Super Dice Roller v2!

For this project, you will extend concepts used in the dice rolling labs to create an application that creates random values in an array and then performs several common array operations.

Specifications

1. Create a java file that contains a main method, along with the following methods.
    a. **void printWelcomeMessage()** - Prints your message header including project and name.
    b. **int promptForInteger(String prompt)** - Prompts the user for an integer, collects it from the user, and returns that value.
    c. **void randomizeArray(int[] intArray, int numSides)** - Takes a list of integers and rolls a numSides sided die, storing a random value at each index in the array.
    d. **void printArray(int[] intArray)** - Prints the contents of an integer array with a space in between each value.
    e. **int linearSearch(int[] intArray, int key)** - Looks through an integer array for the first occurrence of a search value and returns its index, or, a -1 if not found.
    f. **int binarySearch(int[] intArray, int key)** - Looks through a *sorted* integer array for an occurrence of a search integer and returns its index, or, a negative index if not found.
    g. **void bubbleSort(int[] intArray)** - Sorts an integer array of any size using the bubble sort algorithm.
    h. **int findSum(int[] intArray)** - Returns the sum of all the integers in an array
    i. **int findMax(int[] intArray)** - Returns the largest integer in an array.
    j. **int findMin(int[] intArray)** - Returns the smallest integer in an array.
    k. **double findAverage(int[] intArray)** - Returns the average of all the integers in an array.
    l. **double findMedian(int[] intArray)** - Returns the median of all the integer in a *sorted* array.
2. The main method will invoke the methods above to implement the following functionality.
    a. Prompt the user for the number of dice to roll (if the user enters 0 or a negative number skip the following instructions and go to step h. below)
    b. Prompt the user to enter the number of sides on each dice
    c. Create an array of size *number of dice to roll* and randomize each value in the array by rolling the appropriate sided die as requested by the user.
    d. Display all the values in the array, sort the array, and then display the values again.
    e. Display the sum, maximum, minimum, average, and median value for the array of values.
    f. Prompt the user for a value to search for in the array.
    g. Perform a linear search and a binary search in the array for the value specified and display an appropriate message for the result. Keep in mind, the algorithms could return different values if there are repeated values in your array.
    h. Loop back to step a. unless the user has entered a 0 or negative number, in this case break the loop.

```
Super Dice Roller 2

Enter the number of dice to roll (0 to exit): 12
Enter the number of sides on your dice: 6
Rolling 12d6:  4 5 2 4 6 1 6 1 1 4 2 5
Sorting Array:  1 1 1 2 2 4 4 4 5 5 6 6
       Sum: 41
   Maximum: 6
   Minimum: 1
   Average: 3.42
    Median: 4.00
Enter an integer to search for: 1
   Linear search: 1 found at index 0
   Binary search: 1 found at index 2

Enter the number of dice to roll (0 to exit): 9
Enter the number of sides on your dice: 20
Rolling 9d20:  19 3 18 19 10 4 1 5 5
Sorting Array:  1 3 4 5 5 10 18 19 19
       Sum: 84
   Maximum: 19
   Minimum: 1
   Average: 9.33
    Median: 5.00
Enter an integer to search for: 6
   Linear search: 6 not found
   Binary search: 6 not found

Enter the number of dice to roll (0 to exit): 0

Process finished with exit code 0
```

3. Notes
    a. Your code must implement the methods above as specified. This means return types, and argument/parameter names must match the function signature exactly as requested above. Any deviations will result in a loss of points.
    b. Although some of these methods have been provided in the textbook or in class, they will need to be adapted to fit the specifications of this project.
4. All user prompts, messages, and results must be neat, appropriately organized, and easily understood by the user. Your output must match mine as closely as possible.
5. Test your code thoroughly to ensure you are getting the correct results for each array operation including error conditions and exit conditions.
6. Make sure your code has the required documentation, as outlined in the CS Java Documentation Policy under Course Info on D2L. You must comment your methods as defined in this policy.

7. This is an individual assignment. By submitting your work to D2L, you acknowledge you have read the NESCC Computer and Information Science Department's Honor Code and Documentation Policy and are following its policies to the best of your ability.

Deliverables

Make sure your code has the required documentation, as outlined in the CISP Java Documentation Policy on the course website.

This is an individual assignment. By submitting your work to D2L, you acknowledge you have read the NESCC Computer and Information Science Department's Honor Code and Documentation Policy and are following its policies to the best of your ability.

Once you are satisfied with your code, compress your src directory with a **.zip** file and upload it the Project 3 D2L drop box. Your .**zip** file should contain the following. Make sure your code is in the following package structure **edu.northeaststate.cs1.projects.project3**.

1. Your source code under the following directory structure: **src\edu\northeaststate\cs1\projects\project3**. Please **do not** include IDE project files such as .iml, .idea, or the out directory. In your .**zip** file, please prune off any unneeded packages such as examples, labs, or other projects.
2. Provide a citation document in Word .**docx** format with links or a write-up if you utilized any outside resources to complete your assignment.

Evaluation

Five factors will be considered in grading your project:

1. **Compiles** (10%): does the Java code compile with no errors?
2. **User Interface** (10%): does the program interact with the user as expected?
3. **Design** (40%): does the code meet functionality requirements and design specifications?
4. **Deliverables** (30%): are all deliverables included, named, commented, and organized appropriately?
5. **Standards** (10%): does the code follow good programming practices and coding standards?