TOPIC 6
MODIFYING PICTURES USING 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

- How to manipulate digital images by changing pixels
- What is a **loop** in Java
  - while loop
  - for loop

Modifying Pictures

- Recall that we manipulate pictures by manipulating the pixels in the picture
- We can change the color values of
  - Just one pixel
  - Some of the pixels in a picture
    - A whole row of pixels
    - A whole column of pixels
    - Pixels in some sub-area of the picture
  - Every pixel in the picture

Changing a picture

- An Example: One way to change a picture is to reduce the amount of red in it
  - decrease it by half
    - How do we reduce any value by half?
  - increase it by 25%
    - How do we increase any value by 25%?

Changing all the pixels in a picture

- Example: Change the red in caterpillar.jpg
- How?
  - Get the current pixel
  - Get the red value of the current pixel
  - Change this value to half its original value
  - Set the red of the current pixel to the new value
- There are 329*150 = 49,350 pixels in the caterpillar picture

Looping (iteration)

- All programming languages have a construct that allows us to repeat a series of statements some number of times: in Java this is called a **loop**
- Looping is also called **iteration**
- A loop should not go on forever, so we need some way to tell when we are done with the repetition: some **test** to see if the looping should stop

Loops (often) need counters

- If you want to do something x times you often need a **counter**
  - The counter may start with 0 or 1
    - Example: index of an array
  - You add 1 each time you finish whatever it is you are repeating
  - When the counter reaches the appropriate number, you stop the loop

While loops

- Simple example:

```
int count = 1;
while (count <= 5)
{
    System.out.println("This is a test");
    count = count + 1;
}
```

- How many times will "This is a test" be displayed on the screen?
- What is the value of the counter after the statements in the **body of the loop** are executed for the last time?

While loops

- The basic syntax for a **while loop** is:

```
while (test)
{
    body of loop
 }
```

where

- test is a condition that is true or false
- body of the loop consists of the statements to be executed while the condition is true

While loops

- When the condition becomes false, execution continues with the statement after the while loop
  - We say it “falls through” to the statement after the while loop
- A while loop is called a **pretested loop**
- Something must change within the body of the loop, that will cause the condition to become false
  - Otherwise we have an **infinite loop**

Add the numbers from 1 to 100

- You will need something to hold the total
  - What type should it be?
  - What value should it start out with?
- You will need something that counts from 1 to 100
  - And add that value to the total
  - Stop when you get to 100
  - What type should it be? What value should it start with?

Add the numbers from 1 to 100

```
int total = 0;
int counter = 1;
while (counter <= 100)
{
   total = total + counter;
   counter = counter + 1;
 }
System.out.println("Sum of 1 to 100 is " + total);
```

Add the numbers from 1 to 100

- What will be the value of counter after the while loop?
- What would happen if you forgot to add 1 to counter?
- In DrJava, click on Reset to terminate an infinite loop

Exercise

- Write a while loop that will print 40 asterisks on a line:

```
****************************************
```

  - Start the counter at 1
  - Start the counter at 0

Decrease red in a picture

- We will now develop the code for a method to decrease the red in a picture
  - Decrease the red in all the pixels
  - Using the array of pixels
- We will add this method to the Picture class
- Before we start writing the code, we need to work out the steps required
  - We write an **algorithm**

What is an algorithm?

- An **algorithm** is a description of the steps needed to do a task
  - Can be written in English
  - Example: a recipe is an algorithm
- A program is an **implementation of an algorithm**
  - In a particular computer language

Decrease red algorithm

- To decrease the red value in a picture by 50%
  1. Get the array of pixels from the picture
  2. Start the array index at 0
  3. Check if the index is less than the length of the array
     1. If it is, go on to step 4
     2. If it isn't, we're done
  4. Get the pixel at the current index
  5. Get the red value at the pixel
  6. Divide the red value by 2
  7. Set the red value at the pixel to the reduced red value
  8. Add 1 to the array index
  9. Go back to step 3 to process the next pixel in the array

From algorithm to Java code

- How do we get the array of pixels from the current picture object?
  - We have used `Pixel[] pixelArray = PictureObj.getPixels();`
  - In our method, we want to get the array of pixels from the current object (i.e. the object that this method will be invoked on)
  - So we use the keyword `this`

