

MAT 2170: Laboratory 4

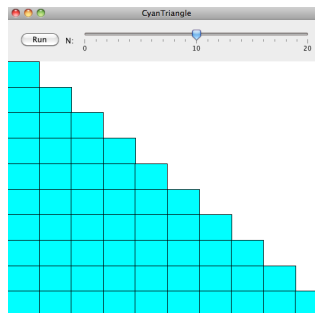
Key Concepts

1. `for`, `while` loops
2. integer arithmetic
3. graphics

The main purpose of this lab is to give you practice using loops. In all the programs, you must consider the **stopping criteria** for each loop. Examples worked out with pencil and paper may help you discover what is needed for *initialization*, *update*, and *termination* in the loops.

Exercises

1. Complete the Prelab exercises by class time on Wednesday.
2.
 - Create the Lab exercises in the directory `mat2170/lab4` under your user account (not in any other directory). This can be done at the time you create each project.
 - Remember: in graphics programs, store the window's width (`getWidth()`) and height (`getHeight()`) as floating point values.
 - You can get the slider value in a `SliderProgram` using the method `getN()`, which returns an integer.
 - **Warning** regarding graphics programs: once your program appears to produce the correct results, enlarge the graphics window (by dragging its bottom-right corner down and to the right) after the graphics objects have been drawn in order to verify that your program isn't drawing anything outside the observable part of the window.
3. (Exercise 6, page 128) Write a dialog program, `Reverse`, that, when given a non-negative integer, **calculates** and **displays** the number that has the same digits, but in reverse order. As an example, if the user enters 1234, your program should output 4321 with a nice message explaining what it represents. Integer arithmetic and a `while` loop should be used to accomplish this.
4. (Lower-left Triangle) Write a `SliderProgram`, `LowerTriangle`, that will produce a series of identical blocks which fill the lower-left triangle of the graphics window. A sample output is shown below, with the slider set to 10. Copy the `LowerTriangle` java skeleton program from our web site, and place it in an empty java file in the `src` directory of your `LowerTriangle` project. Modify the `run()` method, which is invoked each time the run button is pushed while the program executes.



The `init()` method initializes the program, setting the size of the window (during testing) and the range of the slider. When publishing your applet, remember to set the window size to 500 by 500 pixels by modifying the `html` file in the usual places.

Nested `for` loops are required (basically the loops from the Triangle Number Table program that displayed and summed the numbers from 1 to N).

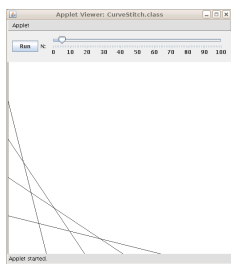
To obtain the width and height of the window, use the messages `getWidth()` and `getHeight()` (do **not** use hard-coded values, aka magic numbers).

Do not create a single block before the nested loops, then use the `move()` message to reposition it since that will erase the block each time before redrawing it. Instead, before entering the loops, determine a block's width and height, and the position of the first block. Then, create a `GRect` object inside the inner loop, based on your pre-calculated values. After displaying each block, modify the position values as necessary using the block's width and height.

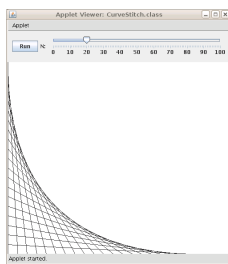
5. (Exercise 7, page 128) Write a dialog program, `DigitalRoot`, which, when given a non-negative integer, calculates and displays its **digital root** — the value obtained from repeated summing of the digits until a single digit remains. This program also requires nested loops. The outer loop should check to see if the number is greater than 9 (and hence, consists of more than one digit), while the inner loop must lop off digits and add them to a sum. The sum should replace the initial integer so it is checked again to see if a single digit has been reached yet.

If the user enters 1234, your program should first calculate $4 + 3 + 2 + 1 = 10$, then $0 + 1 = 1$, and output 1 with a nice message explaining what it represents.

