Software Architecture. Interfaces exercise:

1. Implement a class Person that:
    a. Stores both name and surname.
    b. Has a method to print the name and surname to the screen.

```
public class Person {

    protected String name;
    protected String surname;

    public Person(String name, String surname) {
        this.name = name;
        this.surname = surname;
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public void printPersonName() {
        System.out.println("Person's name is: " + this.name + ", "
+ this.surname);
    }
}
```

2. Implement a class Sorter that it is able to sort an array of Persons (Person[]). It must: (write the algorithm yourself)
    a. Have a method that receives an array of Person
    b. Sort the array by surname, name.
    c. Use a simple algorithm like:
        i. https://en.wikipedia.org/wiki/Bubble_sort
        ii. https://en.wikipedia.org/wiki/Insertion_sort

In this section, I implemented QuickSort as its run time order is nlogn, and it is faster than bubbleSort or InsertionSort on average.

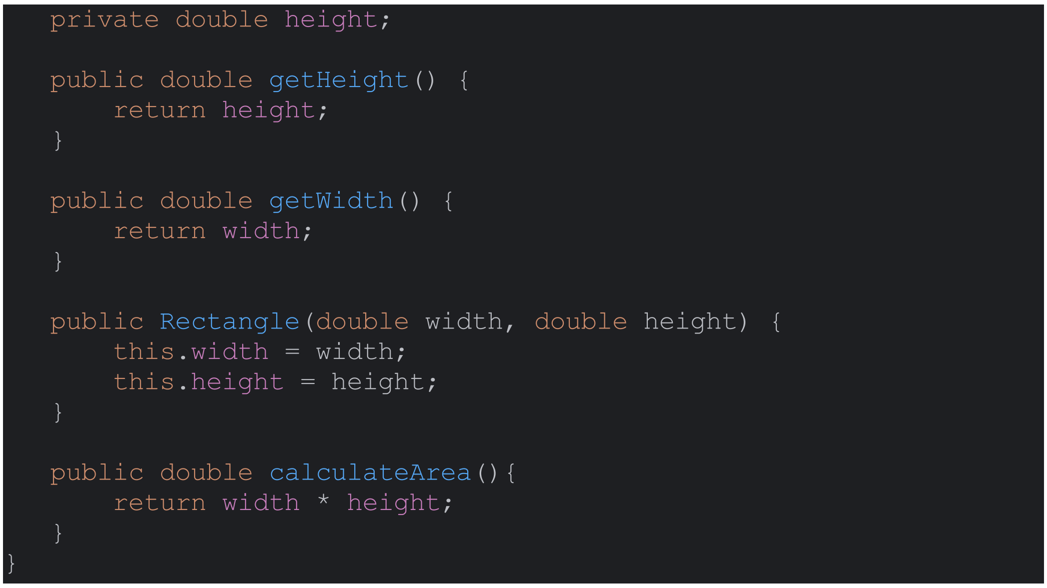
```
public class Sorter {

    public void sort(Person[] persons) {
        quickSort(persons, 0, persons.length - 1);
    }

    // Quicksort recursive method
    private void quickSort(Person[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            // Recursively sort elements before and after partition
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    // Partition method
    private int partition(Person[] arr, int low, int high) {
        Person pivot = arr[high];  // Choose last element as pivot
        int i = low - 1;           // Index of smaller element

        for (int j = low; j < high; j++) {
            if (compare(arr[j], pivot) <= 0) {
                i++;
                // Swap arr[i] and arr[j]
                Person temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // Swap arr[i+1] and arr[high] (pivot)
        Person temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    // Compare by surname, then name
    private int compare(Person p1, Person p2) {
        int surnameComparison =
p1.getSurname().compareToIgnoreCase(p2.getSurname());
        if (surnameComparison != 0) {
            return surnameComparison;
        }
        return p1.getName().compareToIgnoreCase(p2.getName());
    }
}
```