    ```
    Pixel[] pixelArray = this.getPixels();
    ```

From algorithm to Java code

- How do we write the loop?
  - Use a while loop with a counter being the array index starting at 0

    ```
    int index = 0;
    ```
  - Loop while the index is less than the length of the array

    ```
    while (index < pixelArray.length)
    ```
  - Get the current pixel from the array of pixels (i.e. the pixel for the current index)

    ```
    Pixel pixelObj = pixelArray[index];
    ```

From algorithm to Java code

- Get the red value at the pixel

  ```
  int value = pixelObj.getRed();
  ```
- Divide the red value by 2

  ```
  value = value / 2;
  ```
- Set the pixel's red value

  ```
  pixel.setRed(value);
  ```
- Add one to the index (**increment** it)

  ```
  index = index + 1;
  ```

decreaseRed method version 1

```
public void decreaseRed()
 {
   Pixel[] pixelArray =
            this.getPixels();
   Pixel pixelObj = null;
   int index = 0;
   int value = 0;

   // loop through all the pixels
   while(index < pixelArray.length)
   {
     // get the current pixel
     pixelObj = pixelArray[index];
```

```
     // get the red value
     value = pixelObj.getRed();

     // decrease the red value
     value = value / 2;

      // set the pixel's red value
      pixelObj.setRed(value);

      // increment the index
      index = index + 1;
    }
  }
```

Local variables in Java

- When we declare variables inside the body of a method, they are know as **local variables**
- Examples in the decreaseRed method:
  - pixelArray
  - pixelObj
  - index
  - value
- **Scope of a variable**: the area in the program in which the variable is known

Reference variables revisited

- In our method, we have the following statements in the body of the while loop:

```
// get the current pixel
pixelObj = pixelArray[index];
```

  - What object does the variable pixelObj refer to, the first time through the loop?
    - The object at pixelArray[0]
  - What object does the variable pixelObj refer to, the second time through the loop?

Reference variables revisited

- A reference variable can be changed to refer to a different object (of the same type, of course)
  - Another example:

```
Pixel aPixel = pictureObj.getPixel(0,0);
System.out.println(aPixel);
aPixel = pictureObj.getPixel(100,100);
System.out.println(aPixel);
```

  - If there is nothing else referring to an object, it gets automatically **garbage collected** by Java

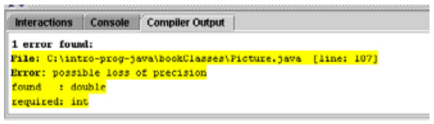
Can we use multiplication by 0.5 ?

- Back to our decreaseRed method
  - You could have also multiplied the red value by 0.5

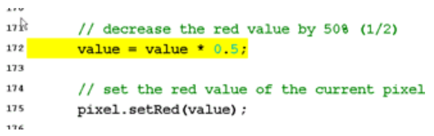
    value = value * 0.5;
  - Try it: change the line in the decreaseRed code that divided by 2, and compile it.

Can we use multiplication by 0.5 ?

- You will get a compiler error, "possible loss of precision"
- It is complaining about putting a double value into an int variable
- Loss of fractional part



```
1 error found:
File: C:\intro-prog-java\bookClasses\Picture.java  [line: 107]
Error: possible loss of precision
found   : double
required: int
```



```
    // decrease the red value by 50% (1/2)
    value = value * 0.5;

    // set the red value of the current pixel
    pixel.setRed(value);
```

Can we use multiplication by 0.5 ?

- It will compile if we tell the compiler we know about the possible loss of precision, and that it is intended
- By using a cast to int

```
value = (int) (value * 0.5);
```

Shortcuts for common operations

- You often need to add 1 to a value
- You may use the **shortcut**

```
index++;        or      ++index;
```

- Similarly, if you wanted to subtract 1:

```
index = index – 1;
index--;  or    -- index;
```

- You can also use these shortcuts:

```
x += y              for     x = x + y
x -= y                      x = x – y
x *= y                      x = x*y
x /= y                      x = x / y
```

decreaseRed method version 2

```
public void decreaseRed()
 {
   Pixel[] pixelArray = this.getPixels();
   Pixel pixelObj = null;
   int index = 0;
   int value = 0;

   // loop through all the pixels
   while(index < pixelArray.length)
   {
     // get the current pixel
     pixelObj = pixelArray[index];

     // get the red value
     value = pixelObj.getRed();

     // decrease the red value
     value = (int) (value * 0.5);

     // set the pixel's red value
     pixelObj.setRed(value);

     // increment the index
     index++;
   }
 }
```

