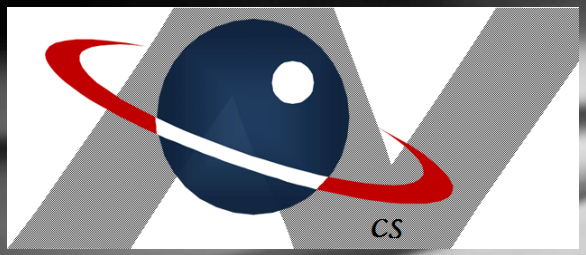
O(N) CS LESSONS

Lesson 8 – Passing Parameters



By John B. Owen

Table of Contents

Objectives

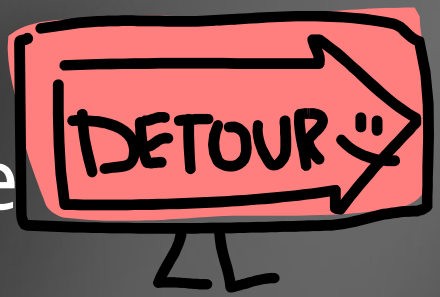
- It is now important to understand the distinctions between different types of parameters (actual, formal, value, and reference) when defining and using methods that pass parameters.
- This lesson will explore these concepts in significant detail…read and study it carefully!

Parameter – a definition

- A parameter is simply a ***variable in the parentheses of a method header or method call.***
- It has a data type and an identifier, just like a variable.
- It is used to pass data in a program from one location to another through the ***parameter list*** of a method.

Object vs primitive

- First we need to review the key difference between an object variable and a primitive variable; object parameters and primitive parameters differ in the same way.
- The next four slides were introduced in lesson 2A and are listed here as a review.



Review

Variables are memory locations

Variables are *locations in the computer's RAM*, or memory *that can be changed*.

When you declare a variable, like

```
int age = 55;
```

the JAVA compiler finds some available memory, carves out enough room for an int value, marks that memory location with the ***identifier*** (variable name) `age`, and places the value `55` into that memory location.

Primitives vs object references

Review

```
int age = 55;
```

- The main reason this is called a ***primitive*** is that the memory location actually contains the value.
- Objects are stored in a different way than primitives…see next slide.

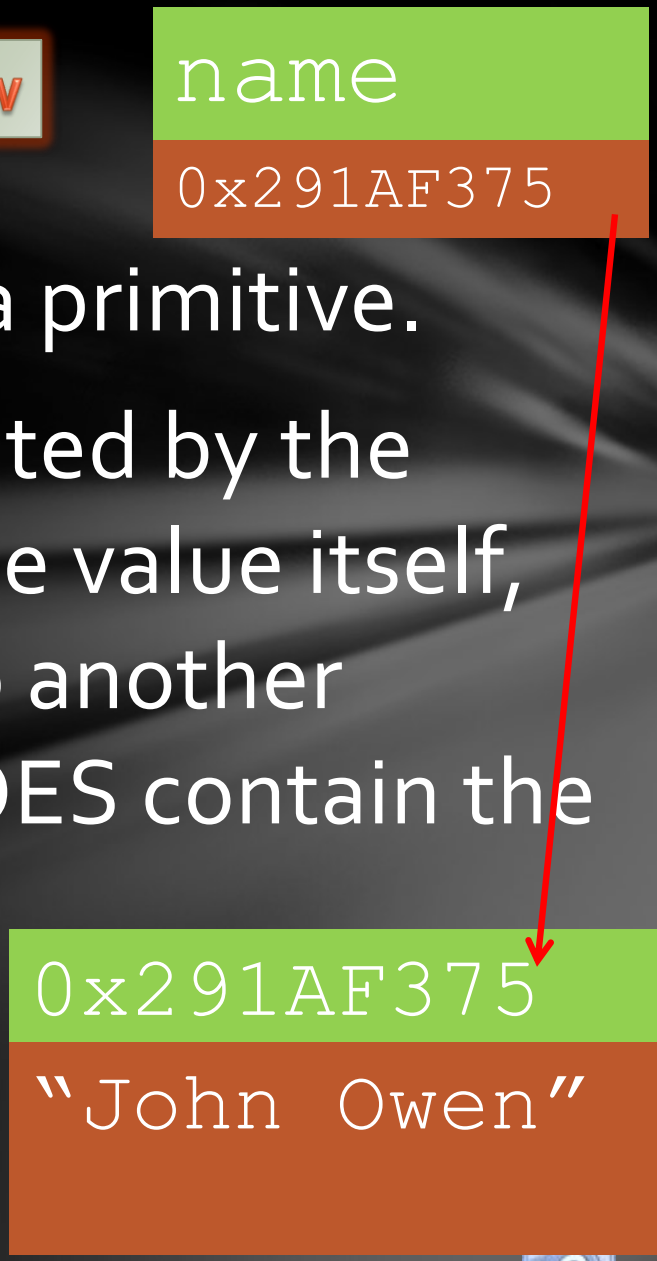
Object reference

Review

An object is not stored like a primitive.

The memory location indicated by the variable does not contain the value itself, but instead is a ***reference*** to another memory location, which DOES contain the value.

```
String name =
"John Owen";
```



Memory Locations…

Review

Other primitive types work the same way as an int, in that the memory location actually stores the data value.

Each type of data requires different amounts of memory to store…more on that later in this lesson.

- `double wage = 54.65;`
- `char initial = 'B';`
- `boolean sailor = true;`

Summary - Object references vs primitives

- The reason to review these concepts is to better understand parameters and how they work.
- To summarize, both object references and primitives are variables, but contain different types of information
  - Primitives contain actual data
  - Object references contain addresses of data somewhere else in memory.
- Now, on with the lesson…

Parameter passing - examples

- Parameter passing has been demonstrated numerous times in the lessons and examples so far.
- For example, the Math class methods all require parameters, sometimes even more than one:
  - ***Math.abs(int a)***
  - ***Math.pow (double a, double b)***
  - ***Math.max (int a, int b)***

Parameter passing - examples

- The String class methods also have required parameter passing:
  - ***String.charAt(int index)***
  - ***String.substring(int beginIndex, int endIndex)***
- The methods you designed in Lesson 5C also had parameters:
- ***MyClass.firstHalf(String word)***
- ***MyClass.firstLastValue(String str)***

Parameter passing - examples

- In one of the custom array methods from Lesson 6C, the parameter passed was an array:
  - ***public static double avgArr (int [] ints)***

Parameter passing

- In order to pass data from one part of a program to another, there must be a sender parameter (actual parameter) and a receiver parameter (formal parameter), a situation similar to that in a ball game…someone to **pass the ball**, and someone to **receive the passed ball**.

Formal parameters

- All of the parameter examples we have just mentioned are called ***formal parameters*** because they are ***located in the header of the method definition***.
- ***Formal parameters are the ones that receive data, much like a ball player catching a passed ball.***

Actual parameters

- Actual parameters are the ones ***found in the method calls***:
  - `System.out.println(Math.abs(-3.4));`
  - `System.out.println("hello".charAt(3));`
  - `int x = 14;`
  - `int y = Math.pow(x,2);`
  - `out.printf("Average = %.1f\n",avgArr(ints));`
- ***Actual parameters are the ones that send the data, much like a ball player throwing a ball.***

Parameter passing is ONE WAY (?)

- The actual transfer of data **from the actual parameter to the formal parameter** is how data gets from one location in a program to another.
- It is a ONE-WAY action…well, sort of.

