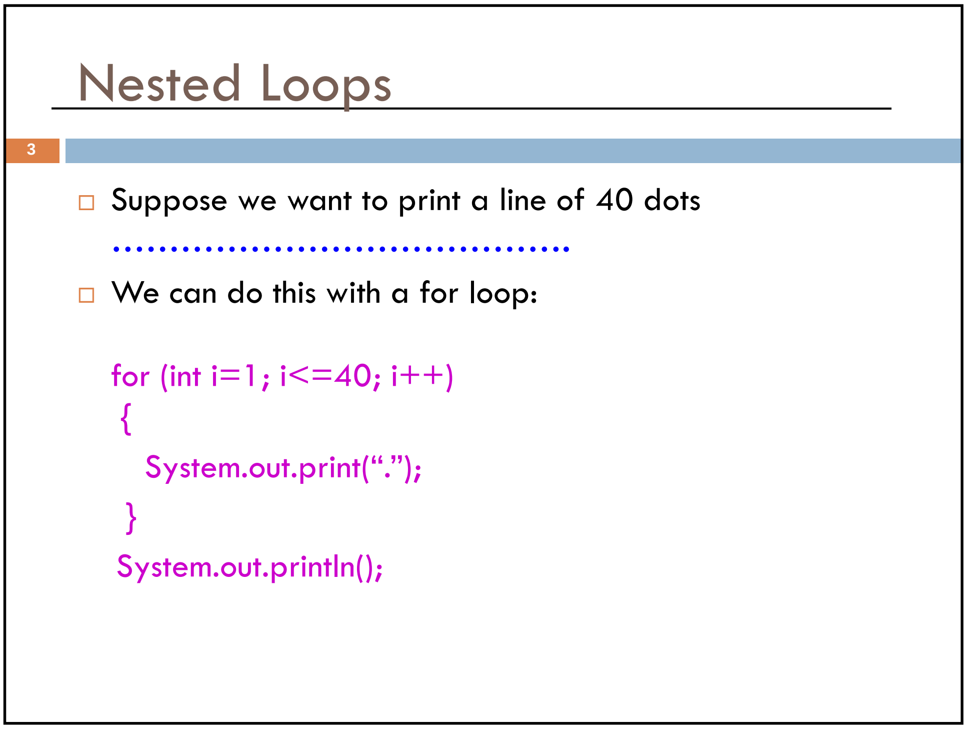
TOPIC 7
MODIFYING PIXELS IN A MATRIX

NESTED FOR LOOPS

Notes adapted from Introduction to Computing and Programming with Java: A Multimedia Approach by M. Guzdial and B. Ericson, and instructor materials prepared by B. Ericson.

Outline

- **Nested loops**
- Using nested loops to process data in a **matrix (2-dimensional array)**
- More advanced ways of manipulating pictures in Java programs



