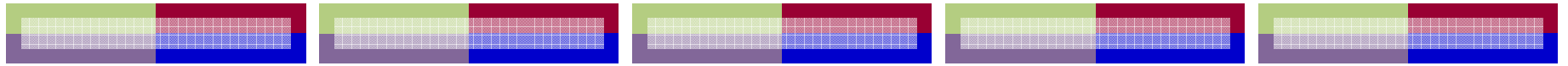


CS101

Problem Solving and Object-Oriented Programming

L21: Arrays II



Arrays

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

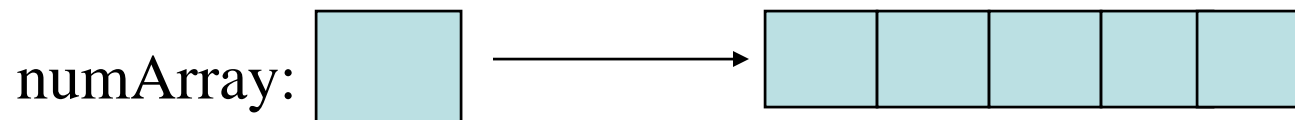
- Declaring an array:

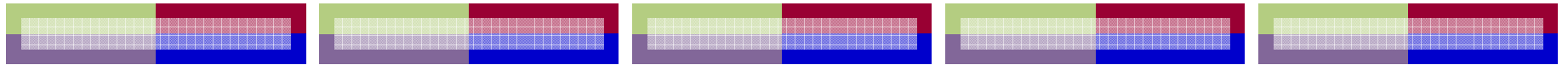
- `private int[] numArray;`

numArray: 

- Constructing an array:

- `numArray = new int [5];`





Arrays

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

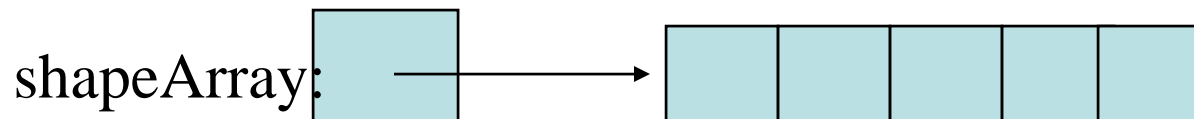
- Declaring an array:

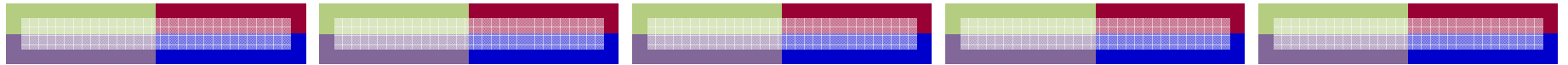
- `private FilledOval[] shapeArray;`

shapeArray: 

- Constructing an array:

