Set 10 Answers

1. How many `ints` are created by the statement: `int[] a = new int[5];`

5

2. How many Strings are created by the statement: `String[] a = new String[5];`
(Hint: The answer to this question and the previous question are different!)

None. An array of null-references is all that is created – so you have created five references to Strings but no actual Strings.

3. Are the elements of an array of primitives automatically initialized? If so, to what values?

Yes, to 0 or false.

4. Are the elements of an array of references to objects initialized? If so, to what values?

Yes, to null.

5. Draw the memory diagram for each of the following code fragments:

```
a.    int[] a = new int[4];
Should have a variable on the call stack which points to an
array on the heap of size 4.  Each box in the array
contains a zero.
</SPAN>
b.    String[] b = new String[4];
   for (int i = 0; i < b.length; i++)
    b[i] = _value _ + i;
Should have a variable on the call stack which points to an
array on the heap of size 4. Each box in the array points
to a String.  The strings say _value0_, _value1_, _value2_,
and _value3_.
```

6. Write a class that has an instance variable which is an array of `Cat` objects, called `kitties`. Write a method that returns a reference copy of `kitties`. Write a method that returns a shallow copy of `kitties`. Write a method that returns a deep copy of `kitties`.

```
public class Circus {
     private Cat[] kitties;

     public Cat[] referenceCopy() {
```

```
            return kitties;
        }

        public Cat[] shallowCopy() {
            Cat[] copy = new Cat[kitties.length];
            for (int i = 0; i < copy.length; i++) {
                copy[i] = kitties[i];
            }
            return copy;
        }

        public Cat[] deepCopy() {
            Cat[] copy = new Cat[kitties.length];
            for (int i = 0; i < copy.length; i++) {
                copy[i] = new Cat(kitties[i]);
            }
            return copy;
        }
}
```

7. Suppose you are passing (or returning) an array of primitives to/from a method. Is it safe to make a reference copy only?

No.

8. Suppose you are passing (or returning) an array of references to immutable objects to/from a method. Is it safe to make a reference copy only? Is it safe to make a shallow copy?

Reference copy is unsafe. Shallow is safe.

9. Suppose you are passing or returning an array of references to mutable objects to/from a method. Is it safe to make a reference copy only? Is it safe to make a shallow copy?

Neither one is safe. Only a deep copy is safe in this case.

10.

a. Write a code fragment that creates a two-dimensional ragged array of ints with 3 rows, initialized with the following data:

5 8 9

4 11 13 15 17

0 1

```
int[][] x = {{5, 8, 9}, {4, 11, 13, 15, 17}, {0, 1}};
```

b. After you have created this array, write code that will print the contents in the same format that you see above.

```
for (int row = 0; row < x.length; row++) {
                    for (int col = 0; col < x[row].length; col++) {
                       System.out.print(x[row][col] + " ");
                    }
                    System.out.println();

      }
```

c. Draw the memory map for this array.

x should appear on the stack.

x should refer to a _row array_ on the heap of size 3. The first element of the _row array_ will refer to an array of size 3 that contains 5, 8, and 9. The second element of the _row array_ will refer to an array of size 5 that contains 4, 11, 13, 15, and 17. The third element of the _row array_ will refer to an array of size 2 that contains 0 and 1.

11.

a. Write a method called catDuplicator, with the following prototype:

```
public Cat[][] catDuplicator(int[] rowSizes, Cat c)
```

The method will create a two-dimensional ragged array, using the array _rowSizes_ to determine how many rows there are, and how long each row must be. Each element of the array that gets created will refer to the very same cat, `c`.

For example, if the array `rowSizes` contains the data: 5 7 2 1

then the return value would be a two-dimensional ragged array with four rows. The first row would be size 5, the second row would be size 7, etc. Each element of the two-dimensional array must be a reference to `c`.

```
public static Cat[][] catDuplicator(int[] rowSizes, Cat c) {
               Cat[][] answer = new Cat[rowSizes.length][];
               for (int i = 0; i < rowSizes.length; i++) {
                  answer[i] = new Cat[rowSizes[i]];
               }
               for (int i = 0; i < answer.length; i++) {
                  for (int j = 0; j < answer[i].length; j++) {
                        answer[i][j] = c;
                  }
               }
               return answer;

}
```

b. Draw the memory map for the method above.

The variable c is on the stack and it refers to ONE cat on the heap. The variable answer is on the stack and it refers to a _row array_ that is the same length as the length of the array rowSizes. Each element of the _row array_ refers to a row. A row is an array of references (the sizes of the rows match the entries in the array rowSizes). Each element of each row refers to the very same cat that c refers to!

12. Write code that asks the user for a value (n), and then creates an n by n two-dimensional array of ints. Fill the array with a multiplication table. For example, if n is 3, the table should be:

1 2 3

2 4 6

3 6 9

```
System.out.println("Enter n: ");
            java.util.Scanner s = new java.util.Scanner(System.in);
            int size = s.nextInt();
            int[][] a = new int[size][size];
            for (int row = 0; row < size; row++) {
                  for (int col = 0; col < size; col++) {
                        a[row][col] = (row + 1) * (col + 1);
                  }

}
```

13. a.

Write a method which has the following prototype:

public static double[] linearize(double[][] array)

The method linearizes the two-dimensional array by returning a one-dimensional array with all the elements of the parameter (selected row-by-row). The original array cannot be modified.

```
public static double[] linearize(double[][] array) {
               int size = 0;
               for (int i = 0; i < array.length; i++) {
                  size += array[i].length;
               }
               double[] answer = new double[size];
               int position = 0;
               for (int i = 0; i < array.length; i++) {
                  for (int j = 0; j < array[i].length; j++) {
                        answer[position++] = array[i][j];
                  }
               }
               return answer;

}
```

b. Write a JUnit test that tests your method.

```
public void testLinearize() {
                double[][] x = {{5, 2, 3}, {8, 9}, {7, 11, 15}};
                double[] correctAnswer = {5, 2, 3, 8, 9, 7, 11, 15};
                double[] answer = MyClass.linearize(x);
                for (int i = 0; i < x.length; i++) {
                   assertEquals(correctAnswer[i], answer[i]);
                }

}
```

14. Write a method called _deepCopy_ that takes a parameter (a two-dimensional array of String objects) and returns a deep copy. (You must make copies of the Strings themselves, even though they are immutable.)

```
public static String[][] deepCopy(String[][] a) {
                String[][] copy = new String[a.length][];
                for (int i = 0; i < copy.length; i++) {
                   copy[i] = new String[a[i].length];
                }
                for (int i = 0; i < copy.length; i++) {
                   for (int j = 0; j < copy[i].length; j++) {
                         copy[i][j] = new String(a[i][j]);
                   }
                }
                return copy;

      }
```