Passing primitives by value

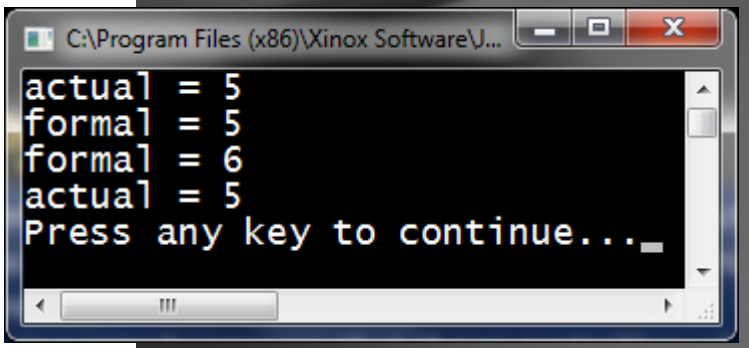
- In Java, ALL primitive parameters are ***passed by value***, which means:
  - *A COPY of the value contained in the **actual parameter** (the sender) is sent to the **formal parameter** (the receiver),* much like a fax machine sends a copy of a document to a remote fax machine.
  - This is indeed a 

transmission.

Passing primitives by value

- Let's clarify all of this discussion by showing exactly what it means to ***pass by value***, and the reason it is so important to understand it.
- Below is a sample program that shows a primitive value being passed to a method.
- Study it carefully and see if you can see the crucial point.

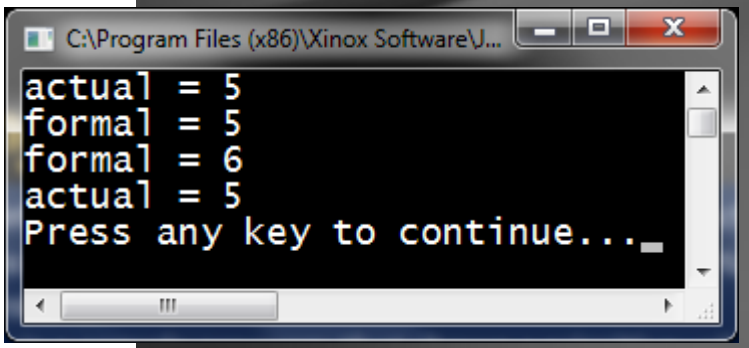
```
public static void doStuff(int x)
{
    out.println("formal = "+x);
    x = 6;
    out.println("formal = "+x);
}
/**main method
 */
public static void main (String [] args)
{
    int y = 5;
    out.println("actual = "+y);
    doStuff(y);
    out.println("actual = "+y);
}
```



Passing primitives by value

- Did you spot it!?
- Look carefully at the sequence.
- First, a variable y is created in main and assigned the value 5.
- It is then output in main as 5.
- In the call to method ***doStuff***, it serves as an actual parameter sent through the parameter list.

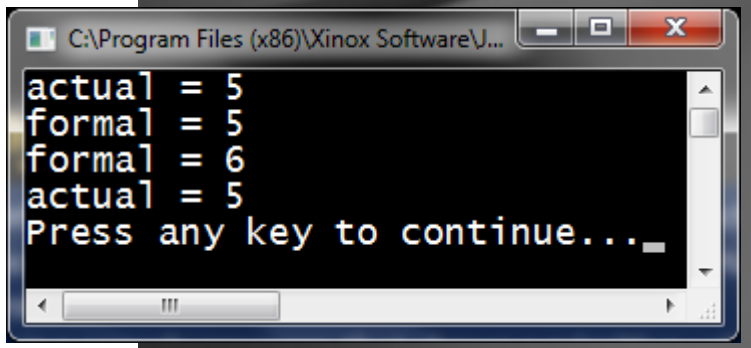
```
public static void doStuff(int x)
{
     out.println("formal = "+x);
     x = 6;
     out.println("formal = "+x);
}
/**main method
 */
public static void main (String [] args)
{
     int y = 5;
     out.println("actual = "+y);
     doStuff(y);
     out.println("actual = "+y);
}
```



Passing primitives by value

- The formal parameter x of the ***doStuff*** method receives the value 5, and then is output as the value 5.
- It is then reassigned the value 6, and output as the value 6.
- The method ***doStuff*** completes its work, and control is passed back to the main method.
- Finally y is output again as the value 5.

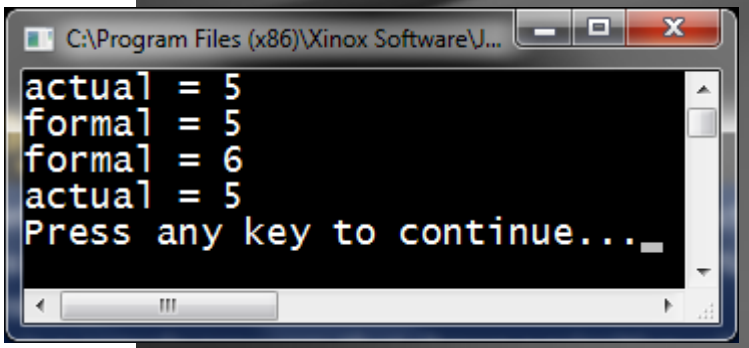
```
public static void doStuff(int x)
{
    out.println("formal = "+x);
    x = 6;
    out.println("formal = "+x);
}
/**main method
 */
public static void main (String [] args)
{
    int y = 5;
    out.println("actual = "+y);
    doStuff(y);
    out.println("actual = "+y);
}
```



Passing primitives by value

- Here is the issue…when the value is changed in the formal parameter (x) of the ***doStuff*** method, the ***change in the formal parameter*** **does NOT affect** the ***value of the actual parameter*** in main (y), therefore, the last output shown below is still the original value of the variable y.
- This is called ***passing by value,*** a true ONE-WAY situation.

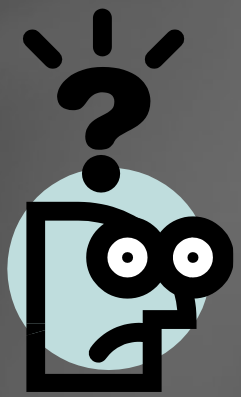
```
public static void doStuff(int x)
{
     out.println("formal = "+x);
     x = 6;
     out.println("formal = "+x);
}
/**main method
 */
public static void main (String [] args)
{
     int y = 5;
     out.println("actual = "+y);
     doStuff(y);
     out.println("actual = "+y);
}
```



Value vs reference

- Now it gets a bit sticky.
- PRIMITIVES are **passed by value**, as we just demonstrated.

- Objects in one sense are **passed by value**, but in reality are **passed by reference**, which makes it a TWO-WAY situation!



Passing objects by reference

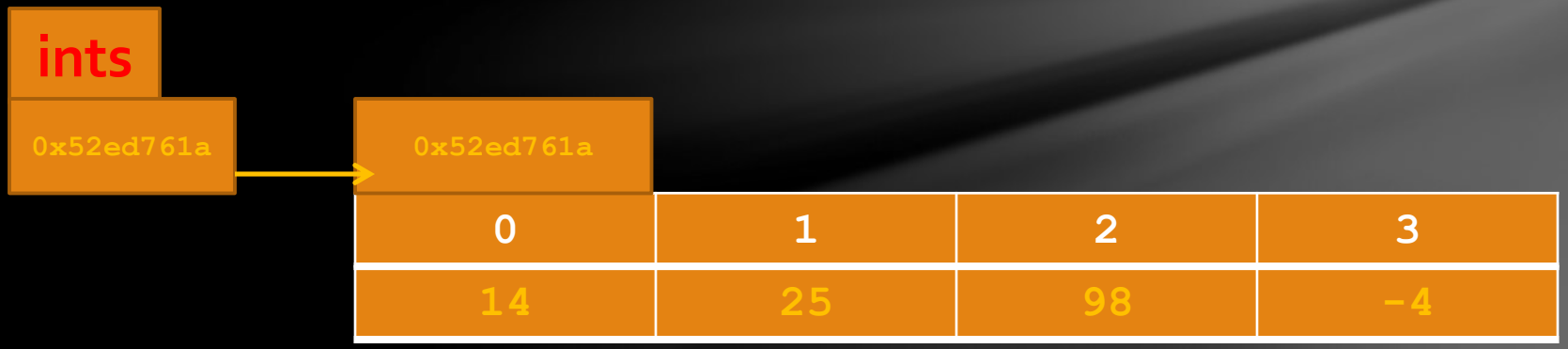
- When objects (like a String or an array) are involved in parameter passing, it is NOT the actual VALUE referenced by the object that is passed through the parameter list.
- Instead, a COPY of the **address** of the object's value, contained in the object reference, is ***passed by value*** through the parameter list (from actual to formal).

