Exercise 1: Løs nedenstående opgave. Det er *Exercise 1* fra bogens kapitel 7.

Write a class `Arrays1` with a method called

```
public static int lastIndexOf(int[] a, int v)
```

that accepts an array of integers and an integer value as its parameters and returns the last index at which the value occurs in the array. The method should return `-1` if the value is not found. For example, in the array `[74, 85, 102, 99, 101, 85, 56]`, the last index of the value `85` is `5`.

End of Exercise 1

Exercise 2: Løs nedenstående opgave. Det er *Exercise 2* fra bogens kapitel 7.

Write a method class `Arrays2` with a method

```
public static int countInRange(int[] a, int min, int max)
```

that accepts an array of integers, a minimum value, and a maximum value as parameters and returns the count of how many elements from the array fall between the minimum and maximum (inclusive). For example, in the array `[14, 1, 22, 17, 36, 7, 43, 5]`, there are four elements whose values fall between `4` and `17`.

End of Exercise 2

Exercise 3: Løs nedenstående opgave. Det er *Exercise 5* fra bogens kapitel 7. Du kan eventuelt springe spørgsmål b) over i første omgang. *Hint til spørgsmål a)*: Se på programmet `Tally` i bogen. *Hint til spørgsmål b)*: Benyt metoden `sort` fra klassen `Arrays`, som ligger i pakken `java.util`. Kommandoen `Arrays.sort(arr)` sorterer elementerne i arrayet `arr`.

Write a class `Arrays3` with the following methods

a) Write a method

```
public static int mode(int[] a)
```

that returns the most frequently occurring element of an array of integers. Assume that the array has at least one element and that every element in the array has a value between `0` and `100` inclusive. Break ties by choosing the lower value. For example, if the array passed contains the values `[27, 15, 15, 11, 27]`, your method should return `15`.

b) Write a version that does not rely on the values being between `0` and `100`. Implement this in method `modeX`.

End of Exercise 3

Exercise 4: Under denne uges fildeling ligger der en fil `vejretDK.txt` som indeholder klimadata for Danmark i perioden 1874 til 2005. Filen indeholder for hver måned en angivelse af gennemsnitstemperaturen i denne måned, angivet som kommatal.
Lav et program `Klimadata` som indlæser navnet på en måned fra konsollen, og dernæst ved hjælp af filen `vejretDK.txt` udfører følgende:

- Printer årstal for laveste gennemsnitstemperatur i den pågældende måned.

- Printer årstal for højeste gennemsnitstemperatur i den pågældende måned
- Grafisk printer fordelingen af gennemsnitstemperaturer i den pågældende måned (afrundet til nærmeste heltal).

Eksempel på programudførsel:

```
Indtast måned: Oktober
Laveste gennemsnitstemperatur for oktober måned: 5.2 grader i 1905.
Højeste gennemsnitstemperatur for oktober måned: 12.0 grader i 2001.
Temperaturfordeling for oktober måned i perioden 1874 til 2005:
5:  2   **
6:  7   *******
7:  20  ********************
8:  26  **************************
9:  42  ******************************************
10: 25  *************************
11: 9   *********
12: 1   *
```

End of Exercise 4

Exercise 5: Løs nedenstående opgave. Det er *Exercise 13* fra bogens kapitel 7.

Write a class `ArrayCompare` with a method called `equals` that accepts two two-dimensional arrays of integers as parameters and returns `true` if the arrays contain the same elements in the same order. If the arrays are not the same length in either dimension, it should return `false`.

End of Exercise 5

Exercise 6: Lav et program der indlæser en fil med tal, gemmer tallene i et array, og beregner gennemsnit. Hint: man kan benytte følgende skelet:

```
public static double gns(String filnavn) throws IOException {
  Path path = Paths.get(filnavn);
  int numLines = (int)Files.lines(path).count();
  Scanner filScanner = new Scanner(new FileReader(filnavn));
  //lav array, beregn ting.
  return ??;
}
```

b) Lav en metode, der returnerer det største tal, som er mindre end gennemsnittet.