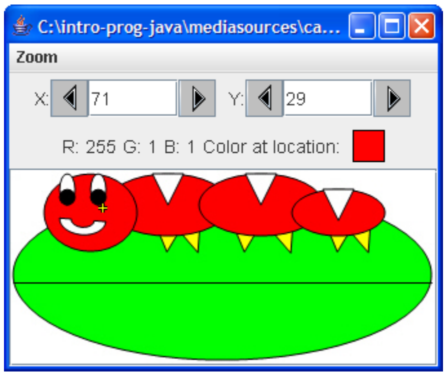
Testing decreaseRed()

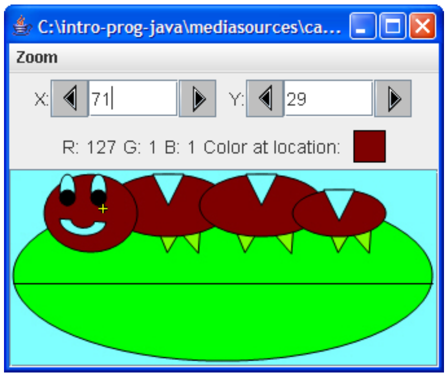
- Add the method decreaseRed() to Picture.java
  - Before the last } which ends the class definition
- Compile the new Picture.java
- Test it by doing the following in the interactions pane:

```
> String fileName = FileChooser.pickAFile();
> Picture picture1 = new Picture(fileName);
> picture1.explore();
> picture1.decreaseRed();
> picture1.explore();
```

  - Check in the picture explorer that the red values were reduced by 50% …

Testing decreaseRed()


C:\intro-prog-java\mediasources\ca...
Zoom
X: 71
Y: 29
R: 255 G: 1 B: 1 Color at location:


C:\intro-prog-java\mediasources\ca...
Zoom
X: 71
Y: 29
R: 127 G: 1 B: 1 Color at location:

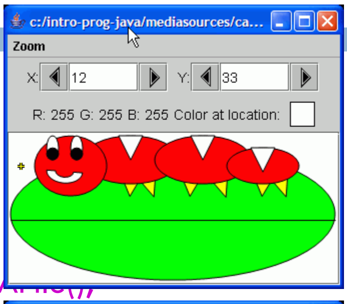
The caterpillar.jpg picture before and after calling our new decreaseRed() method

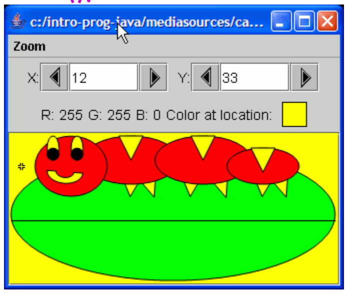
Exercise: clear the blue in a picture

- In Picture.java add the method
  public void clearBlue()

  that sets the blue value to 0 for all
- Test with:

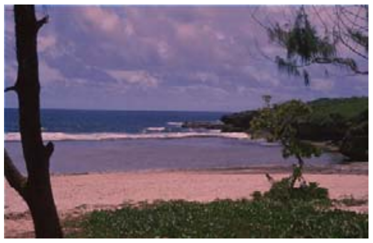
```
>String fileName =  FileChooser.pickAFile();
>Picture p = new Picture(fileName);
>p.explore();
>p.clearBlue();
>p.explore();
```


c:/intro-prog-java/mediasources/ca...
Zoom
X: 12
Y: 33
R: 255 G: 255 B: 255 Color at location:


c:/intro-prog-java/mediasources/ca...
Zoom
X: 12
Y: 33
R: 255 G: 255 B: 0 Color at location:

Example: faking a sunset

- If you want to make an outdoor scene look like it happened during sunset
  - You might want to increase the red , but you can't increase past 255
  - Another idea is to reduce the blue and green
    - To emphasize the red, reduce the blue and green by 30%



