Decrease Red Algorithm

- To decrease the red value in a picture by 50%:
  1. Get the array of pixels from the picture.
  2. Set up an index to start at 0.
  3. If the index is less than the length of the array?
     3.1. True: go to step 4 of the loop.
     3.2. False: go to step 9 (i.e. leave the loop)
  4. Get the pixel at the current index from the array of pixels.
  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. Increment the red value at the pixel.
  9. Done.
What is an Algorithm?

- An **algorithm** is a description of the steps needed to do a task.
  - It can be written in English.
  - A recipe is type of algorithm.

- A **program** is an implementation of an algorithm.
  - in a particular computer language.
  - In our case, Java.

From Algorithm to Program (code)

- How do we get the array of pixels from the current picture object?
  - We have used
    ```java
    Pixel[] pixelArray = picture.getPixels();
    ```
  - But we want to get the array of pixels from the current object
    - So we can use the keyword **this**
      ```java
      Pixel[] pixelArray = this.getPixels();
      ```
    - Or we can leave off the **this**
      ```java
      Pixel[] pixelArray = getPixels();
      ```

Loop Algorithm to Code

- How to write/code/implement the loop?
  - Use a while loop with a counter for the index starting at 0
    ```java
    int index = 0;
    ```
  - Loop while the index is less than the length of the array
    ```java
    while(index < pixelArray.length)
    ```
  - Get the current pixel from the array of pixels for the current index
    ```java
    Pixel pixel = pixelArray[index];
    ```

Loop Algorithm to Code - Continued

- Get the red value at the pixel
  ```java
  int value = pixel.getRed();
  ```
- Divide the red value by 2
  ```java
  value = value / 2;
  ```
- Set the pixel red value
  ```java
  pixel.setRed(value);
  ```
- Add one to (increment) the index
  ```java
  index = index + 1;
  ```
Decrease Red Method

/**
 * Method to decrease the red by half in the current picture
 */
public void decreaseRed()
{
    Pixel[] pixelArray = this.getPixels();
    Pixel pixel = null;
    int value = 0;
    int index = 0;
    // loop through all the pixels
    while(index < pixelArray.length)
    {
        // get the current pixel
        pixel = pixelArray[index];
        // get the red value
        value = pixel.getRed();
        // decrease the red value
        value = value / 2;
        // set the pixel red
        pixel.setRed(value);
        // increment the index
        index = index + 1;
    }
} // end decreaseRed()

Move Declarations Outside Loops

• When you need a variable in a loop it is best to declare it before (outside of) the loop.
  • Otherwise you are declaring a new variable each time through the loop.
  • Which is slower than just changing the value associated with the variable.

The Picture Explorer

• Tool that creates a copy of the current picture and lets you explore it
  – See the color, x, and y values at the cursor
• To use the tool on a picture object
  – pictureObj.explore();
• Use it to see if the colors have changed

Declarations in Java

• In some languages you must declare all variables at the beginning of a method (function).
• In Java you can declare variables anywhere in a method.
  – So long as you declare them before you use them.
• However, you should place all of your declarations together at the beginning of the method.
• This makes it easier to keep track of them as you are developing and debugging your program.
For Loops

• Programmers like shortcuts.
  – Especially those that reduce errors
  – And mean less typing

For Loops

• We have been using a `while` loop with an index
  – We had to declare the index variable and initialize it before the loop.
    • If you forget this you get a compiler error.
  – We had to increment the index in the loop.
    • If you forget this it will be an infinite loop.

For Loops

• The shortcut is a `for` loop.
  – You can use it in many of the same places as a while loop.
  – It keeps track of the index automatically.
    • Less work to do.
    • Fewer opportunities for errors.

For Loop Syntax

• The syntax of the for loop:

  ```plaintext
  for(initialization area; continuation test; change area)
  {
    statements;
  }
  ```

  – Initialization area
    • Declare variable(s) and initialize them
  – Continuation test
    • If true do body of loop
      • If false jump to next statement after the loop
  – Change area
    • Change the loop variable(s)
      – Increment or decrement them
For Loop Syntax

- The syntax of the for loop:
  ```
  for(initialization area; continuation test; change area)
  {
    statements;
  }
  ```
- Each of the three parts of a for loop are optional.
  - However, the ‘;’s have to be there.
- You can declare and initialize more than one variable in the initialization area, just separate them with commas.

Elements of for loops

- Programmers often use `i` as an index variable name.
- You should probably use something more descriptive, like `index`.

Comparison of While and For Loops

```java
int i = 0;
while(i < pixelArray.length)
{
    statements
}
```

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

Elements of for loops

- Programmers often use `i` as an index variable name.
- You should probably use something more descriptive, like `index`.

```java
for(int index = 0;
    index < pixelArray.length;
    index++)
{
    statements
}
```
Elements of for loops

- Even though the variable index is declared and initialized in the for loop it actually is only declared once before the first test.
- It is not repeated each time through the loop.

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

Elements of for loops

- Even though the increment is right after the continuation test in the for loop it doesn’t happen till after the last statement in the loop.

