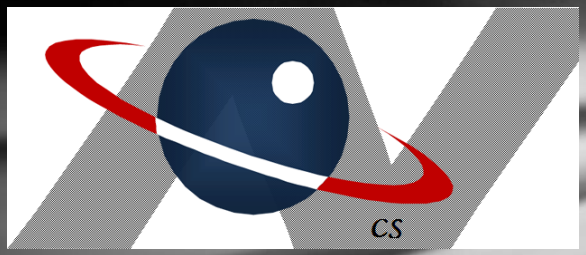
O(N) CS LESSONS

Lesson 7B – Array methods



By John B. Owen

Table of Contents

Objectives

- Arrays were introduced in Lesson 6D with the String "split" process, and further explored in lesson 7A.
- This lesson will discuss the special array processing methods found in the ***System*** and ***Arrays*** class that allow for even more processing techniques, including copying, filling, searching, and sorting.

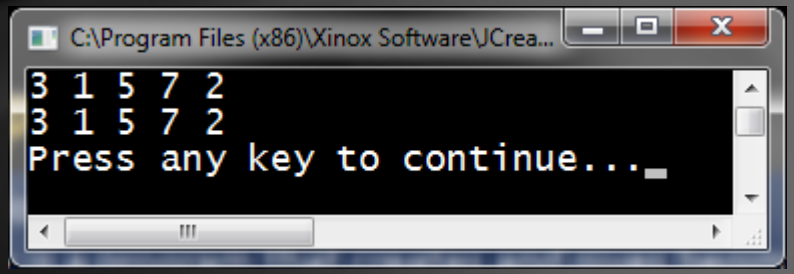
Array copying

- The first part of this lesson is devoted to techniques for creating duplicate arrays, either in whole or in part.
- There are special methods that help do this very quickly and easily, but first we must clarify a few things.

Shallow copy

- Let's look again at an array of integers.
- Below is a program that creates and gives beginning values to the array ***nums***, assigns the existing array to another array variable ***nums2***, then outputs both.
- Notice that the output for each is the same.
- This is because both array variables, ***nums*** and ***nums2***, are referencing, or pointing to, the same array!

```
{
    public static void main (String [] args)
        throws IOException
    {
        int [] nums = {3,1,5,7,2};
        int [] nums2 = nums;
        for(int x:nums)
            out.print(x+" ");
        out.println();
        for(int x:nums2)
            out.print(x+" ");
        out.println();
    }
}
```



Shallow copy

- Further evidence of this can be shown by making a change using one variable, and seeing that it also changes the other.
  - Using ***nums***, we change the value of one element in the array, then output both arrays again.
- Both outputs reflect the change, which proves that any change made to one is also made to the other!

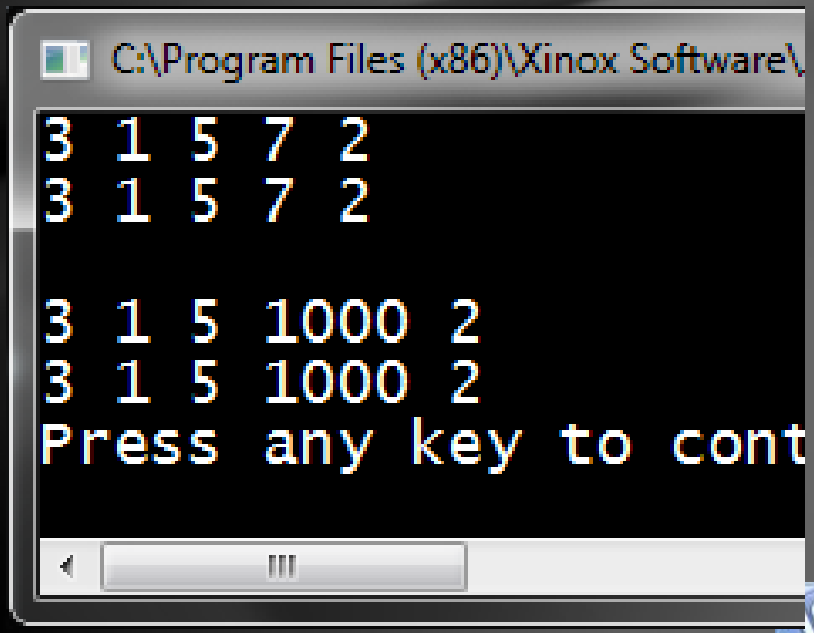
```
        throws IOException
    {
        int [] nums = {3,1,5,7,2};
        int [] nums2 = nums;
        for(int x:nums)out.print(x+" ");
        out.println();
        for(int x:nums2)out.print(x+" ");
        out.println();
        nums[3] = 1000;
        out.println();
        for(int x:nums)out.print(x+" ");
        out.println();
        for(int x:nums2)out.print(x+" ");
        out.println();

    }
}
```

```
3 1 5 7 2
3 1 5 7 2

3 1 5 1000 2
3 1 5 1000 2
Press any key to cont
```

Shallow copy

- This process is called a SHALLOW COPY, because the object reference variable ***nums2*** is simply assigned the **memory location** of ***nums,*** not a true copy of the array.

