

CS101

# Problem Solving and Object-Oriented Programming

## L20: Arrays



## Pre- and Post-fix operators

- Consider the following:

```
int x = 0, y = 0;
```

x:	0	y:	0
----	---	----	---

```
y++; increment y
```

x:	0	y:	1
----	---	----	---

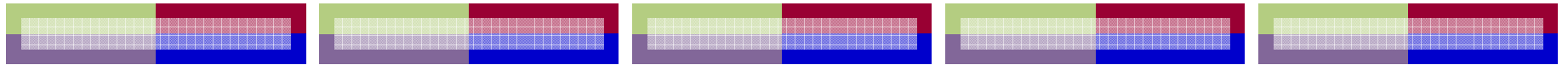
```
x = y++; assign x, then increment y
```

x:	1	y:	2
----	---	----	---

```
x = ++y; increment y, then assign x
```

x:	3	y:	3
----	---	----	---





# Arrays

- Hold a collection of items of the same type and allow access to individual items
- The number of objects is fixed at declaration

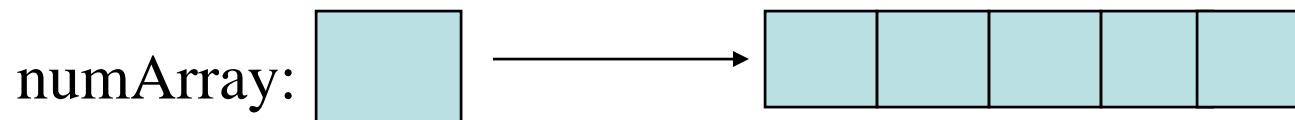
- Declaring an array:

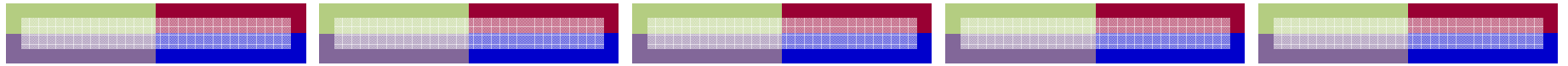
- `private int[] numArray;`

numArray: 

- Constructing an array:

- `shapeArray = new int [5];`





# Arrays

- Collection can be primitive types as on previous slide, or can be an object type

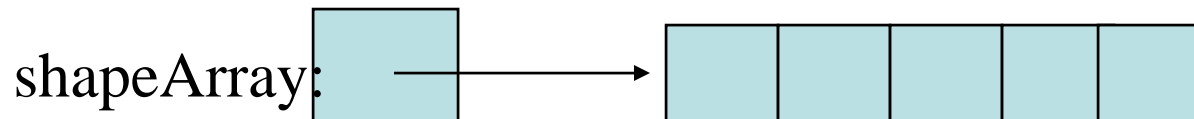
- Declaring an array:

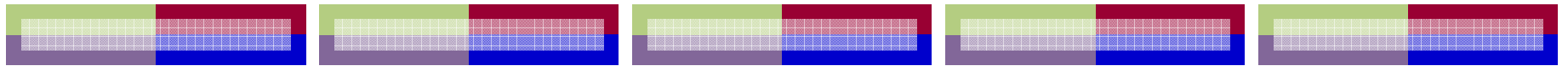
- `private FilledOval[] shapeArray;`

`shapeArray:` 

- Constructing an array:

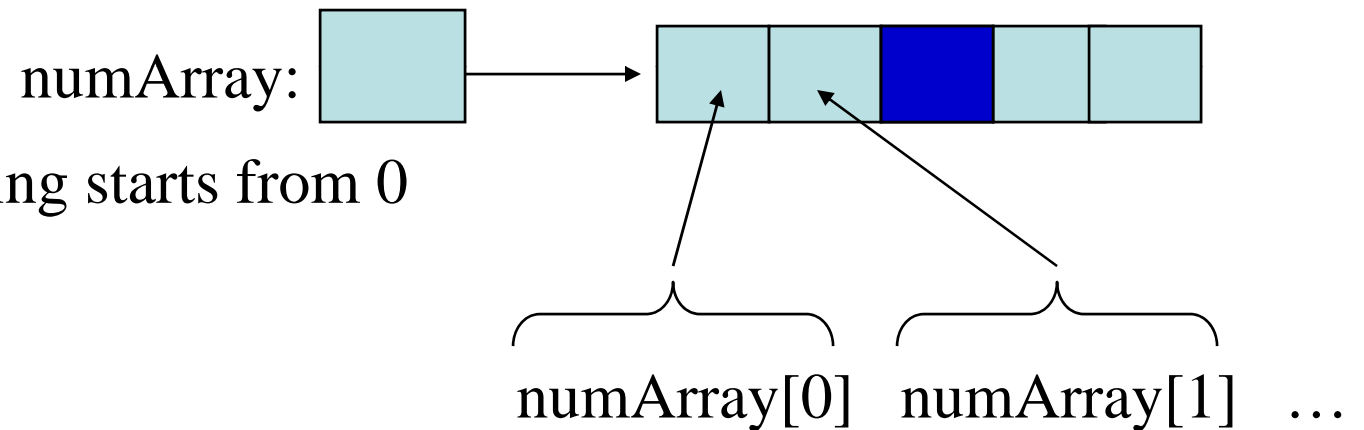
- `shapeArray = new FilledOval [5];`



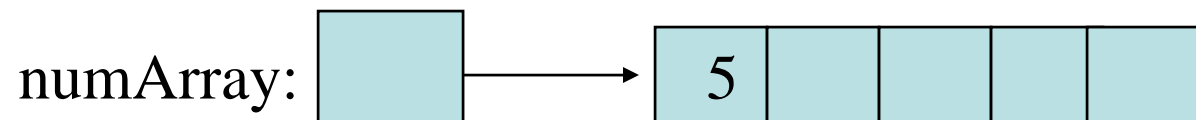


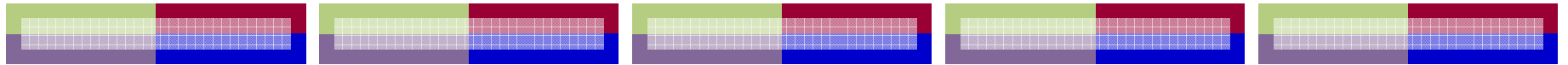
# Arrays

- Accessing an element in the array:
  - `numArray[2]` (the third element in the array)
    - Here 2 is the index of the array



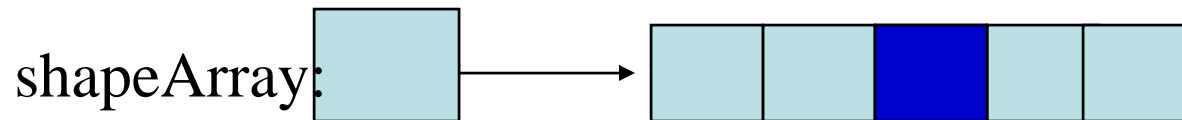
- Assigning a value:  
`numArray[0] = 5;`





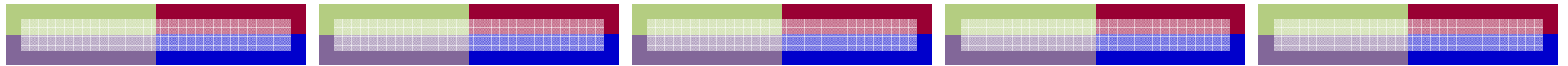
# Arrays

- Accessing an element in the array:
  - `shapeArray[2]` (the third element in the array)
    - Here 2 is the index of the array



- The array length is 5, but index values range from 0-(length-1) or 0-4
- `shapeArray[5]` does not exist and will generate a run-time exception
- Array values initialized to:
  - 0 if Type is int or double
  - false if Type is boolean
  - null if object type





