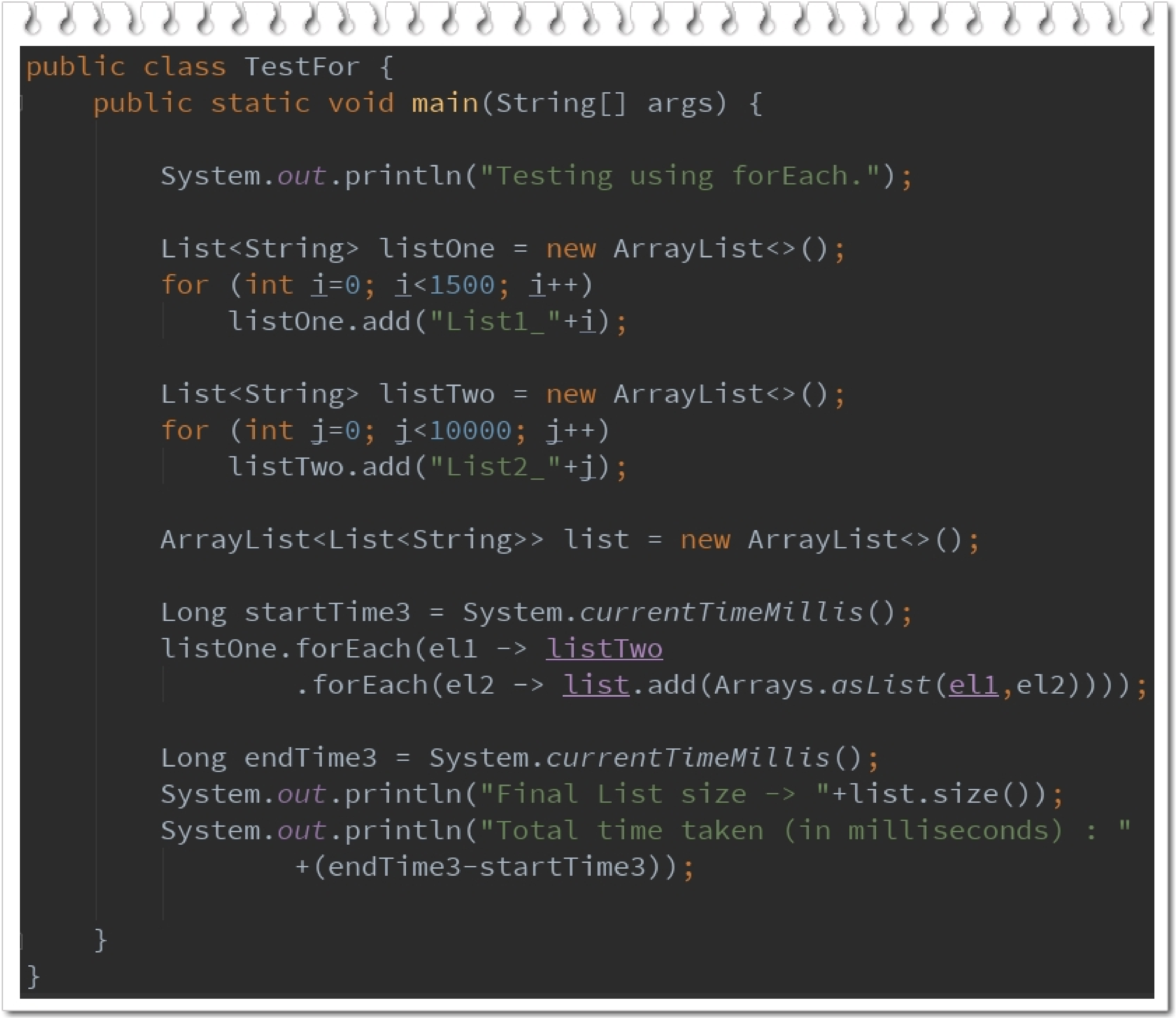
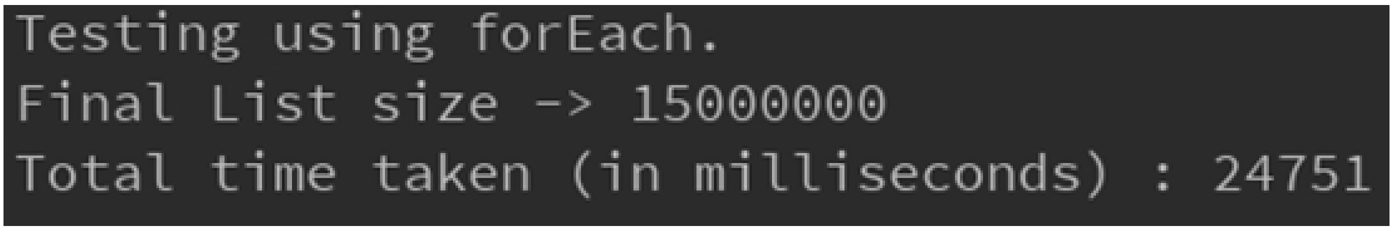
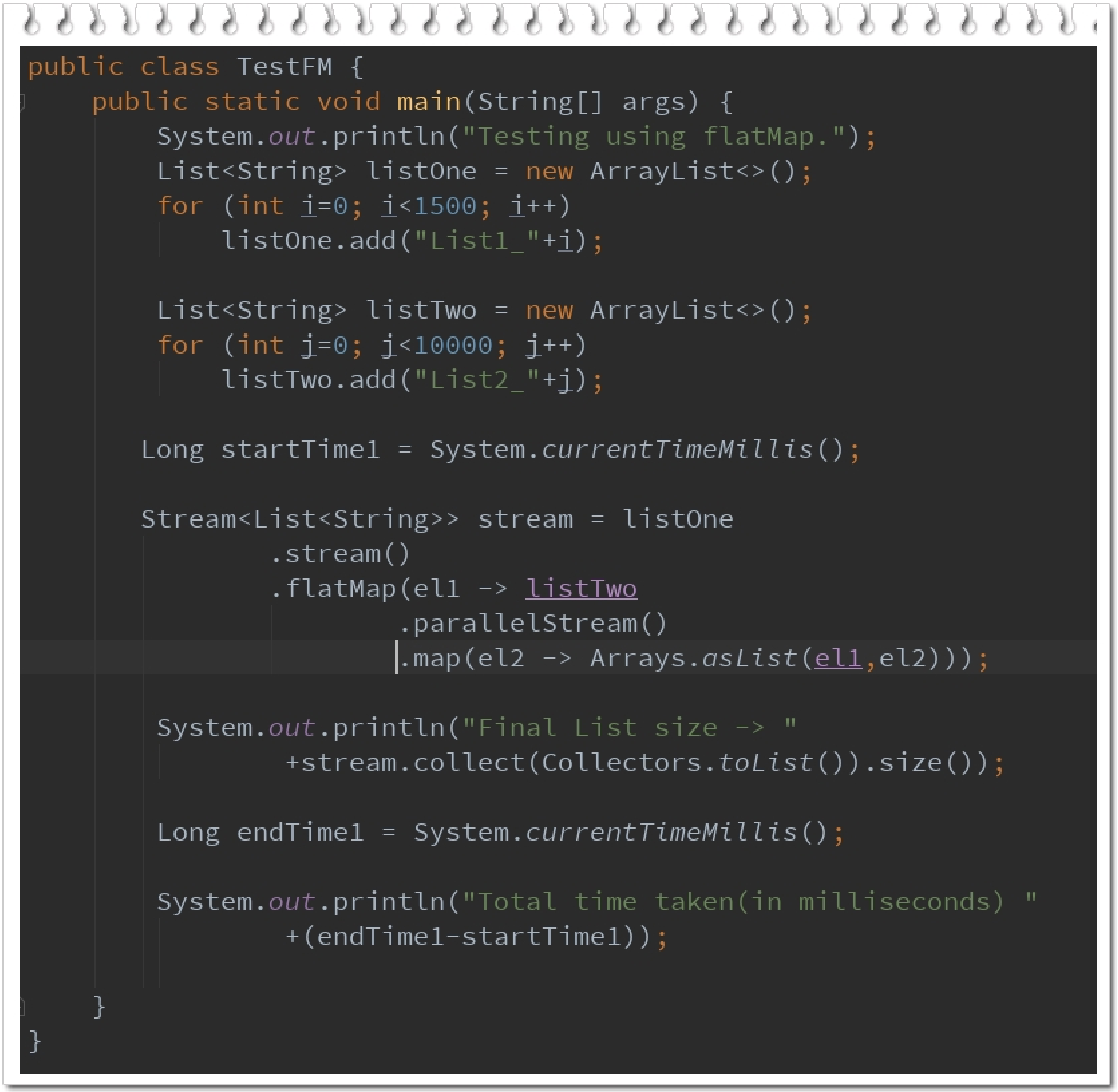


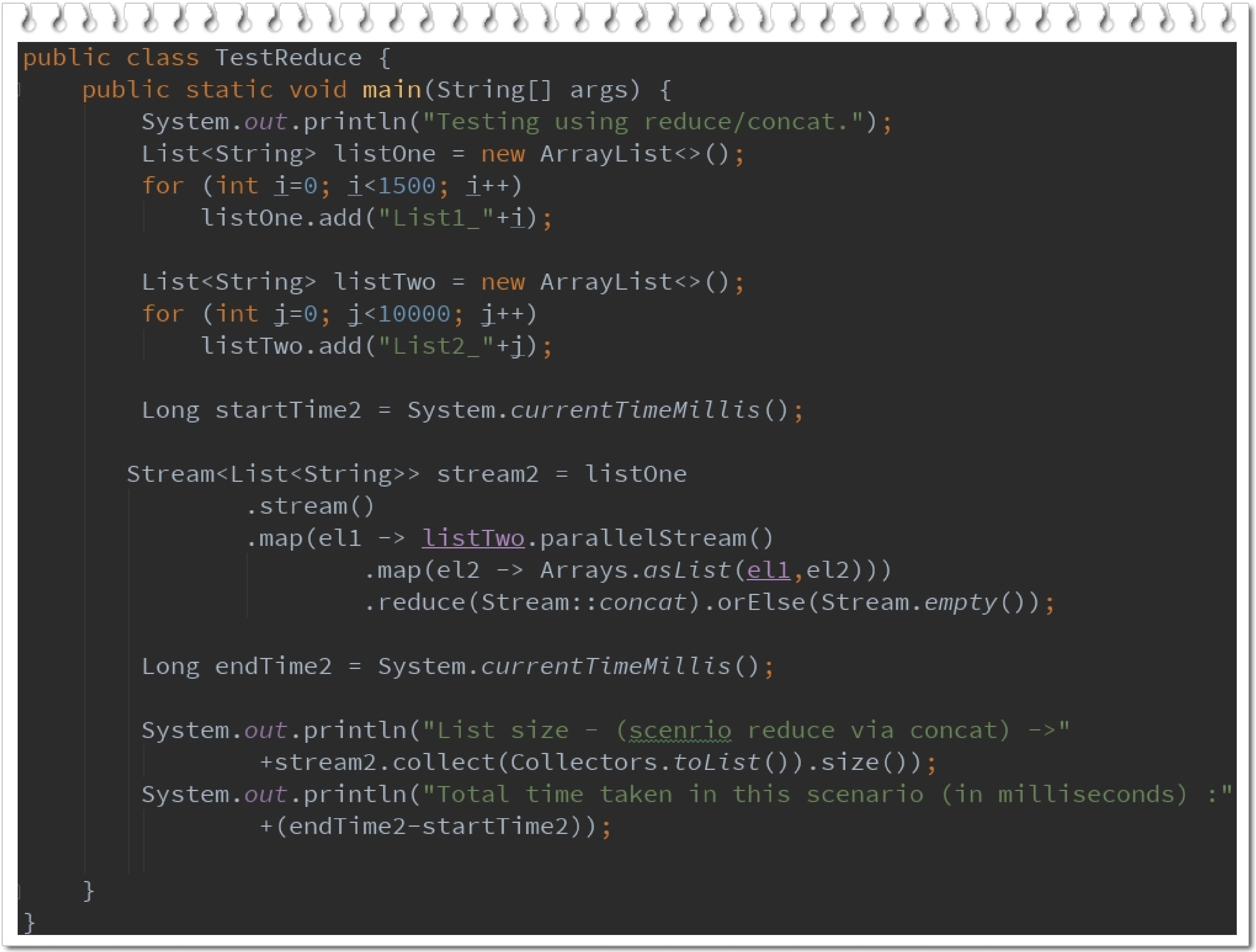
```java
public class TestFor {
    public static void main(String[] args) {

        System.out.println("Testing using forEach.");

        List<String> listOne = new ArrayList<>();
        for (int i=0; i<1500; i++)
            listOne.add("List1_"+i);

        List<String> listTwo = new ArrayList<>();
        for (int j=0; j<10000; j++)
            listTwo.add("List2_"+j);

        ArrayList<List<String>> list = new ArrayList<>();

        Long startTime3 = System.currentTimeMillis();
        listOne.forEach(el1 -> listTwo
                .forEach(el2 -> list.add(Arrays.asList(el1,el2))));

        Long endTime3 = System.currentTimeMillis();
        System.out.println("Final List size -> "+list.size());
        System.out.println("Total time taken (in milliseconds) : "
                +(endTime3-startTime3));

    }
}
```

When we performed cross join between 2 lists using forEach, we got below output.