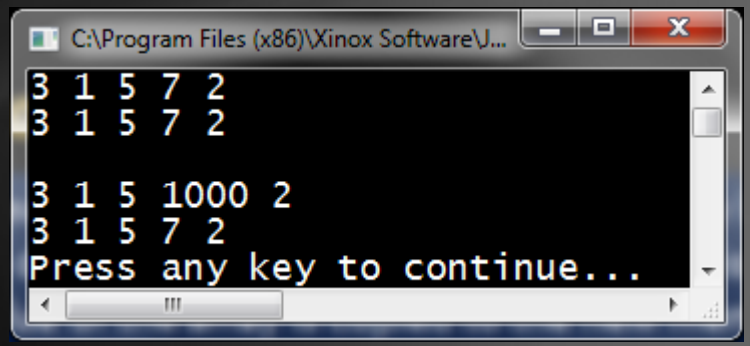
```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    int [] nums2 = nums;
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();

}
}
```



Deep copy

- To achieve a true copy with a separate memory location, you need to do a DEEP COPY, where new memory is created and each value of the array is copied to the new memory.
- Below is an example of a DEEP COPY.

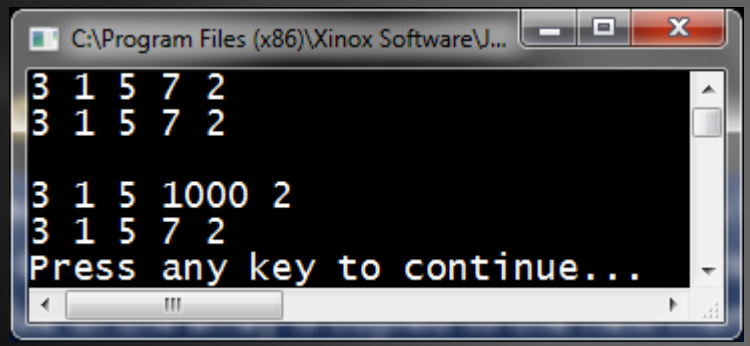
```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    for(int x=0;x<nums.length;x++)
        nums2[x]=nums[x];
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
}
```



Deep copy

- As you can see, the same two arrays exist, ***nums*** and ***nums2***.
- The difference is how ***nums2*** is created and given values.
- Instead of just assigning to it the array ***nums***, a new array of the same length as ***nums*** is created, and a loop is set up to copy each element, one by one, from one array to the other.

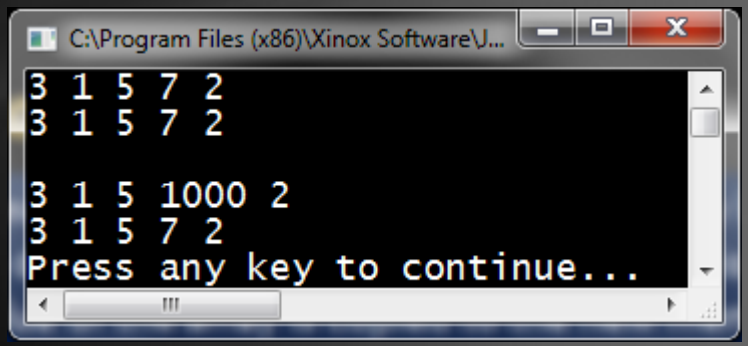
```
    throws IOException
{

    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    for(int x=0;x<nums.length;x++)
        nums2[x]=nums[x];
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
}
```



Deep copy

- This results in a true, or DEEP COPY, and is evidenced by the fact that NOW, the change made in the first array does NOT affect the second array, since they are two truly separate arrays in separate memory locations.

```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    for(int x=0;x<nums.length;x++)
        nums2[x]=nums[x];
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
}
```



Shallow copy vs Deep copy

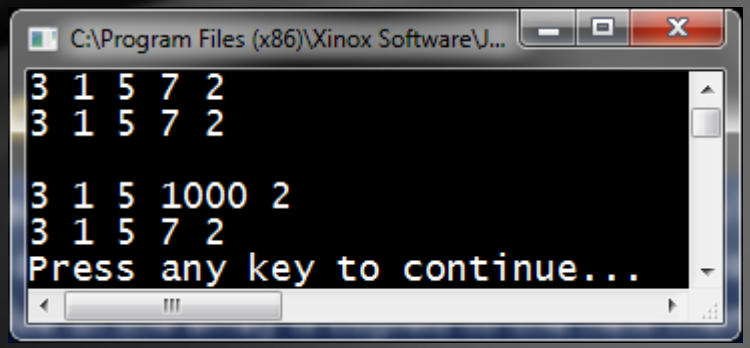
To summarize:

- A **shallow copy** is when two variables are referencing, or "pointing to", the same array.
- A **deep copy** is when completely new memory is created and duplicate values from the original array are placed into the new memory.

System.arraycopy – example #1

- There is a method from the System class that achieves the same thing as the copy loop does…copies one element at a time from one array to another.
- It is called the ***System.arraycopy*** method, as shown below.

