COMP254 Project 3: Computing revenue accurately

For the purposes of this project, imagine you work for a fictitious company called ClickR. ClickR is an online photo-sharing company similar to Flickr. ClickR has two types of customers: *resellers* and *individuals*. Resellers pay ClickR large amounts of money to run the resellers' own websites — a typical payment from a reseller is between $1 million and $10 million. Individuals sign up for ClickR services, and pay a small amount (sometimes called a *micro-payment*) for each interaction with the site. For example, individuals are charged a micro-payment each time they view or upload a photo. Typical micro-payments by individuals are less than 50 cents, and micro-payments are always charged as a whole number of cents.

The *revenue* of a company over a given period of time is the sum of payments by all customers during the period. ClickR computes its revenue for a given period as follows. The value of each payment is stored in a text file, with each payment on a separate line. The values are represented as decimal numbers in ASCII. Three examples of files containing ClickR revenue data are provided with this project. The files are called data1.txt, data2.txt, and data3.txt. The first data file is an artificially small, but manageable, example containing data for only 10 payments. The remaining two data files are more realistic, and each contains data for 1000 payments.

Let's take a closer look at data1.txt. Its contents are:

```
0.29
0.19
0.26
0.16
0.13
0.02
0.02
0.08
1801161.65
0.39
```

As you can see, this file contains 9 micro-payments ranging between $0.02 and $0.39 and one payment from a reseller for about $1.8 million.

In order to compute its revenue, ClickR runs the following Java program (provided with this project as the file `RevenueAdder.java`), which takes as its only argument the name of the revenue data file:

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;

public class RevenueAdder {

      public static float computeRevenue(String filename)
                  throws FileNotFoundException {
            float sum = 0.0f;
            Scanner s = null;
            s = new Scanner(new BufferedReader(new FileReader(
                        filename)));

            while (s.hasNextFloat()) {
                  sum += s.nextFloat();
            }
            s.close();

            return sum;
      }

      public static void usage() {
            System.out.println("usage: java RevenueAdder filename");
            System.exit(0);
      }

      public static void main(String[] args)
                  throws FileNotFoundException {
            if (args.length != 1) {
                  usage();
            }

            String filename = args[0];
            System.out.println("Total revenue is "
                        + computeRevenue(filename));
      }
}
```

A ClickR employee recently noticed that this method of computing revenues often produces incorrect results. For example, if we run the `RevenueAdder` program on data1.txt, we get the following result:

```
> java RevenueAdder data1.txt
Total revenue is 1801163.1
```

However, if we load the data into a spreadsheet program such as Excel and compute the sum, we find that the answer is $1,801,163.19 (see the file data1.xls, provided with this project). So the RevenueAdder is off by 9 cents even on this very small example. Needless to say, the errors get worse when we have more data. For example, on data2.txt, RevenueAdder computes the revenue as $4,770,893.00, which is $5.13 less than the correct answer of $4,770,898.13. On data3.txt, RevenueAdder gets $62,159,844.00, whereas the correct answer is $62,160,080.77 — a difference of $236.77!

Some of your colleagues at ClickR have suggested that these inaccuracies are caused by the use of Java's "float" datatype in the RevenueAdder program. There have also been some suggestions that ClickR should be using either the "double" datatype or the "java.math.BigDecimal" datatype to perform these revenue calculations. However, one of the project managers has expressed concerns that moving to the "double" or "java.math.BigDecimal" datatype might adversely affect the speed of the revenue calculations, or their memory usage. You have been assigned the task of analyzing the problem, explaining why the discrepancies are occurring, and recommending a way of fixing the problem. Specifically, your task is to write a memo to the ClickR development team addressing the following issues:

1. Why are the current revenue calculations producing inaccurate results? Be specific and detailed in your answer to this question. Do not assume that your colleagues have taken a course in computer architecture. Carefully explain all concepts using clear, scientific language.
2. What is your recommendation for fixing the problem? In an appendix to the memo, include some code that will replace the RevenueAdder program with a program that computes the revenue correctly. In the body of the memo, explain why your new code achieves the correct results, and present concrete results that demonstrate this.
3. [Optional, for a small amount of extra credit.] Explain whether or not your solution will adversely affect the speed or memory usage of the revenue calculations, presenting numerical results or other plausible evidence to back up your claims.

Your memo should be at least four pages in length (excluding appendices containing Java code), but you may use up to 10 pages if desired (again excluding appendices).