**01. Answer the following question: What is a Benchmarking?**

A measurement of the quality of an organization's policies, products, programs, strategies, etc., and their comparison with standard measurements, or similar measurements of its peers.

The objectives of benchmarking are (1) to determine what and where improvements are called for, (2) to analyze how other organizations achieve their high-performance levels, and (3) to use this information to improve performance.

**02. Create Eclipse temples (3):**

**1 Second = 1,000 Milliseconds ms**

**1 Second = 1,000,000 Microseconds μs**

**1 Second = 1,000,000,000Nanoseconds ns**

**bs** (Benchmark Start)

|  |
| --- |
| **final long start = System.*nanoTime*();                      // System.*currentTimeMillis*();** |

**bf** (Benchmark Finish)

|  |
| --- |
| **final long finish = System.*nanoTime*();                     // System.*currentTimeMillis*();** |

**br** (Benchmark Result)

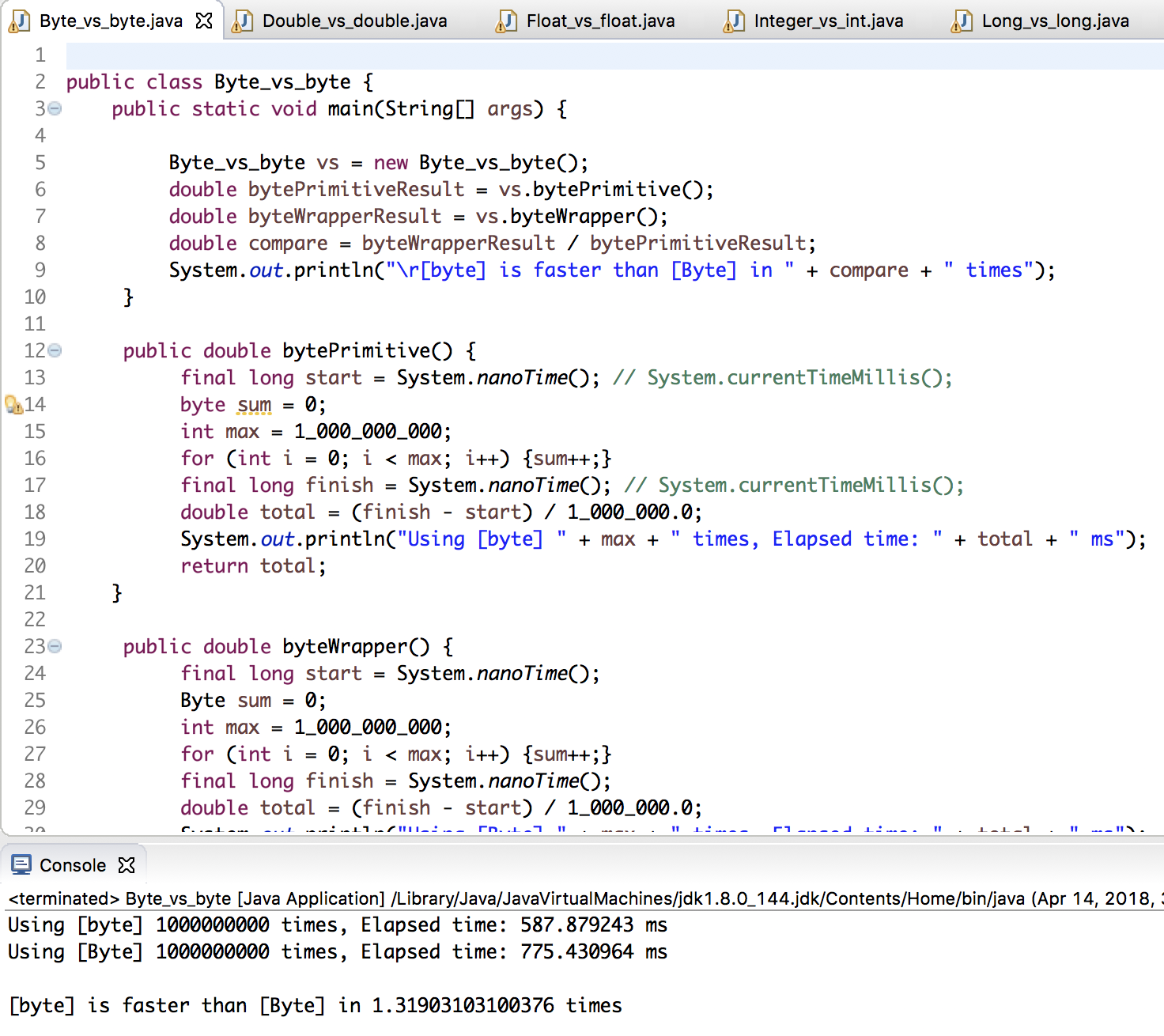
|  |
| --- |
| **System.*out*.println((finish - start) + " ns");              // Nanoseconds** |

or

|  |
| --- |
| **System.*out*.println((finish - start) / 1\_000\_000.0 + " ms"); // Milliseconds** |