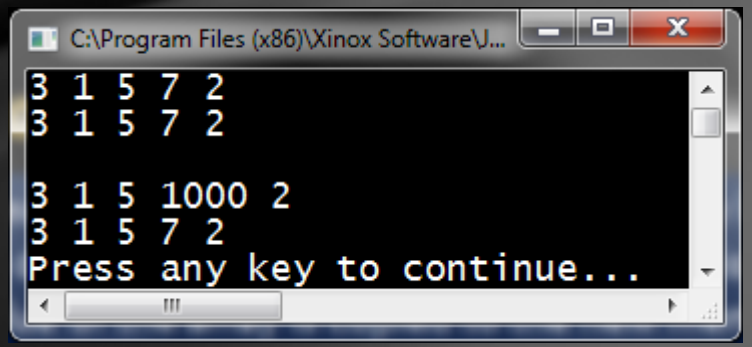
```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    System.arraycopy(nums,0,nums2,0,nums.length);

    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();

}
```



System.arraycopy – example #1

- This method requires five parameters…
  1. the source array
  2. The starting location in the source array
  3. The destination array
  4. The starting location in the destination array
  5. The number of elements to copy

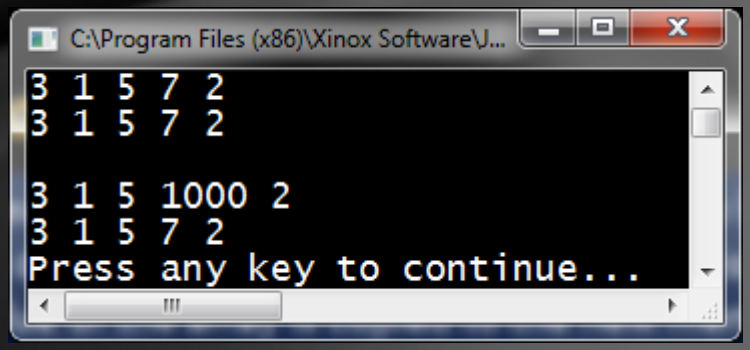
```
int [] nums2 = new int[nums.length];
//copy each element from one to the other
System.arraycopy(nums,0,nums2,0,nums.length);

for(int x:nums)out.print(x+" ");
out.println();
for(int x:nums2)out.print(x+" ");
out.println();
nums[3] = 1000;
out.println();
for(int x:nums)out.print(x+" ");
out.println();
for(int x:nums2)out.print(x+" ");
out.println();

}
```



System.arraycopy – example #1

- This very flexible method can be used to copy the entire array, or just parts of it, from any location in the source array to any location in the destination array.
- Several examples are shown on the next few slides.

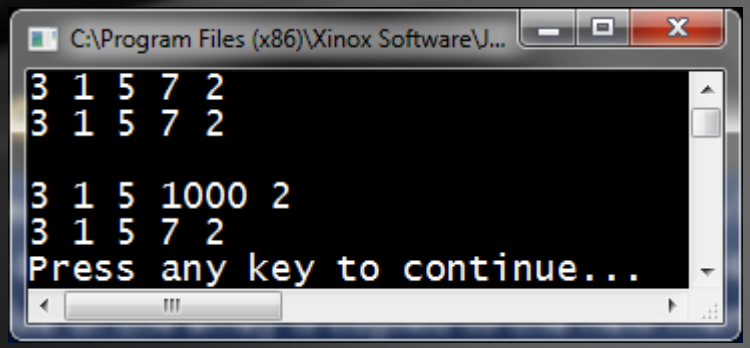
```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    System.arraycopy(nums,0,nums2,0,nums.length);

    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();

}
```



System.arraycopy – example #1

- The program below copies the **entire array**, with ***nums*** as the source array, zero as the starting position in ***nums***, ***nums2*** as the destination array, zero as the starting position of ***nums2***, and ***nums.length*** as the number of elements to copy.

```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    System.arraycopy(nums,0,nums2,0,nums.length);

    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();

}
```

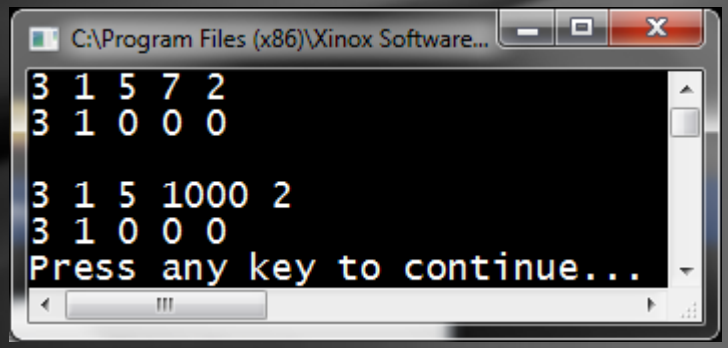


System.arraycopy – example #2

- In this adjusted example, only half of the array is copied.
- Notice the **last parameter is changed**, indicating only half the length of the array is to be copied.

Arrays7C.java

```
    public static void main (String [] args)
        throws IOException
    {
        int [] nums = {3,1,5,7,2};
        //deep copy process...create new memory
        int [] nums2 = new int[nums.length];
        //copy each element from one to the other
        System.arraycopy(nums,0,nums2,0,nums.length/2);

        for(int x:nums)out.print(x+" ");
        out.println();
        for(int x:nums2)out.print(x+" ");
        out.println();
        nums[3] = 1000;
        out.println();
        for(int x:nums)out.print(x+" ");
        out.println();
        for(int x:nums2)out.print(x+" ");
        out.println();
    }
}
```

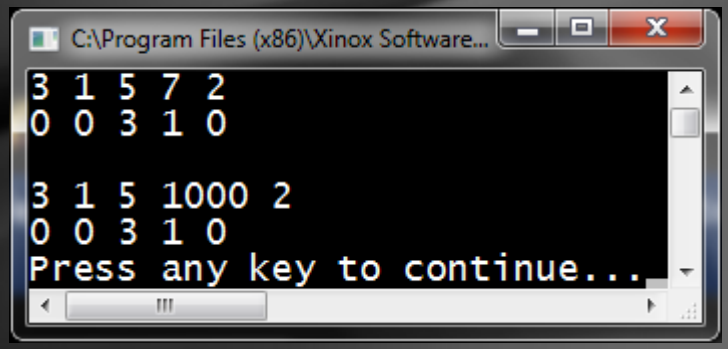


System.arraycopy – example #3

- In another example, the **starting location of the destination array is changed**...the fourth parameter.