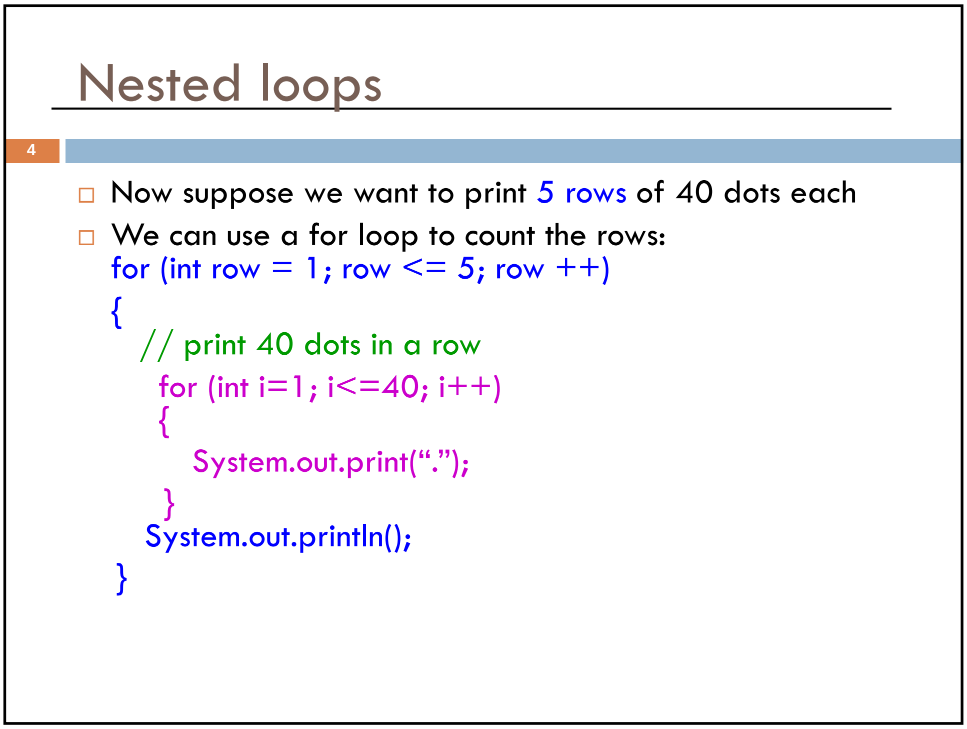
Nested Loops

- Suppose we want to print a line of 40 dots

  ....

- We can do this with a for loop:

```
for (int i=1; i<=40; i++)
 {
    System.out.print(“.”);
 }
System.out.println();
```



Nested loops

- Now suppose we want to print 5 rows of 40 dots each
- We can use a for loop to count the rows:

```
for (int row = 1; row <= 5; row ++)
{
   // print 40 dots in a row
     for (int i=1; i<=40; i++)
     {
        System.out.print(“.”);
     }
   System.out.println();
}
```

Nested loops

- The for loop to print a row of dots is part of the body of the loop that counts the rows
  - This is an example of **nested loops**
  - The loop that counts the rows is called the **outer loop**
  - The loop that prints each row of dots is called the **inner loop**

Nested loops

- Another example: print a triangle of dots

```
•
••
•••
••••
•••••
```

  - The **outer loop** will count the rows
  - The **inner loop** will print the appropriate number of dots

Nested loops

```
for (int row = 1; row <= 5; row ++)
  {
      // print dots in a row
      for (int i=1; i<=??; i++)
      {
         System.out.print(“.”);
       }
    System.out.println();
  }
```

Exercise

- □ What would you change in the code of the previous slide so that it prints

```
.....
....
...
..
.
```

Picture manipulation

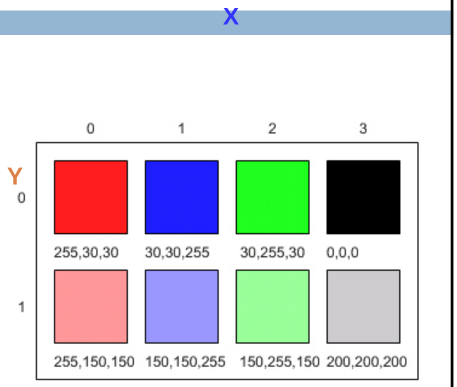
- So far, we have used a single loop when modifying a picture
- We had a one-dimensional array of Pixels returned by the method getPixels()
- But this technique is only useful for simple picture manipulations that change every pixel the same way
  - For example, decreaseRed(), negate(), etc.

