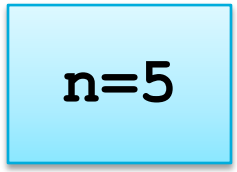
CIS 351-Data Structure-Array
Jan 23, 2020

Dr. Farzana Rahman

Syracuse University

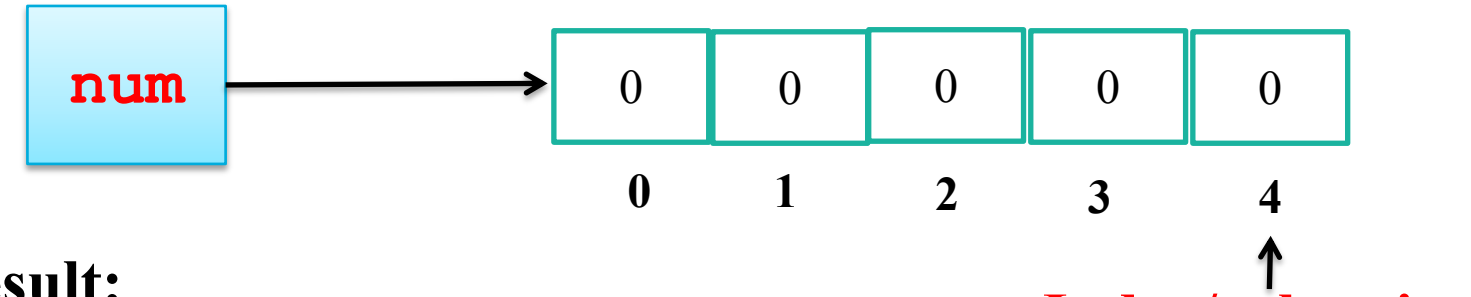
Declaring and Creating Arrays

- Variables can hold one single value



- Arrays can hold multiple values of the same data type.
- Syntax to declare Arrays:

```
int[] num = new int [5];
```



- **Result:**
  - 5 integer locations are allocated
  - They are initialized to 0

Declaring Arrays

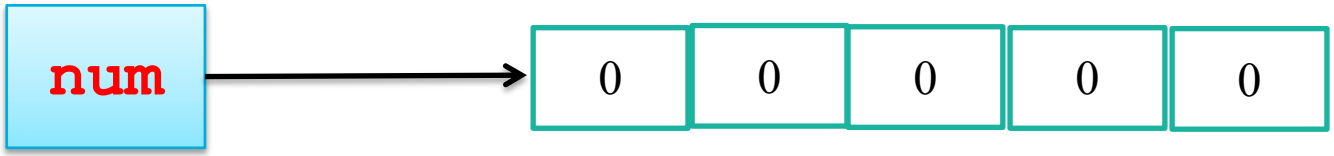
- Some examples of array declarations:

- **`double[] prices = new double[500];`**
- **`boolean[] flags;`**

  `flags = new boolean[20];`
- **`char[] codes = new char[1750];`**

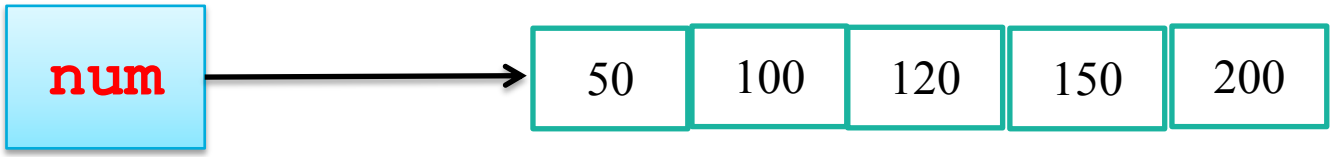
Initializer Lists

- `int[] num = new int [5];`



- Initializing an array:

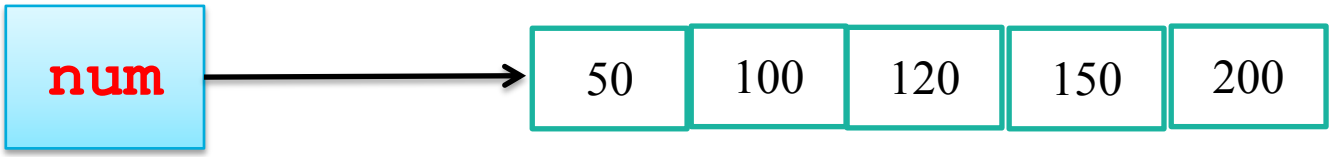
```
num[0] = 50;
num[1] = 100;
num[2] = 120;
num[3] = 150;
num[4] = 200;
```



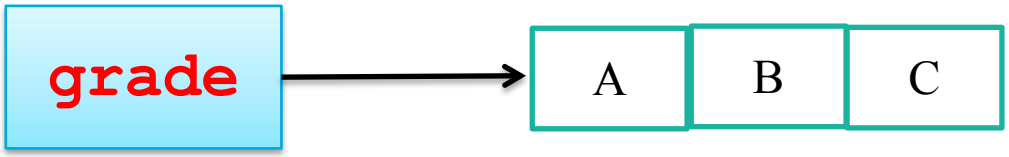
Using initializer to initialize

- An *initializer list* can be used to instantiate and initialize an array in one step: Examples:

```
int[] num= {50, 100, 120, 150, 200};
```




```
char[] grade = {'A', 'B', 'C'};
```



