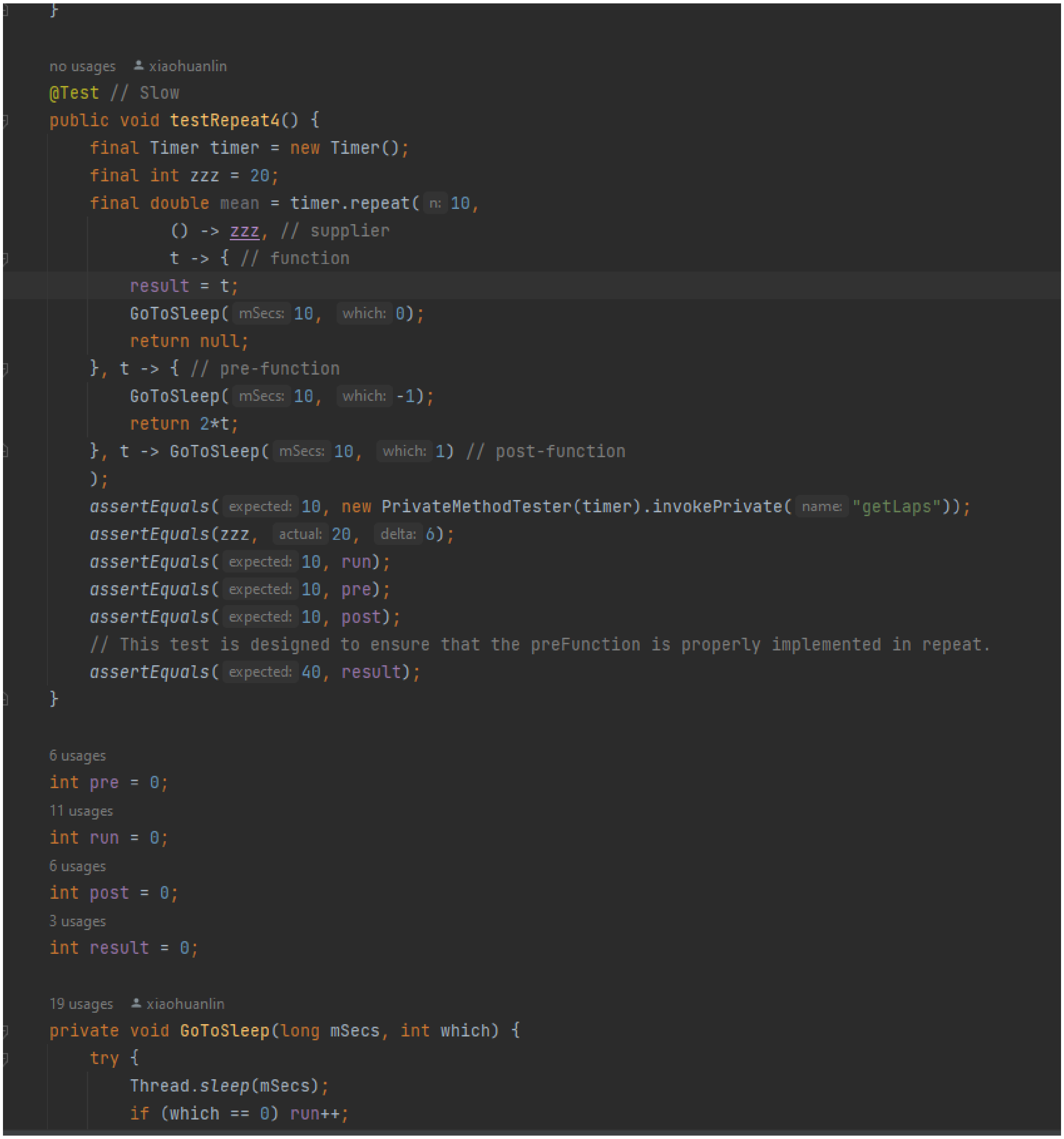
```
no usages   xiaohuanlin
@Test
public void testPauseAndLapResume0() {
    final Timer timer = new Timer();
    final PrivateMethodTester privateMethodTester = new PrivateMethodTester(timer);
    GoToSleep(TENTH,  which: 0);
    timer.pauseAndLap();
    timer.resume();
    assertTrue((Boolean) privateMethodTester.invokePrivate( name: "isRunning"));
    assertEquals( expected: 1, privateMethodTester.invokePrivate( name: "getLaps"));
}

no usages   xiaohuanlin
@Test
public void testPauseAndLapResume1() {
    final Timer timer = new Timer();
    GoToSleep(TENTH,  which: 0);
    timer.pauseAndLap();
    GoToSleep(TENTH,  which: 0);
    timer.resume();
    GoToSleep(TENTH,  which: 0);
    final double time = timer.stop();
    assertEquals(TENTH_DOUBLE, time,  delta: 10.0);
    assertEquals( expected: 3, run);
}

no usages   xiaohuanlin
@Test
public void testLap() {
    final Timer timer = new Timer();
    GoToSleep(TENTH,  which: 0);
    timer.lap();
    GoToSleep(TENTH,  which: 0);
    final double time = timer.stop();
    assertEquals(TENTH_DOUBLE, time,  delta: 10.0);
    assertEquals( expected: 2, run);
}

no usages   xiaohuanlin
@Test
```



