ARRAYS AND A SIMPLE 2D GRAPHICS LIBRARY

Benedict R. Gaster / @cuberoo_

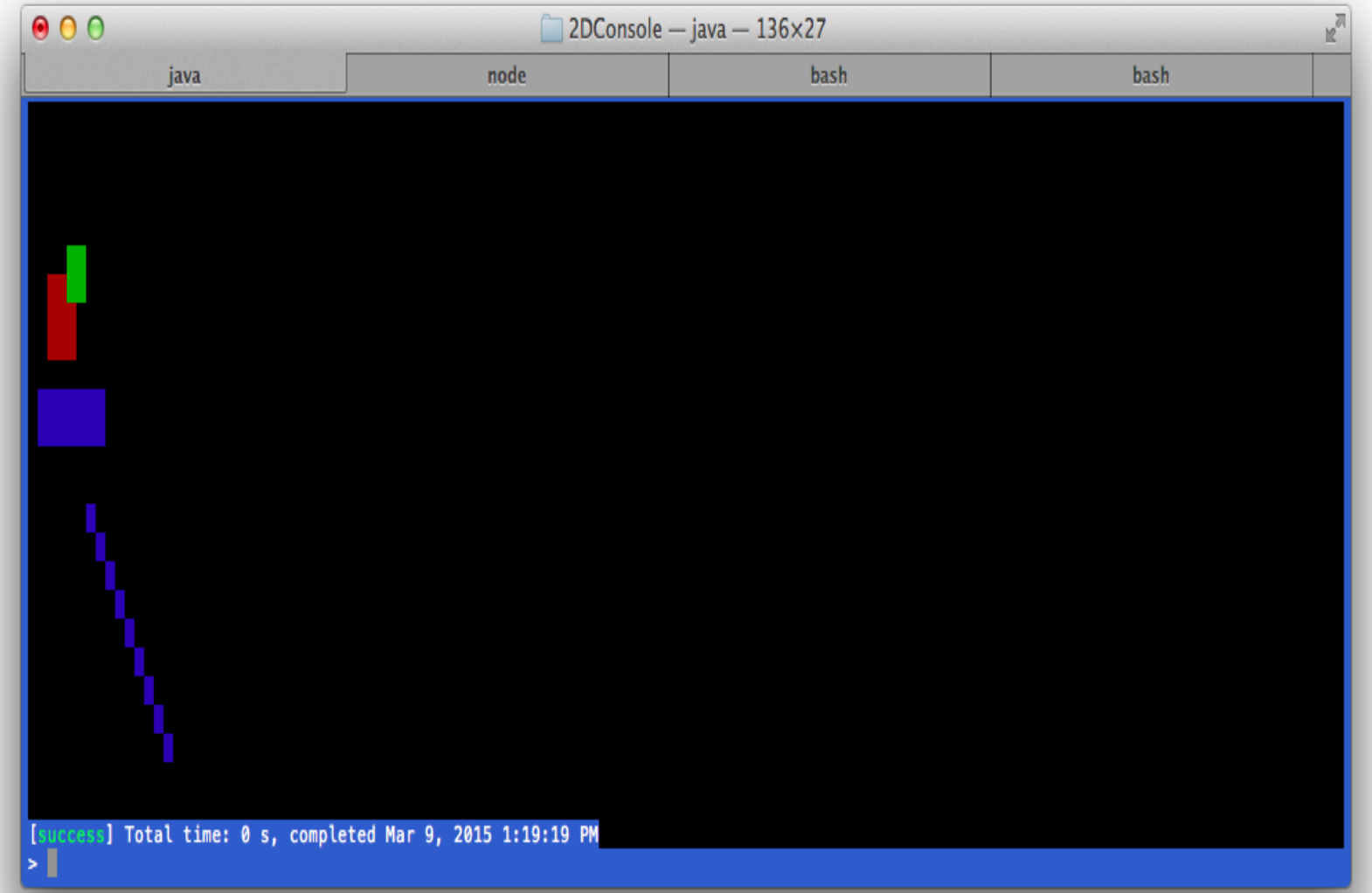
UWE BRISTOL
University of the West of England

bettertogether

2D GRAPHICS LIBRARY

- Represents a scene of 2D objects
- Renders (displays) the scene on the console
- Capable of drawing:
  - Line(x0, y0, x1, y1, color)
  - Square(x, y, size, color)
  - Square(x, y, size, color)
  - Rectangle(x, y, width, height, color)