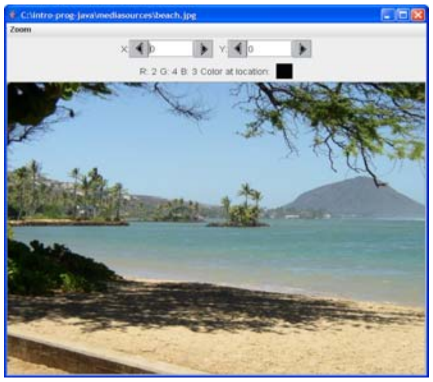
Faking a sunset algorithm

- Sunset: Reduce the blue and green by 30%
  1. Get the array of pixels from the picture
  2. Set up array index to start at 0
  3. Check if the index is less than the length of the array
     1. If it is, go on to step 4
     2. If it isn't, we're done
  4. Get the pixel at the current index from the array of pixels
  5. Set the blue value at the pixel to 0.7 times the original value
  6. Set the green value at the pixel to 0.7 times the original value
  7. Increment the index and go back to step 3 to process the next pixel in the pixel array

Faking a sunset method

```
/* Method to simulate a sunset by
 * decreasing green and blue by 30% */
public void makeSunset()
{
   Pixel[] pixelArray =
          this.getPixels();
   Pixel pixelObj = null;
   int index = 0;
   int value = 0;

   // loop through all the pixels
   while (index < pixelArray.length)
   {
     // get the current pixel
     pixelObj = pixelArray[index];

     // change the blue value
     value = pixelObj.getBlue();
     pixelObj.setBlue((int) (value * 0.7));

     // change the green value
     value = pixelObj.getGreen();
     pixelObj.setGreen((int) (value *
    0.7));

     // increment the index
     index++;
   }
 }
```

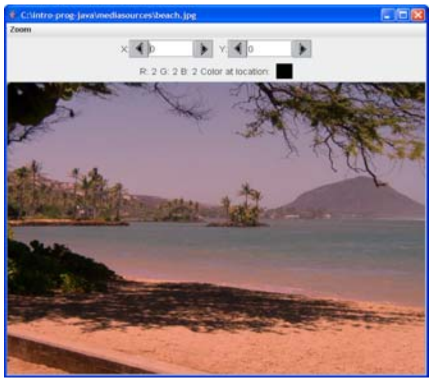
Testing the makeSunset() method


C:\intro-prog-java\mediasources\beach.jpg

Zoom

X: 0 Y: 0

R: 2 G: 4 B: 3 Color at location:


C:\intro-prog-java\mediasources\beach.jpg

Zoom

X: 0 Y: 0

R: 2 G: 2 B: 2 Color at location:

The beach.jpg picture before and after calling the makeSunset() method

Exercise

- Generalize the methods we have made:
  - Create a changeRed() method that takes a double parameter indicate how much to change the red
  - Create an even more generic changeColors() method that takes three double parameters that indicate how much change the red, green, and blue in the image
  - Re-implement makeSunset() using the new changeColors() method

Another kind of loop: for loops

- We have been using a **while loop** with a **counter**
  - We had to declare the counter variable and initialize it before the loop
  - We had to increment the counter in the loop
- The shortcut for this is a **for loop**
  - Programmers like shortcuts!
    - Especially those that reduce errors
    - And mean less typing

Add the numbers from 1 to 100

- Using a while loop:

```
int total = 0;
int counter = 1;
while (counter <= 100)
{
   total = total + counter;
   counter = counter + 1;
}
System.out.println("Sum of 1 to 100 is " + total);
```

Add the numbers from 1 to 100

- Using a for loop:

```
int total = 0;
for (int counter = 1; counter <=100; counter++)
{
   total = total + counter;
}
System.out.println("Sum of 1 to 100 is " + total);
```

Syntax

```
for (start; check; step)
  {
     body of loop
  }
```

- ***Start (initialization area)***
  - Declare loop variable and initialize it
- ***Check (continuation test)***
  - If true, do body of loop
  - If false, jump to next statement after the loop
- ***Step (change area)***
  - Change the loop variable

How does it work?