**03. Provide performance comparison between byte vs. Byte by adding the value to itself**

**1,000,000,000 times. How much one faster than the other?**



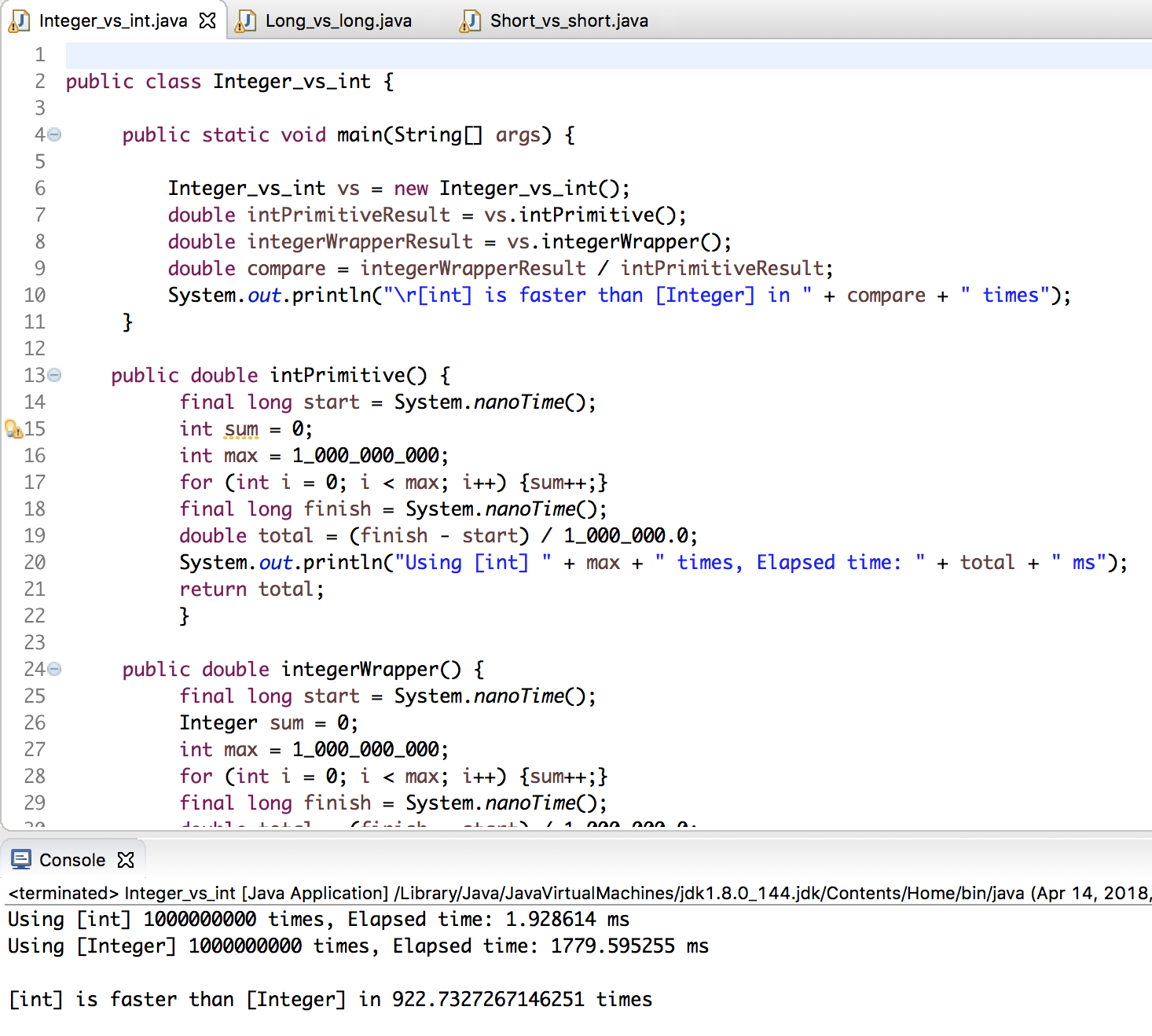
**04. Provide performance comparison between short vs. Short by adding the value to itself**

