ICLICKER QUESTION

What will be output by the following code?

A. 0
   int index = 0;
   int count = 0;
B. 1
   while(index < 3)
   {
      count = count + index;
      index = index + 1;
   }
C. 6
D. 2
E. 3
   System.out.println(count);

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.

• 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.

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

• The syntax of the for loop:
  
  ```
  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

COMPARISON OF WHILE AND FOR LOOPS

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

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

ELEMENTS OF FOR LOOPS

• Programmers often use i as an index variable name.

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

• You should probably use something more descriptive, like index.
**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
    .
    .
    .
}
```

- 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.

```java
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*):
  ```java
  index = index + 1;
  ```

- You may use the shortcut -
  ```java
  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
  
  ```
  int sample = 5;  
  System.out.println(sample++);  
  System.out.println(sample);
  ```
  versus
  
  ```
  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 value = 0;
    int index = 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;
    }
}
```
/**
 * Method to decrease the red by 50%
 */
public void decreaseRed()
{
    Pixel[] pixelArray = this.getPixels();

    // loop through all the pixels
    for(int index = 0; index < pixelArray.length; index++)
    {
        pixelArray[index].setRed(pixelArray[index].getRed() / 2); // decr red
    }
}

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));
    }
}

> 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
  – 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:
  ```java
  while (index < pixelArray.length) {
  ```
- And if so it executes the statements in the body of the loop:
  ```java
  – It sets the variable `pixelObj` to the pixel at the current index in the array of pixels:
  ```java
  pixelObj = pixelArray[index];
  ```
- It gets the red value of that pixel:
  ```java
  value = pixelObj.getRed();
  ```
- It sets the value to the integer part of `(red value * 0.5)`:
  ```java
  value = (int) (value * 0.5);
  ```
- It sets the pixel’s red to the new value:
  ```java
  pixelObj.setRed(value);
  ```
- It increments the index value:
  ```java
  index++;
  ```

MEMORY MAP OF DECREASERED()

- What does memory look like the first time through the loop?
- How about the 2nd time through?
- How about the 3rd time through?
- How about the last time through?

USING SYSTEM.OUT.PRINTLN() IN A LOOP

- One way to check what is happening in your program is to add:
  ```java
  System.out.println(expression);
  ```
- You might add this to the loop to check the value of the variable `i` while the loop is executing:
  ```java
  – And to verify that the increment happens after the last statement in the loop.
NEGATING AN IMAGE

- How would you turn a picture into a negative?
  - White should become black
    • 255,255,255 becomes 0,0,0
  - Black should become white
    • 0,0,0 becomes 255,255,255

NEGATE ALGORITHM

- Subtract current value from 255 for red, green, and blue
  1. Get the array of pixels from the picture
  2. Loop starting an index at 0 and incrementing by 1
  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 red value to 255 – current red value
  6. Set the green value to 255 – current green value
  7. Set the blue value to 255 – current blue value
  8. Increment the index and go back to step 3

CHANGING TO GRAYSCALE

- Grayscale ranges from black to white
  - The red, green, and blue values are the same
- 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
    • \((\text{red} + \text{green} + \text{blue}) / 3\)
  - Example
    • \((15 + 25 + 230) / 3 = 90\)

```java
/**
 * Method to negate an image
 */
public void negate()
{
    Pixel[] pixelArray = this.getPixels();
    Pixel pixelObj = null;

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

        // change the red value
        pixel.setRed(255 - pixel.getRed());

        // change the green value
        pixel.setGreen(255 - pixel.getGreen());

        // change the blue value
        pixel.setBlue(255 - pixel.getBlue());
    }
}
```
**GRAYSCALE ALGORITHM**

- Set color values to the average of the original values
  1. Get the array of pixels from the picture
  2. Loop starting an 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 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. Calculate the average of the current values
     1. \((\text{redValue} + \text{greenValue} + \text{blueValue}) / 3\)
  6. Set the red value to the average
  7. Set the green value to the average
  8. Set the blue value to the average
  9. Increment the index and go to step 3

```java
/**
 * Method to convert a photo to grayscale
 */
public void grayscale()
{
    Pixel[] pixelArray = this.getPixels();
    Pixel pixel = null;
    int intensity = 0;

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

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

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

**TESTING GRAYSCALE**

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

**GRAYSCALE RESULT**
LUMINANCE

• 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)

GRAYSCALE WITH LUMINANCE EXERCISE

• Create a new method
grayScaleWithLuminance()
• Use the new algorithm for calculating intensity.
  • intensity =
      (int) (red * 0.299 +
           green * 0.587 +
           blue * 0.114);
  – Notice that we’re not dividing by 3.

TESTING GRAYSCALE WITH LUMINANCE

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

METHODS THAT CALL METHODS

• We can redesign makeSunset() to use three methods.
  – One to change the green value
  – One to change the blue value
  – One to call the first two methods.
METHODS THAT CALL METHODS

• This gives us two good general methods
  – One to change the green in an image
  – One to change the blue in an image
    that we can possibly use in other situations.

• It also makes the new version of `makeSunset()` very simple.

ICLICKER QUESTION

What will be output by the following code?

A. 5
   int value = 1;
   for(int i = 0; i < 5; i++)
   {
     value *= 2;
   }
   System.out.println(value);

B. 6
   for(int i = 0; i < 5; i++)
   {
     value *= 2;
   }

C. 16

D. 32

E. 64