Introduction to Arrays

Part 2: Copying arrays, Using array as parameter and return value of Methods, Command line argument

Course: CPSC 1150

Instructor: Dr. Bita Shadgar

Lecture 16

Learning Outcomes

- Use array initializer
- Copy an array
- Pass an array to a method
- Return an array from a method
- Use command line to pass argument to java program

Array Initializer

- Array initializer syntax needs to be typed all in one statement, including the declaration
- The following is invalid – the second line causes a compile error:

```
int[] myList;
myList = {-1, -5, -6, -4};  //wrong
```

- Likewise, the following is invalid and will cause a compile error:

```
int[] myList = {4, 3, 8, 2};
myList = {-1, -5, -6, -4}; //wrong
```

- If you want to write this code, must first initialize {-1,-5,-6,-4} with its own name, or must use anonymous array syntax (will see this today)

How to fix it?

- Two possible ways to fix the error:

```
int[] myList = {4, 3, 8, 2};
int[] otherList = {-1, -5, -6, -4};
myList = otherList;
```

```
int[] myList = {4, 3, 8, 2};
myList = new int[]{-1, -5, -6, -4}; //using anonymous array
```

Copying primitive types

- Recall how primitive types are copied
  - The value of one is just assigned to the other

Copying primitive types

```
int x = 4;
int y = x;
```

Question: What happens if we try to copy an array in the same way?

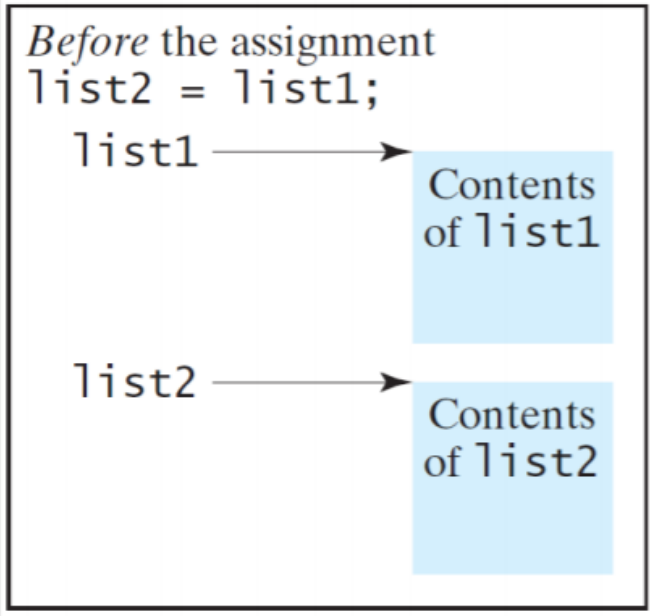
```
int[] xList = {29, 31, 4, -24};
int[] yList = xList;
```

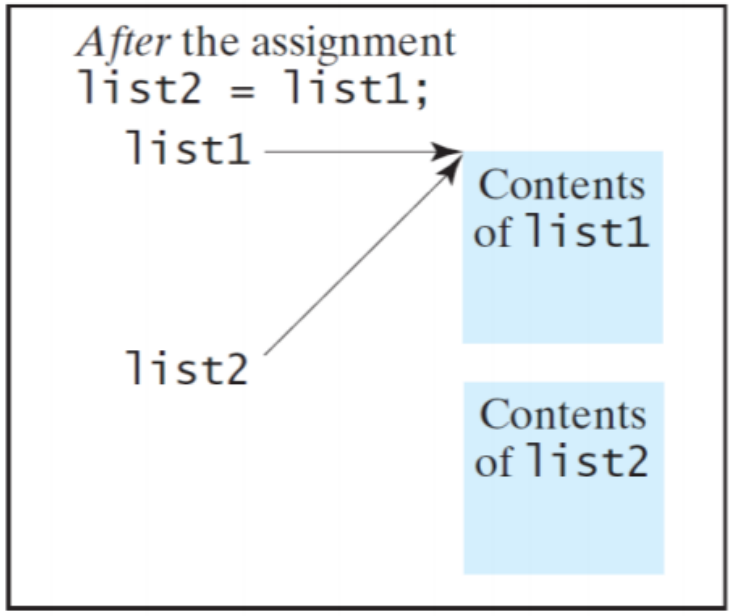
Answer: This code only copies the reference! The variables xList and yList now both refer to the same array. If we modify the contents of the array, both variables will now point to the changed array.

