Java Tasks

Each task should be created as a class. I.e. Task01, Task02 unless a different name is specified. All classes should be created in Java project. Choose appropriate project name (e.g. Tasks) and use packages(e.g. *lv.rcs.tasks*).

As always see Java API doc. For help:

String class:
https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Java Task 05

Create class ***HelloUser***, which

1) Asks user for first name and last name in a **single** line, separated by space
2) Split first name and last name in two variables.
3) Modify first name to start with capital letter, followed by all small letters
4) Modify last name to be in uppercase
5) Print name and surname to console in single quotes(') separated by single space

e.g. ‘joHn LeMOn’ -> 'John LEMON'

Bonus:

1) Add handling for leading/trailing spaces
2) Add handling for 2 last names. In this case use dash as separator when printing them out.
3) Error handling. I.e. if user enters just one string(consider it as first name)
4) Add handling for 2 and more last names(cycle needed here)

e.g. ‘ jOhN lemOn HooK ’ -> ‘John LEMON-HOOK’

Java Task 06

Create class ***DateValidator*** which asks user to enter date, month(year) all in separate lines.

1) Check if the date is valid for entered month .e.g date = 31 and month = 4 is not a valid combination
2) In case of invalid combination print out error message
3) In case of valid combination print out date and month in a format that the month is formated as text.

e.g:

User enters date as 5 and month as 2 and year as 1999

‘5. February, 1999’

Bonus: Add handling for leap years. So that 29. February of 2018 is not valid.

Bonus2: If entered year is negative, format date as “5. February, 326 BC”

Java Task 07

Create class ***GuessNumber*** which implements a game of number guessing

1) Generate a random number between 1 and 10
2) Ask user to enter their guess
3) Generate a message in case user guesses number or ask user to guess again if the answer is wrong.

Bonus: Add handling for invalid input-check if a valid number is entered. Add feedback mechanism which notifys user in case of wrong guess in which direction the number is.

Hint: to generate random number, use this : *int r = new Random().nextInt(10) + 1;* Don't forget to perform import Random class before class declaration, e.g. *import java.util.Random;*

Java Task 08

Create program in class ***SortArray***, which asks user to enter size of array to be generated.
Create an array of *int* with the size specified by user input

1) Fill array with random numbers from 0 to 99 (use hint from Task 7)
2) Print the contents of the array on the screen separated by comma and space (3, 6, 1 etc)
3) Implement a method which sorts all elements in current array in ascending order. Do no use built in Sort() method in Array class(still can be used for checking correctness during development) or other cheats.
4) Print the contents of the array on the screen(now it should be sorted)

Java Task 09

Create program in class ***PrimeGenerator*** which

1) Asks user how many primes to generate(starting from 2 and up)
2) Create an array of specified size and fill with prime numbers
3) Print all generated primes comma separated on the screen

e.g.

input = 10,
Output 2, 3, 5, 7, 11, 13, 17, 19, 23, 29

https://en.wikipedia.org/wiki/Prime_number

Java Task 10

Create program in class CalculatePi, where Pi value should be calculated by formula :

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} + \cdots\right)$$

Tips :

- Sequence preferable length should not be less than 10000
- To perform int to int division and getting double type result, use this construction :

a / (double) b;

Java Task 11

Create class PrimeTest, that tells if entered (or defined) is prime number. Example :

7 – is prime number; 24 – is not prime number

Java Task 12

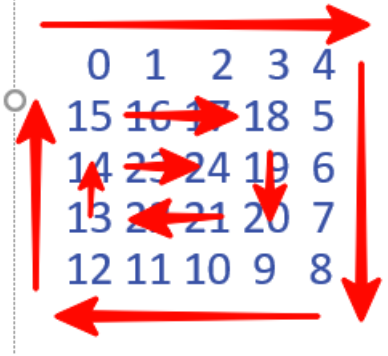
Create class named AwesomeList, that acts like a usual Java collection List implementation, and can do following things (only with integers) :

1. add(<n>) - adds <n> (n – simple int number) to list
2. get(<i>) – returns stored value by index <i>
3. printSelf() – prints stored content
4. Bonus :
    1. remove(<i>) – removes element by index <i>
    2. addAll(<>) – adds all elements from given collection, where collection may be:
        1. Another AwesomeList instance
        2. int[]
    3. add(<n>, <i>) – adds element at specified index
    4. size() – returns current stored content (number) count
    5. equals(<AL>) – returns fact that comparing AwesomeList has same contents

Extra Achievement : Array Master

Create class Spiral, that takes user input of 2 natural numbers (n and m), both strictly below 10, and generates matrix (2d array) of size n x m (respectively horizonal and vertical sizes). Contents of matrix should consist of sequential numbers from 0 to ((n * m) – 1), aligned in “spiral”.
Example for 5 x 5 matrix :

```
 0  1  2  3  4
15 16 17 18  5
14 23 24 19  6
13 22 21 20  7
12 11 10  9  8
```



Hint : for rounding up a floating numer, e.g. 2.5 -> 3, use **Math.ceil(** 2.5 **)**