Problem 1

Write a function **selSort** which sorts a given array of **int**s (orders its elements from the smallest to the greatest) using *selection sort* algorithm. Namely, we iterate over positions (indices) of the array from index 0 to the last but one and for each index (say, i) we find the *index* of the smallest element among those with indices larger than i (i.e., to the right of the i-th element). If the element found is smaller than the i-th, we swap them. Note that the number of swaps will be at most $n-1$ where n is the size of the array (see Problem **??**).

Write two static functions, both taking a *two-dimensional* array of **int**s, which

- **sortRows** — sorts separately each "row" of the array (note that the array need not be rectangular);
- **sortCols** — sorts separately each "column" of the array (this, of course, can be done only for rectangular arrays).

In both cases you can use previously defined **SelSort** function, although in the second case it would be better to implement sorting inside the **SortCols** function.

Write also a function which prints a two-dimensional array on the console.

For example, the following program

download *Arr2DSelSort.java*

```
public class Arr2DSelSort {
    public static void selSort(int[] arr) { ... }
    public static void sortRows(int[][] arr) { ... }
    public static void sortCols(int[][] arr) { ... }
    public static void printArr2D(int[][] arr) { ... }

    public static void main (String[] args) {
       int[][] a = { {3,2,6,1,3,5,6,1,3},
                     {3,1,2,1,5,7,2},
                     {8,9,2,1} };
       System.out.println("Before:");
       printArr2D(a);
       sortRows(a);
       System.out.println("After:");
       printArr2D(a);

       int[][] b = { {3,2,6,1,6},
                     {7,1,2,1,5},
                     {5,3,1,8,7},
                     {8,9,2,7,1} };
       System.out.println("Now columns\nBefore:");
```

```
            printArr2D(b);
            sortCols(b);
            System.out.println("After:");
            printArr2D(b);
        }
    }
```

should print

```
[3, 2, 6, 1, 3, 5, 6, 1, 3]
[3, 1, 2, 1, 5, 7, 2]
[8, 9, 2, 1]
After:
[1, 1, 2, 3, 3, 3, 5, 6, 6]
[1, 1, 2, 2, 3, 5, 7]
[1, 2, 8, 9]
Now columns
Before:
[3, 2, 6, 1, 6]
[7, 1, 2, 1, 5]
[5, 3, 1, 8, 7]
[8, 9, 2, 7, 1]
After:
[3, 1, 1, 1, 1]
[5, 2, 2, 1, 5]
[7, 3, 2, 7, 6]
[8, 9, 6, 8, 7]
```

Problem 2

In the program below

```
public class Arr2DJagged {
    public static void main (String[] args) {
        int[][] arr =
            { {1,3}, {3,4,5,8}, {6,8}, {9} };
        double[] res = getAverages(arr);
        for (double e : res) System.out.print(e + " ");
    }
    // ...
}
```

add a static function **getAverages** which

- takes a two-dimensional array of **int**s (not necessarily rectangular);
- returns an array of numbers of type **double** with its size equal to number of 'rows' in the two-dimensional array passed to the function. Consecutive elements of the resulting array should be equal to arithmetic averages of all elements of consecutive rows of the input array.

For the array as above, the result should be

```
2.0 5.0 7.0 9.0
```

Problem 3

Create two classes:

- **Point** describing points on the Cartesian plane
  - with fields x and y of type **int** (coordinates of the point)
  - and one static method **getPoint** returning an object of the class based on two arguments passed to it (such method plays the rôle of a constructor);
- **Rect** describing rectangles on the Cartesian plane with sides paralel to the axes.
  - It has two fields of type **Point**: ul (upper-left vertex) and br (bottom-right);
  - and a static method **getRect** which returns an object of the class given one point (upper-left vertex) and width and hight of the rectangle;
  - static method **getContainingRect** which accepts an array of points and returns an object of the class corresponding to the smallest rectangle containing all points from the array;
  - and a non-static method **showInfo** printing information on the object.

For example, the following program (classes should be in separate files!)

download *PointsAndRects.java*

```
public class Point {
    int x, y;
    static Point getPoint(int x, int y) {
        // ...
    }
}

public class Rect {
    Point ul;
    Point br;
    static Rect getRect(Point ull, int w, int h) {
        // ...
    }
    static Rect getContainingRect(Point[] arr) {
        // ...
    }
    void showInfo() {
        // ...
    }
}

public class PointsAndRects {
    public static void main(String[] args) {
```

```
        Rect rec = Rect.getRect(Point.getPoint(2, 6), 6, 4);
        rec.showInfo();

        Point[] points = {
            Point.getPoint(3, 4), Point.getPoint(5, 6),
            Point.getPoint(1, 3), Point.getPoint(5, 3),
            Point.getPoint(4, 1), Point.getPoint(3, 7)
        };
        Rect cont = Rect.getContainingRect(points);
        cont.showInfo();
    }
}
```

should print

```
UL=(2, 6) BR=(8, 2)
UL=(1, 7) BR=(5, 1)
```

Problem 4

A producer of cereal inserts a coupon into every box; with equal probability, of one out of N kinds. Customers can win a prize after collecting at least one of every kind. We assume that collectors do not cooperate (by exchanging their coupons).

Write a function **boxesBought** which simulates, using a pseudo-random number generator, the process of buying boxes of cereal until at least one coupon of every kind is collected. The function takes the number of kinds of coupons and returns the number of boxes bought. Run many (e.g., 100000) such simulations and calculate the average number of boxes bought (for various values of N).

Compare the average obtained with the theoretically expected value which is, as can be calculated, NH_N, where H_N is the n-th harmonic number

$$H_N = \sum_{i=1}^{N} \frac{1}{i}$$

(write an auxiliary function **harmo** calculating harmonic numbers.)
The program below

download *Coupons.java*

```
public class Coupons {
    public static void main(String[] args) {
        final int N = 90;
        final int NUM_OF_SIMULATIONS = 100000;
        int totalBoxes = 0;
        for (int  i = 0; i < NUM_OF_SIMULATIONS; ++i) {
            totalBoxes += boxesBought(N);
        }
        double aver = totalBoxes/(double)NUM_OF_SIMULATIONS;
        System.out.println("***** N = " + N);
```

```
            System.out.println("Average : " + aver);
            System.out.println("Expected: " + N*harmo(N));
        }

        static int boxesBought(int coupons) {
            // ...
        }

        static double harmo(int n) {
            // ...
        }
    }
```

should print (of course, the numbers obtained from simulation can be a little different)
For $N = 5$

```
***** N = 5
Average : 11.37195
Expected: 11.416666666666666
```

For $N = 30$

```
***** N = 30
Average : 119.86697
Expected: 119.84961392761174
```

For $N = 90$

```
***** N = 90
Average : 457.93183
Expected: 457.4313542563665
```

The task could be accomplished more effectively by using streams and collections, but plain loops and arrays can also be sufficient.