Arrays7C.java

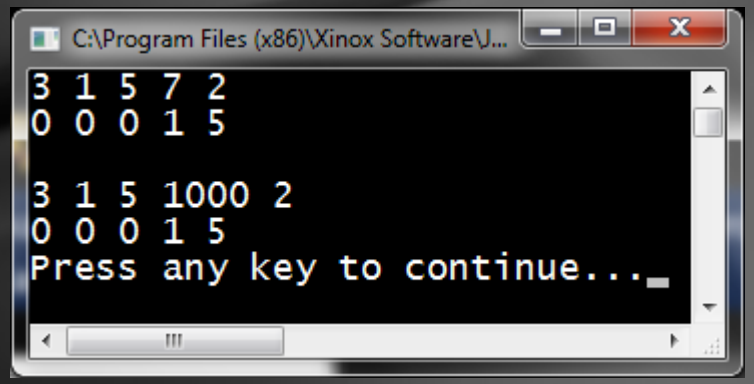
```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    System.arraycopy(nums,0,nums2,2,nums.length/2);

    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
}
```



System.arraycopy – example #4

- In yet another example, the **starting location of the source and destination arrays have been changed**…the second and fourth parameters.

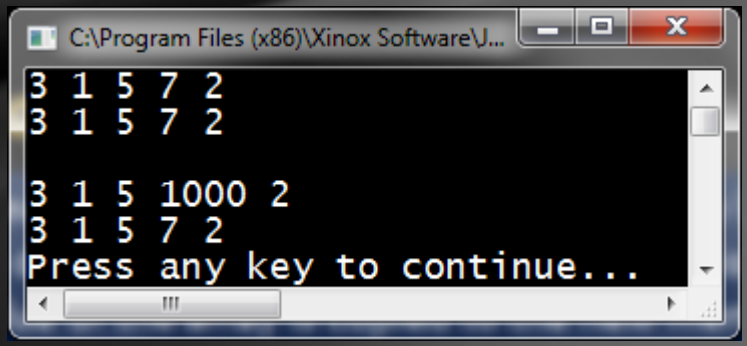
```
public static void main (String [] args)
    throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process...create new memory
    int [] nums2 = new int[nums.length];
    //copy each element from one to the other
    System.arraycopy(nums,1,nums2,3,nums.length/2);

    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
}
```



The "clone" method

- The "clone" method, which automatically "belongs" to any object, achieves the two-step DEEP COPY process ***in one command.*** See the sample program below.
- The result, as you can see, is indeed a DEEP COPY…***both*** ***new memory*** and the ***array copy process*** are achieved with one simple command.

```
throws IOException
{
    int [] nums = {3,1,5,7,2};
    //deep copy process using the clone method
    int [] nums2 = nums.clone();

    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
    nums[3] = 1000;
    out.println();
    for(int x:nums)out.print(x+" ");
    out.println();
    for(int x:nums2)out.print(x+" ");
    out.println();
}
```



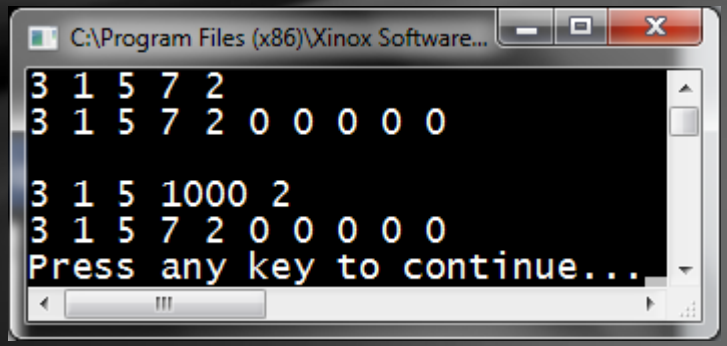
Arrays.copyOf

- Another Arrays class method that is very useful is the ***copyOf*** command. This achieves the same result as clone, but allows for growth or reduction in size of the array.
- The source array **is not changed**, but the resulting deep copy array can have more elements, or fewer elements, according to the need of the program.
- If more elements are required, the newly created elements are filled with the default values of the data type – ***zero*** for numeric arrays, ***false*** for boolean, **spaces** for character, and **null** for objects.

Arrays.copyOf – array of ints

- Below is an example of how this works.
- You can see that the second array is now 10 elements in length, with the first five elements a duplicate of the first array.
- Again, the second set of outputs verifies that the process is indeed a DEEP COPY.

