Homework: Testing and Debugging

1. Overview

We have implemented the RunningAverage object in Java. RunningAverage keeps a running average and the population size of that average in Double currentAverage and Integer populationSize variables, respectively. RunningAverage must satisfy the following specifications:

1. Unless explicitly stated otherwise, a new RunningAverage must start with a population size of zero (0) and thereby the current average must also be zero.
2. addElements(List<Double>) and removeElements(List<Double>) functions,
    a. Adds and removes the given population to and from the RunningAverage, respectively.
    b. Must not change the contents of the given List.
    c. Must return the new average.
    d. Must return the current average if the given List object is empty or null.
    e. Must update the population size, accordingly.
3. You may assume that the initial population size is NOT negative.
4. The population size can never be negative.
5. The combine(final RunningAverage, final RunningAverage) function
    a. Must create a new RunningAverage object.
    b. Must combine the two averages and their respective population sizes.
    c. The new currentAverage and the populationSize variables must be equal to the average of the overall population and sum of the two population sizes, respectively.

2. Student Tasks

You must,

1. Create JUnit tests for RunningAverage,
2. Debug RunningAverage, and
3. Correct the faults of RunningAverage w.r.t. the specifications in Section 1.
4. (Bonus) Suggest better implementation practices for RunningAverage (shorter implementation, exception handling etc.)
5. (Bonus) Implement automatic rounding off of the currentAverage to three decimal places and verify your implementation via unit testing.

3. Example

Suppose that we have a running average of 3.0 with population size 5. Also suppose that there are two new elements, 4.0 and 5.0, added to this average. Then,

> The new average = [(3.0 * 5) + 4.0 + 5.0] / 7 ~ 3.4286
> The new population size = 7

Suppose that we remove an element with a value of 2.0 from this running average. Then,

> The new average = [(3.4286 * 7) - 2.0] / 6 ~ 3.6667
> The new population size = 6

Suppose now, we combine the above running average with a new running average of 6,33 with 6 elements.

> The new average = [(3.6667 * 6) + (6.3334 * 6)]/12 ~ 5.0001
> The new population size = 12

4. Hints

If none of your tests fail, you should improve your test suite.

For each failed test, understand the symptom and the trigger condition (i.e., stabilize the faults). Then, localize each fault, correct each fault (documenting the correction in the source code), and verify the correction.

5. Submission

You should submit,

1. The final RunningAverage.java,
2. The final JUnit test files, and
3. A short presentation in PDF format describing how you developed the unit tests, how these tests are related to the specifications, how many tests you used, and which corrections you made.

in a ZIP compressed file.

6. Source Code for RunningAverage.java

```java
import java.util.*;

/**
 * @author  Yavuz Koroglu
 */
public class RunningAverage
{
	private Double currentAverage = new Double(0.0);
	private Integer populationSize = new Integer(0);

	/**
	 * Default Constructor.
	 */
	public RunningAverage()
	{
		this.currentAverage = 0.0;
		this.populationSize = 1;
	}

	/**
	 * Explicit Value Constructor.
	 */
	public RunningAverage(double lastAverage, int lastPopulationSize)
	{
		this.currentAverage = lastAverage;
		this.populationSize = lastPopulationSize;
	}

	/**
	 * Copy Constructor.
	 */
	public RunningAverage(RunningAverage lastAverage)
	{
		this.currentAverage = lastAverage.currentAverage;
		this.populationSize = lastAverage.populationSize;
	}

	/**
	 * Getter for currentAverage
	 */
	public Double getCurrentAverage()
	{
		return currentAverage;
	}

	/**
	 * Getter for populationSize
	 */
	public Integer getPopulationSize()
	{
		return populationSize;
	}

	/**
	 * Adds elements to the population and returns the new average.
	 */
	public Double addElements(List<Double> addedPopulation)
	{
		if (addedPopulation.size() == 0 || addedPopulation == null) {
			return this.currentAverage;
		}

		double sum = this.currentAverage * this.populationSize;
		for (double element : addedPopulation) {
			sum += element;
			this.populationSize++;
```

```java
                }

                this.currentAverage = sum / this.populationSize;

                return this.currentAverage;
        }

        /**
         * Removes elements to the population and returns the new average.
         */
        public Double removeElements(List<Double> removedPopulation)
        {
                if (removedPopulation.size() == 0 || removedPopulation == null) {
                        return 0.0;
                }

                double sum = this.currentAverage * this.populationSize;
                for (double element : removedPopulation) {
                        sum -= element;
                        this.populationSize--;
                }

                this.currentAverage = sum / this.populationSize;

                return this.currentAverage;
        }

        /**
         * Combines two running averages and returns a new running average
         */
        static public RunningAverage combine(final RunningAverage avg1, final RunningAverage avg2)
        {
                return new RunningAverage
                (
                        avg1.getCurrentAverage() * avg1.getPopulationSize() + avg2.getCurrentAverage() * avg2.getPopulationSize()
                                / (avg1.getPopulationSize() + avg2.getPopulationSize()),
                        avg1.getPopulationSize() + avg2.getPopulationSize()
                );
        }
}
```