Name: ______________________________________

USC Student ID Number:________________________________

USC NetID (e.g., `ttrojan`):____________________________

CS 455 Midterm Exam 1
Fall 2023 [Bono]

Tuesday, Sept. 26, 2023

There are 6 problems on the exam, with 62 points total available. There are 10 pages to the exam (5 pages **double-sided**), including this one; make sure you have all of them. If you need additional space to write any answers or scratch work, pages 9 and 10 are left mostly blank for that purpose. If you use those pages for answers you just need to direct us to look there. ***Do not detach any pages from this exam.***

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name, USC Student ID, and USC username (a.k.a., NetID) at the top of the exam. Also put your NetID at the top right of the front side of each page of the exam (including the last page). Please read over the whole test before beginning. Good luck!

`Arrays` class (selected methods) :

Note: The `Arrays` class contains static methods that operate on arrays.

```
static String toString(int[] array)
```

Returns a string representation of the contents of the specified array. The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (a comma followed by a space). For example: `[3, 7, 2, 14]`

```
static int[] copyOf(int[] original, int newLength)
```

Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. For all indices that are valid in both the original array and the copy, the two arrays will contain identical values.

Problem 1 [6 points]

Consider the following static method that is supposed to return a `String` describing the weather, when given an outside temperature in Fahrenheit. (Approximately equivalent temperatures are also shown in Celsius for those of you who aren't used to Fahrenheit.) It doesn't always do the right thing.

```
public static String getWeather(int temp) {
  String weather = "";
  if (temp >= 90) { weather = "boiling"; }          // 90 is about 32 C
  if (temp < 90 && temp >= 80) { weather = "hot"; } // 80 is about 27 C
  if (temp < 80 && temp >= 70) { weather = "just right"; }// 70 is about 21 C
  if (temp < 70 && temp >= 60) {                   // 60 is about 16 C
     weather = "cool";
  }
  else {
     weather = "cold";
  }
  return weather;
}
```

Do not modify the code. **Show two example data values and the result of calling the method on each of them: the first one should be one where the existing method returns an incorrect weather description, and a second one such that the method returns an accurate weather description:**

	temp	return value of **getWeather(temp)**
1. (wrong)		
2. (right)		

Problem 2 [7 points]

Consider the following incomplete program to find the position of the *last* non-digit in a String read in from the user using Scanner's `next()` method. Here are some examples of input (shown in *italics*) and the corresponding output from runs of a completed version of the program:

Run 1

```
fox
Last non-digit: x
Position: 2
```

Run 2

```
3b4!3279
Last non-digit: !
Position: 3
```

Run 3

```
327
All digits
```

Complete the initialization and loop condition for the program (fill in the boxes below) so it works as described. Do not change anything else in the code. Note: The `charAt(index)` method used below allows you to access individual chars in a `String`. The indices are zero-based (similar to indices in arrays and ArrayLists.) `Character.isDigit(ch)` tells us whether char `ch` is a digit character or not.

```
public class LastNonDigit {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      String word = in.next();
      boolean found = false;
      int position = [                                    ];

      while ( [                                           ] ) {

         char ch = word.charAt(position);
         if (!Character.isDigit(ch)) {
            found = true;
         }
         else {
            position--;
         }
      }
      if (found) {
         System.out.println("Last non-Digit: " + word.charAt(position));
         System.out.println("Position: " + position);
      }
      else {
         System.out.println("All digits");
      }
   }
}
```

Problem 3 [10 pts. total]

Consider the following program (Note: `Arrays.toString` format appears on cover of exam):

```
public class Prob1 {
  public static void main(String[] args) {
    int j = 0;
    int[] arr = new int[3];
    foo(j, arr);
    System.out.println("main1: " + j + " "
                          + Arrays.toString(arr));
    j++;
    foo(j, arr);
    System.out.println("main2: " + j + " "
                          + Arrays.toString(arr));
  }
  private static void foo(int i, int[] vals) {
    i++;
    vals[i]++;
    System.out.println("foo: " + i + " "
                          + Arrays.toString(vals));
  }
}
```

Output of Program

```
foo:

main1:

foo:

main2:
```

Part A. In the space below, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all variables, objects, arrays, and their state as they have changed during the code sequence. You should identify the data in the first call to `foo` as follows: i_1, $vals_1$, and for the second call: i_2, $vals_2$. When a value gets updated, cross it out and show the new value next to the old one rather than drawing it again: e.g., you will only have one box labeled `arr` and one box labeled `j`.

Part B. In the box to the right of the code above, show the output of the program. We started each of the four lines of output for you.

Problem 4 [9 pts]

Consider the test program you wrote in assignment 1. It consisted of two classes:

(1) **`SpiralGeneratorTester`** has a main method plus some static helper methods. It uses another class, (2) **`SpiralGenerator`**.

For each of the following items, describe what part of the program *needs* to know about that item for completing the program, using the answer choices given (i.e., SGT, SG, both, or none). To clarify, by "part…needs to know…", we mean the code in that part has that information or needs to use that information and/or the programmer for that part needs to know that information to write that part of the program.