Passing objects by reference

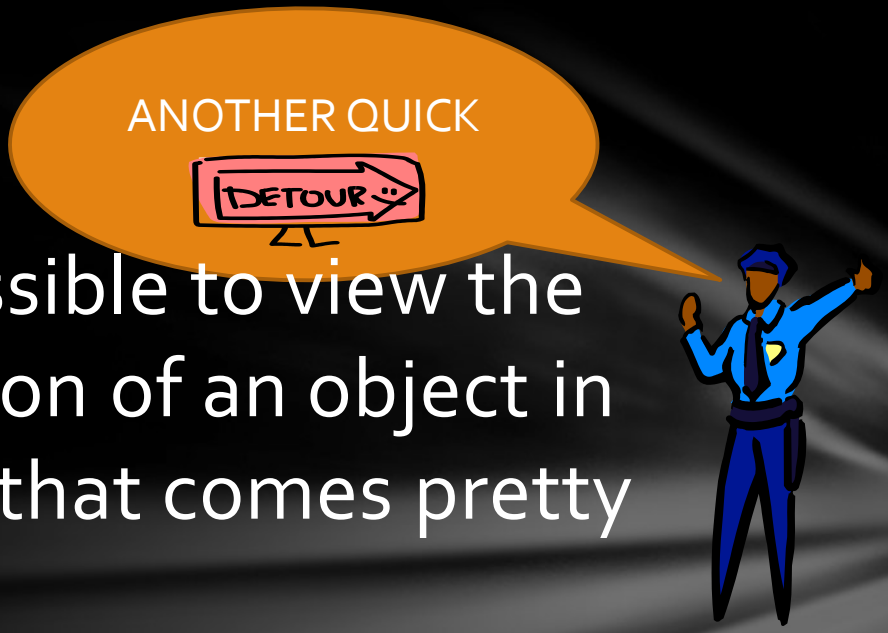
- Since it is the **address** that is **passed by value**, the formal parameter in reality is referencing the ***SAME MEMORY LOCATION AND SAME VALUE*** as the actual parameter is referencing, which is called passing by reference.
- Let's look at an example from the array lesson we discussed earlier.

Passing objects by reference

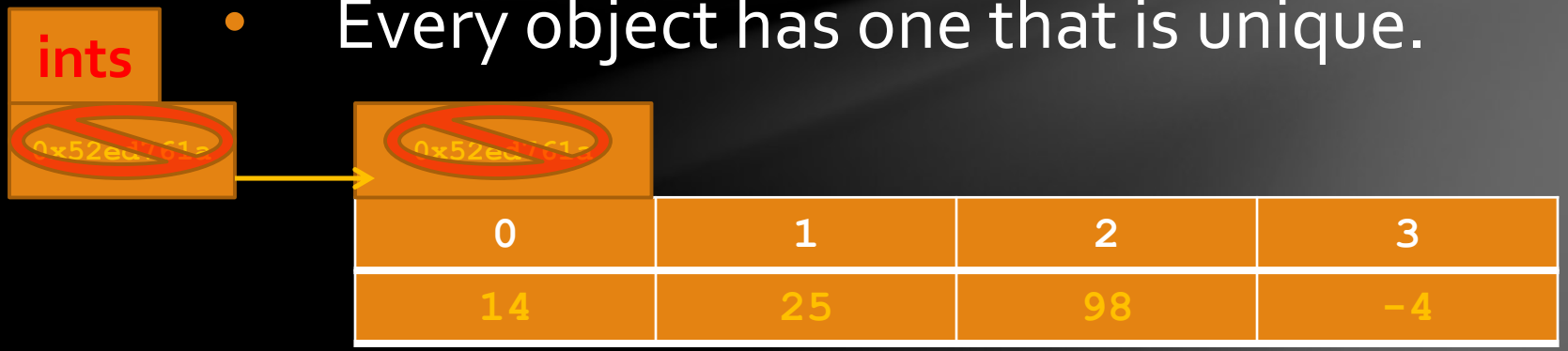
- Here we have an array of integers, referenced by the variable ***ints***, as shown below.



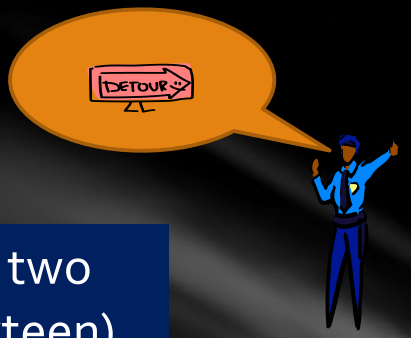
Object hashcode



- Although it is not possible to view the actual memory location of an object in Java, there is a value that comes pretty close.
- It is called the ***hashCode*** of an object.
  - Every object has one that is unique.



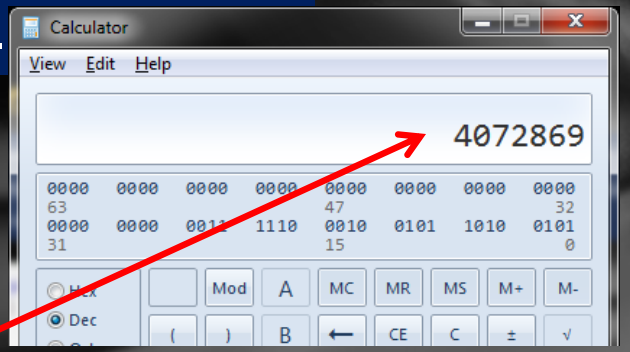
Object hashcode

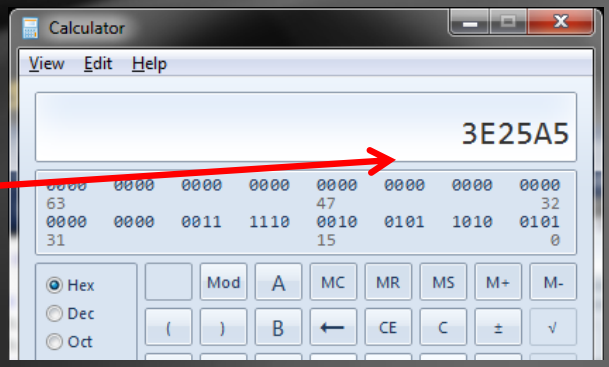


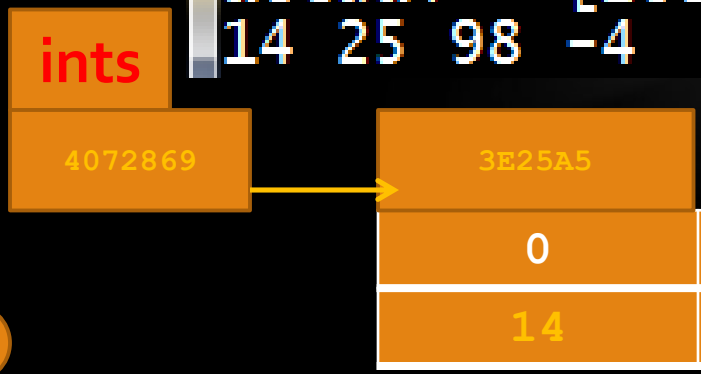
- Below is a program that shows the ***hashcode*** of this array in two forms…a base 10 value, and the hexadecimal value (base sixteen), both of which represent the same hashcode quantity.

```
*/
public static void main (String [] args)
{
    int [] ints = {14,25,98,-4};
    out.println("actual = "+ints.hashCode());
    out.println("actual = "+ints);
    for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
    out.println();
```

