2. One and Two Dimensional Arrays :: Array References and Copying Arrays

Assigning one array variable b to another array variable a does not copy the elements of b to a. Rather the reference in a will be changed to that of b meaning that a and b will both refer to the same array object:

```
int[] a = { 1, 2, 3 };
int[] b = { 4, 5, 6 };
a = b;
b[1] = 10;
System.out.print(a[1]);
```

If the goal is to change the actual contents of a to be the same as b then we can write a **for loop** which copies values from b to a one at a time:

```
int[] a = { 1, 2, 3 }
int[] b = { 4, 5, 6 };
for (int i = 0; i < b.length; ++i) {
  a[i] = b[i];
}
```

2. One and Two Dimensional Arrays :: Passing Arrays to Methods

An array may be the **argument** to a method but remember that what is passed is not the contents of the array but rather the **reference** stored in the array variable. This means that any changes to the array in the method will actually change the contents of the array argument that was passed. Example:

```
public void decEveryOtherElement(int[] anArray) {
  for (int i = 0; i < anArray.length; i += 2) {
    --anArray[i];
  }
}

public void printArray(int[] anArray) {
  for (int value : anArray) {
    System.out.println(value);
  }
}

public void someMethod() {
  int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

  decEveryOtherElement(a);

  // Will print { 0, 2, 2, 4, 4, 6, 6, 8, 8, 10 }.
  printArray(a);
}
```

2. One and Two Dimensional Arrays :: Returning Arrays from Methods

A method may create and return a new array:

```
public int[] createArray(int len, int initValue) {
  int[] newArray = new int[len];
  for (int i = 0; i < newArray.length; ++i) {
    newArray[i] = initValue;
  }
  return newArray;
}

public void someMethod() {
  int a = createArray(5, 10);
}
```