More advanced picture manipulation

- We can only go so far in processing pictures without knowing where certain pixels are in an image, for example:
  - Cropping a picture
  - Copying just part of a picture
  - Performing reflections
  - Performing rotations
- We will now consider a picture as a **matrix** or **two dimensional array**

Review: Pictures as a grid of pixels

- □ Recall that pictures are organized as a **grid of pixels**
- □ The grid has **columns** and **rows**
- □ Each pixel has an **(x, y) position** in the grid
  - ◘ x specifies the column
  - ◘ y specifies the row
- □ We will now call this grid a **matrix** or **2-D array**



Pictures as 2-D arrays

- □ To access each pixel in the picture (i.e. in the 2-D array), we will use nested loops
- □ We can access pixels row by row:
  - ■ The outer loop moves horizontally along rows
  - ■ Then to get each pixel in a row, the inner loop moves vertically along columns
- □ Or we can access pixels column by column:
  - ■ The outer loop moves vertically along columns
  - ■ Then to get each pixel in a column, the inner loop moves horizontally along rows

Nested loops

- To get all the pixels in a picture using their x and y values **row-wise** (left to right, top to bottom)

  x=0, y=0 x=1, y=0 x=2 , y=0 …

  x=0, y=1 x=1, y=1 x=2 , y=1 …

  x=0, y=2 x=1, y=2 x=2, y=2 …

- We have nested loops:
  - The **outer loop** counts rows:

    y from 0 to (height – 1)
  - The **inner loop** counts columns:

    x from 0 to (width – 1)

Nested loop template (row-wise)

```
// Loop through the rows (y direction)
for (int y = 0; y < this.getHeight(); y++)
   {
         // Loop through the columns (x direction)
         for (int x = 0; x < this.getWidth(); x++)
            {
            // Get the current pixel
             pixelObj = this.getPixel(x,y);

            // Do something to its color

             // Set the new color
            pixelObj.setColor(colorObj);
            }
   }
```

Alternative nested loops

- To get all the pixels in a picture using their x and y values **column-wise** (top to bottom, left to right)

  x=0, y=0 x=0 , y=1 x=0, y=2 …
  x=1, y=0 x=1 , y=1 x=1, y=2 …
  x=2, y=0 x=2 , y=1 x=2, y=2 …
- We again have nested loops:
  - The **outer loop** counts columns:
    x from 0 to (width – 1)
  - The **inner loop** counts rows:
    y from 0 to (height – 1)

Nested loop template (column-wise)

```
//  Loop through the columns (x direction)
for (int x = 0; x < this.getWidth(); x++)
{
            //  Loop through the rows (y direction)
            for (int y = 0; y < this.getHeight(); y++)
               {
                //  Get the current pixel
                pixelObj = this.getPixel(x,y);

                //  Do something to its color

                //  Set the new color
                pixelObj.setColor(colorObj);
               }
}
```

Lightening an image

- □ Earlier, we saw how to lighten an image by accessing pixels through getPixels()
- □ This time, we will use **nested loops**
- □ We will do a column-wise implementation
- □ Exercise: write the row-wise version

Lightening an image

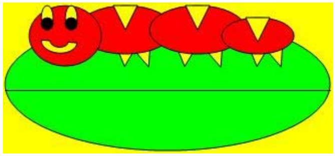
```
public void lighten2()
 {
   Pixel pixelObj = null;
   Color colorObj = null;

   // loop through the columns (x direction)
   for (int x = 0; x < this.getWidth(); x++)
   {
     // loop through the rows (y direction)
      for (int y = 0; y < this.getHeight(); y++)
      {
```

Continued

```
        // get pixel at the x and y location
        pixelObj = this.getPixel(x,y);

        // get the current color
        colorObj = pixelObj.getColor();

        // get a lighter color
        colorObj = colorObj.brighter();

        // set the pixel color to the lighter color
        pixelObj.setColor(colorObj);

      } //end of inner loop
    } // end of outer loop
  }
```

Exercise: Changing to Nested Loops

- Change the method clearBlue() to use nested for loops to loop through all the pixels
- Check that the blue values are all 0 using the explore() method

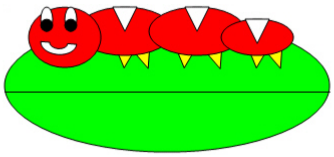


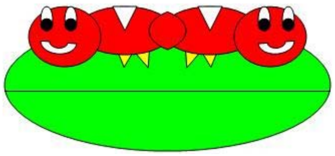
More advanced picture manipulations

- We will now consider image manipulations that do not alter all the pixels in a picture:
  - Vertical mirroring
  - Horizontal mirroring
  - Others (textbook)
    - Cropping
    - Rotating
    - Scaling