3. Implement a Program that uses the Sorter to sort an array of 5 persons.

```
public class Main {
    public static void main(String[] args) {
        Person[] people = {
                new Person("Nika", "Soltani"),
                new Person("Nikoleta", "Manavi"),
                new Person("James", "Press"),
                new Person("John", "Doe"),
                new Person("Fred", "Struik")
        };

        Sorter sorter = new Sorter();
        sorter.sort(people);

        // Print sorted list
        for (Person p : people) {
            p.printPersonName();
        }
    }
}
```

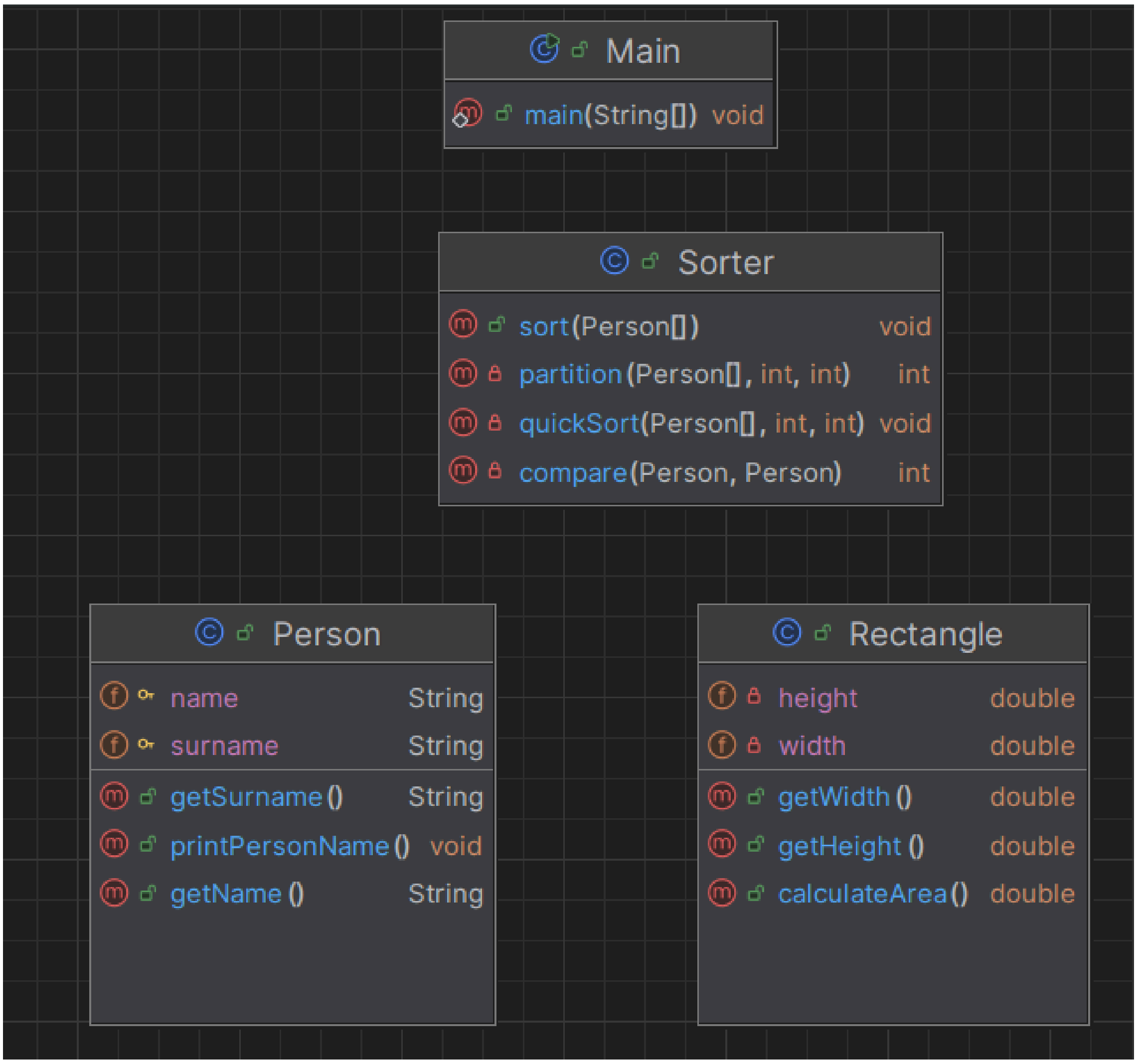
4. Implement a class Rectangle that:
    a. Stores its width and the height
    b. Has a method to calculate its area

```
public class Rectangle {

    private double width;
```



```
    private double height;

    public double getHeight() {
        return height;
    }

    public double getWidth() {
        return width;
    }

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double calculateArea(){
        return width * height;
    }
}
```