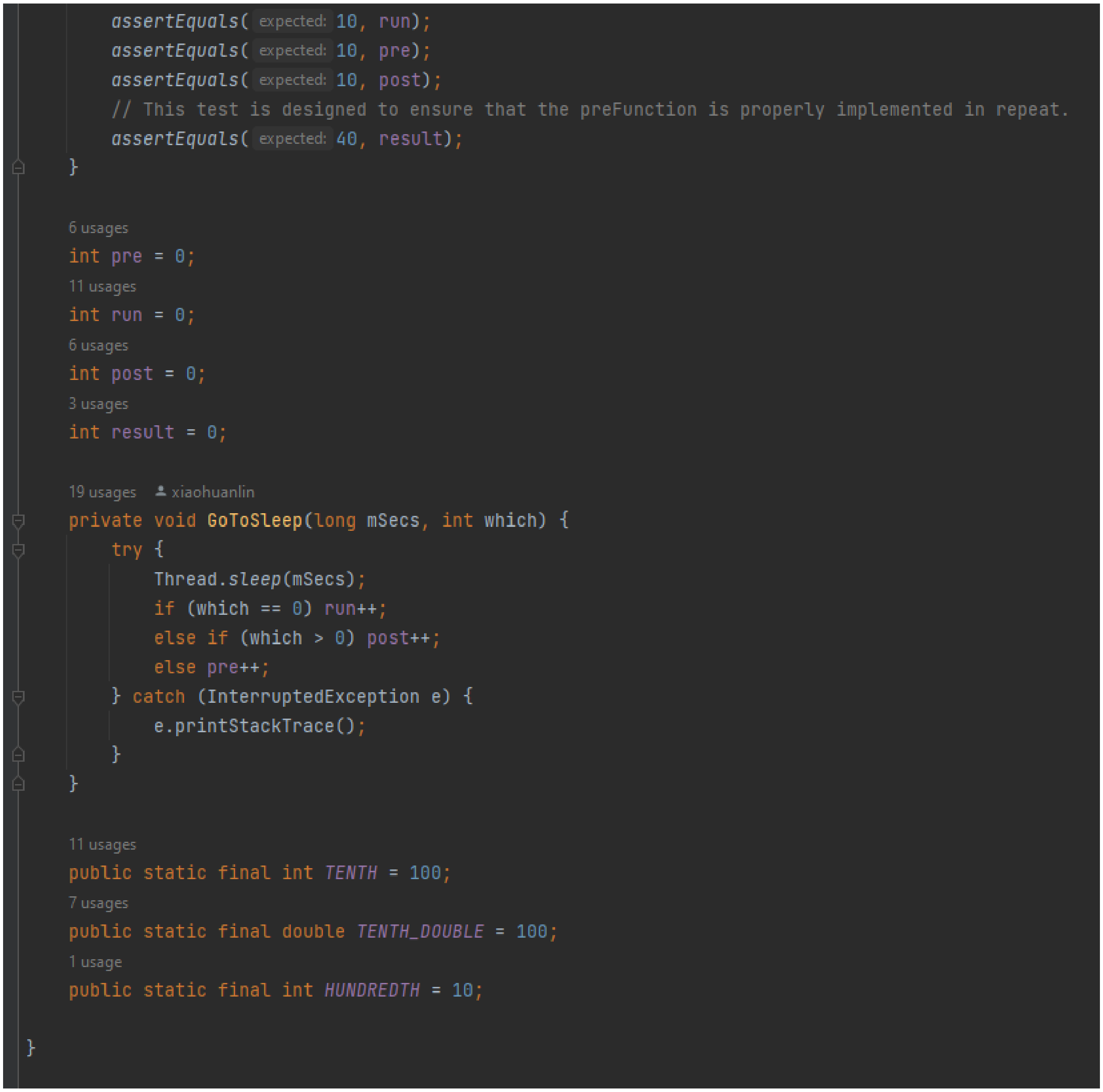
```
no usages  xiaohuanlin
@Test
public void testPause() {
    final Timer timer = new Timer();
    GoToSleep(TENTH,  which: 0);
    timer.pause();
    GoToSleep(TENTH,  which: 0);
    timer.resume();
    final double time = timer.stop();
    assertEquals(TENTH_DOUBLE, time,  delta: 10.0);
    assertEquals( expected: 2, run);
}

no usages  xiaohuanlin
@Test
public void testMillisecs() {
    final Timer timer = new Timer();
    GoToSleep(TENTH,  which: 0);
    timer.stop();
    final double time = timer.millisecs();
    assertEquals(TENTH_DOUBLE, time,  delta: 10.0);
    assertEquals( expected: 1, run);
}

no usages  xiaohuanlin
@Test
public void testRepeat1() {
    final Timer timer = new Timer();
    final double mean = timer.repeat( n: 10, () -> {
        GoToSleep(HUNDREDTH,  which: 0);
        return null;
    });
    assertEquals( expected: 10, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    assertEquals( expected: TENTH_DOUBLE / 10, mean,  delta: 6);
    assertEquals( expected: 10, run);
    assertEquals( expected: 0, pre);
    assertEquals( expected: 0, post);
}

no usages  xiaohuanlin *
```



```
no usages   xiaohuanlin *
@Test
public void testRepeat2() {
    final Timer timer = new Timer();
    final int zzz = 20;
    final double mean = timer.repeat( n: 10, () -> zzz, t -> {
        GoToSleep(t,  which: 0);
        return null;
    });
    assertEquals( expected: 10, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    assertEquals(zzz, mean,  delta: 15);
    assertEquals( expected: 10, run);
    assertEquals( expected: 0, pre);
    assertEquals( expected: 0, post);
}

no usages   xiaohuanlin *
@Test // Slow
public void testRepeat3() {
    final Timer timer = new Timer();
    final int zzz = 20;
    final double mean = timer.repeat( n: 10, () -> zzz, t -> {
        GoToSleep(t,  which: 0);
        return null;
    }, t -> {
        GoToSleep(t,  which: -1);
        return t;
    }, t -> GoToSleep( mSecs: 10,  which: 1));
    assertEquals( expected: 10, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    assertEquals(zzz, mean,  delta: 15);
    assertEquals( expected: 10, run);
    assertEquals( expected: 10, pre);
    assertEquals( expected: 10, post);
}

no usages   xiaohuanlin
@Test // Slow
public void testRepeat4() {
    final Timer timer = new Timer();
    final int zzz = 20;
    final double mean = timer.repeat( n: 10,
```