OUR FINAL RESULT


2DConsole — java — 136×27
java
node
bash
bash
[success] Total time: 0 s, completed Mar 9, 2015 1:19:19 PM
>

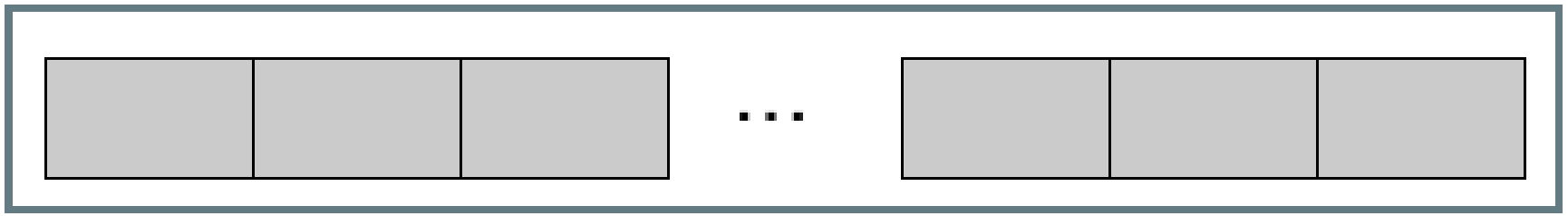
SCALA FUNCTIONLITY WE ARE GOING TO USE

- Base types (**Int**) - lecture 02
- For loops and conditionals - lecture 02
- Objects - lectures 03 and 04
- Classes with inheritance - lectures 03 and 04
- Arrays - today's lecture

WHAT IS AN ARRAY

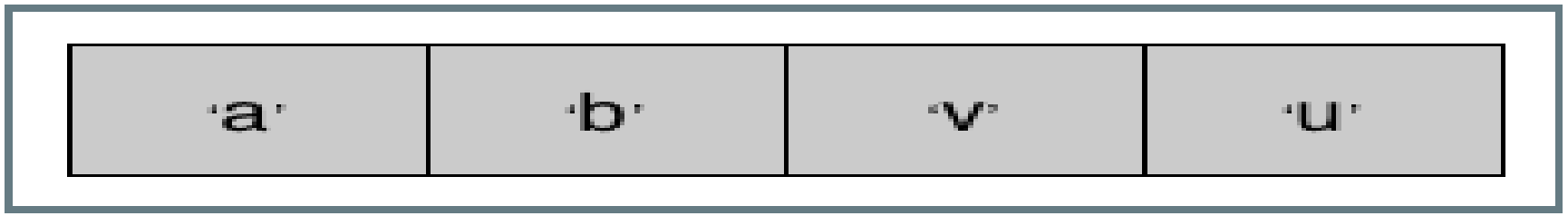
A collection of **things** of the same type