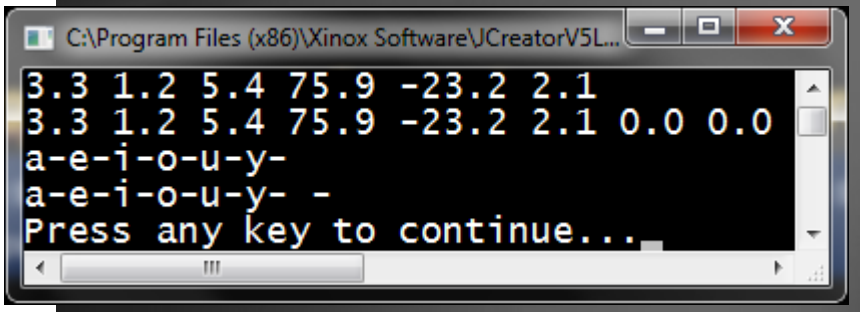
```
    public static void main (String [] args)
        throws IOException
    {
        int [] nums = {3,1,5,7,2};
        //deep copy process using copyOf
        int [] nums2 = Arrays.copyOf(nums,10);
        for(int x:nums)out.print(x+" ");
        out.println();
        for(int x:nums2)out.print(x+" ");
        out.println();
        nums[3] = 1000;
        out.println();
        for(int x:nums)out.print(x+" ");
        out.println();
        for(int x:nums2)out.print(x+" ");
        out.println();
    }
}
```



Arrays.copyOf – double and char arrays

- The same process works for other type arrays, as shown below.
- Below are a double array and a char array, both expanded in size.

```
{
    public static void main (String [] args)
        throws IOException
    {
        double [] nums = {3.3,1.2,5.4,75.9,-23.2, 2.1};
        double [] nums2 = Arrays.copyOf(nums,8);
        for(double x:nums)out.print(x+" ");
        out.println();
        for(double x:nums2)out.print(x+" ");
        out.println();
        char [] chars = {'a','e','i','o','u','y'};
        char [] chars2 = Arrays.copyOf(chars,7);
        for(char x:chars)out.print(x+"-");
        out.println();
        for(char x:chars2)out.print(x+"-");
        out.println();
    }
}
```

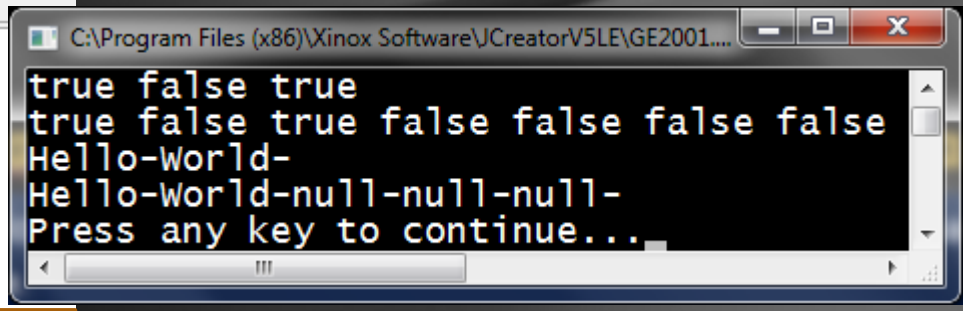


Arrays.copyOf – boolean and String arrays

- Below are a boolean array and a String array, both expanded in size.
- As was mentioned in an earlier lesson, the default values for boolean and Strings are ***false*** and ***null***.

7C.java × Arrays.html

```
    throws IOException
{
    boolean [] flags = {true,false,true};
    boolean [] flags2 = Arrays.copyOf(flags,7);
    for(boolean x:flags)out.print(x+" ");
    out.println();
    for(boolean x:flags2)out.print(x+" ");
    out.println();
    String [] Strings = {"Hello","World"};
    String [] Strings2 = Arrays.copyOf(Strings,5);
    for(String x:Strings)out.print(x+"-");
    out.println();
    for(String x:Strings2)out.print(x+"-");
    out.println();
}
```



Arrays.copyOf – truncating arrays

- Sometimes it is useful to decrease an array in size, called "truncating".
- When you use ***copyOf*** with the length parameter value **less than** the size of the source array, the new array is **shorter** in size than the source array.
- Any elements at the end, beyond the indicated length, are "truncated", or cut off, ***lost forever in the deep abyss of RAM memory!***
- The next slide shows several examples of truncated arrays.

Arrays.copyOf – truncating arrays

- Here you see examples of truncated boolean and String arrays.

java | Arrays.html

```
{
    boolean [] flags = {true,false,true};
    boolean [] flags2 = Arrays.copyOf(flags,2);
    for(boolean x:flags)out.print(x+" ");
    out.println();
    for(boolean x:flags2)out.print(x+" ");
    out.println();
    String [] Strings = {"Hello","World","of","Warcraft"};
    String [] Strings2 = Arrays.copyOf(Strings,3);
    for(String x:Strings)out.print(x+" ");
    out.println();
    for(String x:Strings2)out.print(x+" ");
    out.println();
```

```
true false true
true false
Hello World of Warcraft
Hello World of
```

Arrays.copyOf – truncating arrays

- Here you see examples of truncated double, char, and int arrays.

