

Teaching Unit 5: Arrays

5.1 Array Basics

Solutions

5.1.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
int[] a = {4, 6, 8};  
int[] b = a;  
b[0] = 9;  
System.out.println(a[0]);
```

Output

9

Both references point to the same mutable array, so either reference can observe the mutation.

5.1.2 - Write code

Create an int array of length four. Set its first element to 7 and its last to 12, then print all four values in index order using Arrays.toString.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
int[] values = new int[4];  
values[0] = 7;  
values[values.length - 1] = 12;  
System.out.println(Arrays.toString(values));
```

Teaching Unit 5: Arrays

5.2 Array Traversal

Solutions

5.2.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
int[] values = {2, 5, 1};  
for (int value : values) value *= 2;  
System.out.println(Arrays.toString(values));
```

Output

[2, 5, 1]

Assigning the local enhanced-for variable does not replace primitive array elements.

5.2.2 - Write code

Write public static int countAbove(int[] values, int cutoff). Return the number strictly greater than cutoff. Empty arrays must return 0.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
public static int countAbove(int[] values, int cutoff) {  
    int count = 0;  
    for (int value : values) if (value > cutoff) count++;  
    return count;  
}
```

Check your code

```
System.out.println(countAbove(new int[]{4, 7, 7, 2}, 4));  
System.out.println(countAbove(new int[]{}, 4));
```

Expected output

2
0

Teaching Unit 5: Arrays

5.3 Array Mutation

Solutions

5.3.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
int[] a = {1, 2, 3, 4};  
for (int i = 1; i < a.length; i++) a[i] = a[i - 1];  
System.out.println(Arrays.toString(a));
```

Output

[1, 1, 1, 1]

Each iteration reads the value that the preceding iteration already overwrote.

5.3.2 - Write code

Write public static void reverse(int[] values). Reverse the array in place using swaps. Empty and one-element arrays should remain valid.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
public static void reverse(int[] values) {  
    for (int i = 0; i < values.length / 2; i++) {  
        int opposite = values.length - 1 - i;  
        int temp = values[i];  
        values[i] = values[opposite];  
        values[opposite] = temp;  
    }  
}
```

Check your code

```
int[] a = {1,2,3,4}; reverse(a); System.out.println(Arrays.toString(a));  
int[] b = {}; reverse(b); System.out.println(Arrays.toString(b));  
int[] c = {8}; reverse(c); System.out.println(Arrays.toString(c));
```

Expected output

[4, 3, 2, 1]
[]
[8]

Teaching Unit 5: Arrays

5.4 Array Search

Solutions

5.4.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
int[] values = {4, 2, 4};
int found = -1;
for (int i = 0; i < values.length; i++) {
    if (values[i] == 4) found = i;
}
System.out.println(found);
```

Output

2

Without an early return or stop condition, later matching positions overwrite earlier ones.

5.4.2 - Write code

Write public static int firstIndex(int[] values, int target). Return the first matching index, or -1 when absent.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
public static int firstIndex(int[] values, int target) {
    for (int i = 0; i < values.length; i++) {
        if (values[i] == target) return i;
    }
    return -1;
}
```

Check your code

```
System.out.println(firstIndex(new int[]{6,2,6}, 6));
System.out.println(firstIndex(new int[] {}, 6));
System.out.println(firstIndex(new int[]{1,2}, 6));
```

Expected output

```
0
-1
-1
```