End of Exercise 6

Exercise 7: I denne opgave implementerer vi en sorteringsalgoritme ved navn bubblesort. CodeJudge forventer at metoderne ligger i en klasse ved navn `BubbleSort`.

a) Lav en metode `naboSwap`, der tager et array og et index i som input, og bytter om på arrayets indhold på plads nr. i og plads nr. $i + 1$.

```
public static void swap(int[] array, int i){
        //bytter indholdet af plads i og plads i+1.
}
```

Dvs. f.eks. arrayet `mitArray`:
`[11,16,12,19]`

bliver med `naboSwap(mitArray, 1)` til:
`[11,12,16,19]`

Da `11` befinder sig på plads nr. 0, `12` befinder sig på plads nr. 1, `16` er på plads nr. 2, og `19` er på plads nr. 3.

b) Lav en metode (bubble), som tager et array, og tager et positivt tal c som er mindre end arrayets længde, og modificerer arrayet som følger:

Ved hjælp af flere kald til metoden `naboSwap` flyttes det *største* af arrayets c første tal op, så det står på plads nr. $c-1$.

Hint: sammenlign plads i og $i+1$ startende med $i=0$, og kald `swap` hvis de kommer i den gale rækkefølge.

c) Lav en klasse `BubbleSort` med metoden
```
public static void sort(int[] a)
```

Metoden sorterer arrayet i stigende orden, ved gentagende gange at kalde metoden `bubble`.

Først sørger den for, at arrayets største element er på plads på arrayets sidste plads, så at de to sidste er på de korrekte pladser, så de tre sidste, osv, ind til den er gennem hele arrayet.

Programmet bør printe arrayet ved hjælp af `Arrays.toString` både før start, og efter hver runde.

For eksempel, for arrayet $\{5,6,3,2,1\}$ får vi:
```
[5, 6, 3, 2, 1]
[5, 3, 2, 1, 6]
[3, 2, 1, 5, 6]
[2, 1, 3, 5, 6]
[1, 2, 3, 5, 6]
```

c) $(*)$ Tilføj metoden
```
public static void sort2(int[] a)
```

som undgår unødvendige runder.

Remark. The name *bubblesort* is motivated as follows: the large entries in the array ascend to the end of the array like bubbles in a glass of champagne.

End of Exercise 7

Exercise 8: Lav en metode, der tager et array som input, og som outputter `true` hvis arrayet er sorteret i stigende rækkefølge, og `false` hvis det ikke er.

End of Exercise 8

Exercise 9: [*] Write a program `FlexArray` that fetches integers from standard input, one or more at a time. The numbers are stored in a single array, starting from the first position of the array. The array is initialized with length 5. The program has to be able to store any number of integers. Whenever the number of integers which have been read is grater than 0 and divisible by 3, the all these integers are printed in a line, separated by commas.

End of Exercise 9

Exercise 10: [*] Write a program that reads the file `Integers.txt`. The file contains integers, separated by blanks or new-line. The integers in the file are between a and b, $a < b$. These two number are the only ones in the first line of the file. For the remaining numbers in the files, compute how often each one occurs in the file.

End of Exercise 10

Exercise 11: [**] Write a class `Adders` which implements:

a) a full adder as a method, using the gates implemented in the class `Gates`. If you did not solve the corresponding exercise, use the class `Gates` from Campusnet.

```
public static ????? fullAdder(boolean a1, boolean a2, boolean carry)
```

The method as three booleans as input arguments and returns two booleans (find a way how to do this). A definition of a full adder can be found, for example, at `http://en.wikipedia.org/wiki/Adder_%28electronics%29#Full_adder`

b) a three-bit adder, that is, a method which takes two three bit numbers as input and returns their sum (a four-bit number). Use the full adders from part a).

End of Exercise 11