CS 61B Spring 2021

Introduction to Java

Discussion 2: January 25th, 2021

1 Our First Java Program

Below is our first Java program of the semester. Next to each line, write out what you think the code will do when run. If you think the line will result in an error, correct it, and proceed through the code as if it is running your corrected version. *This exercise is adapted from Head First Java.*

```
size = 27; // Errors. size must be declared as an int or other number type. To fix this error, the line should be "int size = 27;".
name = "Fido"; // Errors. Similar to above, name must be declared as an String.
Dog myDog = new Dog(name, size); // Declares and initializes a new variable of type Dog. Calls the Dog constructor to create a new object of type Dog. You can assume we corrected the errors above related to size and name.
Dog yourDog = new Dog("Scruffy", 1000); // As before, declares and initializes a new Dog object.
Dog[] dogList = new Dog[3]; // Declares and instantiates an array of size 3 that can hold Dog objects.
dogList[0] = myDog; // Sets the 0th spot in the array to myDog.
dogList[1] = yourDog; // Sets the 1st post in the array to yourDog.
dogList[2] = 5; // Errors. This array only holds Dogs. To fix this error, you could delete this line or replace the right hand side with a valid Dog instance.
dogList[3] = new Dog("Cutie", 8); // Errors. Index 3 is out of bounds for an array of size 3. Java arrays are 0 indexed, so an array of size 3 will have valid indices 0, 1, and 2. To fix this error, you could either change the index to a valid option, delete this line, or go back to the instantiation of dogList and make it an array of size 4 to begin with.
int x; // Declares a new variable x of type int.
x = size - 5; // Given that our declaration of size was corrected, assigns x the value 22.
if (x < 15) {
// If x is less than 15, calls bark, an instance method of the Dog class. Since x is 22, myDog.bark() is not called.
    myDog.bark(8);
}
```

2 Mystery

This is a function (a.k.a. method). It takes an array of integers and an integer as arguments, and returns an integer.

```
public static int mystery(int[] inputArray, int k) {
    int x = inputArray[k];
    int answer = k;
    int index = k + 1;
    while (index < inputArray.length) {
        if (inputArray[index] < x) {
            x = inputArray[index];
```

```
            answer = index;
        }
        index = index + 1;
    }
    return answer;
}
```

(a) What `mystery` returns if `inputArray = [3, 0, 4, 6, 3]` and `k = 2`.

The method returns 4.

(b) Can you explain in plain English what `mystery` does?

It returns the index of the smallest element that occurs at or after index `k` in the array, in this case, 4. If `k` is greater than or equal to the length of the array or less than 0, an `ArrayIndexOutOfBoundsException` will be thrown, though this exception is not something you'd know without running the program.

The variable `x` keeps track of the smallest element found so far and the variable `answer` keeps track of the index of this element. The variable `index` keeps track of the current position in the array. The while loop steps through the elements of the array starting from index `k + 1` and if the current element is less than `x`, `x` and `answer` are updated.

Extra: This is another function. It takes an array of integers and returns nothing.

```
public static void mystery2(int[] inputArray) {
    int index = 0;
    while (index < inputArray.length) {
        int targetIndex = mystery(inputArray, index);
        int temp = inputArray[targetIndex];
        inputArray[targetIndex] = inputArray[index];
        inputArray[index] = temp;
        index = index + 1;
    }
}
```

Describe what `mystery2` does if `inputArray = [3, 0, 4, 6, 3]`.

If `mystery2` is called on the array `3, 0, 4, 6, 3` then after the method runs, the array will be `0, 3, 3, 4, 6`. Given any array, the method `mystery2` sorts the elements of the array in increasing order.

At the beginning of each iteration of the while loop, the first `index` elements of the array are in sorted order. Then the method `mystery` is called to find the index of the smallest element of the array occurring at or after `index`. The element at the index returned by `mystery` is then swapped with the element at position `index` so that the first `index` + 1 elements of the array are in sorted order. This algorithm is called `selection sort`. We will talk about it more later on in the course.

3 Writing Your First Program

Implement `fib` which takes in an integer `n` and returns the nth Fibonacci number.

The Fibonacci sequence is $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$. The first two numbers in the sequence are 0 and 1, and every number thereafter it is the sum of the two numbers in the sequence before it.

```
public static int fib(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fib(n - 1) + fib(n - 2);
    }
}
```

Extra: Implement a more efficient version of `fib` in 5 lines or fewer. Here, efficiency might mean making less recursive calls or doing less overall computation. You don't have to make use of the parameter `k` in your solution.

```
public static int fib2(int n, int k, int f0, int f1) {
    if (n == k) {
        return f0;
    } else {
        return fib2(n, k + 1, f1, f0 + f1);
    }
}
```

While the first solution given, `fib`, uses the same amount of lines, it is not as efficient as `fib2`. One reason `fib` is less efficient is that the highest level of recursion, or the first function call, must wait for all calls beneath it to complete before it can return the answer. The function call at a higher level cannot terminate because it must add the result of two subsequent function calls. This means there are lots of frames (recall CS61A environment diagrams and function call frames) open, and a lot of space wasted. Additionally, `fib` will repeat lots of calculations–notice how if you call `fib(5)` it will calculate `fib(3)` twice.