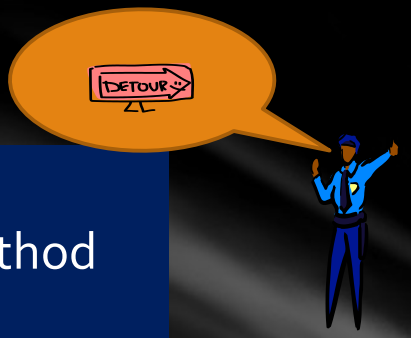
```
actual = 4072869
actual = [I@3e25a5
14 25 98 -4
```







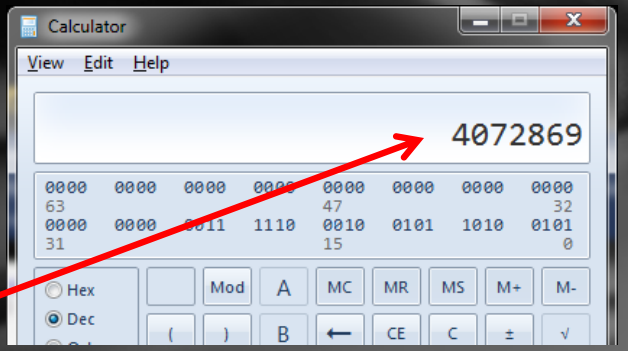
Object hashcode

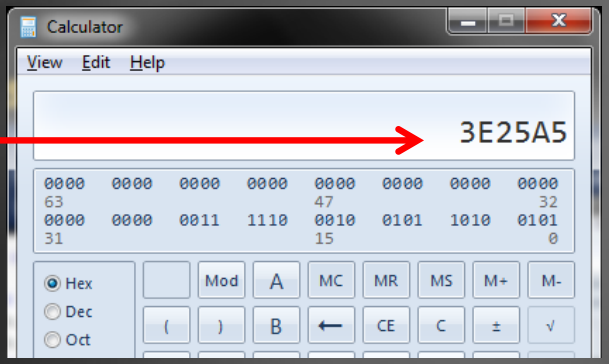


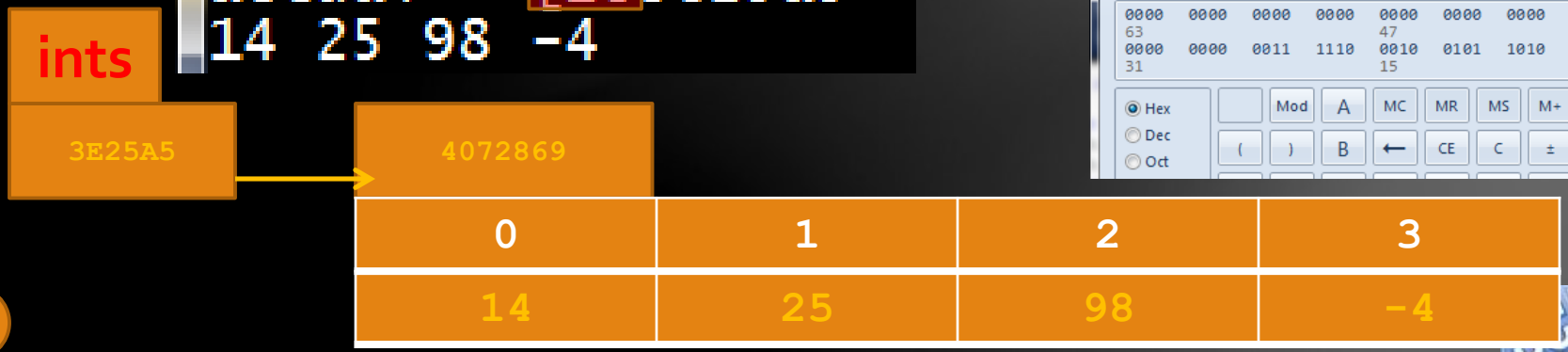
- The "`[I@`" portion of the second output means ***"array of integers at"***, an extra label produced by another special method that belongs to every object, the ***toString*** method, which is automatically called when you simply output the object.

```
public static void main (String [] args)
{
    int [] ints = {14,25,98,-4};
    out.println("actual = "+ints.hashCode());
    out.println("actual = "+ints);
    for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
    out.println();
```

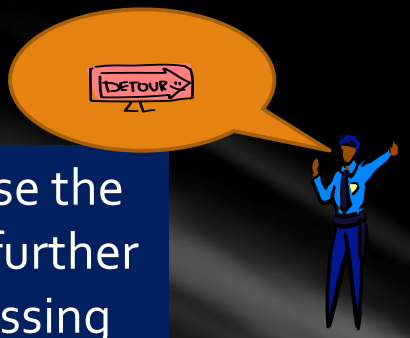
```
actual = 4072869
actual = [I@3e25a5
14 25 98 -4
```







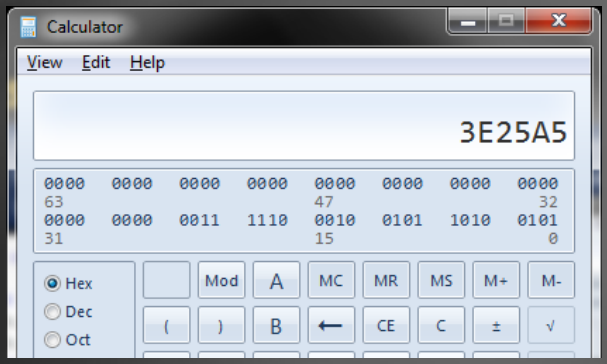
Object hashcode



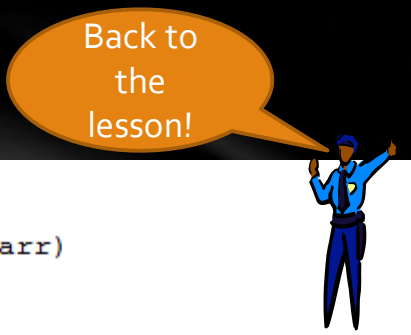
- Since both values represent the same hashcode, we'll just use the ***hexadecimal*** form to continue our lesson as we attempt to further clarify the memory location address aspect of parameter passing with objects.

```
*/
public static void main (String [] args)
{
    int [] ints = {14,25,98,-4};
    out.println("actual = "+ints.hashCode());
    out.println("actual = "+ints);
    for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
    out.println();
```

```
actual = 4072869
actual = [I@3e25a5
14 25 98 -4
```



Passing objects by reference



- Here is the entire program, similar to the one we used with the primitive parameter passing example.
- First the array is created and initialized with values.

```
public class Parameters
{
    public static void doStuff(int [] arr)
    {
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
        arr[1] = 6;
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
    }
    /**main method
     */
    public static void main (String [] args)
    {
        int [] ints = {14,25,98,-4};
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
        doStuff(ints);
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
    }
}
```

```
C:\Program Files (x86)\Xinox Software...
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 6 98 -4
actual = [I@3e25a5
14 6 98 -4
Press any key to continue...
```