```
Testing using forEach.
Final List size -> 15000000
Total time taken (in milliseconds) : 24751
```

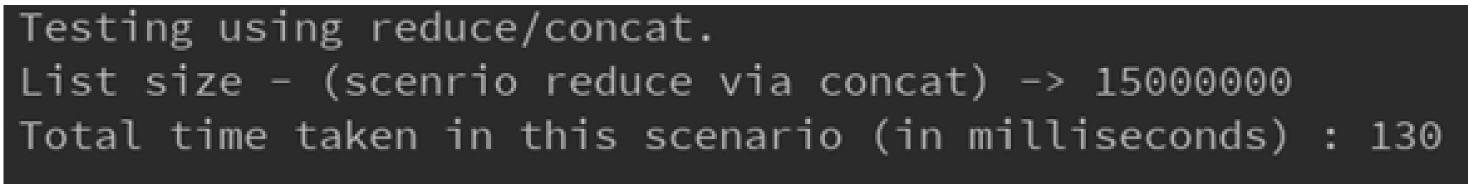


```java
public class TestFM {
    public static void main(String[] args) {
        System.out.println("Testing using flatMap.");
        List<String> listOne = new ArrayList<>();
        for (int i=0; i<1500; i++)
            listOne.add("List1_"+i);

        List<String> listTwo = new ArrayList<>();
        for (int j=0; j<10000; j++)
            listTwo.add("List2_"+j);

       Long startTime1 = System.currentTimeMillis();

       Stream<List<String>> stream = listOne
               .stream()
               .flatMap(el1 -> listTwo
                       .parallelStream()
                       .map(el2 -> Arrays.asList(el1,el2)));

        System.out.println("Final List size -> "
                +stream.collect(Collectors.toList()).size());

        Long endTime1 = System.currentTimeMillis();

        System.out.println("Total time taken(in milliseconds) "
                +(endTime1-startTime1));

    }
}
```



```java
public class TestReduce {
    public static void main(String[] args) {
        System.out.println("Testing using reduce/concat.");
        List<String> listOne = new ArrayList<>();
        for (int i=0; i<1500; i++)
            listOne.add("List1_"+i);

        List<String> listTwo = new ArrayList<>();
        for (int j=0; j<10000; j++)
            listTwo.add("List2_"+j);

        Long startTime2 = System.currentTimeMillis();

       Stream<List<String>> stream2 = listOne
                .stream()
                .map(el1 -> listTwo.parallelStream()