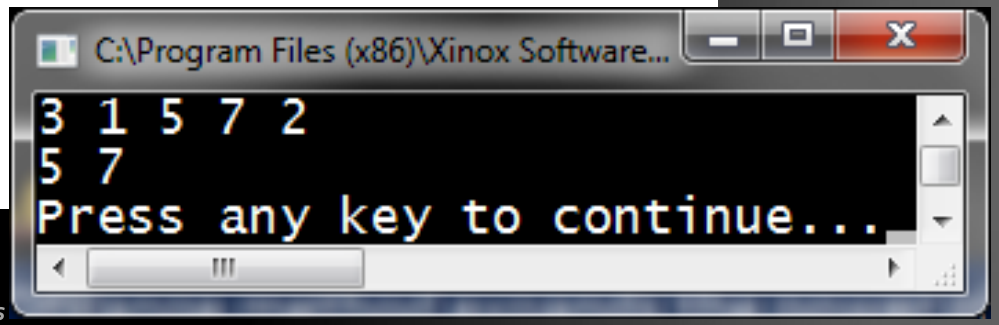
```
double [] nums = {3.3,1.2,5.4,75.9,-23.2, 2.1};
double [] nums2 = Arrays.copyOf(nums,4);
for(double x:nums)out.print(x+" ");
out.println();
for(double x:nums2)out.print(x+" ");
out.println();
char [] chars = {'a','e','i','o','u','y'};
char [] chars2 = Arrays.copyOf(chars,4);
for(char x:chars)out.print(x+"-");
out.println();
for(char x:chars2)out.print(x+"-");
out.println();
int [] inums = {3,1,5,7,2};
int [] inums2 = Arrays.copyOf(inums,2);
for(int x:inums)out.print(x+" ");
out.println();
for(int x:inums2)out.print(x+" ");
out.println ();
```

```
3.3 1.2 5.4 75.9 -23.2 2.1
3.3 1.2 5.4 75.9
a-e-i-o-u-y-
a-e-i-o-
3 1 5 7 2
3 1
```

Arrays.copyOfRange

- The ***copyOfRange*** method expands the power of ***copyOf*** by designating a starting and ending location, in the same exact was as does the ***String.substring*** method.
- Below is an example. The first parameter is the starting location in the source array, and the second is **one step beyond** the ending location.

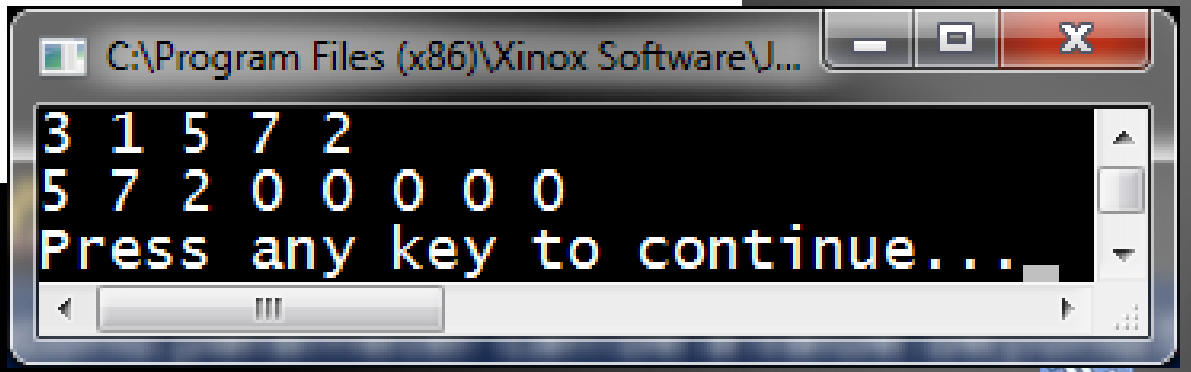
```
{
int [] inums = {3,1,5,7,2};
int [] inums2 = Arrays.copyOfRange(inums,2,4);
for(int x:inums)out.print(x+" ");
out.println();
for(int x:inums2)out.print(x+" ");
out.println ();

}
```



Arrays.copyOfRange

- The second parameter can be a value any distance beyond the length of the source array, as shown below.
- The same process works with all of the other data types as well.

```
{
    public static void main (String [] args)
        throws IOException
        {
        int [] inums = {3,1,5,7,2};
        int [] inums2 = Arrays.copyOfRange(inums,2,10);
        for(int x:inums)out.print(x+" ");
        out.println();
        for(int x:inums2)out.print(x+" ");
        out.println ();

        }
}
```



Arrays.copyOfRange

- Here are examples of ***copyOfRange*** with the other data types.

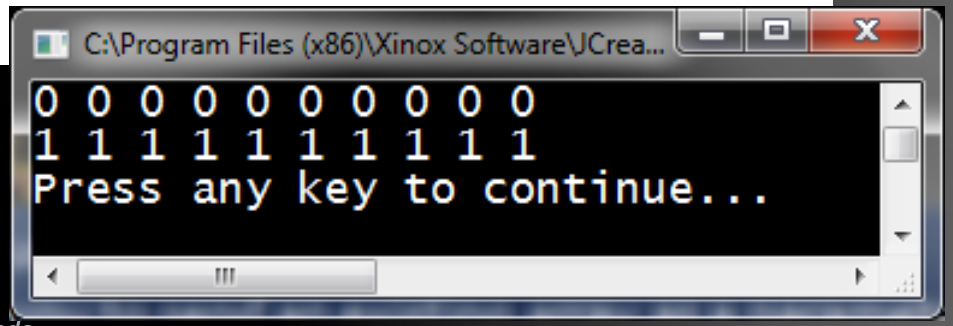
```
boolean [] flags = {true,false,true};
boolean [] flags2 = Arrays.copyOfRange(flags,2,9);
for(boolean x:flags)out.print(x+" ");
out.println();
for(boolean x:flags2)out.print(x+" ");
out.println();
String [] Strings = {"Hello","World","of","Warcraft"};
String [] Strings2 = Arrays.copyOfRange(Strings,3,8);
for(String x:Strings)out.print(x+" ");
out.println();
for(String x:Strings2)out.print(x+" ");
out.println();
double [] nums = {3.3,1.2,5.4,75.9,-23.2, 2.1};
double [] nums2 = Arrays.copyOfRange(nums,4,10);
for(double x:nums)out.print(x+" ");
out.println();
for(double x:nums2)out.print(x+" ");
out.println();
char [] chars = {'a','e','i','o','u','y'};
char [] chars2 = Arrays.copyOfRange(chars,1,5);
for(char x:chars)out.print(x+"-");
out.println();
for(char x:chars2)out.print(x+"-");
out.println();
```

```
true false true
true false false false false false false
Hello World of Warcraft
Warcraft null null null null
3.3 1.2 5.4 75.9 -23.2 2.1
-23.2 2.1 0.0 0.0 0.0 0.0
a-e-i-o-u-y-
e-i-o-u-
```