VISUALLY AN ARRAY LOOKS LIKE



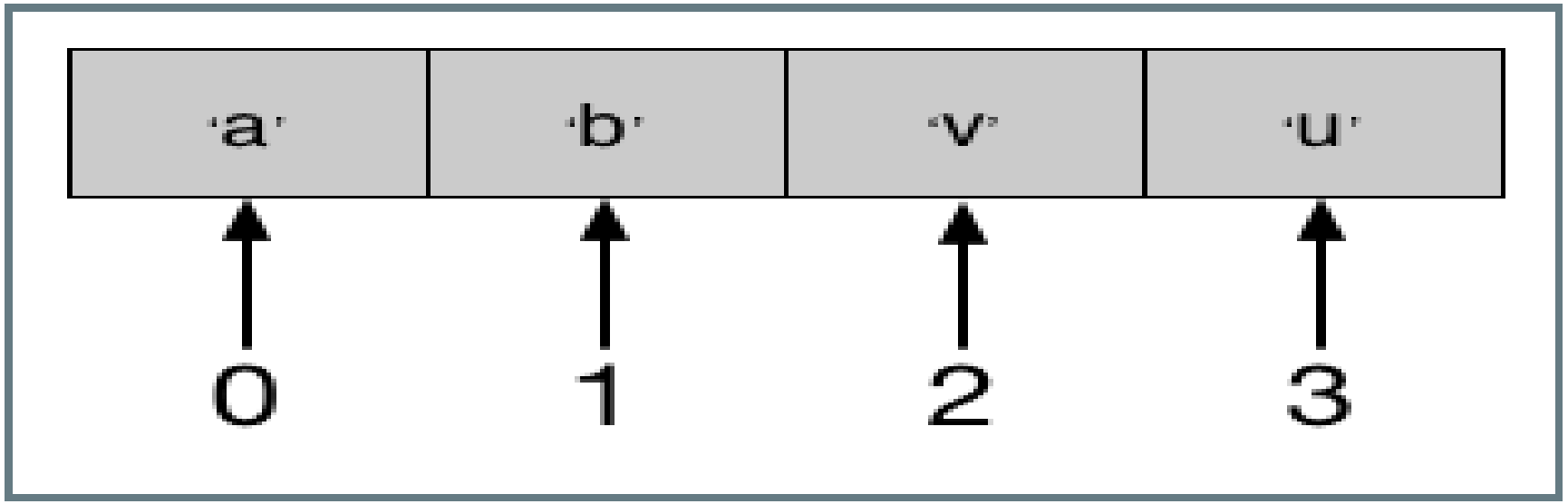
NUMBER OF ELEMENTS IN ARRAY

- An array has a fixed number of elements
- For example, a 4 element array of characters:



ACCESSING AN ARRAY

- An array is accessed
  - from left to right
  - first element has an index of 0
  - last element has an index of (N-1), where N is the number of elements



ARRAYS IN SCALA

- Array class
- type of elements contained in an array
- number of elements contained in an array
- general form looks like this

```
Array[ type ](size)
```

ARRAY DEFINITION EXAMPLES

```
var x = Array[Int](100)      // x coordinates in 2D point
var y : Array[Int](100])     // y coordinate in 2D point
var name = Array[Int](256)   // person's name, array of charaters
```

ARRAY ACCESS

```
x(3)        // access the forth element of x
y(0)        // access the third element of y
name(256); // access the 257 element of name, ERROR as name only has 256
```

- Access to an element NOT in an array is said to be out of bounds!
- Out of bounds access will raise a runtime exception!

USE LOOPS TO ACCESS MULTIPLE ELEMENTS

```
def drawPoints(x : Array[Int], y : Array[Int]) {
   for (i <- 0 until x.length)
       drawPoint(x(i), y(j))
}
```

ALTERNATIVELY

```
def drawPoints(x : Array[Int], y : Array[Int]) {
  for {
    i <- 0 until x.length)
  } drawPoint(x(i), y(j))
}
```

- Why use this approach?

MULTIPLE DIMENSIONAL ARRAYS

- No direct support multi-dimensional arrays, but
  - Provides various methods to process arrays in any dimension
  - For example, the following might define a two-dimensional array for our screen

```
var screen = Array.ofDim[Color.ColorT](width,height)

// Color.ColorT is the type of colors
// width and height are the size of our screen in "pixels"
```

ARRAY ACCESS FOR MULTI DIMENSIONS

```
screen(3)(2)  // accesses column 4 (in X dimension) and
              // row 4 (in Y dimension)
```

ACCESSING LENGHT OF EACH DIMENSION

```
scala> val a = Array.ofDim[Int](2,3)
a: Array[Array[Int]] = Array(Array(0, 0, 0), Array(0, 0, 0))

scala> a.length // length of 1st dimension
res3: Int = 2

scala> a(0).length // length of 2nd dimension
res4: Int = 3
```

