Exercises on Arrays

Exercise 1

Write a Java method with the following signature

```
public static void arrayCopy(int[] source, int[] destination)
```

The method copies the entire contents of the array `source` to the array `destination`

Initialize `source` to any integers you want; initialize `destination` to `-1` for every element

Print the contents of `destination` before and after a call to `arrayCopy()` to test that all the elements have been copied correctly

Make the array sizes the same for both `source` and `destination`

Exercise 2

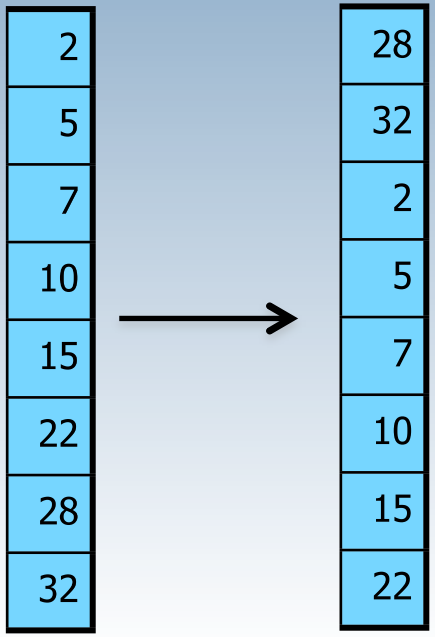
Write a Java method with the following signature

```
public static void arrayShift(int[] numbers, int shift)
```

The method shifts all the elements of the array `numbers` by the amount given in the parameter `shift`. If `shift` is positive, the elements are shifted down; if `shift` is negative the elements are shifted up. If `shift` is zero, no shifting happens. The elements at the top or bottom of the array must be wrapped around

For example, the call `arrayShift(numbers, 2)` would have this effect.

Notice that the two bottom numbers reappear at the top of the array after shifting



Exercise 3

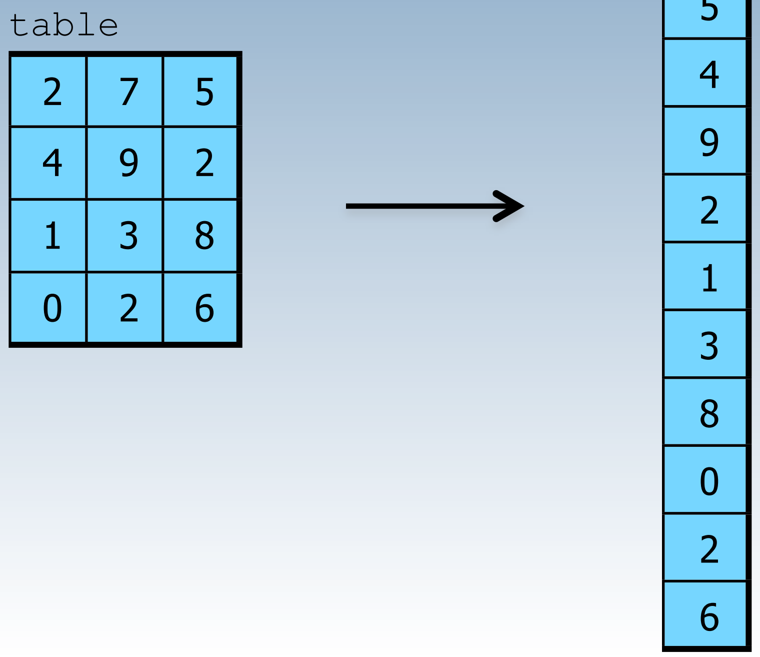
Write a Java method with the signature

```
public static void transfer2to1(int[][] table, int[] list)
```

The method copies the contents of the two dimensional array `table` to the one dimensional array `list`, where `table` can have any dimensions.

The size of `list` needs to be determined at run-time from the dimensions of `table`.