**1,000,000,000 times. How much one faster than the other?**

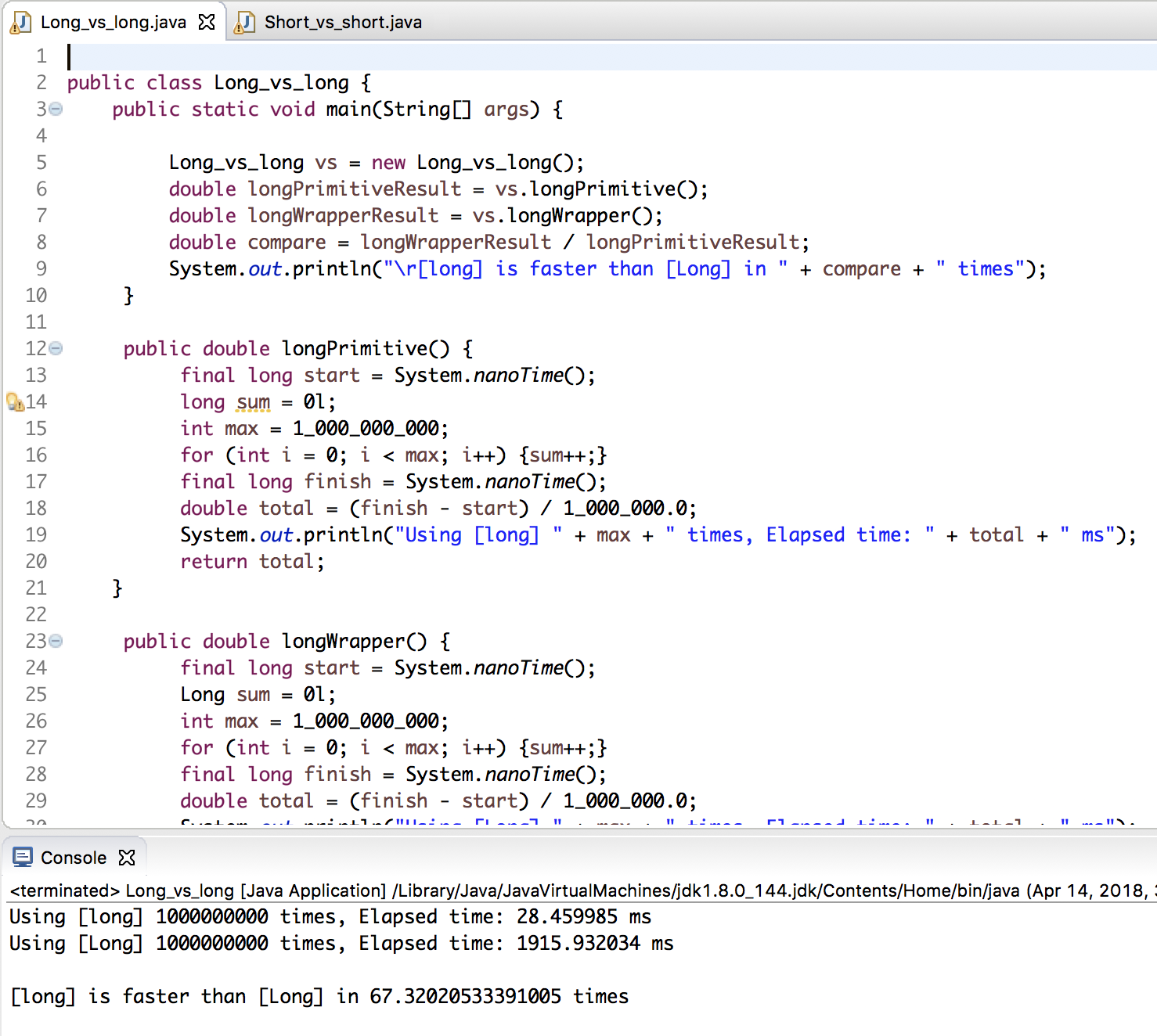


**05. Provide performance comparison between int vs. Integer by adding the value to itself**

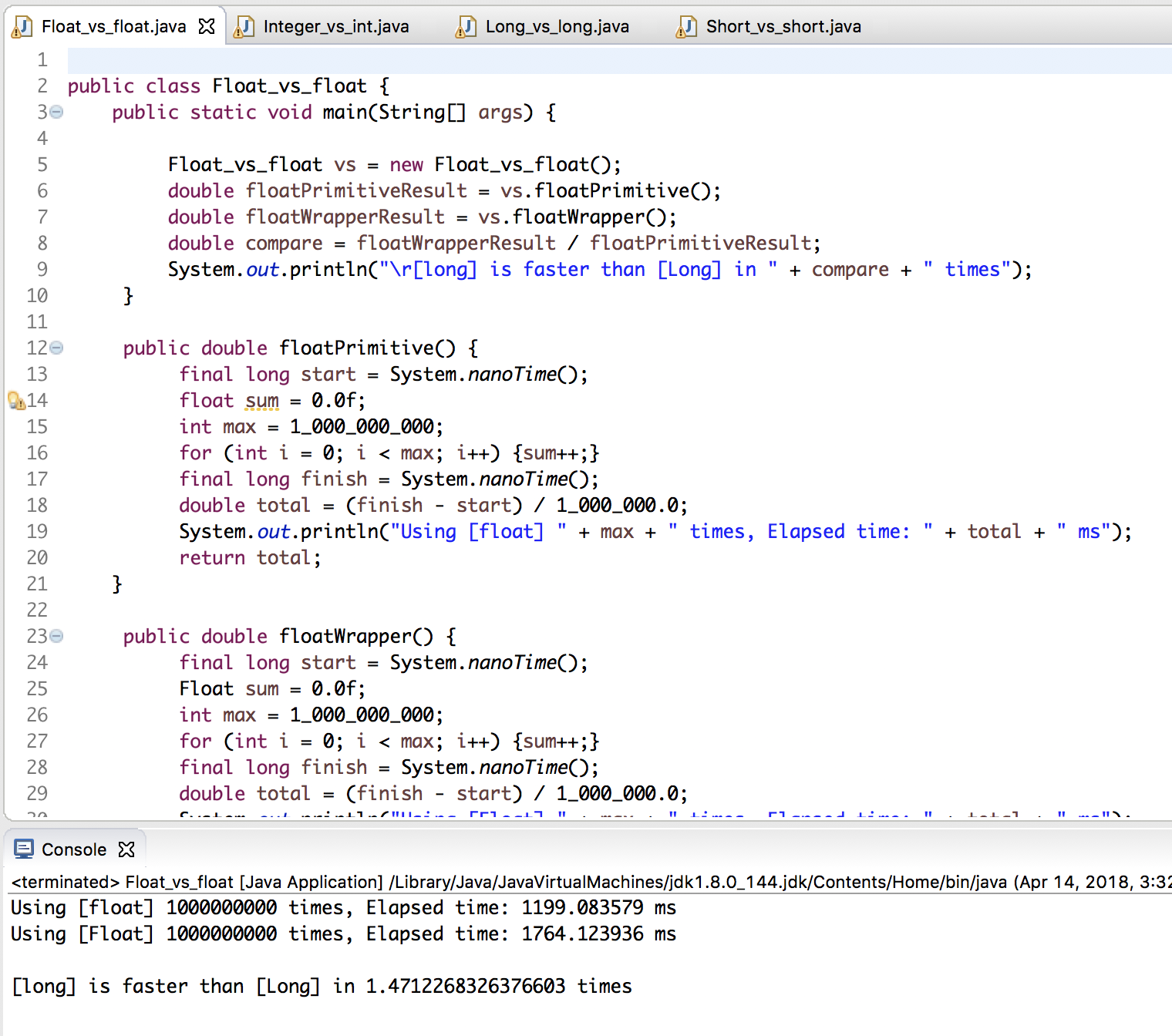