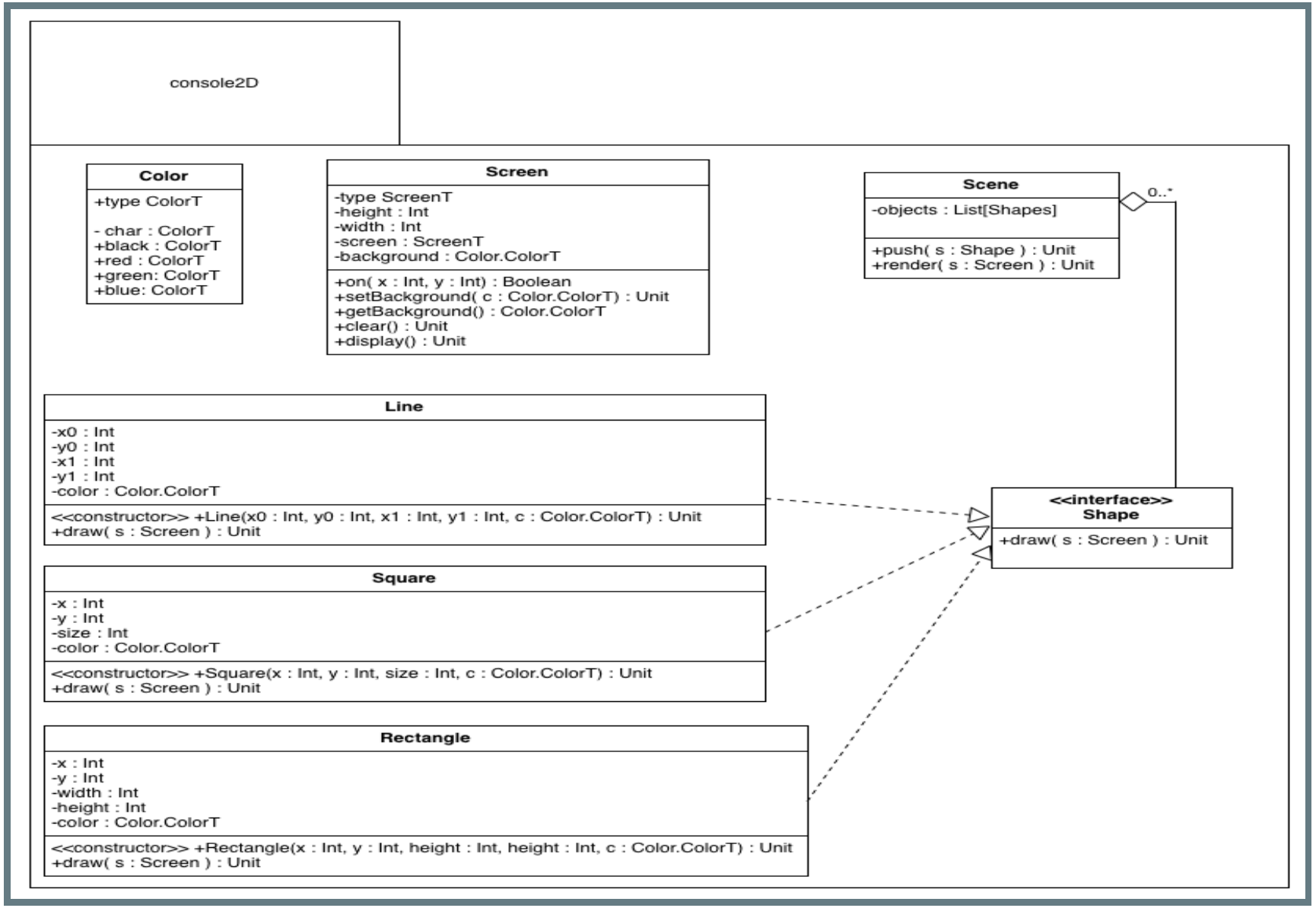
LOOPS AGAIN - MULTIPLE FOR LOOPS

```
def display(screen : Array[Array[Color.ColorT]]) {
  for (i <- 0 until screen.length)
    for (j <- 0 until screen(0).length)
      drawPoint(screen(i)(j))
}
```

LOOPS AGAIN - ALTERNATIVE

```
def display(screen : Array[Array[Color.ColorT]]) {
  for {
    i <- 0 until screen.length
    j <- 0 until screen(0).length
  } drawPoint(screen(i)(j))
}
```

2D GRAPHICS LIBRARY



DRAWING OUR SAMPLE SCENE

```
// create a new scene
var scene = new Scene()

// draw a couple of overlapping squares
scene.push(new Square(2,2,3,Color.red))
scene.push(new Square(4,1,2,Color.green))

// draw a line
scene.push(new Line(6,10,15,20, Color.blue))

// draw a rectangle
scene.push(new Rectangle(1,6,7,2,Color.blue))
```

RENDERERING OUR SAMPLE SCENE

```
// Setup the display
val screen = new Screen(20,20)
screen.setBackground(Color.black)

scene.render(screen)

// Finally make the screen visible
screen.display()
```

SOURCE CODE

- Contained in the introduction examples on github!
  - https://github.com/bgaster/scala-intro

EXERCISES

1. Implement a method to draw a rectangle of a given length and height at point x,y
2. Reimplement the library to use a class to represent points
3. Reimplement the **draw** method for square and rectangle interms of **Line**
    - What do you notie about this approach?
4. Modify the **Screen** class to use the length method of the **screen** array, rather than **width** and **height**
5. Extend the library to support rendering text