The elements are copied from left to right for each row. For example:



If `table` is a square array, what is the order of complexity of your method with respect to the number of rows? Why?

Exercise 4

Write a Java method with the following signature

```
public static void uniqueEntries(int[] list)
```

The method fills the array `list` with unique random integers

Let the range of integers be from 0 to five times the size of `list`. For example, if `list` is an array of 30 elements, then the range is [0, 150)

The integers must be randomly generated and the array must not contain any repeats, that is, every number in the array must appear only once

Exercise 5

Write a Java program that calculates the average number of comparisons that a linear search performs on an array of 800 integers*. Use the following algorithm

```
Declare and instantiate an array of 800 integers
Initialize array with integers from 0 to 799 inclusive

Do the following 1000000 times {
    generate a random number in [0, 800)
    use linear search to find the number
    accumulate the total number of comparisons so far
}
print (total comparisons / 1000000)
```

The program outputs around 400 as an average number of comparisons. After complexity analysis of the number of comparisons with respect to the size of the array, we also obtain a number close to 400. Why?

**The array is uniformly distributed for this exercise: All numbers in the given range are present in the array*

Exercise 6

The game of Tic Tac Toe can be easily represented with a 3x3 array, where `0` stands for an empty spot, `1` stands for the moves of one player and `2` stands for the moves of the other player.

An example of what the board might look like during a game is

2	0	0
0	2	1
0	0	1

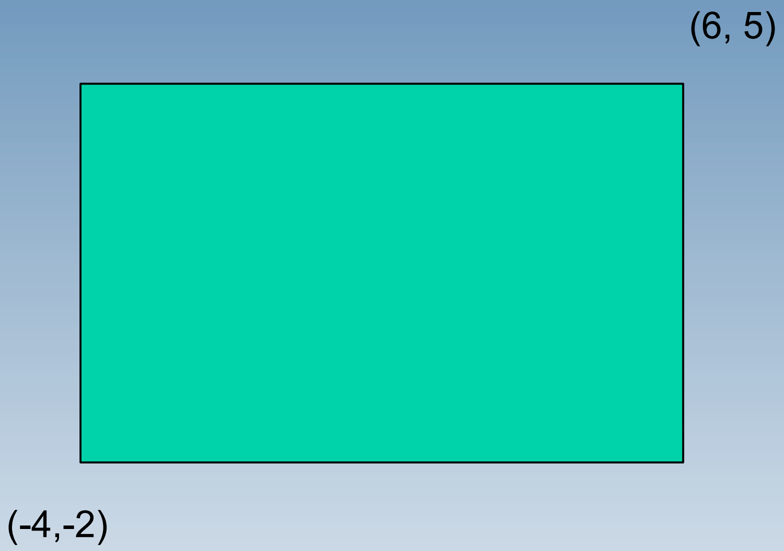
Write Java method with the signature

```
public static int gameOver(int[][] board)
```

that determines from the board if the game is over. A game is over if there are any three 1's or 2's in a horizontal row, a vertical column, or a diagonal, or if all squares have been filled with a tie. The method returns `-1` if the game is not over, `0` if a tie, or the number of the winner

Exercise 7

A rectangle can be represented with a 2x2 array where the first row contains the lower left (x, y) coordinates and the second row contains the upper right (x, y) coordinates. For example,



-4	-2
6	5

See next slide

Exercise 7 (Continued)

Write a Java method with the signature

```
  public static boolean rectOverlap(int[][] r1, int[][] r2)
```

that determines if rectangles `r1` and `r2` overlap. This means that any portion of one rectangle is inside the other. If a rectangle has any side on top of the other rectangle, we consider this overlapping.