Vertical mirroring

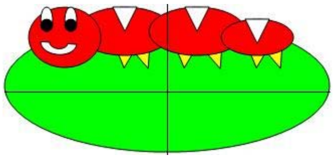
- We place a mirror in the middle of a picture





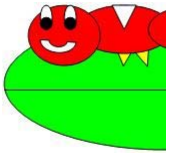
Vertical mirroring

- To do this, we want to take the mirror image of the left half of the caterpillar and copy it to the right half

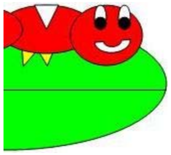


Vertical mirroring

- Left half:

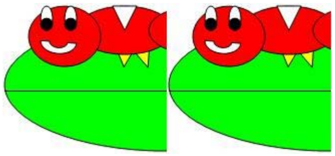


- Copy to right half:



Vertical mirroring

- Bad approach: copy column 0 to column 164, column 1 to column 165, etc.



Algorithm

- Loop through the rows (y values)
  - Loop from x starting at 0 and going to just before the midpoint (mirror) value
    - Get the left pixel, at x and y
    - Get the right pixel, at (width -1 - x) and y
    - Set the color for the right pixel to be the color of the left pixel

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)

Algorithm to code

- We are going to need the midpoint

  ```
  int midpoint = this.getWidth() / 2;
  ```

- Loop through the rows (y values)

  ```
  for (int y = 0; y < this.getHeight(); y++) {
  ```

  - Loop through x values (starting at 0)

    ```
    for (int x = 0; x < midpoint; x++) {
    ```

    - Set right pixel color to left pixel color

      ```
      Pixel leftPixel = this.getPixel(x, y);
      Pixel rightPixel = this.getPixel(this.getWidth() - 1 -
        x, y);
      rightPixel.setColor(leftPixel.getColor());
      ```

Mirror vertical method

```
public void mirrorVertical()
 {
   int mirrorPoint = this.getWidth() / 2;
   Pixel leftPixel = null;
   Pixel rightPixel = null;

   // loop through the rows
   for (int y = 0; y < this.getHeight(); y++)
   {
     // loop from 0 to just before the mirror point
     for (int x = 0; x < mirrorPoint; x++)
     {
```

Continued

```
      leftPixel = this.getPixel(x, y);
      rightPixel = this.getPixel(this.getWidth() – 1 – x, y);
      rightPixel.setColor(leftPixel.getColor());
    }
  }
}
```

Trying the method

- Create the picture

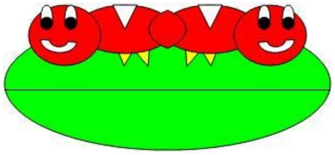
```
Picture p1 = new Picture(
              FileChooser.getMediaPath("caterpillar.jpg"));
```

- Call the method on the picture

```
p1.mirrorVertical();
```

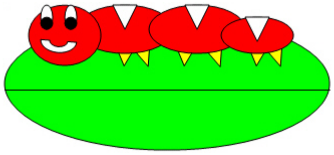
- Show the picture

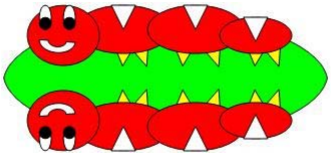
```
p1.show();
```



Horizontal mirroring

- Next: mirroring horizontally, i.e. around a mirror held horizontally in the vertical center of the picture