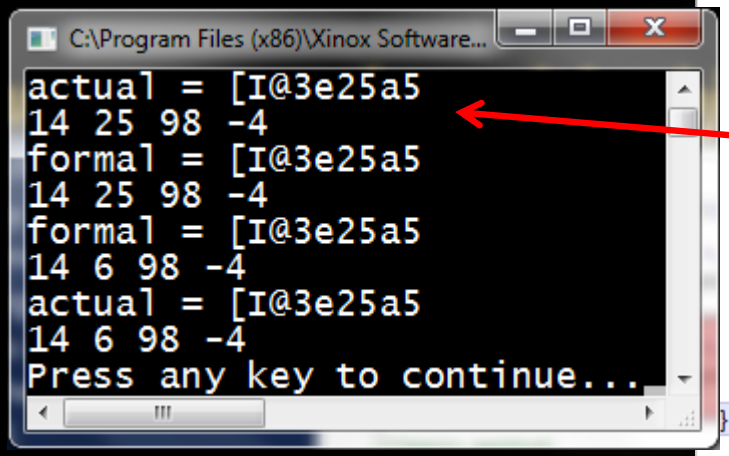
ints →

0	1	2	3
14	25	98	-4

Passing objects by reference

- The hashcode is output, and the contents of the array as well.
- The ***doStuff*** method is then called, sending ***ints*** as the actual parameter.

```
public class Parameters
{
    public static void doStuff(int [] arr)
    {
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
        arr[1] = 6;
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
    }
    /**main method
     */
    public static void main (String [] args)
    {
        int [] ints = {14,25,98,-4};
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
        doStuff(ints);
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
    }
}
```



0	1	2	3
14	25	98	-4

Passing objects by reference

- The ***doStuff*** method receives the array with the formal parameter ***arr***, and outputs the hashcode and contents, ***both of which have the same output as the actual parameter in main!!!***

```
public class Parameters
{
    public static void doStuff(int [] arr)
    {
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
        arr[1] = 6;
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
    }
    /**main method
     */
    public static void main (String [] args)
    {
        int [] ints = {14,25,98,-4};
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
        doStuff(ints);
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
    }
}
```

```
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 6 98 -4
actual = [I@3e25a5
14 6 98 -4
Press any key to continue...
```

ints

arr

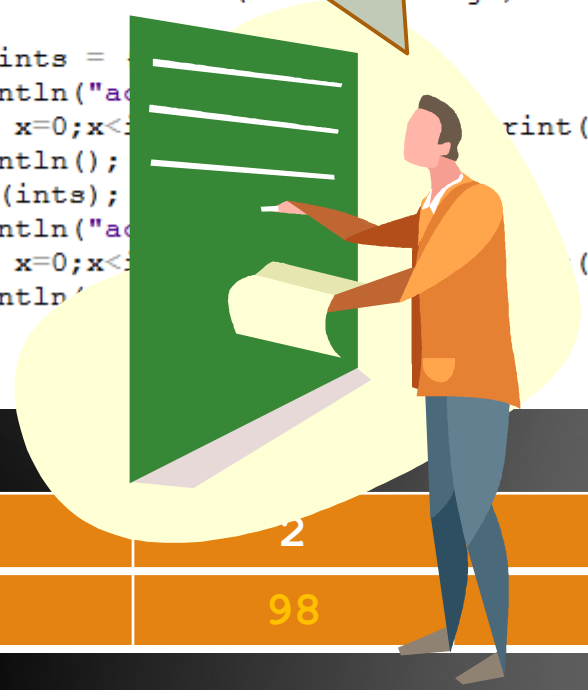
0	1	2	3
14	25	98	-4

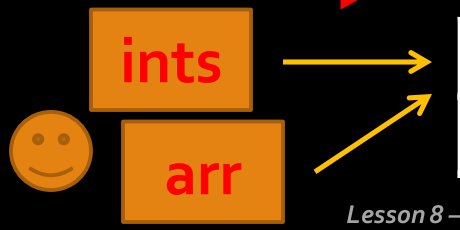
Passing objects by reference

- The ***doStuff*** method receives the array with the formal parameter ***arr***, and outputs the hashcode and contents, ***both of which have the same output as the actual parameter in main!!!***

```
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 6 98 -4
actual = [I@3e25a5
14 6 98 -4
Press any key to continue...
```

This means that both ints and arr are pointing to, or referencing, the same array!





0	1	2	3
14	25	98	-4

Passing objects by reference

- Then a change is made in the array arr, the formal parameter.
- The element in position 1 is reassigned the value 6, and all is output again.
- ***Same memory location, state of the object changed!!***

```
public class Parameters
{
    public static void doStuff(int [] arr)
    {
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
        arr[1] = 6;
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
    }
    /**main method
     */
    public static void main (String [] args)
    {
        int [] ints = {14,25,98,-4};
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
        doStuff(ints);
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
    }
}
```

```
C:\Program Files (x86)\Xinox Software...
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 6 98 -4
actual = [I@3e25a5
14 6 98 -4
Press any key to continue...
```

ints

arr

0	1	2	3
14	6	98	-4

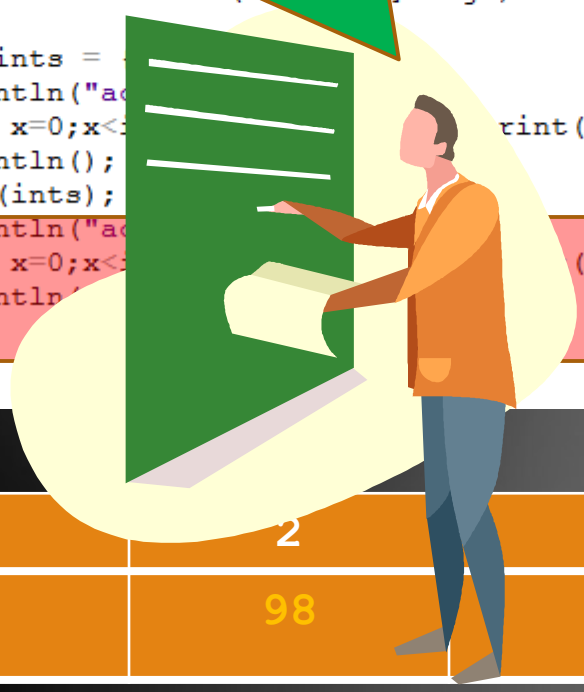
Passing objects by reference

- The ***doStuff*** method completes its job, and program control reverts back to main.
- The array is output once more, with the same hashcode, but ***with the altered contents!!!***

```
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 6 98 -4
actual = [I@3e25a5
14 6 98 -4
Press any key to continue...
```

```
public class Parameters
{
    public static void doStuff(int [] arr)
    {
                                          rint(arr[x]+" ");
                                          (arr[x]+" ");
     */
    public static void main (              ] args)
    {
        int [] ints =
        out.println("a
        for(int x=0;x<
        out.println();
        doStuff(ints);
        out.println("a
        for(int x=0;x<
        out.println
    }
}
```

The change that was made by the formal parameter in the method also affected the actual parameter in main.



ints →

0	1	2	3
14	6	98	-4

Passing objects by reference

There is an important clarification regarding actual and formal parameters:

- *Formal parameters ONLY EXIST during the time the method is active.*
- *Once the method is finished, the formal parameter ceases to exist.*
- *Actual parameters exist before, during, and after the method call.*

```
tuff(int [] arr)

l = "+arr);
length;x++)out.print(arr[x]+" ");

l = "+arr);
length;x++)out.print(arr[x]+" ");

String [] args)

rint(in

(in

out.println
}
}
```

```
Press any key to continue...
```

ints →

0	1	2	3
14	6	98	-4

Changing the formal reference

Another issue also needs to be addressed:

- The TWO-WAY situation *(changes in formal also make changes in actual)* can only occur when the formal parameter keeps the same reference *(points to the same memory location)* .
- If the ***formal reference*** changes during the method (points to another object), anything the formal parameter does has ***no effect*** on the actual parameter since they are no longer referencing the same object.