The coordinates are passed in the standard `xy` plane with (0, 0) at the bottom left corner of the screen. Java has (0, 0) at the top left, so adjustments may need to be made in the calculations

Use a Java drawing method to draw both rectangles and confirm the result of your method

Exercise 8

Write a Java method with the signature

```
    public static boolean circleOverlap(int[] c1, int[] c2)
```

that determines if circles `c1` and `c2` overlap

Each of the array parameters contains 3 numbers: The first two numbers are the coordinates for the centre of the circle and the last number is the size of the radius

Exercise 9

Write a Java method with the signature

```
public static void bubbleSort(String[] items)
```

that sorts the array of strings `items` in ascending order

Since Java allows overloading, this method can have the same name as the Bubble Sort that we already wrote previously that sorts integers.

When making the call, Java will invoke the correct method because of the parameters. Java will be able to distinguish one method from the other because the call will contain either an array of integers or an array of strings. The methods have different signatures.

Exercise 10

Write a Java method with the signature

```
public static String lastName(String entry)
```

that returns the last name from an entry that contains a first name and a last name

The parameter `entry` contains two words: A first name and a last name such as `Adam Smith`. Between the first name and the last name there are one or more spaces. You can assume that in `entry` there are no spaces before the first name and there are no spaces after the last name. In our example, there are no spaces before `Adam` and there are no spaces after `Smith`

The method must return the last name without any spaces before or after it. In our example, `Smith` is returned without any spaces

Exercise 11

Write a Java method with the signature

```
 public static void bubbleSort(String[] items, int orderKey)
```

that sorts the array `items` in ascending order depending on the value of the parameter `orderKey`

The array `items` contains first names and last names such as

```
Adam Smith
John Donahue
Suzie   Mango
```

The method sorts by first name if the value of `orderKey` is `0`. The method sorts by last name if the value of `orderKey` is `1`. Use your methods from exercises 9 and 10 to create this new method

You can assume that the array `items` and the parameter `orderKey` contain valid values

Exercise 12

Write a Java method with the signature

```
public static int[] oneRow(int[][] matrix, int n)
```

that returns the `n`th row of the array `matrix`

The method returns a one dimensional array whose size is equal to the number of columns of `matrix`. For example, if `matrix` is

```
10  32  45  44
15  75  62   7
 9  32   0  82
```

then the call `oneRow(matrix, 2)` returns the one dimensional array

```
9
32
0
82
```

See next slide

Exercise 12 - Continued

To obtain the dimensions of a two dimensional array,

A call to `matrix.length` will give the number of rows of `matrix`

A call to `matrix[0].length` will give the number of columns of `matrix`

The array that will be returned needs to be declared and instantiated inside the method and have the correct size. Here is the code to do it:

```
int[] temp = new int[matrix[0].length];
```

Exercise 13

The game of Sudoku can be easily represented with a 9x9 array, where `0` stands for an empty spot and the numbers `1` to `9` are the values that the user has entered so far

An example of what a board might look like during a game is

0	6	9	0	3	0	0	1	0
0	3	4	0	0	0	0	0	0
0	2	0	0	0	0	0	4	0
0	5	0	0	0	0	8	7	2
0	0	0	0	0	0	0	0	4
0	0	0	0	8	0	0	6	0
3	0	7	0	0	0	0	0	0
2	4	0	5	6	0	0	0	0
0	0	0	0	0	0	3	2	0

See next slide

Exercise 13 (Continued)

One of the rules of the game is that the player cannot have a number repeated on the same row. We need to check that this does not happen

Write a Java method with the signature

```
public static boolean validRow(int[][] sudokuBoard, int num, int row)
```

where `num` is the number that the user wishes to enter. The method scans the `row` of `sudokuBoard` and returns `true` if `num` is not found in the row, i.e., the row is valid. Otherwise the method returns `false`

In the board from the previous slide the call

`validRow(board, 2, 3)` returns `false` because the row is not valid, i.e., it already contains a `2`

The call `validRow(board, 4, 8)` returns `true` because the row is valid

Exercise 14

A second rule of Sudoku is that a column cannot have a repeated number. Write a Java method with the signature

```
 public static boolean validCol(int[][] sudokuBoard, int num, int col)
```

that uses the same guidelines as exercise 13, but with the columns of the board instead of the rows