- Our example to add up 1 to 100:
  `for (int counter = 1; counter <=100; counter++)`
- Step 1: the loop variable counter is declared and initialized
- Step 2: the test is performed
  - If the condition counter <=100 is true, go to Step 3
  - If the condition is false, go to the statement after the body of the loop
- Step 3: the body of the loop is executed
- Step 4: the loop variable counter is incremented
- Go back to the test at Step 2

The loop variable

- The variables i, j, k are commonly used for the loop counter
- Our example to add up 1 to 100:
  for (int i = 1; i <=100; i++)
- If the loop variable is declared within the for loop, its scope is only within the body of the loop
- Example: what would happen if we had this?

```
for (int i = 1; i <=100; i++)
      {
          total = total + i;
      }
      System.out.println(i);
```

Examples

- Example 1:

```
int total = 0;
for ( int i = 1; i <=100; i = i + 2 )
{   total = total + i;
}
System.out.println(total);
```

- Example 2:

```
int total = 0;
for ( int i = 100; i >0; i -- )
{   total = total + i;
}
System.out.println(total);
```

Examples

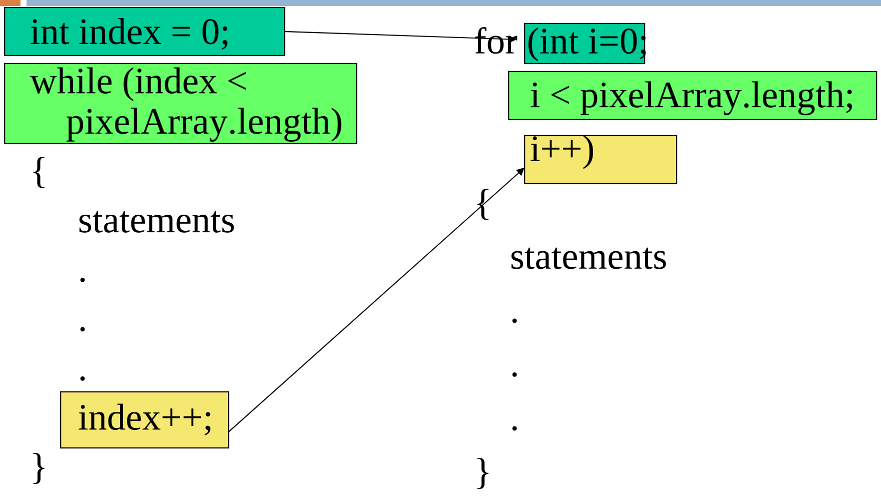
- Example 3 `int total = 0;`

```
for ( int i = 1; i <= 0; i++ )
{   total = total + i;
}
System.out.println(total);
```

- Example 4

```
int total = 0;
 for ( int i = 1; i > 0; i++ )
 {   total = total + i;
 }
 System.out.println(total);
```

Comparison of While and For Loops



```
int index = 0;
while (index <
    pixelArray.length)
{
     statements
     .
     .
     .
     index++;
}
```

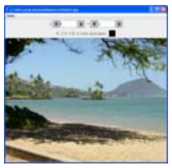
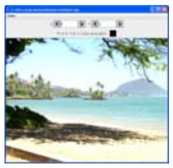
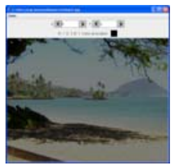
```
for (int i=0;
  i < pixelArray.length;
  i++)
{
   statements
   .
   .
   .
}
```

Method to clear blue in a picture

```
public void clearBlue()
{
    Pixel pixelObj = null;
    // get the array of pixels
    Pixel[] pixelArray = this.getPixels();
    // loop through all the pixels
    for (int i = 0;  i < pixelArray.length; i++)
    {
      // get the current pixel
      pixelObj = pixelArray[i];
      // set its blue to 0
      pixelObj.setBlue(0);
    }
  }
```

Lightening and darkening pictures

- Lightening and darkening images is now quite simple
  - We loop through all the pixels of an image
  - Instead of adjusting individual color components of each pixel, we tune the overall pixel color
  - We make use of the Color.brighter() and Color.darker() methods we saw earlier

Negating image algorithm

- Negating:
  1. Get the array of pixels from the picture
  2. Loop, starting array index at 0
  3. Check if the index is less than the length of the array
     1. If it is, go on to step 4
     2. If it isn't, we're done
  4. Get the pixel at the current index from the array of pixels
  5. Set the red value to (255 – current red value)
  6. Set the blue value to (255 – current blue value)
  7. Set the green value to (255 – current green value)
  8. Increment the index and go back to step 3 to process the next pixel in the pixel array