Arrays.fill

- Another useful method of the ***Arrays*** class is ***fill***.
- It allows you to send an existing array as a parameter, a value to fill it with, and the method loads that value into every location in the **existing array**, or into a desired range inside the array.
- Below is an example of an integer array using the ***fill*** method.

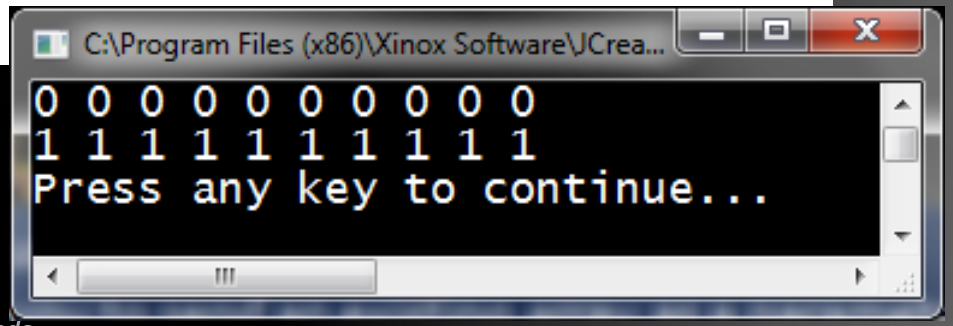
```
public static void main (String [] args)
    throws IOException
    {
        int [] nums = new int[10];
        for(int x:nums) out.print(x+" "); out.println();
        Arrays.fill(nums,1);
        for(int x:nums) out.print(x+" "); out.println();
    }
```



Arrays.fill

- The newly created array is automatically filled with the default value zero, but the fill command can fill every element with any other desired value.

```
public static void main (String [] args)
    throws IOException
    {
        int [] nums = new int[10];
        for(int x:nums) out.print(x+" "); out.println();
        Arrays.fill(nums,1);
        for(int x:nums) out.print(x+" "); out.println();
    }
```



Arrays.fill

- The fill process works with the other data types as well.

```
double [] nums2 = new double[10];
for(double x:nums2) out.print(x+" "); out.println();
Arrays.fill(nums2,3.4);
for(double x:nums2) out.print(x+" "); out.println();out.println()

char [] chars2 = new char[10];
for(char x:chars2) out.print(x+"-"); out.println();
Arrays.fill(chars2,'*');
for(char x:chars2) out.print(x+"-"); out.println();out.println();

String [] Strings2 = new String[10];
for(String x:Strings2) out.print(x+"-"); out.println();
Arrays.fill(Strings2,"yo");
for(String x:Strings2) out.print(x+"-"); out.println();out.printl

boolean [] flags2 = new boolean[10];
for(boolean x:flags2) out.print(x+"-"); out.println();
Arrays.fill(flags2,true);
for(boolean x:flags2) out.print(x+"-"); out.println();out.println
```

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3.4 3.4 3.4 3.4 3.4 3.4 3.4 3.4 3.4 3.4

 - - - - - - - - - -
*-*-*-*-*-*-*-*-*-*-

null-null-null-null-null-null-null-null-null-null-
yo-yo-yo-yo-yo-yo-yo-yo-yo-yo-

false-false-false-false-false-false-false-false-false-false-
true-true-true-true-true-true-true-true-true-true-
```

Arrays.fill

- To fill only a certain range of elements, two integer parameters (start, stop) can be placed between the array and fill value.
- Again, as in the copyOfRange method, this process functions exactly as the String.substring method does.
- Study these examples carefully.

```
int [] nums = new int[10];
for(int x:nums) out.print(x+" "); out.println();
Arrays.fill(nums, 3,6,1);
for(int x:nums) out.print(x+" "); out.println();out.println();

double [] nums2 = new double[10];
for(double x:nums2) out.print(x+" "); out.println();
Arrays.fill(nums2,2,7,3.4);
for(double x:nums2) out.print(x+" "); out.println();out.println()
```

```
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0

0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 3.4 3.4 3.4 3.4 3.4 0.0 0.0 0.0
```

Arrays.fill

- More examples…

```
char [] chars2 = new char[10];
for(char x:chars2) out.print(x+" "); out.println();
Arrays.fill(chars2,1,5,'*');
for(char x:chars2) out.print(x+" "); out.println();out.println();