ints ⟶

0	1	2	3
14	6	98	-4

Changing the formal reference

Here is an example:

- ***ints*** first points to the original array containing `14, 25, 98,` and `-4,` which is output in main.

```
public static void doStuff(int [] arr)
{
    out.println("formal = "+arr);
    for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
    out.println();
    arr = new int[4];
    out.println("formal = "+arr);
    for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
    out.println();
}
/**main method
 */
public static void main (String [] args)
{
    int [] ints = {14,25,98,-4};
    out.println("actual = "+ints);
    for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
    out.println();
    doStuff(ints);
    out.println("actual = "+ints);
    for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
    out.println();
}
```

ints

0	1	2	3
14	25	98	-4

```
actual = [I@3e25a5
14 25 98 -4
```

Changing the formal reference

- ***arr*** receives and outputs the same array as ***ints.***

Parameters.java

```
public class Parameters
{
    public static void doStuff(int [] arr)
    {
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
        arr = new int[4];
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
    }
    /**main method
     */
    public static void main (String [] args)
    {
        int [] ints = {14,25,98,-4};
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
        doStuff(ints);
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
    }
```

arr

ints

0	1	2	3
14	25	98	-4

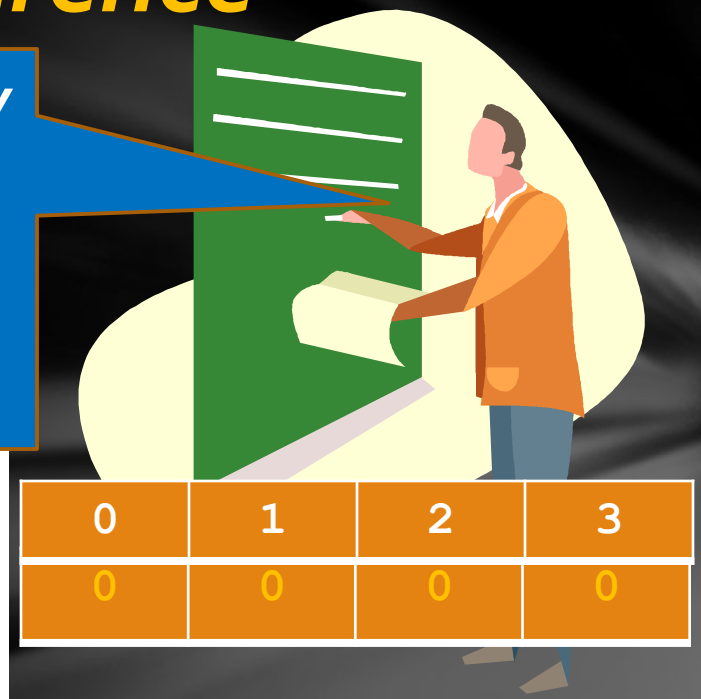
```
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
```

Changing the formal reference

- ***arr*** is then reassigned and outputs a ***new array***, in a different memory location which initially contains all zeroes.
- Notice the hashcode has changed.
- ***ints*** still has the original array.

```
    {
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
        arr = new int[4];
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
    }
    /**main method
     */
    public static void main (String [] args)
    {
        int [] ints = {14,25,98,-4};
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
        doStuff(ints);
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
    }
```

arr



0	1	2	3
0	0	0	0

ints

0	1	2	3
14	25	98	-4

```
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
formal = [I@19821f
0 0 0 0
```

Changing the formal reference

- When program control returns to main, no changes have been made to the original array, still referenced by the actual parameter, and the formal parameter has ceased to exist, ***gone with the wind!***

```
        out.println();
        arr = new int[4];
        out.println("formal = "+arr);
        for(int x=0;x<arr.length;x++)out.print(arr[x]+" ");
        out.println();
    }
    /**main method
     */
    public static void main (String [] args)
    {
        int [] ints = {14,25,98,-4};
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
        doStuff(ints);
        out.println("actual = "+ints);
        for(int x=0;x<ints.length;x++)out.print(ints[x]+" ");
        out.println();
    }
```

ints

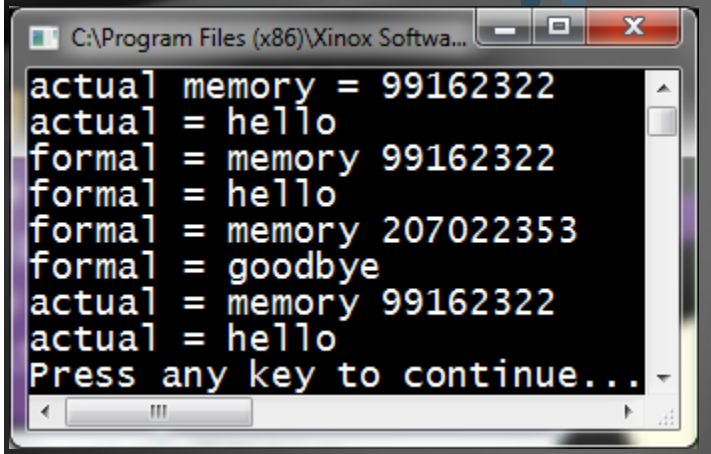
0	1	2	3
14	25	98	-4

```
actual = [I@3e25a5
14 25 98 -4
formal = [I@3e25a5
14 25 98 -4
formal = [I@19821f
0 0 0 0
actual = [I@3e25a5
14 25 98 -4
Press any key to continue
```

Passing String objects

- Finally, we need to discuss passing Strings as parameters.
- Since they are **objects**, they are indeed passed by reference.
- Let's study this program example.

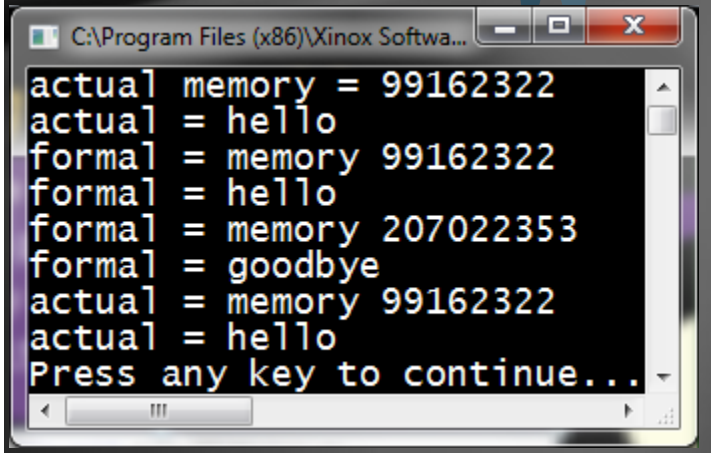
```
29      public static void doStuff(String x)
30      {
31          out.println("formal = memory "+x.hashCode());
32          out.println("formal = "+x);
33          x = "goodbye";
34          out.println("formal = memory "+x.hashCode());
35          out.println("formal = "+x);
36      }
37      /**main method
38       */
39      public static void main (String [] args)
40      {
41          String word = "hello";
42          out.println("actual memory = "+word.hashCode());
43          out.println("actual = "+word);
44          doStuff(word);
45          out.println("actual = memory "+word.hashCode());
46          out.println("actual = "+word);
47      }
48  }
```



Passing String objects

- The original String, ***word*** that contains "hello", is output along with its memory location (the base 10 version).
- It is then passed to ***doStuff***, output with the same value and hashcode, and then changed.