5. Draw a diagram of the current classes of the system



Main

main(String[]) void

Sorter

sort(Person[]) void

partition(Person[], int, int) int

quickSort(Person[], int, int) void

compare(Person, Person) int

Person

name String

surname String

getSurname() String

printPersonName() void

getName() String

Rectangle

height double

width double

getWidth() double

getHeight() double

calculateArea() double

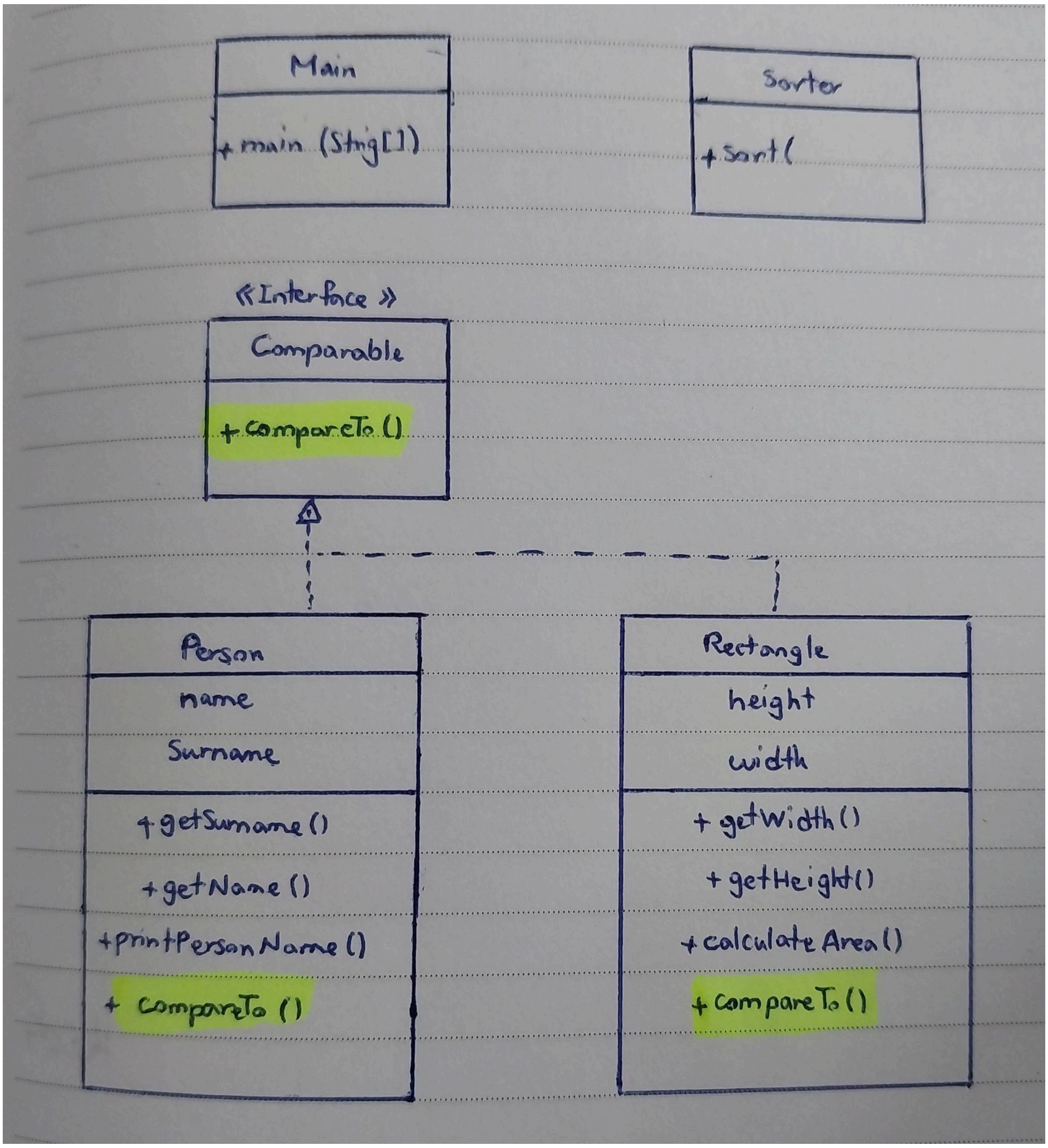
6. Which modifications do you have to do to make the Sorter class capable of sorting both Rectangles and Persons?
   a. Rectangles must be sorted by their area.

