Steve Cooper
CS 106A

Handout #24
April 29, 2013

Section Handout #4: String Processing

Portions of this handout by Eric Roberts and Patrick Young

Please remember to work on the problem with ** prior to section.**

1. Adding commas to numeric strings (Chapter 8, Exercise 13, page 290)

When large numbers are written out on paper, it is traditional—at least in the United States—to use commas to separate the digits into groups of three. For example, the number one million is usually written in the following form:

```
1,000,000
```

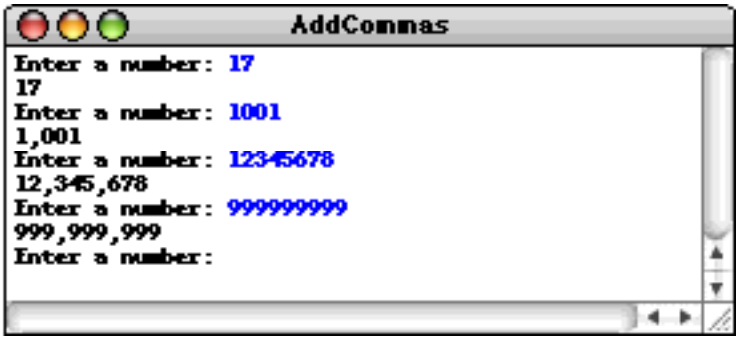
To make it easier for programmers to display numbers in this fashion, implement a method

```
private String addCommasToNumericString(String digits)
```

that takes a string of decimal digits representing a number and returns the string formed by inserting commas at every third position, starting on the right. For example, if you were to execute the main program

```
public void run() {
   while (true) {
      String digits = readLine("Enter a numeric string: ");
      if (digits.length() == 0) break;
      println(addCommasToNumericString(digits));
   }
}
```

your implementation of the `addCommasToNumericString` method should be able to produce the following sample run:



2. *******Deleting characters from a string*******

Write a method

```
private String removeDecreasingChars(String str)
```

that removes letters within the string that are not increasing lexicographically, according to the ASCII char-to-number chart. For example, your method should return the values shown:

```
removeDecreasingChars("earthy")
        returns                                    "erty"
removeDecreasingChars("asteroid")
        returns                                    "ast"
removeDecreasingChars("Thegirlisgood!")
        returns                                    "Thirs"
removeDecreasingChars(''zebrafish'')
        returns                                    ''z''
removeDecreasingChars(''SPARTAorperish'')
        returns                                    ''STors''
```

You can assume that you will not be passed any non-alphabetic chars.

3. Heap/stack diagrams

Using the style of heap/stack diagram introduced in Chapter 7, show the state of both the heap and the stack at the point in the computation indicated by the arrow in the following code, where the `Rational` class is the one defined in Chapter 6.

```
public void run() {
   Rational r = new Rational(1, 2);
   r = raiseToPower(r, 3);
   println("r ^ 3 = " + r);
}

private Rational raiseToPower(Rational x, int n)
   Rational result = new Rational(1);
   for (int i = 0; i < n; i++) {
      result = result.multiply(x);
   }
   return result;                          ←Diagram at this point
}
```

Indicate which values in the heap are garbage at this point in the calculation.

4. Tracing method execution

For the program below, show what output is produced by the program when it runs.

```
/*
 * File: Mystery.java
 * ------------------
 * This program doesn't do anything useful and exists only to test
 * your understanding of method calls and parameter passing.
 */

import acm.program.*;

public class Mystery extends ConsoleProgram {

   public void run() {
      ghost(13);
   }


   private void ghost(int x) {
      int y = 0;
      for (int i = 1; i < x; i *= 2) {
         y = witch(y, skeleton(x, i));
      }

      println("ghost: x = " + x + ", y = " + y);
   }


   private int witch(int x, int y) {
      x = 10 * x + y;
      println("witch: x = " + x + ", y = " + y);
      return x;
   }


   private int skeleton(int x, int y) {
      return x / y % 2;
   }
}
```