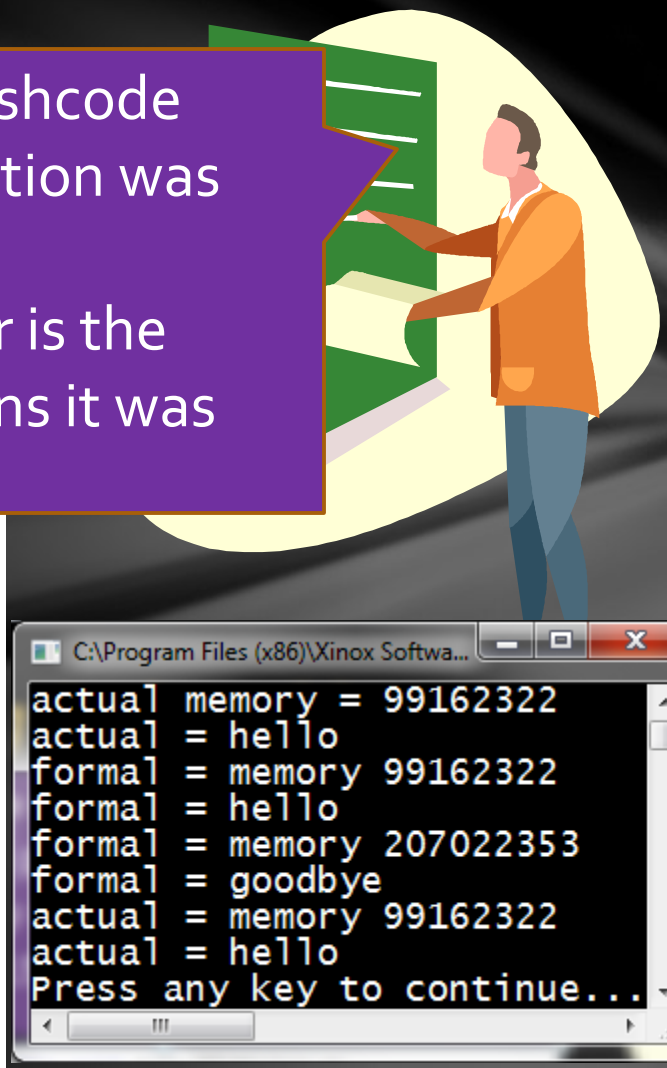
```
29      public static void doStuff(String x)
30      {
31          out.println("formal = memory "+x.hashCode());
32          out.println("formal = "+x);
33          x = "goodbye";
34          out.println("formal = memory "+x.hashCode());
35          out.println("formal = "+x);
36      }
37      /**main method
38       */
39      public static void main (String [] args)
40      {
41          String word = "hello";
42          out.println("actual memory = "+word.hashCode());
43          out.println("actual = "+word);
44          doStuff(word);
45          out.println("actual = memory "+word.hashCode());
46          out.println("actual = "+word);
47      }
48  }
```



Passing String objects

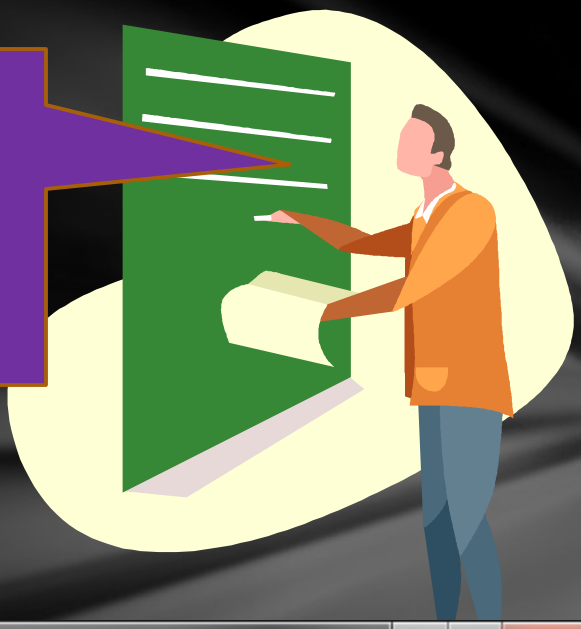
- This change is evidenced by the new hashcode value, which means a new memory location was created with new contents
- The final output of the actual parameter is the same as the original output, which means it was NOT changed.

```
31          out.println("formal = memory "+x.hashCode());
32          out.println("formal = "+x);
33          x = "goodbye";
34          out.println("formal = memory "+x.hashCode());
35          out.println("formal = "+x);
36      }
37      /**main method
38       */
39      public static void main (String [] args)
40      {
41          String word = "hello";
42          out.println("actual memory = "+word.hashCode());
43          out.println("actual = "+word);
44          doStuff(word);
45          out.println("actual = memory "+word.hashCode());
46          out.println("actual = "+word);
47      }
48  }
```

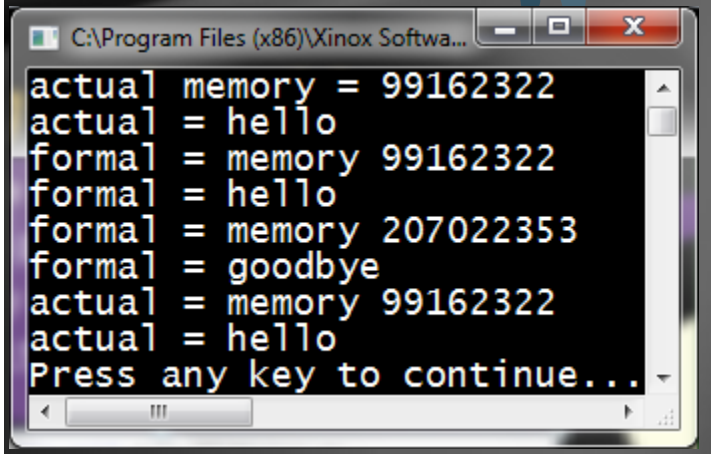


Passing String objects

- This means that although they are indeed passed by reference, Strings actually behave as if they are passed by value…strange, but true.



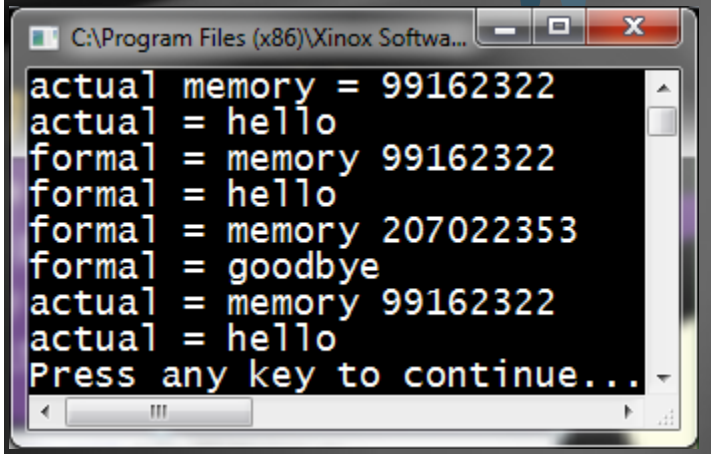
```
29      public static void doStuff(String x)
30      {
31          out.println("formal = memory "+x.hashCode());
32          out.println("formal = "+x);
33          x = "goodbye";
34          out.println("formal = memory "+x.hashCode());
35          out.println("formal = "+x);
36      }
37      /**main method
38       */
39      public static void main (String [] args)
40      {
41          String word = "hello";
42          out.println("actual memory = "+word.hashCode());
43          out.println("actual = "+word);
44          doStuff(word);
45          out.println("actual = memory "+word.hashCode());
46          out.println("actual = "+word);
47      }
48  }
```



Passing String objects

- Furthermore, this reinforces the fact that Strings are immutable (their current state cannot be changed), so any new assignment is actually a reference to a new String object, and the old object is gone!

```
29      public static void doStuff(String x)
30      {
31          out.println("formal = memory "+x.hashCode());
32          out.println("formal = "+x);
33          x = "goodbye";
34          out.println("formal = memory "+x.hashCode());
35          out.println("formal = "+x);
36      }
37      /**main method
38       */
39      public static void main (String [] args)
40      {
41          String word = "hello";
42          out.println("actual memory = "+word.hashCode());
43          out.println("actual = "+word);
44          doStuff(word);
45          out.println("actual = memory "+word.hashCode());
46          out.println("actual = "+word);
47      }
48  }
```



Lesson Summary

- In this lesson, you learned how parameter passing occurs using actual parameters and formal parameters, passing by value or by reference.
- The crucial issue is whether or not changes in the method header's formal parameter affect the original data in the actual parameter of the method call.