(SGT) only `SpiralGeneratorTester` needs to know about it

(SG) only `SpiralGenerator` needs to know about it

(both) both `SpiralGeneratorTester` and `SpiralGenerator` need to know about it

(none) neither `SpiralGeneratorTester` nor `SpiralGenerator` need to know about it

Items:

________ 1. data structure(s) used in a `SpiralGenerator` object

________ 2. the types of the parameters for methods of `SpiralGenerator`

________ 3. contents of failure messages, such as one reporting that the segment wasn't horizontal or vertical.

________ 4. preconditions on `SpiralGenerator` methods

________ 5. names of `SpiralGenerator` instance variables

________ 6. the class name `SpiralGenerator`

________ 7. names of methods of `SpiralGeneratorTester`

________ 8. the types of local variables in methods of `SpiralGenerator`

________ 9. the class name `SpiralGeneratorTester`

Problem 5 [15 points]

Complete the implementation of the class `Car`, defined on this and the next page and described further below.

Consider a `Car` class that keeps track of gas and miles driven, assuming a particular miles per gallon fuel efficiency for the car. You specify the MPG in the constructor; the car starts out with no gas in the tank. It will keep track of the fuel level and miles driven as the car gets driven (the `drive` method), or as gas gets put in the tank (the `addGas` method), and you can ask about the fuel level and the miles driven. If you try to drive farther than the amount of gas in the tank will allow, it will only go as far as it can before running out of gas. Below is an example of the use of this class:

```
public static void main(String[] args) {

   Car myCar = new Car(20);         // my car gets 20 miles per gallon (MPG)

   System.out.println(myCar.gasLeft());    // 0.0: gas tank starts out empty
   myCar.addGas(5.0);                      // put 5 gallons in the tank
   myCar.drive(80);                        // drive 80 miles (returns 80.0)
   System.out.println(myCar.gasLeft());    // 1.0 gallon left
   System.out.println(myCar.milesDriven());  // 80.0
   double distance = myCar.drive(50);      // attempt to drive 50 more miles
          // returns actual distance driven before running out of gas (20.0)

   System.out.println(myCar.gasLeft());      // 0.0
   System.out.println(myCar.milesDriven());  // 100.0
   distance = myCar.drive(10);           // returns 0.0 (already out of gas)
   myCar.addGas(2.0);
   myCar.drive(30);
   myCar.addGas(3.0);
   System.out.println(myCar.gasLeft());      // 3.5
   System.out.println(myCar.milesDriven());  // 130.0

   . . .
}
```

The class interface and method comments appear on this and the next page. We left space so you can add in your code to complete the class.

```
//   Car keeps track of gas and miles driven, assuming a particular miles
//   per gallon fuel efficiency for the car.
public class Car {







   //   Creates a car with the given miles per gallon (MPG).  It starts
   //   out with no fuel in the tank and no miles driven.
   //   PRE: milesPerGallon > 0
   public Car (int milesPerGallon) {






   }
```

Problem 5 (cont.)

```
    // Returns the amount of gas in the tank, in gallons
   public double gasLeft() {









   }

   // Returns the number of miles driven since this object was created.
   public double milesDriven() {









   }

   // Add the given amount of gas to the tank. (NOTE: no limit on gas tank size.)
   // PRECONDITION: gallons >= 0.0
   public void addGas(double gallons) {









   }

   // Attempt to drive the distance given in miles, returning the actual distance
   // driven as a result of this call.  If we run out of gas before going the
   //  specified distance, value returned will be less than miles.
   // PRECONDITION: miles > 0
   public double drive(int miles) {









   }
}
```

Problem 6 [15 points]

Implement the static method `rotateLeft`, which takes an array and a non-negative int *k*, and returns an array like the one passed in, but with the values rotated left by *k* positions. A rotation moves values that "fall out" of the front of the array around to the end of the array. ***The original array is unchanged by the method.*** Here are some examples (array indices shown on the first one):

	`arr:`	`k:`	return value of `rotateLeft(arr):`
	0 1 2 3 4 5 6		*0 1 2 3 4 5 6*
Ex1:	`1 2 3 4 5 6 7`	`3`	`4 5 6 7 1 2 3`
Ex2:	`1 2 3 4 5 6 7`	`10`	`4 5 6 7 1 2 3`
Ex3:	`1 2 3 4 5 6 7`	`1`	`2 3 4 5 6 7 1`
Ex4:	`12 2 9`	`0`	`12 2 9`
Ex5:	`12 2 9`	`3`	`12 2 9`

Note: Part of your score will be based on whether your solution performs more data movements than necessary (fewer is better). Hint: the % (modulus) operator may be helpful. You may not use any Java classes apart from the `Arrays` class.

```
//   Returns an array with the same values as nums, but with the values
//   rotated left by k positions.  nums is unchanged by the method.
//   PRE: nums.length > 0 and k >= 0
public static int[] rotateLeft(int[] nums, int k) {
```

Extra space for answers or scratch work.

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.