Reassigning an array reference variable

– **Question:** Can you illustrate what is happening in memory when the following code is executed?

```
int[] theList = {2, 4, 6};
int[] otherList = {1, 3, 5};
theList = otherList;
```





Understanding arrays (1)

- How many bytes of memory are allocated for a float array of length 12?
- Will the following code compile and run? If so, what is the output? How can we access the first array?

```
int[] myList = {4, 3, 8, 2};
myList = new int[]{-1, -5, -6, -4};
System.out.print(myList[1]);   //prints -5
```

- Will the following code compile and run? If so, what is the output?

```
int[] myList = new int[300];
myList = new int[]{-1, -5, -6, -4};
System.out.println(myList[1]);   //prints -5
System.out.println(myList.length); //prints 4
```

Understanding arrays (2)

- Will the following code compile and run? If so, what is the output?

```
int[] myList = {-1, -5, -6, -4};
myList[4] = -12;
System.out.println(myList[4]); //ArrayIndexOutOfBoundsException
```

- What is the output of the following code? Explain.

```
int[] list1 = {-1, -5, -6, -4};
int[] list2 = list1;
list2[2] = 13;
System.out.println(list1[2]);   //prints 13
```

Two techniques for copying arrays

- Copy individual elements

1. Start with array A
2. Create a new array, B, of the same length as A
3. In a for loop, copy each element
   3.1. Set each B[i] = A[i]

- Use arraycopy

1. Static void method from the System class
2. Pass in both arrays, starting indices for each, and number of elements to copy
   2.1. Both arrays must already have been created
3. Note that this method violates the naming conventions

◦ Syntax for arraycopy

```
System.arraycopy(srcArray, srcPos, tgtArray, tgtPos, length);
```

Array copying example

Example

Let’s take a look at the program CopyArrays, which will demonstrate both techniques for copying an array.

Passing arrays to methods: Syntax

- An array (or several arrays) can be passed to a method
- For example, we may define a method with the following header:

```
public static double getMax(double[] values)
```

- Invoking getMax:

```
double[] myList = {1.03, 31.0, -7.66};
double maxVal = getMax(myList);  //invoking getMax
```

- Another option is to create an **anonymous array**. . .

Anonymous arrays

- An anonymous array is an array with no explicit reference variable
- Can be useful if you need an array to pass into a method, and don't want to write a lot of code
- The syntax is: new type[]{val1, val2, ...}