**1,000,000,000 times. How much one faster than the other?**



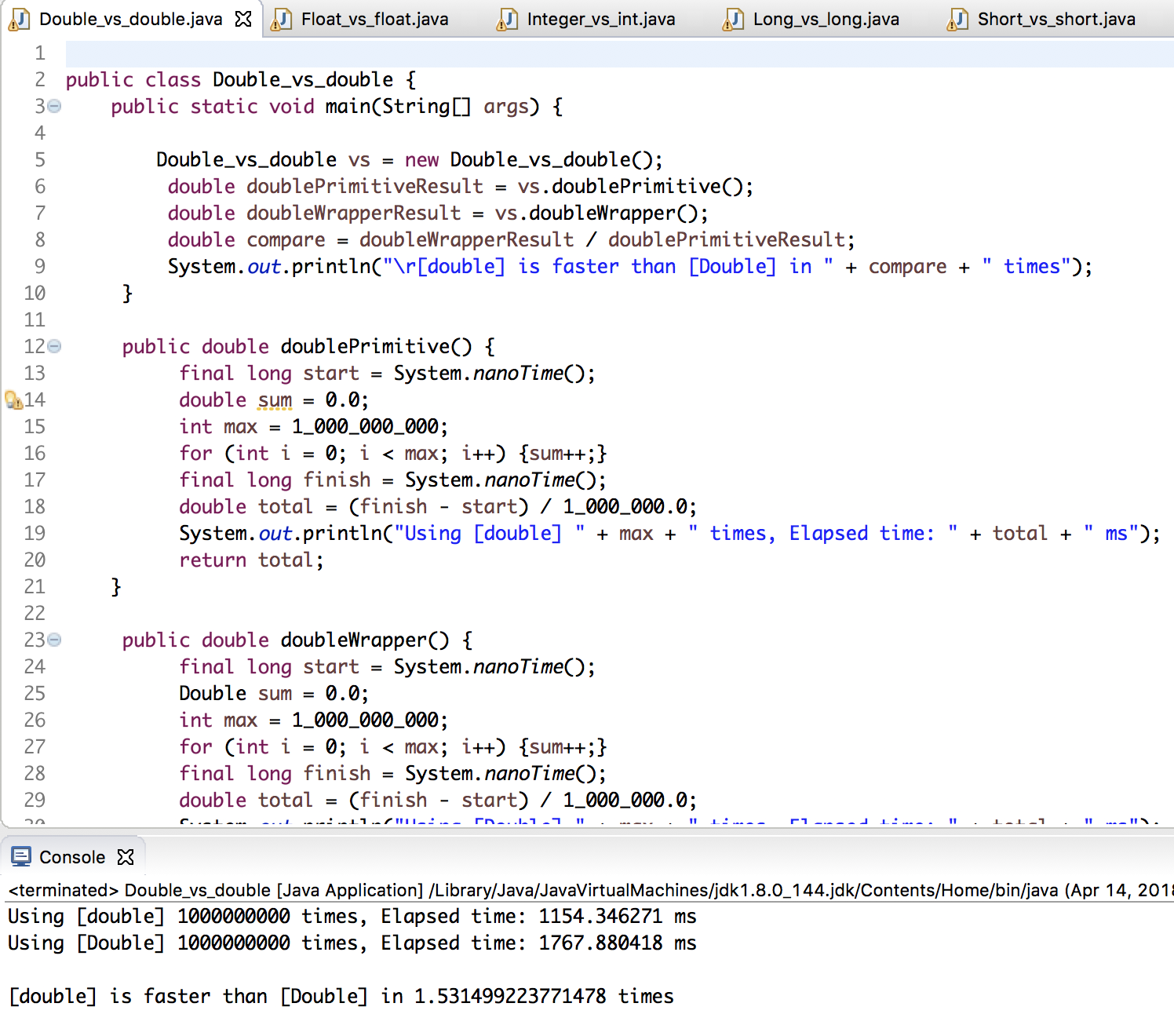
**06. Provide performance comparison between long vs. Long by adding the value to itself 1,000,000,000 times. How many times one faster than another?**