                        .map(el2 -> Arrays.asList(el1,el2)))
                        .reduce(Stream::concat).orElse(Stream.empty());

        Long endTime2 = System.currentTimeMillis();

        System.out.println("List size - (scenrio reduce via concat) ->"
                +stream2.collect(Collectors.toList()).size());
        System.out.println("Total time taken in this scenario (in milliseconds) :"
                +(endTime2-startTime2));

    }
}
```

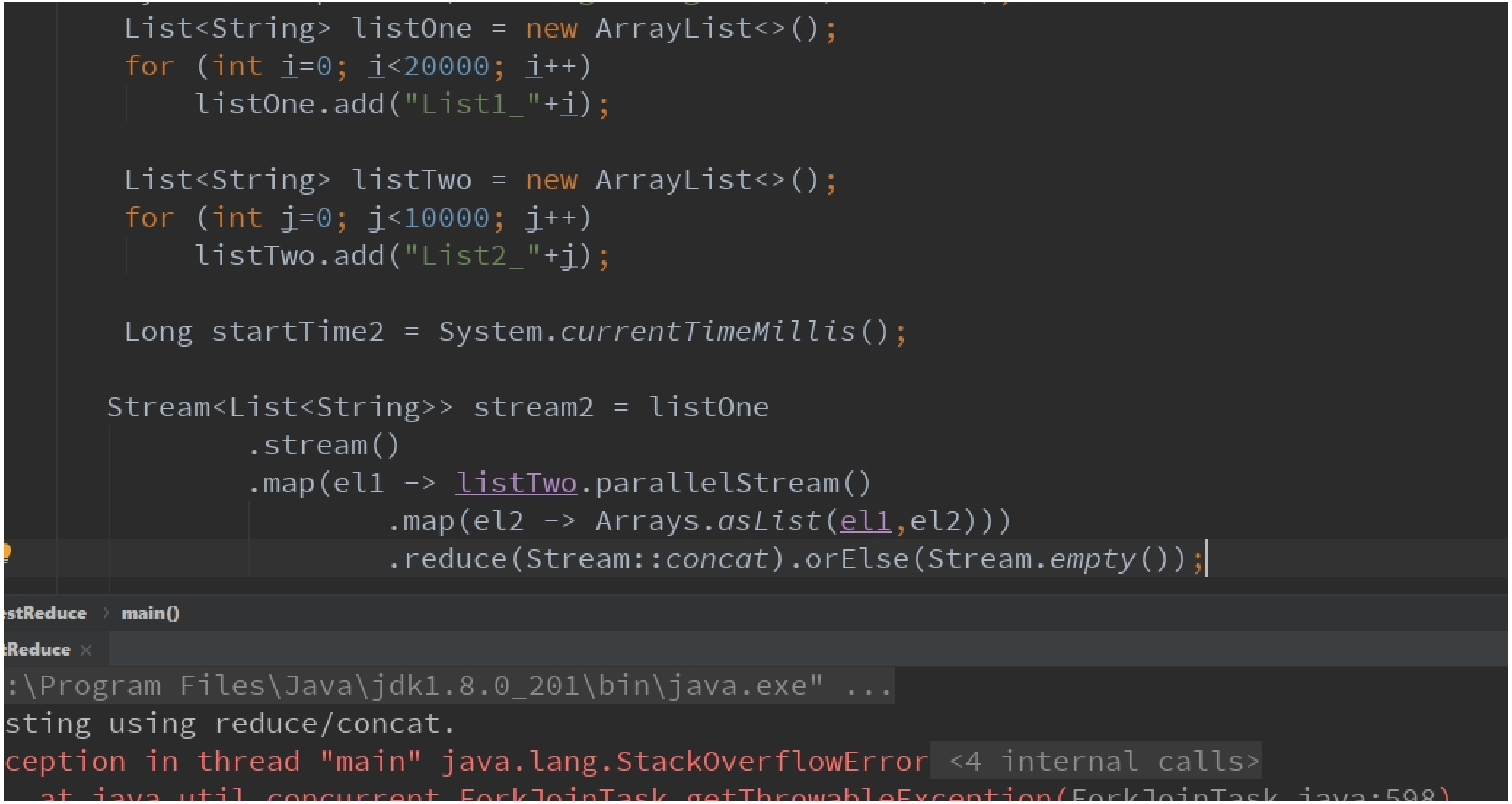
When we performed cross join between 2 lists
using map and reduce and parallel stream,
for internal stream, we got below output.



```
Testing using reduce/concat.
List size - (scenrio reduce via concat) -> 15000000
Total time taken in this scenario (in milliseconds) : 130
```

Performance wise this is what we have observed-
Reduce/Concat > flatMap or forEach

Reduce/Concat with map performed better.
But there is chance of stackoverflowerror, let's
see that in the next slide.



```
    List<String> listOne = new ArrayList<>();
    for (int i=0; i<20000; i++)
        listOne.add("List1_"+i);

    List<String> listTwo = new ArrayList<>();
    for (int j=0; j<10000; j++)
        listTwo.add("List2_"+j);

    Long startTime2 = System.currentTimeMillis();

   Stream<List<String>> stream2 = listOne
           .stream()
           .map(el1 -> listTwo.parallelStream()
                   .map(el2 -> Arrays.asList(el1,el2)))
                   .reduce(Stream::concat).orElse(Stream.empty());
```

```
:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...
sting using reduce/concat.
ception in thread "main" java.lang.StackOverflowError <4 internal calls>
```