The call `validCol(board, 8, 0)` returns `true` because the column is valid, i.e., the number `8` is not in column `0`

The call `validCol(board, 3, 6)` returns `false` because the column is not valid

Exercise 15 - Extending

The last rule of Sudoku is that a number cannot be repeated within a 3x3 matrix as shown below. Write a Java method with the signature

```
public static boolean validBox(int[][] sudokuBoard,
                                   int num, int row, int col)
```

0	6	9	0	3	0	0	1	0
0	3	4	0	0	0	0	0	0
0	2	0	0	0	0	0	4	0
0	5	0	0	0	0	8	7	2
0	0	0	0	0	0	0	0	4
0	0	0	0	8	0	0	6	0
3	0	7	0	0	0	0	0	0
2	4	0	5	6	0	0	0	0
0	0	0	0	0	0	3	2	0

The method checks if `num` is in the 3x3 matrix surrounding the entry at `(row,col)`

See next slide

Exercise 15 - Extending (Continued)

In the Sudoku board, there are nine 3x3 matrices with top left corners at

```
0,0    0,3    0,6
3,0    3,3    3,6
6,0    6,3    6,6
```

From a given `row` and `col`, we can calculate the top left corner of the 3x3 matrix that surrounds the entry at `(row,col)` and then check if `num` is already contained in this 3x3 matrix. This check will require nested loops

In the board of the previous slide, the call `validBox(board, 5, 4, 0)` will return `false` because the number `5` is already in the 3x3 matrix that surrounds the entry at `(4,0)`

The call `validBox(board, 1, 8, 2)` will return `true` because the number `1` is not in the 3x3 matrix that surrounds the entry at `(8,2)`

Exercise 16 - Extending

Write a Java method with the signature

```
public static boolean validSudoku(int[][] sudokuBoard,
                                  int num, int row, int col)
```

that returns `true` if `num` does not break any of the three rules of the game as outlined in exercises 13, 14, 15. Otherwise the method returns `false`

Be sure to use the methods from exercises 13, 14, and 15. If these methods are all in working order then `validSudoku()` will turn out to be a surprisingly short method

Selected Solutions

Solution to Exercise 3

```
public static void transfer2to1(int[][] table, int[] list) {
    for (int row = 0; row < table.length; row++) {
        for (int col = 0; col < table[row].length; col++) {
            list[row * table[row].length + col] = table[row][col];
        }
    }
}
```

Solution to Exercise 4

```
public static void uniqueEntries(int[] list) {
    int upper = list.length * 5;  // set range to five times the array size
    int number = -1;

    for (int i = 0; i < list.length; i++) {
        boolean repeatFound = false;
        do {
            repeatFound = false;
            number = (int) (Math.random() * upper);    // generate a new number
            for (int j = 0; j < i && !repeatFound; j++) { // check if it's in list
                repeatFound = (number == list[j]);
            }
        } while (repeatFound);       // keep generating numbers until
                                     // a unique one is found
        list[i] = number;
    }
}
```

Solution to Exercise 13

For testing purposes, a 4x4 array is used in this program

```
public class Sudoku {
    public static boolean validRow(int[][] sudokuBoard, int num, int row) {
        int col = 0;
        for (col = 0;
             col < sudokuBoard[0].length && sudokuBoard[row][col] != num;
             col++)
            ;

        return (col == sudokuBoard[0].length);
    }

    public static void main(String[] args) {
        int[][] board = {{1, 2, 3, 4},
                          {2, 1, 5, 8},
                          {3, 6, 7, 2},
                          {9, 8, 6, 5}};
        System.out.println(validRow(board, 5, 3));
        System.out.println(validRow(board, 9, 0));
        System.out.println(validRow(board, 6, 1));
    }
}
```

The output of this program is

```
false
true
true
```