

Objectives: methods; images; colors

2. First, implement the Add method with three integer parameters. Return the sum of the parameters - unless if any of the values are less than zero return zero; if the sum is greater than 1000 return 1000.

Second, implement an Add method with two integer parameters with the same rules except that if the first value is -1 return -1. *Hint: Call your first Add method.*

```
public class Program {
    public static void main(String[] args) {
        int a, b, c, sum;
        a = TextIO.getlnInt();
        b = TextIO.getlnInt();
        c = TextIO.getlnInt();
        sum = Add(a, b, c);
        System.out.println("Total is " + sum);
    }
}
```

}

Third ... Why can you not make a third method Add in the same class that takes two ints and returns a double? double Add(int x, int y)

1. Love...

Complete the following code to print out a random love letter. Choose a random phrase from each string array.

```
public static void main(String[] args) {
    String [][] letter = {
        {"Hi", "Dear", "Dearest"},
        {"Mike,", "Jenny,", "sugar,", "sweetheart,"},
        {"\n"},
        {"I can no longer", "I want to", "I need to"},
        {"think", "swim", "chill and watch Netflix movies",
         "break up", "sing country music"},
        {"for you.", "with you.", "about you."},
        {"\n"},
        {"Bye,", "Your loving friend,"},
        {"\n"},
        {"Jenny", "Jim"}
    };
}
```

3a. Introducing class methods. Think about Math.random()...

3b. Demo ... write a class method (subroutine) that solves the quadratic equation:

Given: $ax^2 + bx + c = 0$ then solution is: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

5. Create a **class** method '*debugArray*' that will print each value of a string array parameter.

Can you think of an application where a `debugArrayInt()` would be useful?

6. What is `(0xff << 16) | (9 << 8) | 21` in hexadecimal?

4. Merge...

Complete the following code to merge two sorted integer arrays together into a single output array

```
.
public static int[] merge(int[] A, int[] B) {

    int done = 0;

    int countA = 0;

    int countB = 0;

    int[] result = new int[_____];

    while ((countA < A.length) ____ _____) {

        if (_____) result[done++] = A[ countA++];

        else

            result[_____] = B[ _____];

    }

    while (countA < A.length)

        result[done++] = A[countA++];


}
```

7. Using 2D arrays to represent an image.

Create a picture of the JVMs memory and use memory pointers to explain why the following code swaps two rows.

```
int[][] pixels;
pixels = new int[480 /*row or 'y' coordinate*/][640 /* column or 'x'*/];
// initialize pixel array : Odd rows are black.
// Even rows are white
for(int y=0;y< 480; y++)
    for(int x = 0; x< 640; x++)
        if(y % 2 ==0) pixels[____][____] = 0xffffffff;

//0xffffffff = all white (red=255,green=255,blue=255)
int[] temp = pixels[10];
pixels[10] = pixels[11];
pixels[11] = temp;
```



```
int redComponent = 0xFF = 25510 = 111111112
int greenComponent = 0xA7 = 16710 = 101001112
int blueComponent = 0x33 = 5110 = 001100112
```

Encode these components into a single 4-byte integer...

```
int pixelValue = (redComponent<<16)|(greenComponent<<8)|(blueComponent)
```

```
(redComponent << 16) = 00000000011111111000000000000000000000002
(greenComponent << 8) = 00000000000000000000000101001110000000002
(blueComponent) = 000000000000000000000000000000000000001100112
```

```
pixelValue = 000000000111111110100111001100112
```

alpha

red

green

blue

Objectives: Subroutines and functions.

Up next: Friday - introducing Object Oriented Programming (OOP)!

MP4 due Monday.

1. First, implement the Add method with three integer parameters. Return the sum of the parameters - unless if any of the values are less than zero return zero; if the sum is greater than 1000 return 1000.

Second, implement an Add method with two integer parameters with the same rules except that if the first value is -1 return -1. *Hint: Call your first Add method.*

```
public class Program {
    public static void main(String[] args) {
        int a, b, c, sum;
        a = TextIO.getlnInt();
        b = TextIO.getlnInt();
        c = TextIO.getlnInt();
        sum = Add(a, b, c);
        System.out.println("Total is " + sum);
    }
}
```

3. Complete the following bucket sort code to sort the data array.

```
int[] data = {5,22,5,18,4,... 74623 more values between 0 & 999};
int max = 1000;
int[] histogram = new int[max];
// Phase 1, count the number of occurrences of 0,1,2,3... max-1
for(int i = 0; i < data.length; i ++)
```

```
// Phase 2, Use histogram to create the sorted output data
int ptr=0; // we will write values into data[ptr]
for(int value=0; value<max; value ++)
```

?