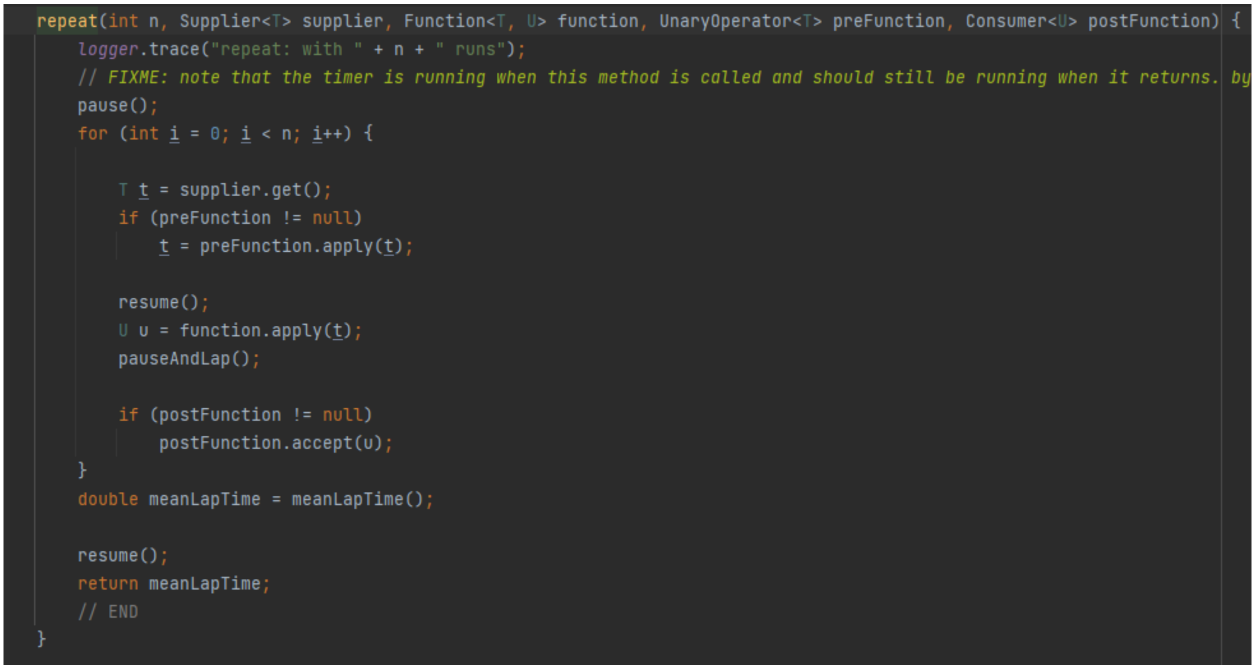


```
}

no usages   xiaohuanlin
@Test // Slow
public void testRepeat4() {
    final Timer timer = new Timer();
    final int zzz = 20;
    final double mean = timer.repeat( n: 10,
            () -> zzz, // supplier
            t -> { // function
        result = t;
        GoToSleep( mSecs: 10,  which: 0);
        return null;
    }, t -> { // pre-function
        GoToSleep( mSecs: 10,  which: -1);
        return 2*t;
    }, t -> GoToSleep( mSecs: 10,  which: 1) // post-function
    );
    assertEquals( expected: 10, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    assertEquals(zzz,  actual: 20,  delta: 6);
    assertEquals( expected: 10, run);
    assertEquals( expected: 10, pre);
    assertEquals( expected: 10, post);
    // This test is designed to ensure that the preFunction is properly implemented in repeat.
    assertEquals( expected: 40, result);
}

6 usages
int pre = 0;
11 usages
int run = 0;
6 usages
int post = 0;
3 usages
int result = 0;

19 usages   xiaohuanlin
private void GoToSleep(long mSecs, int which) {
    try {
        Thread.sleep(mSecs);
        if (which == 0) run++;
```

