CS 61BL Summer 2026

Introduction to Java Programming

Exam Prep 01: June 26th, 2026

1 Our First Java Program

This exercise is adapted from Head First Java.

Consider the following implementation of a `Dog` class.

```
class Dog {
  public static int cuteness = 100;
  public Dog(String name, int size) {
    // Implementation not provided.
  }

  public void bark(int loudness) {
    // Implementation not provided.
  }

  public static void evolve(int newCuteness) {
    System.out.println("Becoming cuter...");
    cuteness = newCuteness;
  }
}
```

Next to each line below, 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. The `Dog` class is included below for your reference.

```
size = 27; /* Example answer: 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";
Dog myDog = new Dog(name, size);
Dog yourDog = new Dog("Scruffy", 1000);
Dog[] dogList = new Dog[3];
dogList[0] = myDog;
dogList[1] = yourDog;
dogList[2] = 5;
dogList[3] = new Dog("Cutie", 8);
int x;
x = size - 5;
if (x < 15) {
  myDog.bark(8);
}
myDog.evolve(200);
Dog.evolve(300);
Dog.bark(10);
```

```
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 spot 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()

```
myDog.bark(8); }
```

```
myDog.evolve(200);
```

Calls the static method evolve on the Dog class, changing the value of cuteness to 200.

```
Dog.evolve(300);
```

Calls the static method evolve on the Dog class, changing the value of cuteness to 300.

```
Dog.bark(10);
```

Errors. bark is an instance method, not a static method. To fix this error, you could either change the method to be static or call it on an instance of Dog.

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 would `mystery` return if `inputArray = [3, 0, 4, 6, 3]` and `k = 2`?

The method returns 4.

(b) Explain, in natural language, what `mystery` does.

It returns the index of the smallest element that occurs at or after index 1k 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.

(c) Use `mystery` to sort a given array of integers in ascending order. *Hint: Can you use what mystery returns to swap integers within the array?*

```
public static void sortArray(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;
   }
}
```

3 Two Sum

Given an array of numbers **nums**, find two numbers in **nums** such that adding them would result in the value **target**. Assume that only one such pair exists, and there is always a pair that fulfills the condition.

Return the numbers as an array with two terms.

You may not need all lines provided.

```
int[] twoSum(int[] nums, int target) {
  for (int i = 0; i < nums.length; i++) {
    for (int j = i + 1; j < nums.length; j++) {
      if (nums[i] + nums[j] == target) {
        return new int[]{nums[i], nums[j]};
      }
    }
  }
  return new int[]{-1, -1}; // No solution, technically wouldn't happen.
}
```

4 Reversals

In this problem, we are trying to implement a **reverseString** method. This method takes in a string and reverses it.

```
public static String reverseString(String s) {
  // Implementation to be filled in
}
```

(a) For each of the test cases we have written below, write down whether it is correct, and briefly explain why it is, or isn't useful.

Some questions to guide your thinking: Does the test case actually test the method **reverseString**? Does it test the method with a variety of inputs? Does it test edge cases?

```
assertThat("hello world").isEqualTo("dlrow olleh");
```

This wouldn't work because we didn't actually call the method reverseString. Key takeaway: always call the method you are trying to test!

```
assertThat(reverseString("Hello")).isEqualTo("olleH");
```

This works well and tests the reversal of a given word.

```
assertThat(reverseString(1234)).isEqualTo(4321);
```

This would give a compiler error: the method **reverseString** is supposed to take in a **String**, while an **int** is given.

```
assertThat(reverseString("@#_!")).isEqualTo("!_#@");
```

This works well and tests whether reversals work for special characters.

```
assertThat(reverseString("")).isEqualTo("");
```

This works well and tests whether reversals work for empty strings - a common edge case.

```
assertThat(reverseString("baaaa")).isEqualTo("aaaab");
```

This would also work as a test case, it tests the reversal of a given word.

(b) Consider the following implementation for **reverseString**

```
public static String reverseString(String s) {
  return s.substring(1) + s.charAt(0);
}
```

Here is an explanation of some features of the code, as well as an example: **substring(i)** returns a substring that begins with the character at the specified index and extends to the end of this string. **charAt(i)** returns the character at index **i**. Note that 0-indexing also applies to strings. The + sign concatenates the two things returned by the two function calls into a longer string.

If the index is out of bounds, then an **IndexOutOfBoundsException** is thrown, and the program errors.

```
s = "Circle";
s.substring(2); // Returns "rcle"
s.charAt(1); // Returns 'i'
s.substring(2) + s.charAt(1) // Returns "rclei"
s.substring(6) // Errors!
```

Some of the test cases from the previous part are replicated below. Write down what our method would return (could be an error), and determine whether each of these would pass for our implementation.

```
assertThat(reverseString("Hello")).isEqualTo("olleH");
```

This test would fail because our implementation would return "elloH".

```
assertThat(reverseString("")).isEqualTo("");
```

This test would fail because our implementation would be trying to access index 1 for a string of length 0, resulting in an IndexOutOfBoundsException.

```
assertThat(reverseString("baaaa")).isEqualTo("aaaab");
```

This test would pass because our implementation would return "aaaab", which is the same as the expected answer. The key takeaway from this is that buggy implementations sometimes won't be caught if your test cases are not comprehensive.

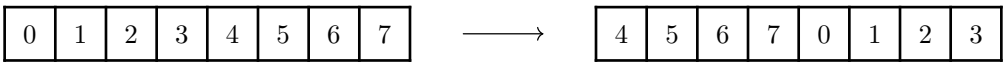
(c) Write a correct implementation for **reverseString**. You may not need all lines provided.

Hint: use **charAt(i)**, which takes in an integer and returns the character at that index. Remember that strings are 0-indexed, so the last character of a string **s** is **s.charAt(s.length() - 1)**.

```
public static String reverseString(String str) {
  String reversed = "";
  for (int i = str.length() - 1; i >= 0; i--) {
    reversed += str.charAt(i);
   }
  ___
  ___
  return reversed;
}
```

5 Rotate

Write a function that, when given an array **A** and integer **k**, returns a new array whose contents have been shifted **k** positions to the right, wrapping back around to index 0 if necessary. For example, if **A** contains the values 0 through 7 inclusive and **k = 12**, then the array returned after calling **rotate(A, k)** is shown below on the right:



k can be arbitrarily large or small - that is, **k** can be a positive or negative number. If **k** is negative, shift **k** positions to the left. After calling **rotate**, **A** should remain unchanged. (This means that **rotate** is **nondestructive!**)

Hint: you may find the modulo operator % useful. Note that the modulo of a negative number is still negative (i.e. (-11) % 8 = −3).

```
/** Returns a new array containing the elements of A shifted k positions to the right. */

public static int[] rotate(int[] A, int k) {

    int rightShift = k % A.length;

    if (rightShift < 0) {

        rightShift += A.length;
    }


    int[] newArr = new int[A.length];

    for (int i = 0; i < A.length; i++) {

        int newIndex = (i + rightShift) % A.length;

        newArr[newIndex] = A[i];
    }
    return newArr;
}
```