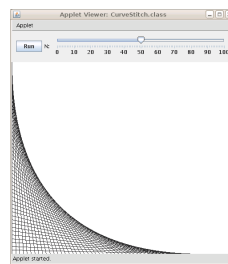
6. (Curve-stitch design¹) Write a `SliderProgram`, `CurveStitch`, which will produce a “curve-stitch” parabola. A single `for` loop is sufficient. Your program should produce a graphics window that is 600 by 600 pixels and the slider should range between the values 3 and 100. When publishing your applet remember to set the window size to 600 by 600 pixels by modifying the `html` file in the usual places. Within a graphics or slider program, the methods `getWidth()` and `getHeight()` can be used to determine the width and height of the graphing window in pixels.



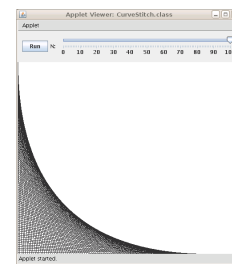
slider = 5



slider = 20



slider = 50



slider = 100

Begin by determining the endpoints for the vertical line along the left side of the graphics window (utilize `getWidth()` and `getHeight()`), as well as the `deltaX` and `deltaY` values, which will step the endpoints down the left-hand side and across the bottom of the window. Once these values are known, begin a loop which will create a `GLine` in the initial position (along left side of the window), then update the endpoints for the next line to be drawn. The last line segment drawn should be the horizontal line along the bottom of the window.

7. When you have completed the lab:

- Submit an electronic copy of lab 4
- Publish these programs to your web site
- Complete the Postlab worksheet, staple printouts of your Java programs to it (in the order assigned in this handout), and hand in at the beginning of Lab 5.

¹www.deimel.org/rec_math/circular_3.htm

1. Complete the following **Reverse** test suite table, adding a few of your own inputs and expected output:

Input	Expected Output
1234	4321
4321	
10	

2. The general algorithm for **Reverse** is:

- (a) Prompt for and get a **number** from user (integer)
- (b) Step through the digits of **number** in reverse, displaying each digit as it is extracted.

Give an algorithm for the second step, “Stepping through **number** in reverse & display each digit” It will need to repeatedly display the units digit, then *lop it off* of **number**.

3. Give the Java statements for the **LowerTriangle** program which:

- (a) Stores the width of the graphics window in a variable named **wWidth**:
- (b) Stores the height of the graphics window in a variable named **wHeight**
- (c) Establishes a variable, **maxBlocks**, set to the current value of the slider.
- (d) Calculates and stores in **bWidth**, the width of one block such that **maxBlocks** blocks fits exactly into a window **wWidth** wide.
- (e) Calculates and stores in **bHeight** the height of one such block.

- (f) Establishes the nested loops to fill the lower triangle of the graphics window with colored blocks. The outer loop should count through the rows, 0 up to `maxBlocks`. The inner loop, which should count across columns based on which row is being displayed, should create and display a `block` at the current row and column.

4. Complete the following `DigitalRoot` test suite table, adding a few of your own inputs and expected output:

Input	Expected Output
1234	1
54321	6
0	0
10	1

5. The general algorithm for `DigitalRoot` is:
- Prompt for and get a `number` from user (integer)
 - Calculate the digital root of `number`
 - Output original and digital root values

Give an algorithm for the second step, “Calculate the digital root of `number`”. Nested loops will be necessary.

5. **while** loops equivalent to the **for** loops in the last problem are desired. Give the initialization, **while** loop header, and update statements necessary to make the loops equivalent.

(a) Counting from 1 to 100

i. **initialization:**

ii. **while header:**

iii. **update:**

(b) Counting by sevens starting at 0 until the number has more than three digits

i. **initialization:**

ii. **while header:**

iii. **update:**

(c) Counting backwards by twos from 100 to 0

i. **initialization:**

ii. **while header:**

iii. **update:**

6. Given `blockWidth`, `blockHeight`, and `maxBlocks`, provide an algorithm (with two nested loops) for displaying a triangle which covers the **upper right** portion of a graphics window. Of particular interest is the initialization of `x`, and the inner loop (for `columns`).

```
double y = 0.0;

for (int row = 0; row < maxBlocks; row++)
{
    x =

    for (int col =          ; col          ;          )
    {
        GRect ThisRow = new GRect(x, y, blockWidth, blockHeight);
        ThisRow.setFilled(true);
        ThisRow.setFill(Color.BLUE);
        add(ThisRow);
        x += blockWidth;
    }
    y += blockHeight;
}
```