Writing Methods

For each exercise below, write the method described. Give all of the methods public visibility. Assume all ranges are inclusive (include both end points).

1. Write a method called **powersOfTwo** that prints the first 10 powers of 2 (starting with 2). The method takes no parameters and doesn't return anything.

2. Write a method called **alarm** that prints the word "Alarm!" multiple times on separate lines. The method should accept an integer parameter that specifies how many times the output line is printed.

3. Write a method called **sum100** that returns the sum of the integers from 1 to 100.

4. Write a method called **sumRange** that accepts two integer parameters that represent a range. You may assume the first parameter is less than or equal to the second. The method should return the sum of the integers in that range.

5. Write a method called **maxOfTwo** that accepts two integer parameters and returns the larger of the two.

6. Write a method called **larger** that accepts two floating point parameters (of type `double`) and returns true if the first parameter is greater than the second, and false otherwise.

7. Write a method called **countA** that accepts a `String` parameter and returns the number of times the letter 'A' is found in the string.

8. Write a method called **evenlyDivisible** that accepts two integer parameters and returns true if the first parameter is evenly divisible by the second, or vise versa, and false otherwise. You may assume that neither parameter is zero.

9. Write a method called **average** that accepts three integer parameters and returns their average as a floating point value.

10. Overload the **average** method of the previous exercise such that if four integers are provided as parameters, the method returns the average of all four.

11. Overload the **average** method once more to accept five integer parameters and return their average.

12. Write a method called **multiConcat** that takes a `String` and an integer as parameters, and returns a `String` that is the parameter string concatenated with itself n number of times (where n is the second parameter). For example, if the parameters are "hi" and 4, the return value is "hihihihi".

13. Overload the **multiConcat** method from the previous example such that if the integer parameter is not provided, the method returns the string concatenated with itself. For example, if the parameter is "test" the return value is "testtest".

14. Write a method called **isAlpha** that accepts a character parameter and returns true if that character is either an uppercase or lowercase alphabetic letter.

15. Write a method called **validate** that accepts three integer parameters. The first two parameters represent a range, and the purpose of the method is to verify that the value of the third parameter is in that range. You may assume that the first parameter is less than or equal to the second. If the third parameter is not in the specified range, the method should prompt the user and read a new value. This new value should be tested for validity as well. The method should only return to the calling method once a valid value has been obtained, and it should return the valid value.

16. Write a method called **floatEquals** that accepts three floating point values as parameters. The method should return true if the first two parameters are essentially equal, within the tolerance of the third parameter.

17. Write a method called **reverse** that accepts a `String` as a parameter and returns a `String` that contains the characters of the parameter in reverse order. Note: there is actually a method in the `String` class that performs this operation, but for the sake of this exercise you will write your own.

18. Write a method called **isIsoceles** that accepts three integer parameters that represent the lengths of the sides of a triangle. The method should return true if the triangle is isosceles but not equilateral, meaning that exactly two of the sides have an equal length, and false otherwise.

19. Write a method called **randomInRange** that accepts two integer parameters representing a range. You may assume that the first parameter is less than or equal to the second, and that both are positive. The method should return a random integer in the specified range.

20. Overload the **randomInRange** method of the previous exercise such that if only one parameter is provided, the range is assumed to be from 1 to that value. You may assume the parameter value is positive.

21. Write a method called **randomColor** that creates and returns a `Color` object that represents a random color. Recall that a `Color` object can be defined by three integer values between 0 and 255 representing the contributions of red, green, and blue (its RGB value).

22. Write a method called **darken** that accepts a `Color` object as a parameter and returns a new Color object that is "darker" than the parameter. Create the darker color using RGB values that are 10 percent less than the original.

23. Write a method called **drawCircle** that draws a circle based on the method's parameters: a `Graphics` object through which to draw the circle, two integer values that define the (x, y) coordinate of the center of the circle, another integer that represents the circle's radius, and a `Color` object that defines the circle's color. The method does not return anything.

24. Overload the **drawCircle** method of the previous exercise such that if the `Color` parameter is not provided, the circle's color will default to black.

25. Overload the **drawCircle** method again such that if the radius is not provided, a random radius in the range 10 to 100 will be used.

26. Overload the **drawCircle** method yet again such that if both the color and radius of the circle are not provided, the color will default to red and the radius will default to 40.