String [] Strings2 = new String[10];
for(String x:Strings2) out.print(x+"-"); out.println();
Arrays.fill(Strings2,5,10,"yo");
for(String x:Strings2) out.print(x+"-"); out.println();out.printl

boolean [] flags2 = new boolean[10];
for(boolean x:flags2) out.print(x+"-"); out.println();
Arrays.fill(flags2,4,5,true);
for(boolean x:flags2) out.print(x+"-"); out.println();out.printlr
```

```
 - - - - - - - - - -
 -*-*-*-*- - - - - -

null-null-null-null-null-null-null-null-null-null-
null-null-null-null-null-yo-yo-yo-yo-yo-

false-false-false-false-false-false-false-false-false-false-
false-false-false-false-true-false-false-false-false-false-
```

Lesson Summary

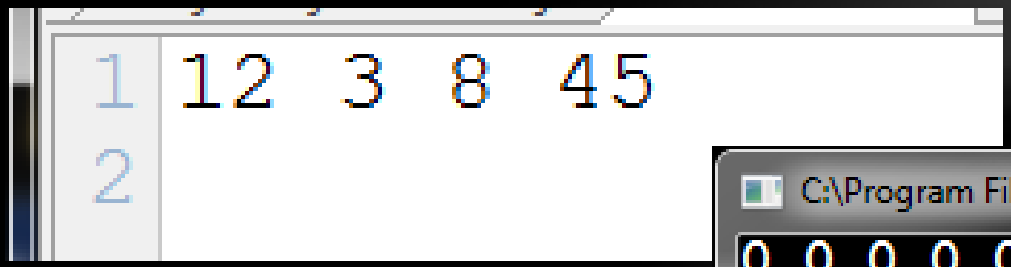
- In this lesson, you learned about some useful array methods to help with copying and filling arrays in various ways, including
  - System.arraycopy
  - Arrays.clone
  - Arrays.copyOf
  - Arrays.copyOfRange
  - Array.fill
- Now it is time to practice with a couple of labs.

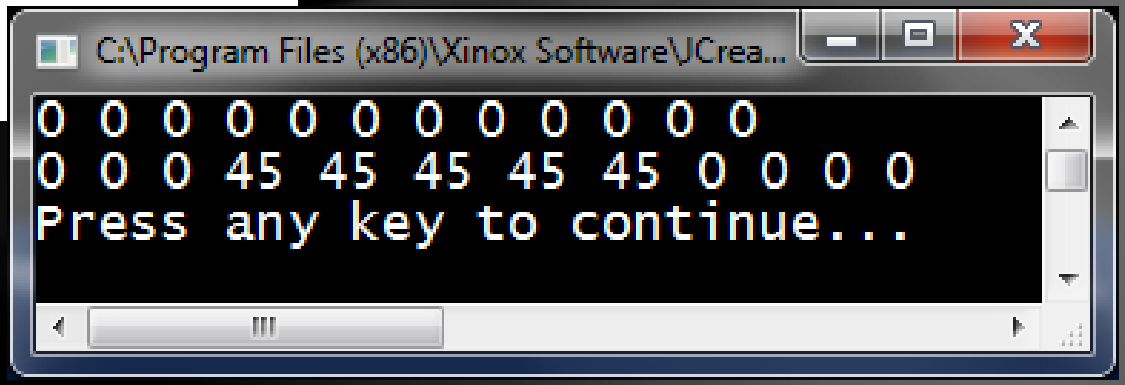
Labs

- MyArrayMethods will be the name of this set of labs. As you did before, create a separate folder and file called MyArrayMethods and do your work there.

Lab7B1

- WAP to input from a data file ("lab7b1.in") four integer values, N1, N2, N3, and N4.
- Create an integer array of size N1, fill it with N4 values from location N2 up to but not including location N3.
- Output the array before and after the fill, as shown below.





Lab7B2

- WAP to read from a data file ("lab7B2.in") two lines of double values into two separate double arrays, output both separately, then combine both into one larger array, output the combined array in ascending sorted order.

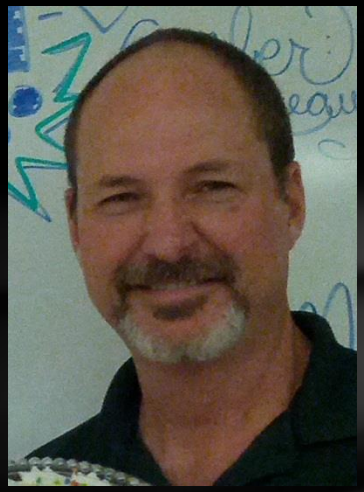
```
1 1.2 4.3 5.6 8.3
2 33.4 -2.6 0.1 99.2 82.4
3
```

```
Array 1:
1.2 4.3 5.6 8.3
Array 2:
33.4 -2.6 0.1 99.2 82.4
Arrays combined:
1.2 4.3 5.6 8.3 33.4 -2.6 0.1 99.2 82.4
Arrays combined and sorted:
-2.6 0.1 1.2 4.3 5.6 8.3 33.4 82.4 99.2
Press any key to continue...
```

CONGRATULATIONS!

- You now know how to use several useful array methods in your work.
- *Lesson 8 will discuss the topic of passing parameters.*
- *Lesson 9 will explore the world of matrices.*

Thank you, and have fun!



To order supplementary materials for all the lessons in this package, such as lab solutions, quizzes, tests, and unit reviews, visit the O(N)CS Lessons website, or contact me at

John B. Owen
captainjbo@gmail.com