Using loop to initialize array with values

`array.lenght` = how many elements is the array

- An *initializer list* can be used to instantiate and initialize an array in one step: Examples:

```
for (int i=0; i<=num.length -1; i++)
{
    num[i]=keyboard.nextInt();
}
```

i = 0
i = 1
i = 2
i = 3
i = 4
i = 5

Bounds Checking

- For example, if the array `num` can hold 100 values, it can be indexed using only the numbers 0 to 99
- The following reference will cause an exception to be thrown:

```
System.out.println (num[100]);
```

- It's common to introduce *off-by-one errors* when using arrays

problem

```
for (int index=0; index <= 100; index++)
    num[index] = index + 10;
```

Fix = always use **index < length** or **index <= (length-1)**

Processing Array Elements
Array Length

To display the elements of the array referenced by *values*, we could write:

```
int count;
int[ ] values = {13, 21, 201, 3, 43};

for (count = 0; count < values.length; count++)
{
    System.out.println("Value " + (count+1) + " in the
     list of values is " + values[count]);
}
```

Enhanced `for` Statement

- Enhanced `for` statement
  - Allows iterates through elements of an array or a collection without using a counter: **Syntax:**

```
for ( datatype parameter : arrayName )
    Statement
```

Example:

```
int[] num= {50, 100, 120, 150, 200};
for (int val : num)
    System.out.print(val +" \t");
```

Printed Result:

50	100	120	150	200

Do’s and Don'ts – Enhanced for loop

- **When to use it:**
- good for displaying array from first to last element

- **When not to use it:**
- Accessing elements in reverse order
- Manipulating array elements
- Changing the array elements
- Need to access only couple of array elements

Processing Array Elements
reassigning array reference variables

```
int[ ] oldValues = {10, 100, 200, 300};
int[ ] newValues = new int[4];
```

- This assignment operation does not copy the content of **`oldValues`** array to the **`newValues`** array

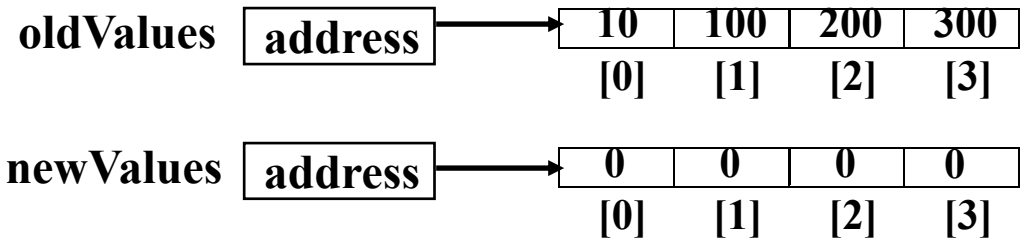
- To copy the contents of one array to another you must copy the individual array elements.

Processing Array Elements

reassigning array reference variables

- After the following statements execute:

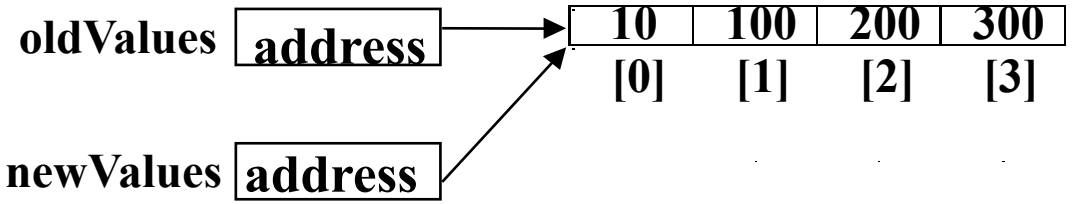
```
int[ ] oldValues = {10, 100, 200, 300};
int[ ] newValues = new int[4];
```



- Now, if we have the following statement

```
newValues = oldValues;
```

// It will copy the address in oldValues into newValues



Processing Array Elements
reassigning array reference variables

- To copy the contents of *oldValues* to *newValues* we could write:

```
int count;
int[ ] oldValues = {10, 100, 200, 300};
int[ ] newValues = new int[4];
```

// If newValues is large enough to hold the values in oldValues

```
if (newValues.length >= oldValues.length)
{
    for (count = 0; count < oldValues.length; count++)
    {
        newValues[count] = oldValues[count];
    }
}
```

Some Useful Array Operations
comparing array elements

The two arrays are not stored in the same memory location, so their equality testing returns false

```
char[ ] array1 = {'A', 'B', 'C', 'D', 'A'};
char[ ] array2 = {'A', 'B', 'C', 'D', 'A'};
boolean equal = false;

if (array1 == array2)
// This is false - the addresses are not equal
{
     equal = true;
}
```

•To compare the contents of two arrays, you must compare the individual elements of the arrays.

Passing array element parameter

- An array element can be passed as an argument to a method.

```
int[] num= {50, 100, 120, 150, 200};
int minNum = findMin(num [0]);
                ……..
```

```
public int findMin(int value)
{
     ……..
}
```

value = 50

Pass by Value

Passing array parameter

- An array can be passed as an argument to a method.

```
int[] num= {50, 100, 120, 150, 200};
int minNum = findMin(num);
                ……..
```

Pass by Ref

```
public int findMin(int[] numArray)
{
     ……..
}
```