Chad Philip Johnson
CSCI20, Worthington
November 08th, 2012

Pencil & Paper 12

1. What is a partially filled array?
A partially filled array is an array that has a helper variable of type integer that keeps track of how much of the array has been used by the program. Our textbook offers the following comments on this use of an array: "Partially filled arrays require some care. The program must keep track of how much of the array is used and must not reference any indexed variable that has not been given a meaningful value.... the program uses only as much of the array as it needs…. [a variable] keeps track of how many elements are stored in the array" (369). Partially filled arrays are not always filled completely during the execution of the program. Oftentimes an array is declared "to be the largest size a program could possibly need. The program is then free to use as much or as little of the array as needed" (369).

2. In a class containing a partially filled array as a member variable, how would you keep track of how much of the array has been used?
Declare a helper variable of type integer within the same class and increment it as additions to the array are made and, conversely, decrement it when subtractions to the array are made (that is, increase the variable by one when an element is put into the array and decrease the variable by one when an element is taken from the array). This variable can then be used to check when a valid index value for the array is greater than the number of elements of the array that have actually been filled. According to our textbook, the helper variable "usually must be an argument to any method that manipulates the partially filled array…. to ensure that only meaningful array indices are used" (369).

3. What is the appropriate return value for a method that accesses an object in an array, when there is no object present?
When no object is present within an element of an array, the constant null should be used for a return value. According to our textbook, "null is a special constant that can be used to give a value to any variable of any class type. The constant null is not an object but a sort of placeholder for a reference to an object" (297).

4. Write a for-each loop to count the number of available "slots" in the array messages.

```
        String [] messages = {"Hello", null, "Goodbye", null};
        int count = 0;
```

The following code will count the number of available "slots" in the array messages using a for-each loop:

```
String [] messages = {"Hello", null, "Goodbye", null};
int count = 0;
for( String message : messages ) {
        if( message == null ) {
                count++;
        }
}
```

Note: Our textbook offers the following comments when checking for null values: “Because [null] is like a reference (memory address), use == and != rather than the method equals() when you test to see whether a variable contains null” (297).

5. Write a for-each loop to count the number of occupied "slots" in the array messages.

```
	String [] messages = {"Hello", null, "Goodbye", null};
	int count = 0;
```

The following code will count the number of occupied “slots” in the array messages using a for-each loop:

```
String [] messages = {"Hello", null, "Goodbye", null};
int count = 0;
for( String message : messages ) {
	if( message != null ) {
		count++;
	}
}
```

6. Use FruitBasket (see link in code section above) to answer this question. Does the removeFruit method "pack" the array after the removal, or does it leave an empty slot at the removal location?

No, the removeFruit method does not “pack” the array after removing a fruit object. The method first checks to make sure that the requested index is valid for the array. Provided the index is valid it then assigns the constant null to that index location in the array after assigning the previous object to the temporary variable theFruit. The variable theFruit is then returned by the method. Instead of packing the array the method simply fills this “slot” with a null value.

7. Use FruitBasket (see link in code section above) to answer this question. Does the addFruit method add the Fruit at the end of the occupied "slots", or does it insert the Fruit at the first available (null) spot it finds?

No, the method addFruit inserts the new fruit object in the first available null spot it finds (which will be at the end of the occupied “slots” if no null values are found in between the elements of the array containing non-null values). It checks that the variable count has not exceeded the length of the array and then searches for an element that contains the null constant. If one is found it assigns the fruit to that element in the array and then increments the count variable by one.

8. Use FruitBasket (see link in code section above) to answer this question. Assuming there is at least one Fruit in the array, and that the supplied index value is greater than or equal to zero and less than the size of the array, is it possible to get a null value returned by the method getFruit?

Yes, it is possible to get a null value returned by the method getFruit when at least one fruit object exists in the array. This is because the method’s if statement only checks whether the supplied index value is valid for the size of the array basket and not whether the element at that index location is actually a fruit: getFruit simply returns the currently stored object/value at the provided index of the array (provided it is within the boundaries of the array).

9. Use FruitBasket (see link in code section above) to answer this question. Does the method getFruit remove (delete) a Fruit object from the array?

No, the method getFruit does not delete/remove the fruit object from the array at the given index--it only retrieves the value that is stored there. This value is then returned by the method, leaving the contents of the basket array unchanged.

10. What is the EXACT output of this week's "Coding" files after they have been completed, compiled, and executed?
The following text is printed to the console upon program execution:

```
AN APPLE A DAY
```