# Walking an Array

```
public void colorShapes ( ) {  
    Color newColor = new Color(colorGen.nextValue(),  
                               colorGen.nextValue(), colorGen.nextValue());  
    for (int i = 0; i < shapesArray.length; i++) {  
        if (shapesArray[i] != null ) {  
            shapesArray[i].setColor(newColor);  
        }  
    }  
}
```

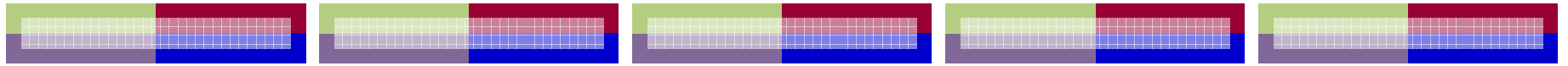
- Notes:

shapesArray.length tells us how big the array is

arrays are indexed from 0 to its length - 1

for loops are commonly used to walk an array



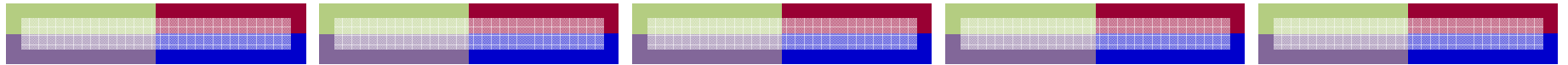


## Simpler way to Walk an Array

```
public void colorShapes ( ) {  
    Color newColor = new Color(colorGen.nextValue(),  
                               colorGen.nextValue(), colorGen.nextValue());  
    for (int i = 0; i < numShapes; i++) {  
        shapesArray[i].setColor(newColor);  
    }  
}
```

Only call setColor on those shapes that have been instantiated; no need to check for null

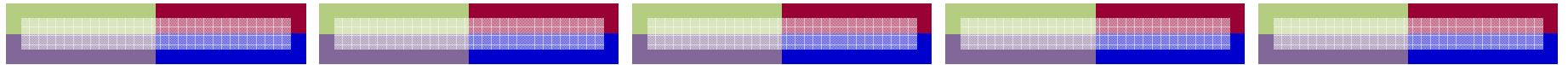




# Strings

- A sequence of characters:
  - “I’m touched”, “Lisa” // string literals enclosed in “”
    - Character literal uses ‘’: ‘L’
  - Recall: `new Text(“I’m touched”, x, y, canvas);`
- Let `counter = 5;`
  - `System.out.println(“Loop iterations: “ + counter);`
  - Loop iterations: 5





# Strings

## • Declaration

- String Name;

## • Initialization

- Name = "Lisa";
- String Name = "Lisa";

## • What if I want to add the 1st initial of my last name?


- Name = Name + " B"; // Name is now "Lisa B"





# Classes and Primitive Types

	Primitive Types	Classes
Examples	int, double, boolean	FilledRect, Flower
Values	1, 2.5, false	new FilledRect(...);
Operations	x+y, a&&b	rect.setColor(...);
Comparison	x == y	rect1.equals(rect2);





# Strings

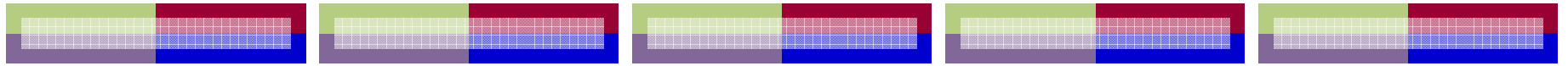
	Primitive Types	Classes
Values	“Lisa”	<code>new String(...);</code>
Operations	<code>x+y</code>	<code>str.charAt(0)</code>
Comparison		<code>str1.equals(str2)</code>

Note: `str1 == str2` compiles, but might not always do what you want!

Tests whether `str1` and `str2` refer to the same object.

So avoid `==` with strings!





# Strings and Characters

- Getting a character at a particular location in a string

```
String name = "Lisa B";
```

```
char firstChar = name.charAt(0); // firstChar => 'L'
```

```
int len = name.length(); // len contains the string's length
```

```
char lastChar = name.charAt(len-1); // lastChar => 'B'
```