b. Which element of your design enables the class Sorter and its sort method to accept both Persons and Rectangles?

c. Draw a new diagram with the proposed changes before trying to implement it. (by hand, take a picture and put it in the documentation)

d. DO NOT CONTINUE READING UNTIL YOU HAVE A CLEAR IDEA OF HOW TO IMPLEMENT IT !!

As each of these two classes need to be sorted but the comparison method is different in each of them, we need to have a contract (interface) for the comparison concept that each of Person and Rectangle implement this interface. So, at the end, the Sorter class only needs to call the comparison functionality of the objects.



7. This element must have a method that receives two objects and return a boolean indicating if the second is bigger or not (in sort order) than the the first object. Make sure that your design follows this tip.

Here is the implementation of the interface:

```
public interface MyComparable<T> {
   int compareTo(T other);
}
```

Rectangle class implementing MyComparable Interface

```
public class Rectangle implements MyComparable<Rectangle> {

   private double width;
   private double height;

   public Rectangle(double width, double height) {
       this.width = width;
       this.height = height;
   }

   public double calculateArea() {
       return width * height;
   }

   @Override
   public int compareTo(Rectangle other) {
       double area1 = this.calculateArea();
       double area2 = other.calculateArea();
       return Double.compare(area1, area2);
   }
}
```

Person class implementing MyComparable Interface

```
public class Person implements MyComparable<Person> {

   protected String name;
   protected String surname;

   public Person(String name, String surname) {
       this.name = name;
       this.surname = surname;
   }

   public String getName() {
       return name;
   }

   public String getSurname() {
       return surname;
   }

   public void printPersonName() {
       System.out.println("Person's name is: " + this.name + ", "
+ this.surname);
   }

   @Override
   public int compareTo(Person other) {
       int surnameCmp =
surname.compareToIgnoreCase(other.surname);
       if (surnameCmp != 0) return surnameCmp;
```

```java
        return name.compareToIgnoreCase(other.name);
    }
}
```

8. Implement your design and make a program that sorts an array of 5 Persons and another array of 5 Rectangles.

9. Your solution should be similar to the one, already present in standard Java. Have a look at the following documents:
    a. https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html
    b. https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#sort(java.util.List)

10. Implement your program to sort Persons and Rectangles by using the Comparable interface and the Collections class.
In this phase, we use the Comparable and Collection Classes in Java, and the implementation looks as follows (All the codes are on GitHub).

