Object Oriented Analysis and Design
CS 6359.001: Reading Assignment #4

Due on Sunday September 29, 2019 at 11:59am

Instructor: Prof.Mehra Nouroz Borazjany

Shyam Patharla **(sxp178231)**

1. Write a brief functional description for the function.
the function average() takes two parameters

- **k**, an integer
- **arr**, an array of integers

and returns the average of the first **k** elements in the **arr** array. If the value of k is greater than the size of the list, the function returns the average of the elements in the list.

2. Generate functional test cases based on functional description.
In the following table we use the variable **n** to indicate the size of the array . Input: a

Description	Input	Expected Output
Test for empty list	arr of size n=0, any k	0
Test for single element list when $k < 1$	arr of size n=1, k=0	0
Test for single element list	arr of size n=1, $k >= 1$	arr[0]
Test for $k <= n$	arr of size n, $k < n$	average of first k elements in arr
Test for $k > n$	arr of size n, $k >= n$	average of elements in arr

Table 1: Test Case Specifications

Description	Input	EO	AO	Passing Criteria	Test Result
Test for empty list	(), k=2	0	TBD	EO=AO	TBD
Test for n=1 and $k < 1$	(12), k=0	0	TBD	EO=AO	TBD
Test for n=1 and $k >= 1$	(17), k=3	17	TBD	EO=AO	TBD
Test for $k < n$	(2, 6 ,1, 15, 20, 14), k=4	6	TBD	EO=AO	TBD
Test for $k > n$	(2, 6, 1, 15, 20, 10), k=9	9	TBD	EO=AO	TBD

Table 2: Tests

* EO=Expected output, AO=Actual Output, TBD=To be determined

3. Identify and specify the partitions and generate partition test cases.
Let L be the set of all arrays and K be the set of all possible k values. The input domain will be **LxK** and consists of (list,k) tuples.

Let n = list.size()

Equivalence classes:

- Partition 1: Set of all (list, k) tuples where $k < n$
- Partition 2: Set of all (list,k) tuples where $k >= n$

For the first set, the function average() returns the average of the first k elements in the array, while for the second set, the function returns the average of all values in the array.

4. Generate boundary value test cases.
Let
nmax = maximum possible value of k
nmin = minimum possible value of n
kmax=max possible value of k
kmin=min possible value of k

boundary test cases

- Partition 1: (k=nmin-1, k=nmin, k=nmin+1, k=n-1, k=n, k=n+1) x (n=nmin-1, n=nmin,n=nmin+1, n=nmax-1, n=nmax, n=nmax+1)
- Partition 2: (k=n-1, k=n, k=n+1, k=nmax-1, k=nmax, k=nmax+1) x (n=nmin-1, n=nmin,n=nmin+1, n=nmax-1, n=nmax, n=nmax+1)

5. Implement the average function in a class Average and generate test cases using Junit.

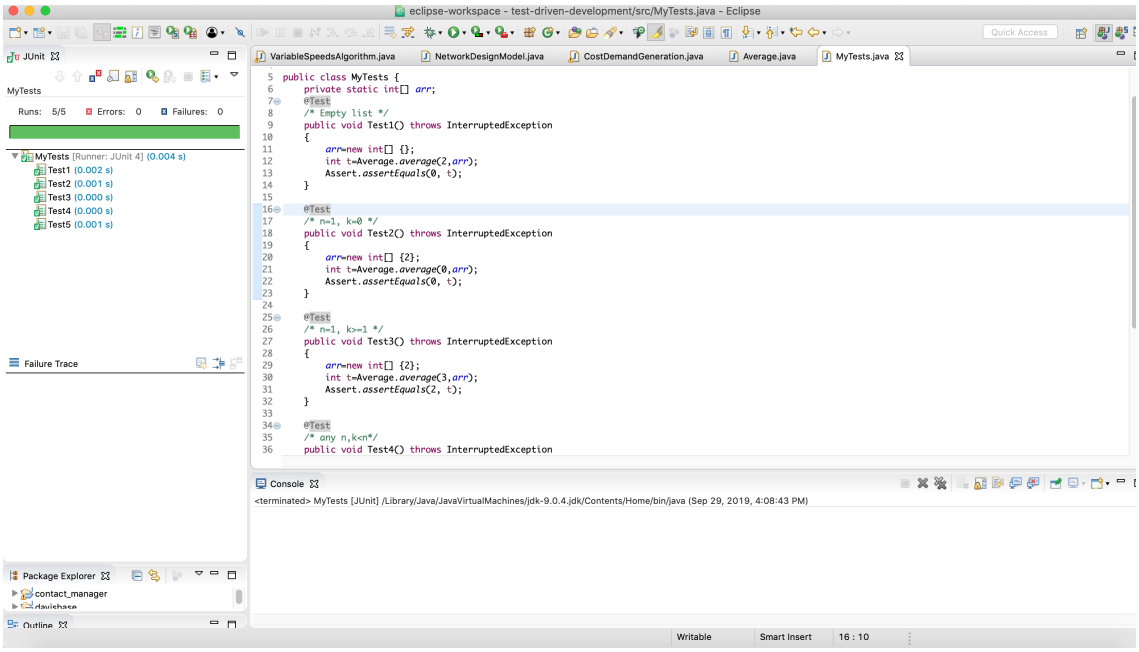
```
public class Average {


  public static int average(int k,int[] list)
  {
    int average=0;
    int n=Math.min(k,list.length);
    if(n==0 )
    {
      return 0;
    }

    for(int i=0;i<n;i++)
    {
      average+=list[i];
    }
    average=average/n;
    return average;
  }


}
```

```
import org.junit.Assert;
import org.junit.Test;
```