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

Shortcuts for Common Operations

- In programming you often need to add one to a value *(incrementing)*:
  ```
  index = index + 1;
  ```
- You may use the shortcut -
  ```
  index++;  
  or  
  ++index;
  ```

Shortcuts for Common Operations

- In programming you often need to subtract one to a value *(decrementing)*:
  ```
  index = index - 1;
  ```
- You may use the shortcut -
  ```
  index--;  
  or  
  --index;
  ```
**Shortcuts**

- The difference between 
  - `index++`
  and
  - `++index`
  is that
  - in `index++`, the variable is referenced first, and then incremented,
  - in `++index`, it is incremented first, and then referenced.
- The same holds for decrementing.

**Shortcuts**

- What does this mean?
  - Try
    ```java
    int sample = 5;
    System.out.println(sample++);
    System.out.println(sample);
    ```
    versus
    ```java
    int sample = 5;
    System.out.println(++sample);
    System.out.println(sample);
    ```
    and see the difference.

**Shortcuts for Common Operations**

- You can also use these shortcuts:
  - `x += y` for `x = x + y`
  - `x -= y` for `x = x - y`
  - `x *= y` for `x = x * y`
  - `x /= y` for `x = x / y`

**PUT IT TOGETHER**

Let’s apply what we’ve learned to `decreaseRed()`
/**
 * Method to decrease the red by 50%
 */
public void decreaseRed()
{
    Pixel[] pixelArray = this.getPixels();
    Pixel pixel = null;
    int index = 0;
    int value = 0;

    // loop through all the pixels
    while(index < pixelArray.length)
    {
        // get the current pixel
        pixel = pixelArray[index];
        // get the value
        value = pixel.getRed();
        // decrease value
        value = value / 2;
        // set the red value
        pixel.setRed(value);
        // increment the index
        index = index + 1;
    }
}
Decrease Red Exercise

- In DrJava
  - Add the method `decreaseRed()` to `Picture.java`
    - Put it before the last `}` which ends the class definition
  - Compile the method
    - Click the Compile All button
  - Test it by doing the following in the interactions pane
    (note that the file may be in a different location on your computer):

```
> String fileName = "C:/intro-prog-java/mediasources/caterpillar.jpg";
> Picture picture1 = new Picture(fileName);
> picture1.explore();
> picture1.decreaseRed();
> picture1.explore();
```

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.
    - Try to reduce the blue and green by 30%

Faking a Sunset Algorithm

- Reduce the blue and green by 30%
  1. Get the array of pixels from the picture
  2. Set up an 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 of the loop
     2. If it isn’t jump to the first instruction after the loop
  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

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

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

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

        // change the blue value
        value = pixel.getBlue();
        pixel.setBlue((int) (value * 0.7));
    }
}
```
Testing makeSunset

```java
> String file = "c:/intro-prog-java/mediasources/beach.jpg";
> Picture pictureObj = new Picture(file);
> pictureObj.explore();
> pictureObj.makeSunset();
> pictureObj.explore();
```

How do we Know if it Worked?

- A very important part of programming is **testing** the result:
  - Just because code compiles and runs without dying from an error doesn’t mean it is correct.
  - There could be an error in the **logic**.
  - It could fail to get the right answers.
    - Either always, or sometimes - under certain conditions.
    - It could even return the correct answer but for the wrong reason.

Tracing Code

- An important skill to develop is the ability to **trace** code.
  - Also called
    - **walking through**
    - **stepping through** your code.
  - Look at each line and predict what will happen.
  - Show the variables and their values.

Step Through decreaseRed()

- A picture object was created from the file **caterpillar.jpg** and then was sent the message **decreaseRed()**
- The picture object was implicitly passed to the method **decreaseRed()** and can be referred to by **this**
- The array of pixel objects was returned from sending **getPixels()** to the picture object
  ```java
  Pixel[] pixelArray = this.getPixels();
  ```
- Some variables were declared for later use in the loop
  ```java
  Pixel pixelObj = null;
  int index = 0;
  int value = 0;
  ```
Step Through decreaseRed() - cont

• The while loop tests if the index is less than the length of the array
  
  \[
  \text{while}(\text{index} < \text{pixelArray.length}) \{
  \]
  – And if so it executes the statements in the body of the loop \{ \ … \} 
  • It sets the variable \text{pixelObj} to the pixel at the current index in the array of pixels 
    \[
    \text{pixelObj} = \text{pixelArray}[\text{index}];
    \]
  • It gets the red value of that pixel 
    \[
    \text{value} = \text{pixelObj}.\text{getRed}();
    \]
  • It sets the value to the integer part of (red value * 0.5) 
    \[
    \text{value} = (\text{int})(\text{value} * 0.5);
    \]
  • It sets the pixel’s red to the new value 
    \[
    \text{pixelObj}.\text{setRed}(\text{value});
    \]
  • It increments the index value 
    \[
    \text{index}++;\]

Using System.out.println() in a Loop

• One way to check what is happening in your program is to add
  – \text{System.out.println}(\text{expression});
• You might add this to the loop to check the value of the variable \text{i} while the loop is executing.
  – And to verify that the increment happens after the last statement in the loop.