// This sort is fast but what limitations can you see with this algorithm?

}

Third ... Why can you not make a third method Add in the same class that takes two ints and returns a double?

```
double Add(int x, int y)
```

2. Where are the scoping errors in the following code? Can you fix them by adding parameters and return types?

```
public class Scope {
    private static final String BONJOUR = "hi"; // Class variable
    private static int count = 0; // Class variable
    public static void main(String [] args) {
        int i = 6; // Local (temporary) variable only accessible inside main
        friendlyMethod();
        Scope.friendlyMethod();
        TextIO.putln(hello);
    }
    public static void friendlyMethod() {
        Scope.printer("Welcome");
        printer("Huh?");
        String hello = "hello!";
        printer(hello + Scope.BONJOUR);
        printer(hello + BONJOUR);
        TextIO.putln("i = " + i);
        count++;
    }
    public static void printer(String h) {
        TextIO.putln(h+"...");
    }
}
```

3. Complete the following bucket sort code to sort the data array.

```
int[] data = {5,22,5,18,4,... 74623 more values between 0 & 999};
int max = 1000;
int[] histogram = new int[max];
// Phase 1, count the number of occurrences of 0,1,2,3... max-1
for(int i = 0; i < data.length; i ++)
```

?

```
// Phase 2, Use histogram to create the sorted output data
int ptr=0; // we will write values into data[ptr]
for(int value=0; value<max; value ++)
```

?

// This sort is fast but what limitations can you see with this algorithm?

4. Create a **class** method '*debugArray*' that will print each value of a string array parameter.

Can you think of an application where a `debugArrayInt()` would be useful?

5. What is $(0xff \ll 16) \mid (9 \ll 8) \mid 21$ in hexadecimal?

5b. Funky Town...

```
class PixelEffects {
...
public static int[][] funky(
    int[][] source, int[][] sourceB) {
    int width = source.length;
    int height = source[0].length;
    int[][] result = new int[width][height];
    for (int i = 0; i < width; i++)
        for (int j = 0; j < height; j++) {
            int rgb = source[i][j];
            int red = RGBUtilities.toRed(rgb);
            int green = RGBUtilities.toGreen(rgb);
            int blue = RGBUtilities.toBlue(rgb);

            result[i][j] =
                RGBUtilities.toRGB(0, Math.max(green, blue), 0);
        }
    return result;
}
```

6. Represent **Red-Green-Blue** Color Information as a single integer.

[<http://math.hws.edu/javanotes/c13/s1.html#GUI2.1.2>] The red, green, and blue components of a color are represented as 8-bit integers, in the range 0 to 255. When a color is encoded as a single int, the blue component is contained in the eight low-order bits of the int, the green component in the next lowest eight bits, and the red component in the next eight bits. (The eight high order bits store the "alpha component or transparency" of the color.) It is easy to translate between the two representations using the shift operators \ll and \gg and the bitwise logical operators $\&$ and \mid .

Briefly: If A and B are integers, then $A \ll B$ is the integer obtained by shifting each bit of A, B bit positions to the left; $A \gg B$ is the integer obtained by shifting each bit of A, B bit positions to the right; $A \& B$ is the integer obtained by applying the logical **and** operation to each pair of bits in A and B; and $A \mid B$ is obtained similarly, using the logical **or** operation. For example, using 8-bit binary numbers, we have:

$$\begin{array}{r} 01100101 \\ \& 10100001 \\ \hline 00100001 \end{array} \qquad \begin{array}{r} 01100101 \\ \mid 10100001 \\ \hline 11100101 \end{array}$$

Here are incantations that you can use to work with color codes.

***Can you write an alternative version for green?**

```
/* Suppose that rgb is an int that encodes a color.
   To get separate red, green, and blue color components: */
```

```
int red = (rgb >> 16) & 0xFF;
int green = (rgb >> 8) & 0xFF;
int blue = rgb & 0xFF;
```

where the binary representation of the integer 0xFF is:

```
0000 0000 0000 0000 0000 0000 1111 1111
  {      }      {      }      {      }      {      }
  most    most    most    most    least
significant byte
```

```
/* Suppose that red, green, and blue are color components in the range 0
   to 255. To combine them into a single int: */
```

```
int rgb = (red << 16) | (green << 8) | blue;
```