**07. Provide performance comparison between float vs. Float by adding the value to itself 1,000,000,000 times. How much one faster than the other?**



**08. Provide performance comparison between double vs. Double by adding the value to itself 1,000,000,000 times. How much one faster than the other?**



**09. Answer the following question: What are the differences between String and StringBuffer and StringBuilder in Java?**

1. String is immutable whereas StringBuffer and StringBuider are mutable classes.
2. StringBuffer is thread safe and synchronized whereas StringBuilder is not, thats why [StringBuilder is more faster than StringBuffer](https://www.journaldev.com/137/stringbuffer-vs-stringbuilder).
3. String concat + operator internally uses StringBuffer or StringBuilder class.
4. For String manipulations in non-multi threaded environment, we should use StringBuilder else use StringBuffer class.

### StringBuffer vs StringBuilder

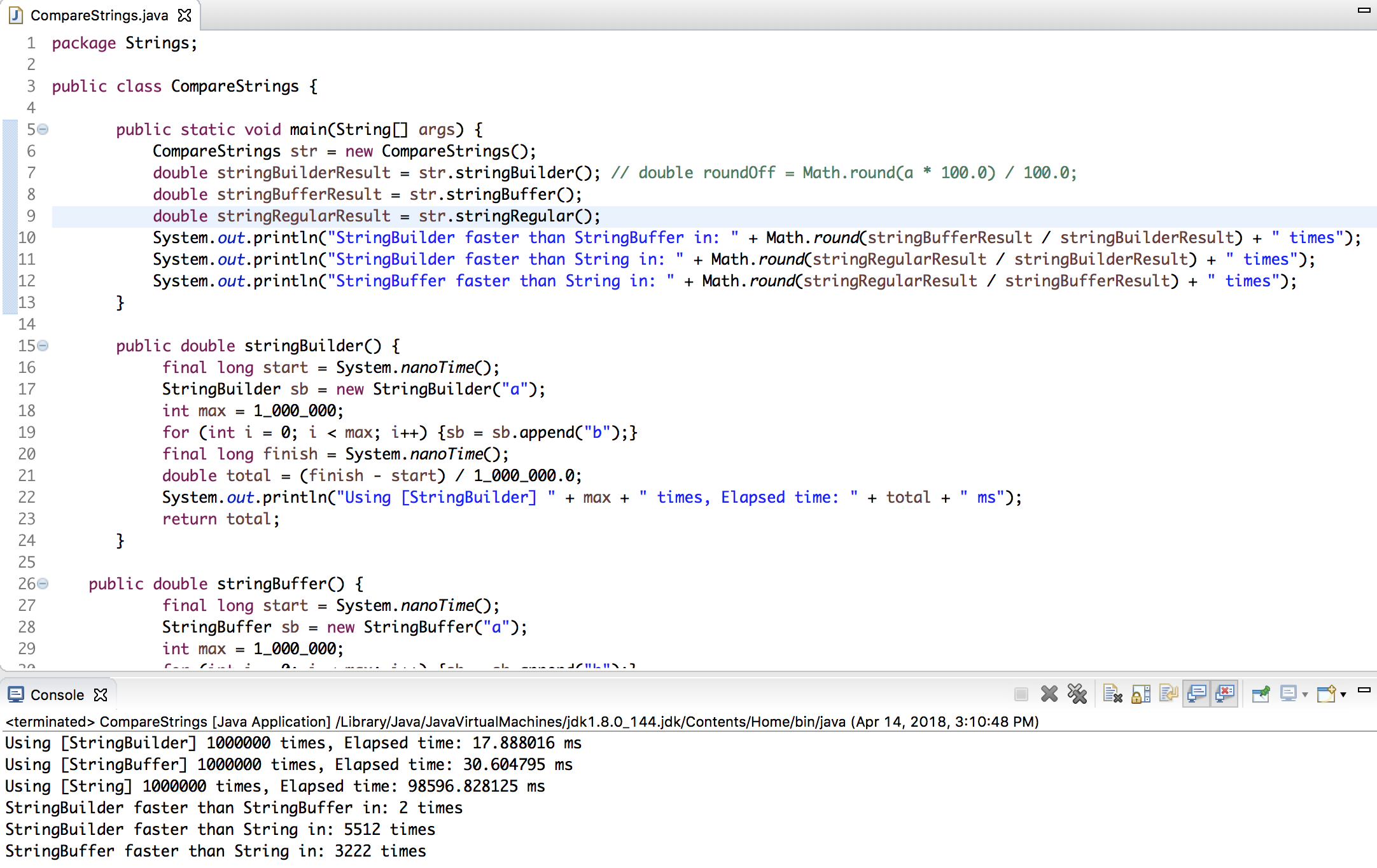
StringBuffer was the only choice for String manipulation till Java 1.4 but it has one disadvantage that all of its public methods are synchronized. StringBuffer provides Thread safety but on a performance cost.

