Assignment 1

Grading
Your submission will receive a grade from A to F where F is Failed.
You are allowed to improve your work after the initial submission before the deadline.
Please note that higher grades are subject to the following criteria:
i) correct functionality, ii) quality of the solution, iii) quality of code documentation, code structure, and organisation. You may use the attached coding conventions document as a guideline to document your code. **Don't forget to submit your assignments on time!**

The grade is final, i.e., you will not get an opportunity to correct/improve after grading.

Your answers should be your own! You are not allowed to copy code from other students, books articles, blogs, wikis or any other source! Each submission will pass through a plagiarism/clone detection system before correction. **If plagiarism is detected, the assignment will be failed and a formal investigation will be initiated.**

Help
Charilaos Skandylas is the teaching assistant to help you with problems related to the practical assignments. Don't hesitate to contact him if you need help. He will be present during laboratory lessons, see the time plan, to assist you in the assignments' tasks. Moreover, you may use the slack channel for help and questions related to practical assignments.

Preliminaries
1. We recommend using Eclipse but you are free to use any IDE or tool of your choice to write Java code.
2. **Setup Eclipse Workspace**
   - Create an Eclipse *workspace* (a folder) with the name `java_courses` on some location in your home directory.
   - Create a *Java project* with the name `2DV600` inside the workspace.
   - Create a *package* with the name `YourLnuUserName_assign1` inside the project. For example, it might look something like `wo222ab_assign1`.
   - Save all program files from the exercises in this assignment inside the package `YourLnuUserName_assign1`.
   - In the future, create a new package (`YourLnuUserName_assignX`) for each assignment and a new project (with the course code as name) for each new course using Java.

Submission: We require following two files for submission:
1. A **zip file** archiving all your `.java` (source) files.
2. A **report** document in doc, docx or pdf format. The report must contain the Java source code for all your solutions and other details required for the solution. Images are not

allowed to show the code. The report must be submitted as a **stand-alone file, i.e., it must not be compressed as zip or any other file format.**

3. Separate submission of both the **report** and the **zip file** is mandatory to pass the assignment, **otherwise, a submission will get a grade "F"**.
4. Submissions through Email or any way other than Moodle are not accepted.

Exercises

1. An ISBN-10 (International Standard Book Number) consists of 10 digits: $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}$. The last digit, d10, is a checksum, which is calculated from the other nine digits using the following formula:
   (d1 * 1 + d2 * 2 + d3 * 3 + d4 * 4 + d5 * 5 + d6 * 6 + d7 * 7 + d8 * 8 + d9 * 9) % 11

   Note: % represents the modulus operator

   If the checksum is 10, the last digit is denoted as X according to the ISBN-10 convention.

   Write a program **ISBN.java** that prompts the user to enter the first 9 digits and displays the 10-digit ISBN (including leading zeros). Your program must read the input as an integer. Below are two sample runs of the program:

   Run 1:
   ```
   Enter the first 9 digits of an ISBN as integer: 013601267
   The ISBN-10 number is: 0136012671
   ```

   Run 2:
   ```
   Enter the first 9 digits of an ISBN as integer: 013031997
   The ISBN-10 number is 013031997X
   ```

2. An Armstrong number is an n-digit number that equals the sum of the nth power of its digits. For example 153 is a three-digit number where the sum of the cubes of the individual digits (1 + 125 + 27) equals the number itself (153).
   Write a program **ArmstrongNumber.java** that prompts user to enter a range for Armstrong numbers. The range is entered by asking user to enter a *starting* and an *ending* number for the range. The program then computes and prints Armstrong numbers, if any, in the entered range. Next, the program should prompt for a new range until the user decides that she or he is through. Use variables of the type integer to store the start and end numbers of the range. Below is an example of the program execution:

   ```
   Enter the starting number of the range :100
   Enter the ending number of the range :1000

   The  Armstrong numbers between the given range are :
   153
   370
   371
   407
   Do you want to repeat? (Y/N): Y
   ```

```
Enter the starting number of the range :200
Enter the ending number of the range :300

The  Armstrong numbers between the given range are :
Do you want to repeat? (Y/N) : N
```

3. The Babylonian algorithm to compute square root of a positive number *n* is as follows:
   1. Make a guess at the answer (you can pick n/2 as your initial guess).
   2. Compute r = n / guess.
   3. Set guess = (guess + r) / 2.
   4. Go back to step 2 until the last two guess values are within 1% of each other.