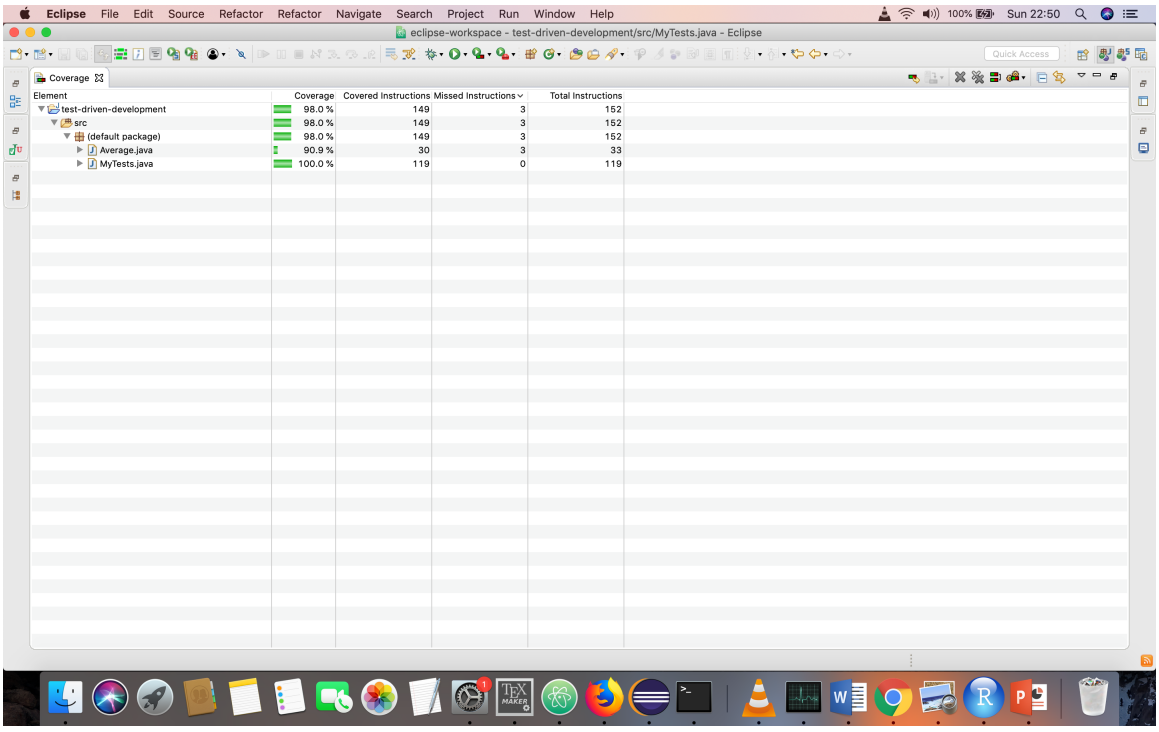
```
public class MyTests {



  private static int[] arr;
  @Test
  /* Empty list */
  public void Test1() throws InterruptedException
  {
    arr=new int[] {};
    int t=Average.average(2,arr);
    Assert.assertEquals(0, t);
  }

  @Test
  /* n=1, k=0 */
  public void Test2() throws InterruptedException
  {
    arr=new int[] {2};
    int t=Average.average(0,arr);
    Assert.assertEquals(0, t);
  }

  @Test
  /* n=1, k>=n */
  public void Test3() throws InterruptedException
  {
    arr=new int[] {2};
    int t=Average.average(3,arr);
    Assert.assertEquals(2, t);
  }

  @Test
  /* any n,k<n*/
  public void Test4() throws InterruptedException
  {
    arr=new int[] {2,6,1,15,20,14};
    int t=Average.average(4,arr);
    Assert.assertEquals(6,t );
  }

  @Test
  /* any n, k>=n*/
  public void Test5() throws InterruptedException
  {
    arr=new int[] {2,6,1,15,20,10};
    int t=Average.average(8,arr);
    Assert.assertEquals(9, t);
```

```
    }



}
```

6. Compile and run the test cases. Record any failures and errors that are reported. Analyze and briefly explain why each of the failures and errors occurs and how you fix them. Correct all the failures and errors until the CUT passes all the test cases.



7. Measure the code coverage using Cobertura or other coverage tool.
We use EclEmma extension in Eclipse to measure coverage of our test cases. We get a 98% coverage.



Source: Object Oriented Software Engineering: An Agile Unified Methodology, David C. Kung, McGrawHill