COS125 - Precept 6 (Arrays I)

1 Code Tracing

Analyse the following code snippets do by tracing the values of the variables across their executions.

```
String[] b = {"A", "B", "C", "D", "E"};
int n = b.length;
for (int i = 0; i < n / 2; i++) {
     String temp = b[i];
     b[i] = b[n - i - 1];
     b[n - i - 1] = temp;
}
```

```
int[] a = new int[5];
int n = a.length;
a[2] = 10;
a[3] = 30;
boolean result = true;
for (int i = 0; i < n - 1; i++) {
   if (a[i] > a[i + 1])
     result = false;
}
```

Can you describe (in plain English) what purpose the idioms in the `for` loops serve in each of the two cases above? (Tracing the loops with a few different arrays helps a lot!)

2 Sound Loop

Write a program `SoundLoop.java` that takes two command-line arguments: `filename.wav`, the name of and audio file, and an integer `n`. The program should create and play a new array corresponding to the audio file repeated `n` times; it should also use `StdAudio.save()` to save the resulting loop in a new file `nx-filename.wav`.

For example, running the following code should play and save the audio file `10x-heartbeat.wav`.

```
> javac-introcs SoundLoop.java
> java-introcs SoundLoop heartbeat.wav 10
```

3 Visualizing Sound

Write a program `SoundWave.java` that reads a single command-line argument, corresponding to an audio file, then *draws* the sound wave that it encodes with the `StdDraw` library. Other than the starter code, you should only need the `StdDraw.point(double x, double y)` function.
(Hint: if the program runs too slowly, you may find it useful to declare a constant ***RESOLUTION*** *and plot one out of every* ***RESOLUTION*** *many points, instead of every single point.)*

4 Compute the Maximum

Write a program `Maximum.java` that takes an arbitrary number of `int` command-line arguments and prints the largest number entered by the user.

Below, we show a couple of sample executions:

```
> javac Maximum.java
> java Maximum 0 -125 6 125 -1
125
> java Maximum -12
-12
```

5 Bonus: Fraud Detection

Write a program `FraudDetection.java` that takes an arbitrary number of `double` command-line arguments, corresponding to the amounts of a sequence of credit card transactions, and flags suspicious transactions according to the following rule:

- if a transaction value is larger than 3× the average transaction value of the previous 5 transactions; or

- if a transaction value is repeated 5× in a row.

Your program should not flag any of the first 5 transactions. Below, we show a few sample executions:

```
> javac FraudDetection.java
> java FraudDetection 100 200 300 400 500 600 700 800 900 1000
> java FraudDetection 5 6 5.5 4.7 7.2 100 5.3
Fraud alert! Transaction #6, amount: $100.0.
> java FraudDetection 5 5 5 5 5 50 50 50 50 50
Fraud alert! Transaction #5, amount: $5.0.
Fraud alert! Transaction #6, amount: $50.0.
```

6 Bonus: Unique

Write a program `Unique.java` that takes an arbitrary number of `int` command-line arguments and verifies whether all of the integers are unique (i.e., no number is repeated more than once).

Below, we show a couple of sample executions:

```
> javac Unique.java
> java Unique 0 -125 6 125 -1
true
> java Unique -12 1 1
false
> java Unique
true
```