Midterm Exam

Course CS2040, Spring 2020, American University of Paris

The use of book or notes is allowed. Internet access is not allowed.
Instructions: create a folder with your name. Then inside this folder create a folder/pachage for each exercise. At the end of the test, give/send the main folder with all sub-folders and files to the proctor.
At the beginning of each file add a comment line with your name and ID number.

1. (The MyRectangle2D class, 22 pt points)

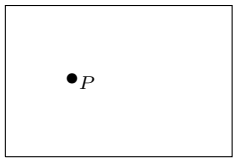
   - Define an abstract class

     MyGeometricObject

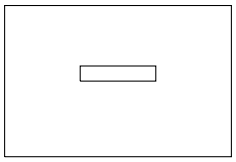
     which contains:
     - a static int data field named myGeometricObjectCounter set as 0, with getter and setter methods;
     - a static method howManyMGO() which prints the counter myGeometricObjectCounter;
     - a static method increaseCounter() which increases myGeometricObjectCounter by 1.
   - Define a class

     MyRectangle2D

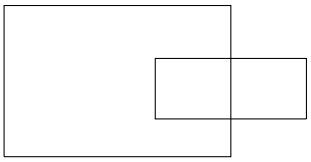
     that extends the abstract class MyGeometricObject. The objects of this class are rectangles with sides parallel to x- or y- axes. It contains:
     - two double data fields named x and y that specify the center of the rectangle, with getter and setter methods.
     - a double data field width and a double data field height, with getter and setter methods.
     - a no-argument constructor that creates a default rectangle with center coordinate $(0, 0)$ and both width and height equals to 1 (ATTENTION: remember that 1 is an int value while the constructor wants two doubles!);
     - a constructor that creates a rectangle with specified values for x , y, width and height
     - both constructors have to increase the myGeometricObjectCounter by 1;
     - a method getArea() that returns the area of the rectangle.
     - a method getPerimeter() that returns the perimeter of the rectangle.
     - a method contains(doublex, doubley) that returns true if the point (x, y) is inside this rectangle, false otherwise. See Figure (1a)
     - a method contains(MyRectangle2Dr) that returns true if the specified rectangle is inside this rectangle. See Figure (1b).
     - a method overlaps(MyRectangle2Dr) that returns true if the specified rectangle overlaps with this rectangle. See Figure (1c).
   - Write a main class with a main method to test the methods:
     - define a rectangle $R1$ with center $(1.5, 1)$, width 3 and height 2;
     - define a rectangle $R2$ using the no-argument constructor;
     - print the area and the perimeter of $R1$;



(a) A point P inside the rectangle



(b) A rectangle is inside another rectangle (it also overlap)



(c) A rectangle overlaps another rectangle (it is not inside)

- test if the point $(1, 1)$ is contained in $R1$ and print the result;
- test if $R2$ is contained in $R1$ and print the result;
- test if the rectangle $R3$ with center $(3, 0.9)$, width 2 and height 0.8 overlaps with $R1$, and print the result.

[TO HELP YOU WITH THE CHECK: the bigger rectangle each Figures (1a,1b and 1c) is $R1$ with width 3, height 2 and lower-left corner in $(0, 0)$. The point P in Figure (1a) has coordinates $(1, 1)$ and the small rectangle in Figure (1c) has center $(3, 0.9)$, width 2 and height 0.8.]

EXTRA/OPTIONAL:

- overload the constructor method to create a rectangle with lower-left corner in $(1, 1)$ with a given length and height and use it in the main function to test if the rectangle with lower-left corner in $(1, 1)$, length 1 and height 0.2 is contained in $R1$.
- override the toString() method in order to print the four corners of the rectangle and use it in the main function to print the rectangle $R1$.

2. (Selection sort, 10pt points) The *selection sort* is an in-place sorting algorithm. In this exercise we implement the algorithm to order an array of integers.

   The algorithm sorts the array arr from left to right as follows:

   - it looks for the bigger element between all the elements of the array, that is, it looks all the elements form the first to the last one;
   - it exchanges this maximal element with the last element of the array (if it is already in this position, do nothing);
   - then it looks for the bigger element between all the remaining elements of the array, that is, it looks all the elements form the first to the second-last one;
   - it exchanges this maximal element with the second-last element of the array (if it is already in this position, do nothing);
   - ... and so on until when it finish arrives at the beginning of the array.

   Example:

   - Let us consider the array $[7, 12, 11, 29, 6]$.
   - The biggest element between the first and the last is 29. Then I swap the 29 with the last element of the array which is 6.
   - I obtain the array $[7, 12, 11, 6, 29]$.
   - The biggest element between the first and the second-last is 12. Then I swap the 12 with the second-last element of the array which is 6.
   - I obtain the array $[7, 6, 11, 12, 29]$.
   - The biggest element between the first and the third-last is 11, which is already the third-last element of the array. I do nothing
   - I still have the array $[7, 6, 11, 12, 29]$.
   - The biggest element between the first and the fourth-last is 7. Then I swap the 7 with the fourth-last element of the array which is 6.
   - I obtain the array $[6, 7, 11, 12, 29]$.
   - The next element to check is the first element of the array. Then, the array is sorted. I stop.

   Implement the method

```
void SelectionSort (int[] arr)
```

   HINTS:

   - the method length returns the length of an array, that is, if arr is an array of length n, then arr.length is the integer n.
   - it may be useful (but not mandatory) to implement a method

     int maxBetween (int[] arr, int minIndex, int maxIndex)

     which returns the max value between the elements of an array with index included between minIndex and maxIndex.