In most of the scenarios, we don’t use String in multithreaded environment, so Java 1.5 introduced a new class StringBuilder that is similar with StringBuffer except thread safety and synchronization.

So, if you are in a single threaded environment or don’t care about thread safety, you should use StringBuilder else use StringBuffer.

**10. Provide performance comparison between String vs. StringBuffer vs. StringBuilder by**

**concatenating (appending) string "a" and "b" 1,000,000 times. How much one faster than the other?**



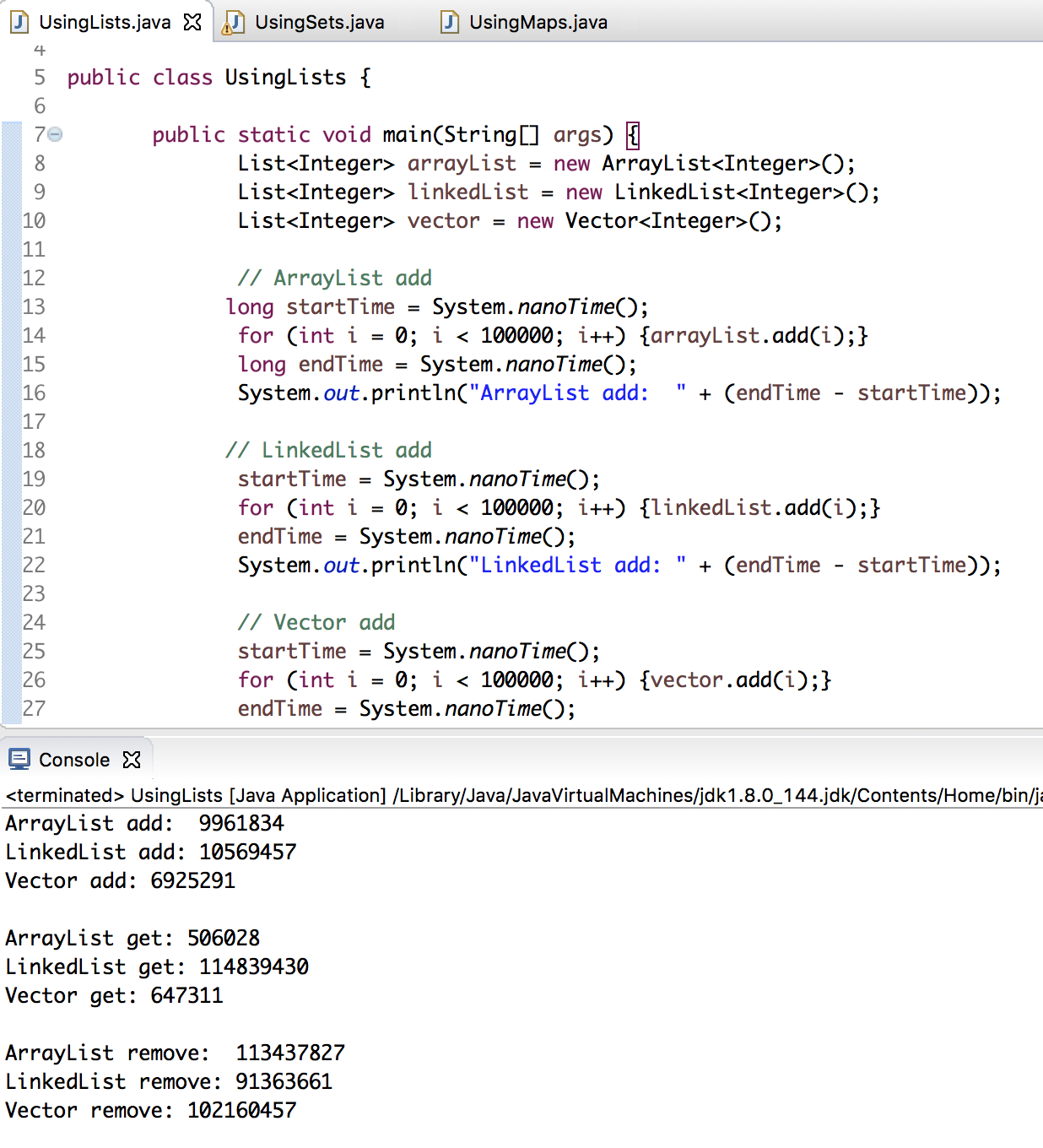
**11. Provide performance comparison between Vector vs. LinkedList vs.**[**ArrayList**](http://www.learn2test.net/cec/vol/lab/java_01/ArrayList.html)**by:**

