CS 61B
Spring 2022

Introduction to Java

Discussion 2: January 24, 2022

1 Old Town Code

Next to each line, write out in words what you think the code will do when it is run. Assume the `Singer` class exists and that the code below compiles. You can assume that the `sing` function in Singer returns a `String` and prints nothing.

```java
int x = 7;
String chorus = "Thank u, next";
Singer queen = new Singer("Ariana");

while (x > 0) {
    x -= 1;
    queen.sing(chorus);
}

String[] phrases = {"love", "patience", "pain", "what does the fox say?"};

for (int i = 0; i < 3; i += 1) {
    System.out.println("One taught me " + phrases[i]);
}

System.out.println(phrases[phrases.length - 1]);
```

Hint: For reference, here is an equivalent Python program.

```python
x = 7
chorus = "Thank u, next"
queen = Singer("Ariana")

while (x > 0):
    x -= 1
    queen.sing(chorus)



phrases = ["love", "patience", "pain", "what does the fox say?"]

for i in range(3):
    print("One taught me " + phrases[i])



print(phrases[len(phrases) - 1])
```

Comments are above the line they describe.

```java
/* Declares a variable of type int and assigns it the value 7. In Java, all variables must be
    declared before
  they are used. */
int x = 7;

/* Declares a variable of type String and assigns it the value
  "Thank u, next". */
String chorus = "Thank u, next";

/* Declares a variable of type Singer and initializes it using the Singer
  constructor with the argument "Ariana". Assigns the object to a variable called queen. */
Singer queen = new Singer("Ariana");

/* Checks if x is greater than 0. If so, subtracts 1 from x, then calls
  queen's sing method with argument "Thank u, next", then goes back
  to the beginning of the loop. The queen object calls the sing function
  with parameter "Thank u, next" a total of 7 times. Since this String
  is returned and not printed, it does not appear in the console. */
while (x > 0) {
    x -= 1;
    queen.sing(chorus);
}

/* Declares a variable of type String array and initializes it to
  hold four Strings. */
String[] phrases = {"love", "patience", "pain", "what does the fox say?"};

/* Declares variable i, initializes to 0, and checks if it is less than 3.
  If so, prints the String at index i, adds one to i, then goes back to
  the beginning of the loop and re-checks that i < 3. Prints out
  "One taught me love", "One taught me patience", and "One taught me pain"
  on separate lines. */
for (int i = 0; i < 3; i += 1) {
    System.out.println("One taught me " + phrases[i]);
}

/* Prints "what does the fox say?" to standard output. */
System.out.println(phrases[phrases.length - 1]);
```

2 A Mystery

Below is a function (or method) called `mystery1`. It takes in two arguments and returns an integer, `answer`. The first argument it takes in is an array of integers called `inputArray`, and the second argument it takes in is an integer, `k`.

```java
public static int mystery1(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;
}
```

Write the return value of `mystery1` if `inputArray` is the array {3, 0, 4, 6, 3} and `k` is 2. Then, explain in English what the mehtod `mystery1` does.

The `mystery1` function returns 4. `mystery1` returns the index of the smallest element that occurs at or after index `k` in the array. If `k` is greater than or equal to the length of the array or less than 0, an `ArrayIndexOutOfBoundsException` will be thrown at runtime.

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 Below is another function called `mystery2`. It takes an array of integers called `inputArray` as an argument and returns nothing.

```java
public static void mystery2(int[] inputArray) {
    int index = 0;
    while (index < inputArray.length) {
        int targetIndex = mystery1(inputArray, index);
        int temp = inputArray[targetIndex];
        inputArray[targetIndex] = inputArray[index];
        inputArray[index] = temp;
        index = index + 1;
    }
}
```

Describe what `mystery2` will do and return if `inputArray` is the array {3, 0, 4, 6, 3}. Then, explain in English what the method `mystery2` does.

`mystery2` doesn't return anything because its return type is **`void`**.

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. (For those of you who are interested, `mystery2` performs selection sort.)

At the beginning of each iteration of the while loop, the first `index` elements of the array are in sorted order. Then the method `mystery1` 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 `mystery1` is then swapped with the element at position `index` so that the first `index + 1` elements of the array are in sorted order.

3 Fibonacci

Implement `fib1` recursively. `fib1` takes in an integer `N` and returns an integer representing the Nth Fibonacci number. The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, . . . , where 0 is the 0th Fibonacci number. As a reminder, the Nth Fibonacci number is calculated as follows:

```
fib(N) = fib(N - 1) + fib(N - 2)
```

```
public static int fib1(int N) {









}
```

Note: The solution assumes that N $\geq$ 0, which works because it does not make sense to have a Nth fibonacci number where N is negative (for example, the -1st fibonacci number does not exist).

```
public static int fib1(int N) {
    if (N <= 1) {
        return N;
    } else {
        return fib1(N - 1) + fib1(N - 2);
    }
}
```

Extra Implement `fib2` in 5 lines or fewer that avoids redundant computation. `fib2` takes in an integer `N` and helper arguments `k`, `f0`, and `f1` and returns an integer representing the Nth Fibonacci number. To compute the Nth fibonacci number, you should call `fib2(N, 0, 0, 1)`. If you're stuck, try implementing `fib1` iteratively and then see how you can transform your iterative approach to implement `fib2`.

```java
public static int fib2(int N, int k, int f0, int f1) {









}
```

```java
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);
    }
}
```