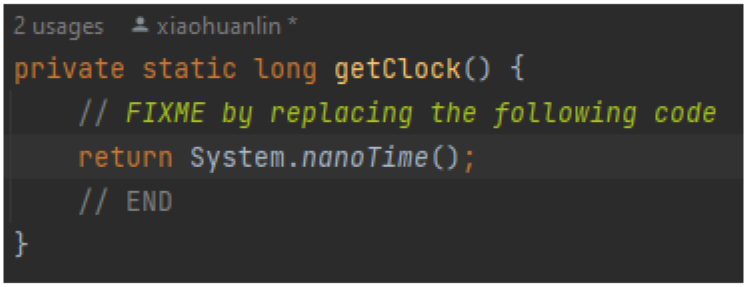


```
        assertEquals( expected: 10, run);
        assertEquals( expected: 10, pre);
        assertEquals( expected: 10, post);
        // This test is designed to ensure that the preFunction is properly implemented in repeat.
        assertEquals( expected: 40, result);
    }

    6 usages
    int pre = 0;
    11 usages
    int run = 0;
    6 usages
    int post = 0;
    3 usages
    int result = 0;

    19 usages  xiaohuanlin
    private void GoToSleep(long mSecs, int which) {
        try {
            Thread.sleep(mSecs);
            if (which == 0) run++;
            else if (which > 0) post++;
            else pre++;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    11 usages
    public static final int TENTH = 100;
    7 usages
    public static final double TENTH_DOUBLE = 100;
    1 usage
    public static final int HUNDREDTH = 10;

}
```

Timer.java

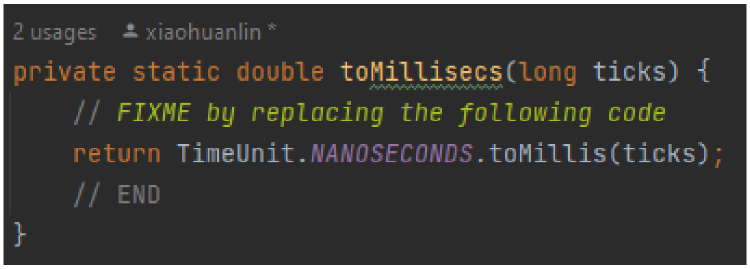


```java
repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
    logger.trace("repeat: with " + n + " runs");
    // FIXME: note that the timer is running when this method is called and should still be running when it returns. by
    pause();
    for (int i = 0; i < n; i++) {

        T t = supplier.get();
        if (preFunction != null)
            t = preFunction.apply(t);

        resume();
        U u = function.apply(t);
        pauseAndLap();

        if (postFunction != null)
            postFunction.accept(u);
    }
    double meanLapTime = meanLapTime();

    resume();
    return meanLapTime;
    // END
}
```



2 usages   xiaohuanlin *

```java
private static long getClock() {
    // FIXME by replacing the following code
    return System.nanoTime();
    // END
}
```



2 usages   xiaohuanlin *

```java
private static double toMillisecs(long ticks) {
    // FIXME by replacing the following code
    return TimeUnit.NANOSECONDS.toMillis(ticks);
    // END
}
```