Negate method

```
/* Method to negate the picture */

public void negate()
{
   Pixel[] pixelArray =

    this.getPixels();
   Pixel pixelObj = null;
   int redValue, blueValue,

          greenValue = 0;

   // loop through all the pixels
   for (int i = 0;
    i < pixelArray.length; i++)
   {
     // get the current pixel
     pixelObj = pixelArray[i];

     // get the values
     redValue = pixelObj.getRed();
     greenValue = pixelObj.getGreen();
     blueValue = pixelObj.getBlue();

     // set the pixel's color
    pixelObj.setColor(
          new Color(255 - redValue,
                    255 - greenValue,
                    255 - blueValue));
    }
   }
```

Changing an image to grayscale

- **Grayscale** ranges from black to white
  - The red, green, and blue values are equal to each other
- How can we change any color to gray?
  - What number can we use for all three values?
    - The intensity of the color
  - We can average the colors
    - (red + green + blue) / 3
  - Example:
    - (15 + 25 + 230) / 3 = 90

Grayscale Algorithm

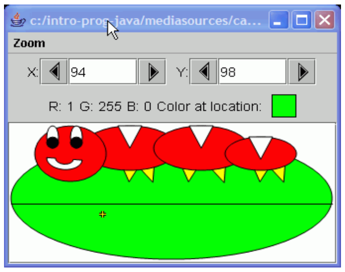
- Grayscale

```
1. Get the array of pixels from the picture
2. Loop, starting array index at 0
3. Check if the index is less than the length of the array
    1. If it is, go on to step 4
    2. If it isn't, we're done
4. Get the pixel at the current index from the array of pixels
5. Calculate the average of the current values
    (redValue + greenValue + blueValue ) / 3
6. Set the red value to the average
7. Set the blue value to the average
8. Set the green value to the average
9. Increment the index and go to step 3
```

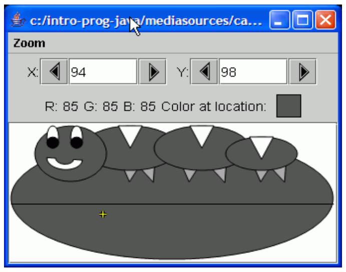
Grayscale method

```
/**
 * Method to change the picture to gray
   scale
 */
public void grayscale()
{
  Pixel[] pixelArray = this.getPixels();
  Pixel pixelObj = null;
  int intensity = 0;

  // loop through all the pixels
  for (int i = 0; i <

         pixelArray.length; i++)
  {
    // get the current pixel
    pixelObj = pixelArray[i];

    // compute the average intensity
    intensity =(pixelObj.getRed() +
                pixelObj.getGreen() +
                pixelObj.getBlue()) / 3;

    // set the pixel color
    pixelObj.setColor(new
   Color(intensity,intensity,intensity));

  }
}
```

Grayscale Result



R: 1 G: 255 B: 0 Color at location:



R: 85 G: 85 B: 85 Color at location:

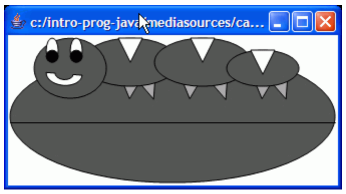
Luminance

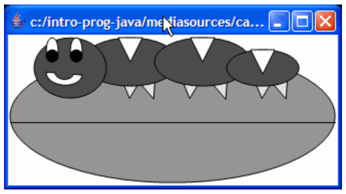
- **Luminance** is our perception of how light/dark things are
  - We perceive blue to be darker than red, and green
  - Even when the same amount of light is reflected
- A better grayscale model should take this into account:
  - Weight green the highest (* 0.587)
  - Red less (* 0.299) and
  - Blue the very least (* 0.114)

Exercise: grayscale with luminance

- Create a new method grayscaleWithLuminance()
- Using the new algorithm for calculating intensity:

  intensity = (int) (red * 0.299 +
  green * 0.587 + blue * 0.114)
- You should get results like the bottom image
  - This is better grayscaling than the top image, which resulted from the previous method grayscale()





Summary

- While loops, For loops
- Algorithms
- Local variables
- Shortcut operators ++ and --