   Write a program **SquareRoot.java** that prompts user to enter an integer value for *n*, iterates through the Babylonian algorithm until the guess is within 1% of the previous guess and outputs the answer as a real number to two decimal places.

   Below is an example of the program execution:

```
This program estimate square roots.
Enter an integer to estimate the square root of: 25
Current guess: 7.25
Current guess: 5.349137931034482
Current guess: 5.011394106532552
Current guess: 5.000012953048684

The estimated square root of 25 is   5.00
```

4. Write a program **TextProcessor.java** that reads a line of text from the keyboard and then prints the text after making following changes in the text:
   i. Replace every alphabetic letter in the text with the letter following it in the alphabet (i.e., replace a with b, b with c, … and z with a.)
   ii. Capitalize every vowel (a, e, i, o, u) in the output
   iii. Characters other than letters (alphabets) remain unchanged.

   An execution might look like this:
   Type a line of text: Hi! I'm Nadeem, 38 years old.
   After Processing: Ij! J'n ObEffn, 38 zfbst pmE.

5. Write a program named **Anagram.java** that reads a word from the keyboard and then prints all anagrams of that word contained in the ***wordlist*** file provided with this assignment. An anagram is a word formed by rearranging the letters of a different word using all original letters exactly once, see https://en.wikipedia.org/wiki/Anagram for details.

   For example, the word eat would produce the following output:
   ate, eat, eta, tea
   While the word master would produce:
   master, maters, stream, tamers

6. Create a program named **Codestrip.java** that reads a java file and prints its contents after removing blank lines and comments. Your program should finally print the following statistics: number of actual lines of code, number of blank lines and comments removed.
   Note: Java supports three types of comments:
   i) single line comments, starting with //
   ii) multiline comments, starting with /* and ending with */
   iii)javadoc comments that start with /** and end with */

   For example, for following input file:
```
/**
* A simple hello world class
*/
public class HelloWorld{

        /*
        This is the main method
        */
        public static void main(String []args){
                //prints hello world
                System.out.println("Hello World");
        }

}
```

   The output shall be:
```
public class HelloWorld{
        public static void main(String []args){
                System.out.println("Hello World");
        }
}

Number of actual lines of code: 5
Number of blank lines removed: 2
Number of comments removed: 3
```

7. Create a class Point that when executed using this code:
```
Point p1 = new Point();
Point p2 = new Point(3,4);

System.out.println(p1.toString());   // ==> (0,0)
System.out.println(p2.toString());   // ==> (3,4)

if (p1.isEqualTo(p2))           // False!
        System.out.println("The two points are equal");

double dist = p1.distanceTo(p2);
```

```
System.out.println("Point Distance: "+dist);

p2.move(5,-2);        // ==> (8,2)
p1.moveToXY(8,2);     // ==> (8,2)

if (p1.isEqualTo(p2))            // True!
        System.out.println("The two points are equal");
```

results in the following console print-out:

```
(0,0)
(3,4)
Point Distance: 5.0
The two points are equal
```

The class Point should of course be able to handle other points with different (x,y) values.

- The coordinates (x,y) are always integers.
- The method toString returns a string with coordinates suitable for print-outs.
- Distance between two points (x,y) and (p,q) is computed as Sqrt((x-p)^2 + (y-q)^2).
- Two points are *equal* if they have the same coordinates.
- Method move moves the point certain steps in x- and y-direction.
- Method moveToXY provide a new set of coordinates.

8. Create a class called **Invoice** that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables:
   - a part number (type String),
   - a part description (type String),
   - a quantity of the item being purchased (type int)
   - a price per item (double)

   The class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named **InvoiceTest** that demonstrates class Invoice's capabilities.