**add 1,000,000 items (Integer - 1, 2, 3, …)**

**get 1,000,000 items**

**remove 1,000,000 items,**

**How much one faster than the other**?



**12. Answer the following question: What are the differences between List and Set in Java?**

here are few noteworthy *differences between List and Set in Java*. Remember that both of them are used to store objects and provides convenient API to insert, remove and retrieve elements, along with to support Iteration over collection.  
  
1) Fundamental difference between List and Set in Java is **allowing duplicate elements**. List in Java allows duplicates while **Set doesn't allow any duplicate**. If you insert duplicate in Set it will replace the older value. Any implementation of Set in Java will only contains unique elements.

2) Another significant difference between List and Set in Java is order. **List is an Ordered Collection** while Set is an unordered Collection. List maintains **insertion order of elements**, means any element which is inserted before will go on lower index than any element which is inserted after. Set in Java doesn't  maintain any order. Though Set provide another alternative called SortedSet which can store Set elements in specific Sorting order defined by [Comparable and Comparator](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) methods of Objects stored in Set.

3) Set uses [equals() method](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) to check uniqueness of elements stored in Set, while SortedSet uses [compareTo() method](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) to implement natural sorting order of elements. In order for an element to behave properly in Set and SortedSet, [equals and compareTo must be consistent](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) to each other.