Lesson Summary

- When a parameter is passed by value, as with primitive data, changes in the formal parameter DO NOT affect the actual parameter.
- When passing by reference, as with objects, changes to the state of the object by the formal parameter DO affect the actual parameter object, **as long as the reference stays the same.**

Lesson Summary

- Finally, although Strings are passed by reference, since they are immutable and cannot be changed in current memory, in effect, they behave as if they are passed by value.

Labs

- The labs for this lesson essentially duplicate the situations presented in the lesson.
- Create a class called MyParameters and do all your work there.

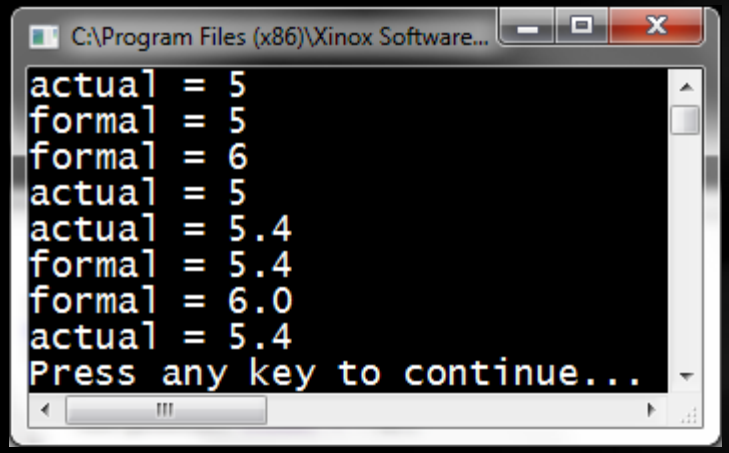
Lab 8A

- Duplicate the program found on slide 18 of this lesson, defining the doStuff method and calling it exactly as shown.
- Then define another method, also called ***doStuff***, but change the formal parameter to type double:

```
public static void doStuff (double x)
```

Lab 8A

- The output and program shell are shown here.



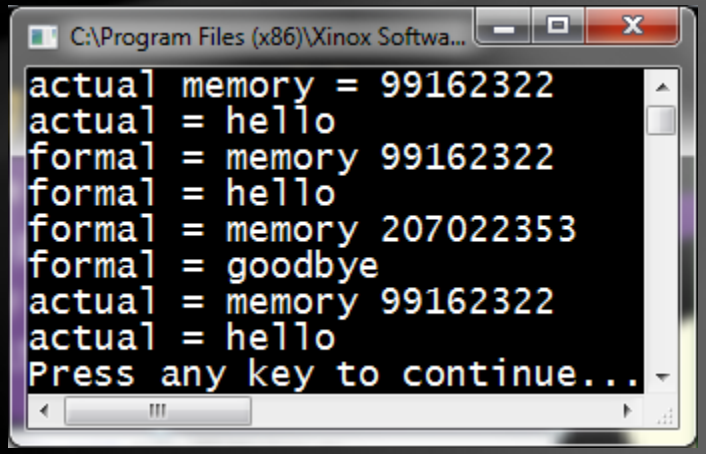

```
public static void doStuff(int x)
{
    [redacted]
}
public static void doStuff(double x)
{
    [redacted]
}
/**main method
 */
public static void main (String [] args)
{
    int y = 5;
    out.println("actual = "+y);
    doStuff(y);
    out.println("actual = "+y);
    double z = 5.4;
    out.println("actual = "+z);
    doStuff(z);
    out.println("actual = "+z);

}
```

Lab 8B

- In this lab, define a third ***doStuff*** method that duplicates the String example shown at the end of the lesson, producing the output shown here.

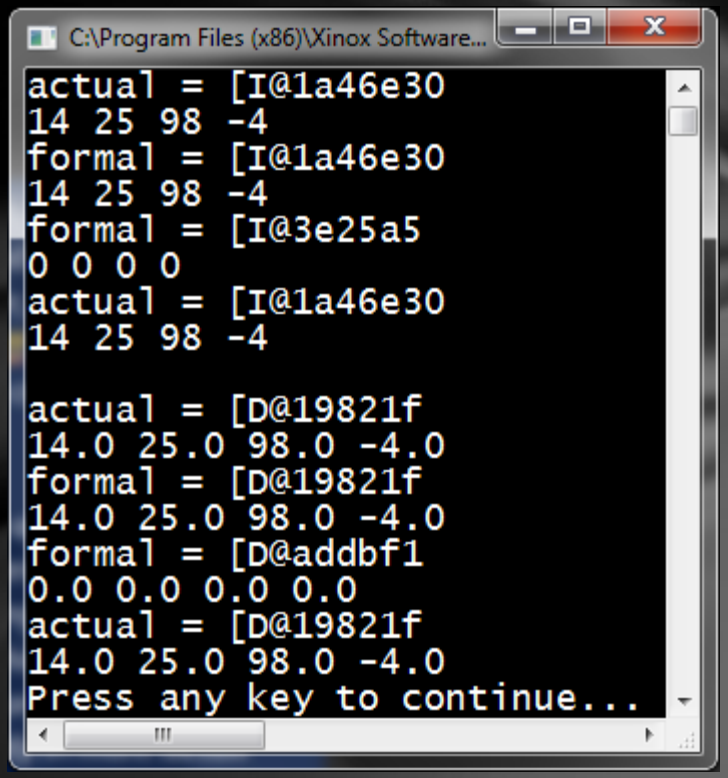
```
29      public static void doStuff(String x)
30      {
31          out.println("formal = memory "+x.hashCode());
32          out.println("formal = "+x);
33          x = "goodbye";
34          out.println("formal = memory "+x.hashCode());
35          out.println("formal = "+x);
36      }
37      /**main method
38       */
39      public static void main (String [] args)
40      {
41          String word = "hello";
42          out.println("actual memory = "+word.hashCode());
43          out.println("actual = "+word);
44          doStuff(word);
45          out.println("actual = memory "+word.hashCode());
46          out.println("actual = "+word);
47      }
48  }
```



Lab 8C

- For this final lab, duplicate the ***doStuff*** method (slide 30) that receives the integer array parameter, and then create another ***doStuff*** method to receive a double array parameter, creating the output shown here.

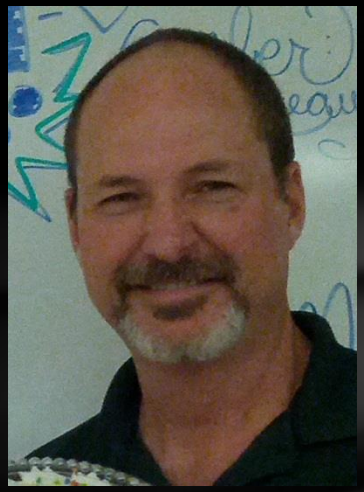
- *Just in case you are wondering, it is OK to have several methods named the same, like **doStuff**, as long as the parameter signature is unique for each one.*
- *This is called "overloading", or **polymorphism**, which we'll study in more detail later on.*



CONGRATULATIONS!

- You now understand more about parameter passing.
- *Lesson 9 will introduce matrices (two-dimensional arrays).*

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