9. Create a Java class **Time** to represent time by hours, minutes, and seconds. The Time class must have the following features:
   - Three instance variables for the hours (range 0 - 23), minutes (range 0 - 59), and seconds (range 0 - 59)
   - Three constructors:
     - default (with no parameters passed; is should initialize the represented time to 0:0:0)
     - a constructor with three parameters: hours, minutes, and seconds.
     - a constructor with one parameter: the value of time in seconds (seconds should be converted to the time value in hours, minutes, and seconds)

- a set-method method setClock() with one parameter seconds (seconds should be converted to the time value in hours, minutes, and seconds).
- get-methods getHours(), getMinutes(), getSeconds() with no parameters that return the values of the corresponding instance variables.
- set-methods setHours(), setMinutes(), setSeconds() with one parameter each that set values of the corresponding instance variables.
- method tick() with no parameters that increments the time stored in a Time object by one second.
- method tickDown() which decrements the time stored in a Time object by one second.
- method addTime() accepting an object of type Time as a parameter. The method should add the time represented by the Time object passed as input parameter to the time represented by this Time object on which the method addTime() is invoked. The sum of the two times should be returned as a Time object, leaving the time represented by the current Time object unchanged.
- Add an instance method subtracTime() that takes one Time parameter and returns the difference between the time represented in the current Time object and the one represented by the Time parameter. Difference of time should be returned as a Time object.
- Add an instance method toString() with no parameters to your class. toString() must return a String representation of the Clock object in the form "hh:mm:ss", for example "08:10:34".

Write a separate class TimeDemo with a main() method. The program should:

- instantiate a Time object timeA using an integer value obtained from the keyboard. The integer value represents seconds passed since midnight.
- tick the clock ten times by applying its tick() method and print out the time after each tick.
- Extend your code by appending to it instructions instantiating a Time object timeB by using three integers (hours, minutes, seconds) read from the keyboard.
- Then tick the clock ten times, printing the time after each tick.
- Add the timeB time in timeA by calling method addTime method and print sum of the time returned by the addTime method.
- Create a reference timeC that should reference to object of difference of timeA and timeB by calling the method subtracTime (), print the time represented by the reference timeC.

10. Write a class **Arrays .java** with the following static methods.
    - Method int average(int[] arr) that takes an integer array as parameter and returns average of the elements in the array arr.
    - Method int max(int[] arr) that takes an integer array as parameter and returns the array element with maximum value in the array “arr”.
    - Method int[] addN(int[] arr, int n) that returns, an array where we have added the number n to all elements in the array arr. For example, in the case addN({1,2,3,4,5}, 2) the result is an array {3,4,5,6,7}. The array gets changed.
    - Method int[] reverse(int[] arr) that creates, and returns, a new array with all the elements in array arr but in reverse order. The array arr should be left unchanged.
    - Method void replaceAll(int[] arr, int old, int nw) that replaces all occurences of the element old with nw in arr.

- Method int[] sort(int[] arr) that returns a new sorted array (least element first) with the same set of elements as in arr. The array arr should be left unchanged.
- Method boolean hasSubString(int[] arr, int[] sub) that returns true if the array sub is a part of the array arr, otherwise false. For example, in the case hasSubString({1,2,3,4,5}, {3,4,5}) the result is true since {3,4,5} is a part of {1,2,3,4,5}, but in case hasSubString({1,2,3,4,5}, {2,3,5}) the result is false.
- Method int[] absDif(int[] arr1, int[] arr2) returns a new array that is the absolute difference between array arr1 and array arr2. That is result array dist(i) = |arr1(i) - arr2(i)|. For example, in the case absDif({1,2,3,4,5}, {1,1,1,1,1}) the result is {0,1,2,3,4}. Notice, that the sizes of both arrays should be the same, in case they are different the method should throw an exception and be handled within the program.

Write also a program ArraysMain.java that demonstrates how all these methods work.