Algorithm

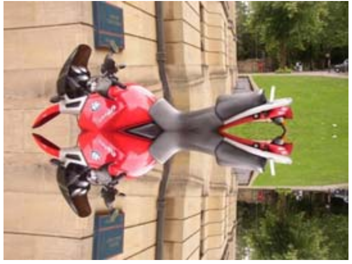
- We will need the horizontal midpoint this time
- Loop through the columns (x values)
  - Loop from y=0 to y < midpoint
    - Get the top pixel, at x and y
    - Get the bottom pixel, at x and (height -1 - y)
    - Set the color for the bottom pixel to be the color of the top pixel

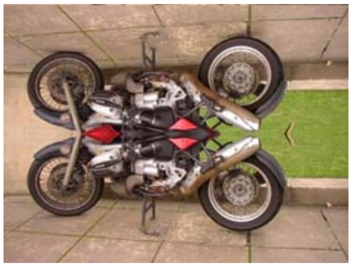
(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)
(0,2)	(1,2)	(2,2)

(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)
(0,2)	(1,2)	(2,2)

Exercise

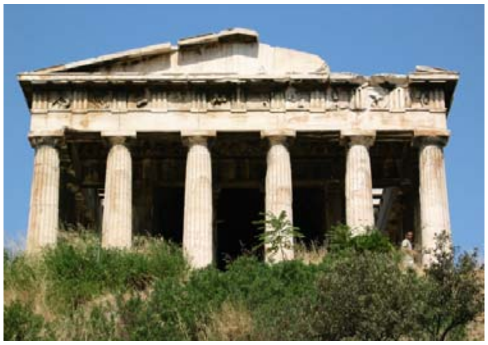
- Write the method to mirror the top half of the picture to the bottom half
  - This is the motorcycle image in redMotorcycle.jpg
- Next: mirroring bottom to top





Useful Mirroring

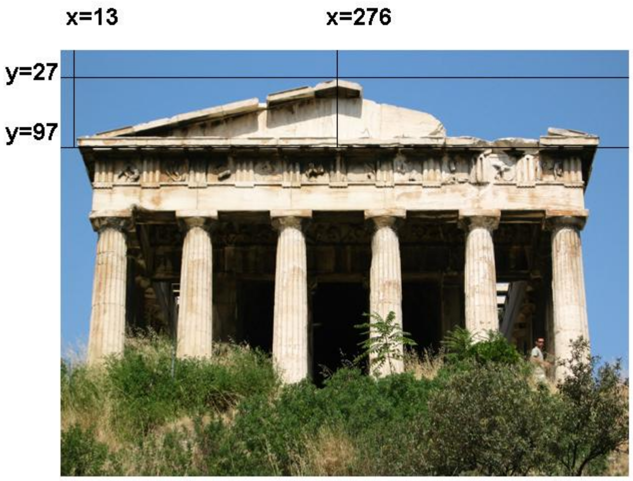
- The Temple of Hephaistos in Athens, Greece has a damaged pediment. Goal: fix the temple.



Useful Mirroring

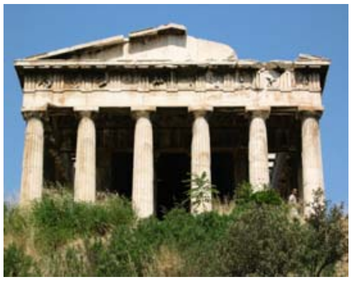
- □ We can’t just mirror the left half onto the right
- □ We don’t mirror all pixels
- □ How can we accomplish this?
  - Choose a point to mirror around vertically
  - Start with the row of pixels at the top of the pediment
  - End with the row of pixels at the bottom of the pediment

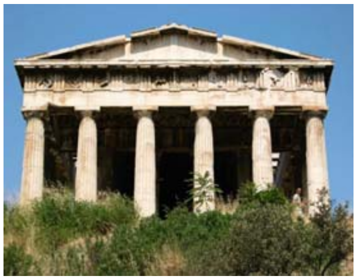
Determining the region



Result

- Before and after: how can you tell that the picture has been digitally manipulated?





Summary

- Nested loops
- Pictures as 2-D arrays of pixels
- Algorithms on Pictures:
  - Vertical mirroring
  - Horizontal mirroring