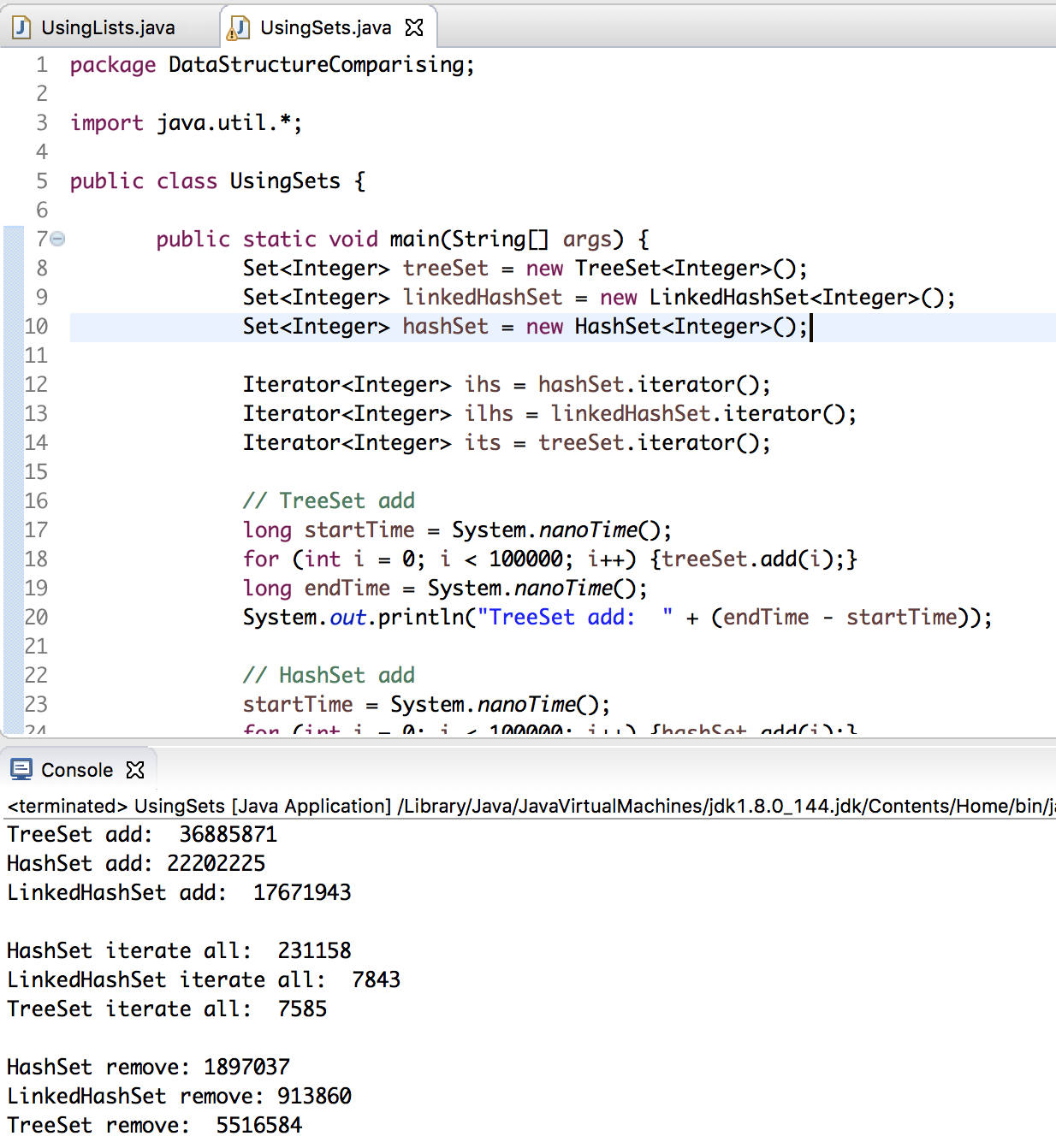
4) Popular implementation of List interface in Java includes ArrayList, Vector and LinkedList. While popular implementation of Set interface includes HashSet, TreeSet and LinkedHashSet.

**13. Provide performance comparison between LinkedHashSet vs. HashSet vs.**[**TreeSet**](http://www.learn2test.net/cec/vol/lab/java_01/TreeSet.html)**by:**

**add 1,000,000 items (Integer - 1, 2, 3, …)**

**iterate 1,000,000 items**

**remove 1,000,000 items, How much one faster than the other?**



**14. Answer the following question: What are the differences between: HashMap, Hashtable, LinkedHashMap, TreeMap in Java?**

* HashMap is implemented as a hash table, and there is no ordering on keys or values.
* TreeMap is implemented based on red-black tree structure, and it is ordered by the key.
* LinkedHashMap preserves the insertion order
* Hashtable is synchronized, in contrast to HashMap. It has an overhead for synchronization.

This is the reason that HashMap should be used if the program is thread-safe.

## **When to use LinkedHashMap, TreeMap, and HashMap**

You can use a **LinkedHashMap** when you need to keep your mappings in either **insertion order** or **access order**. LinkedHashMap by default keeps elements in the order, on which they are inserted, and this order is reflected when you [traverse over LinkedHashMap](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html), but it also provides a constructor, which allows you to keep entries in *access order*, the. order in which they are accessed. One of the clever use of Java LinkedHashMap is to use it as Least Recently Use or **LRU Cache**.  
  
**TreeMap** is your go to map implementation if you want to keep keys  in a sorted order, either in their natural order defined by Comparable interface or a custom order imposed by Comparator interface, though it's worth remembering that your compareTo() or compare() method must be [consistent with equals() method](http://java67.blogspot.sg/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html), because Map interface is defined in terms of equals and TreeMap uses compareTo for comparing keys. So if keys compare() or compareTo() implementation is not consistent, then it will fail to obey Map's general contract.  
  
**HashMap** is your general purpose hashing based collection, whenever you need to use a hash table data structure in Java to store key-value pairs, the first choice goes to HashMap in a single threaded environment. If you happened to use a Map in a multi-threaded environment consider using [Hashtable, synchronized HashMap or ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) from Java Collection Framework.  
  
Since LinkedHashMap solved the problem of chaotic ordering provided by Hashtable and HashMap, without incurring the high cost associated with TreeMap, you can also use LinkedHashMap to create a copy of a Map in Java, as shown in below example.

**15. Provide performance comparison between:**[**HashMap**](http://www.learn2test.net/cec/vol/lab/java_01/HashMap.html)**vs. Hashtable vs. LinkedHashMap vs.  TreeMap by:**

**put 1,000,000 items, where key is an int (1, 2, 3, …) and value is your First and Last name.**

**get 1,000,000 items**

**remove all (1,000,000) items. How much one faster than the other?**