Example

```
double maxVal = getMax(new double[]{1.03, 31.0, -7.66});
```

  - This array can not be accessed again after getMax returns

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++)
    System.out.print(array[i] + " ");
  System.out.println();
}
```

1. Invoke the method

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

2. Invoke the method using anonymous array

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous array

Passing arrays to methods: Mechanism

- Recall that when a method is invoked:
- Its arguments are stored on the stack for later
- The method's parameters are created on the stack, by copying values from the arguments
- If we invoke a method with an array as an argument. . .

  Question: What gets stored on the stack when an array gets passed to a method?

  Answer: The value of the array's reference variable.
- The array itself is stored elsewhere, in the heap
  - An array does not get copied when passed to a method
- **Key point:** When an array is passed to a method and modified inside, the changes are **visible everywhere** (unlike primitive types)

Recall pass-by-value

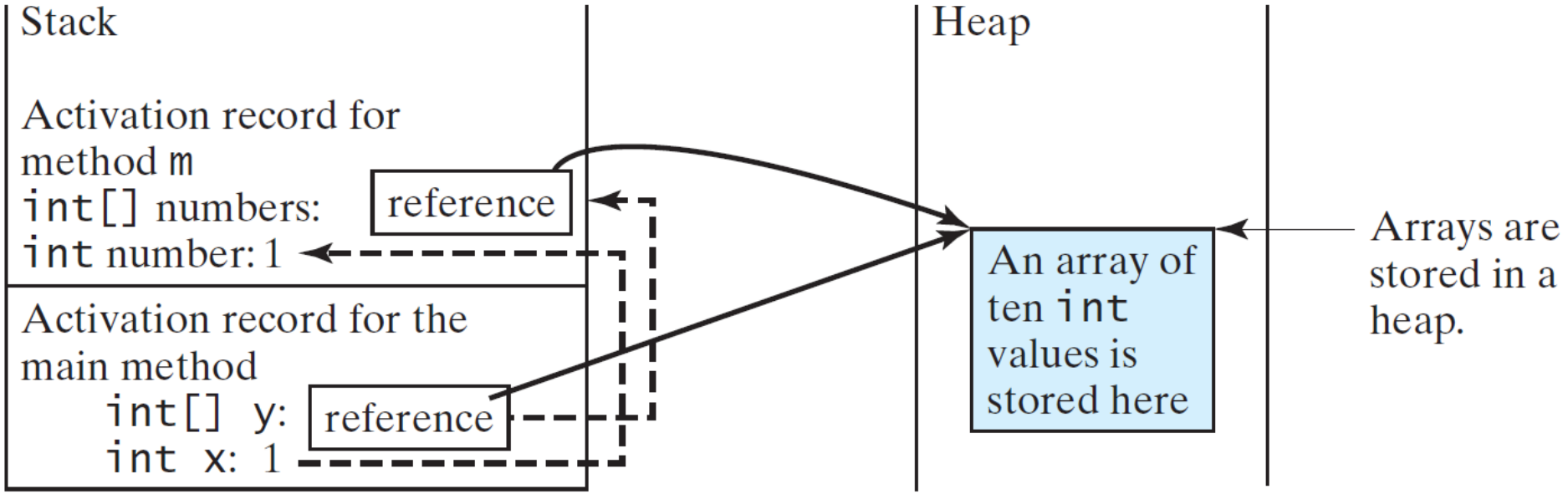
- Recall that when primitive types are passed to methods, they are **passed by value**
  - Changing the value of a parameter inside a method can not impact the argument that was passed in
- For arrays, Java still uses **pass-by-value**, but what value gets passed?

  Answer: The value of the reference variable.
- Contents (values) of an array are not copied to the method's activation frame
  - The original array can be accessed (and modified!) using the reference parameter
- Reassigning the reference parameter inside the method does not impact the original array

Passing array example

Example

Let's take a look at the program TestPassArray, which will demonstrate the difference between passing primitive types and passing arrays, by creating two different swap methods.



Returning an array from a method

- To return an array from a method, make the return type (in the method header) an array type, i.e., float[]
- In the method body, return an array reference variable of matching type

```
public static int[] getReversedArray(int[] source){
    int length = source.length;
    int[] dest = new int[length];
    for(int i = 0; i < length; i++){
        dest[i] = source[length - i - 1];
    }
    return dest;
}
```

Question: If a method is meant to simply modify an array, do we need to return the modified array?

Command Line Arguments

Header for main

```
public static void main(String[] args)
```

- `main` is a method just like any other
  - Its return type is void
  - It always has one parameter – an array of Strings
- When you execute a program, i.e., type `java myClass`, you are really telling the JVM to invoke the main method in myClass
  - Until now, you've done this without passing in any arguments

Passing arguments to main

- When running a program from the command line, use the following syntax to pass in arguments:

java MyClass arg_0 arg_1 arg_2 ... arg_n

- The arguments should be separated by spaces
- Inside the main method of myClass, you can access the arguments
  - They are stored as Strings in the array args
  - Just type args[i] to get the i th argument
- Often these arguments need to be converted into other types or processed before they can be used as intended

```
public class myClass {
  public static void main(String[] args) {
      for(int i =0; i < args.length; ++i)
          System.out.printf("arg[%d] is %s.\n", i, args[i]);
  }
}
```

More Practice - Calculator

Example

Let's look at the Calculator program from the textbook to see how to use/process command line arguments.

- Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

```
java Calculator 2 + 3
java Calculator 2 - 3
java Calculator 2 / 3
java Calculator 2 . 3
```