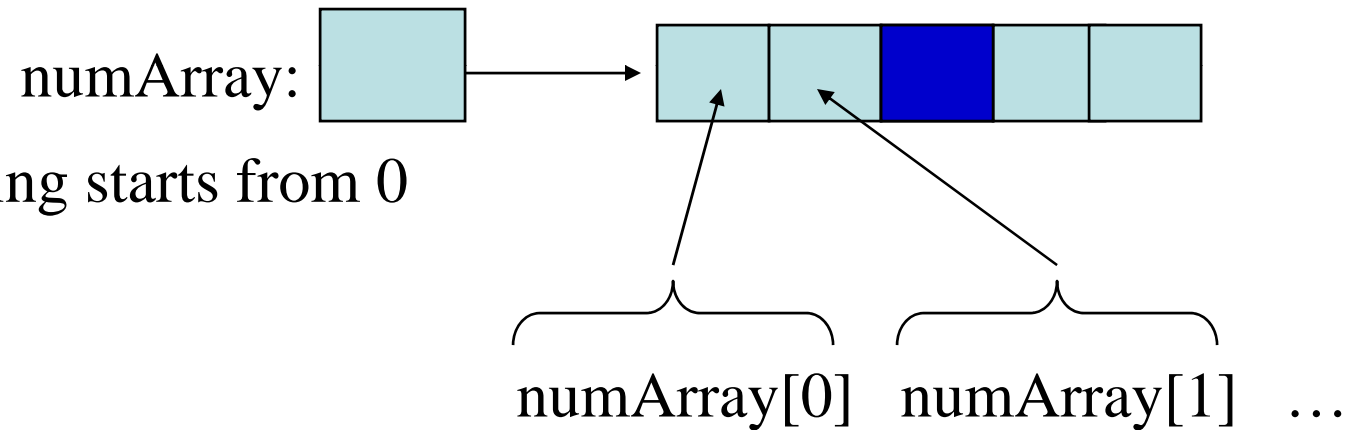
- `shapeArray = new FilledOval [5];`



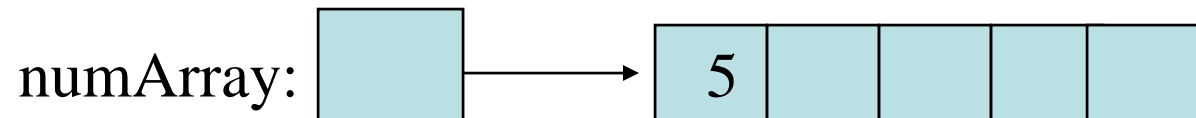


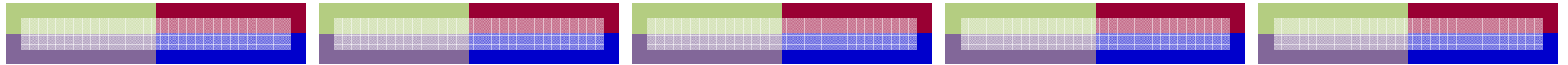
Arrays

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



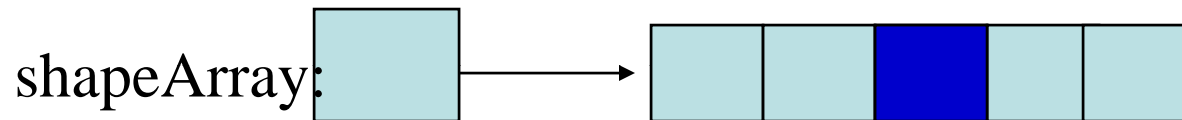
- Assigning a value:
`numArray[0] = 5;`





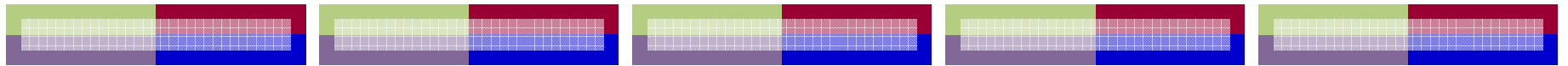
Arrays

- 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);  
        }  
    }  
}
```

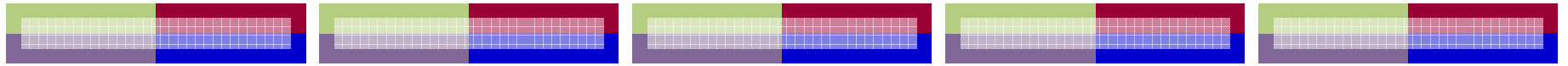
- Notes:

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



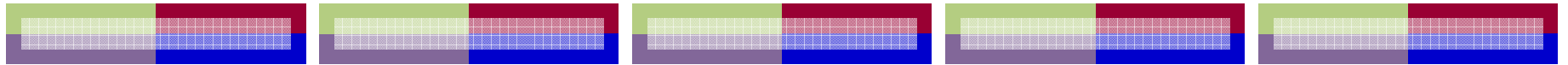


Simpler way to Walk an Array

```
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



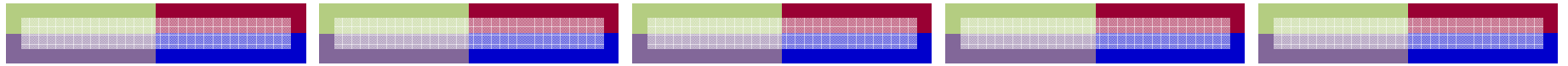


Let's create a simple drawing program that allows the user to place colored dots on the canvas, drag them around and change their color when the mouse exits the canvas:

In the Spring lab, we could only drag (grow) or change the color of the most recently created flower. Why?

We'll need 2 classes. 1 for the collection of shapes,
1 to control the window



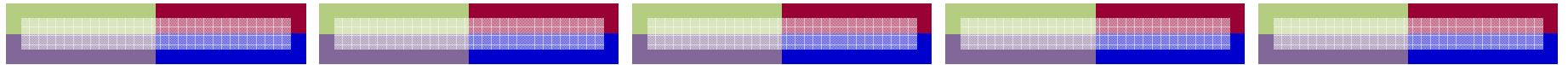


Let's create a simple drawing program that allows the user to place colored dots on the canvas, drag them around and change their color when the mouse exits the canvas:

We'll need 2 classes: 1 for the collection of shapes,
1 to control the window

Want each class to be responsible for its own data and actions. In other words, if an object of one class needs information about an object of another class, it must request that information. This means that the class of the object that will get the request, must have some way to take action given the request.





ShapesCollection

• Data

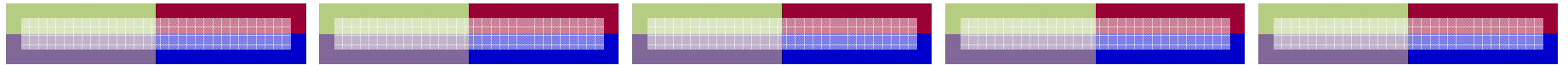
- The collection (place to store references to the shapes)
- Collection size `private static final int MAX_SHAPES;`
- Number of shapes so far `private int numShapes;`
- Color generator

• Constructor

• Actions

- Add a shape to the collection
- Retrieve a shape from the collection
- Color the shapes





```
public class ShapeCollection {
```

```
    private static final int MAX_SHAPES = 200; // max number of shapes in collection
```

```
    private FilledOval[] shapesArray; // The actual shapes
```

```
    private int numShapes; // The number of shapes currently in the collection
```

```
    /*** Create a new, empty collection**/
```

```
    public ShapeCollection () {
```

```
        shapesArray = new FilledOval[MAX_SHAPES];
```

```
        numShapes = 0;
```

```
    }
```

```
    // remove the shape at index specified by position
```

```
    public void removeShapeAtPosition (int position) {
```

```
        if (position < numShapes) {
```

```
            numShapes--;
```

```
            for (int place = position; place < numShapes; place++) {
```

```
                shapes[place] = shapes[place+1];
```

```
            }
```

```
            shapes[numShapes] = null;
```

```
        }
```

```
    }
```

