COLBY COLLEGE
Computer Science Department

CS151 **Allen B. Downey**
Fall 1999

Assignment 6: Iteration

Due: Wednesday 20 March

Recursion

1. Create a new project called `Sum.prj`, and type in the following two methods.

```
public static int methOne (int m, int n) {
  if (m == n) {
    return n;
  } else {
    return m + methOne (m+1, n);
  }
}

public static int methTwo (int m, int n) {
  if (m == n) {
    return n;
  } else {
    return n * methTwo (m, n-1);
  }
}
```

2. Add a few lines to `main` to test these methods. Invoke them a couple of times, with a few different values, and see what you get. By some combination of testing and examination of the code, figure out what these methods do, and give them more meaningful names. Add comments that describe their function abstractly.

3. Add a `println` statement to the beginning of both methods so that they print their arguments each time they are invoked. This is a useful technique for debugging recursive programs, since it effectively "follows the stack" for you.

4. Write a recursive method called `power` that takes a double `x` and an integer `n` and that returns x^n. Hint: a recursive definition of this operation is `power (x, n) = x * power (x, n-1)`. Also, remember that anything raised to the zeroeth power is 1.

Iteration

1. Read Chapter 6, and type the following loop:

```
public static void sequence (int n) {
  while (n != 1) {
    System.out.println (n);
    if (n%2 == 0) {            // n is even
      n = n / 2;
    } else {                   // n is odd
      n = n*3 + 1;
    }
  }
}
```

2. Add some code in `main` that invokes this method and test it with a variety of starting value. Notice that the first thing it does during each iteration is print the value of the loop variable. This is a useful technique for debugging iterative programs.

3. Write an iterative version of the `power` method in the previous section.

4. Write an iterative version of `factorial`.

Moiré patterns

1. Create a new project named `Moire.prj` based on the `cs115SlateStationery`.

2. Add the following code to your project, and replace the contents of `draw` with a single line that invokes `moire`.

```
public static void moire
       (Graphics g, int x, int y, int width, int height) {
  int i = 1;
  while (i<width) {
    g.drawOval (0, 0, i, i);
    i = i + 2;
  }
}
```

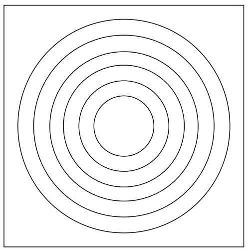
3. Look at the code before you run it and draw a sketch of what you expect it to do. Now run it. Did you get what you expected? For a partial explanation of what is going on, see the following:

   `http://math.hws.edu/xJava/other/Moire1.html`
   `http://tqd.advanced.org/3543/moirelesson.html`

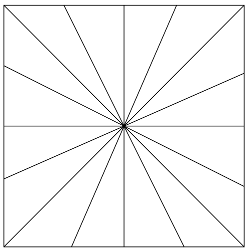
4. Modify the program so that the space between the circles is larger or smaller. See what happens to the image.

5. Modify the program so that the circles are drawn in the center of the screen and concentric, as in the following figure. Unlike in the figure, the distance between the circles should be small enough that the Moiré interference is apparent.

concentric circles



radial Moire pattern



6. Write a method named `radial` that draws a radial set of line segments as shown in the figure, but they should be close enough together to create a Moiré pattern.

7. Just about any kind of graphical pattern can generate Moiré-like interference patterns. Play around and see what you can create.

What to turn in

1. Please prepare and print your code as usual. If you have a Moiré pattern you are particularly proud of, feel free to print a copy and turn that in, too.