The Sorter Class

```java
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class Sorter {

    public <T extends Comparable<T>> void sort(T[] array) {
        List<T> list = Arrays.asList(array);
        Collections.sort(list);
    }

}
```

Person Class

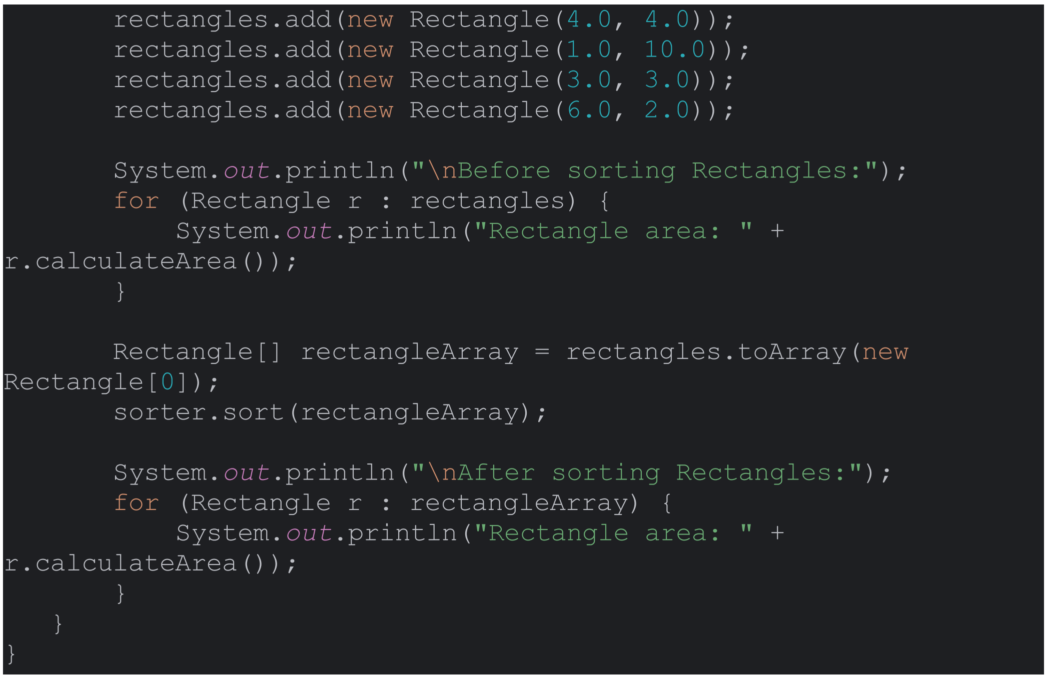
```java
public class Person implements Comparable<Person> {

    protected String name;
    protected String surname;

    public Person(String name, String surname) {
        this.name = name;
        this.surname = surname;
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public void printPersonName() {
        System.out.println("Person's name is: " + this.name + ", "
+ this.surname);
    }

    @Override
    public int compareTo(Person other) {
        int surnameCmp =
surname.compareToIgnoreCase(other.surname);
        if (surnameCmp != 0) return surnameCmp;
        return name.compareToIgnoreCase(other.name);
    }
}
```

Rectangle Class

```
import java.awt.*;

public class Rectangle implements Comparable<Rectangle> {

    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double calculateArea(){
        return width * height;
    }

    @Override
    public int compareTo(Rectangle other) {
        double area1 = width * height;
        double area2 = other.width * other.height;
        return Double.compare(area1, area2);
    }
}
```

Main

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Sorter sorter = new Sorter();

        List<Person> people = new ArrayList<>();
        people.add(new Person("Nika", "Soltani"));
        people.add(new Person("Nikoleta", "Manavi"));
        people.add(new Person("James", "Press"));
        people.add(new Person("John", "Doe"));
        people.add(new Person("Fred", "Struik"));

        System.out.println("Before sorting Persons:");
        for (Person p : people) {
            p.printPersonName();
        }

        Person[] peopleArray = people.toArray(new Person[0]);
        sorter.sort(peopleArray);

        System.out.println("\nAfter sorting Persons:");
        for (Person p : peopleArray) {
            p.printPersonName();
        }

        List<Rectangle> rectangles = new ArrayList<>();
        rectangles.add(new Rectangle(2.0, 5.0));
```



```
        rectangles.add(new Rectangle(4.0, 4.0));
        rectangles.add(new Rectangle(1.0, 10.0));
        rectangles.add(new Rectangle(3.0, 3.0));
        rectangles.add(new Rectangle(6.0, 2.0));

        System.out.println("\nBefore sorting Rectangles:");
        for (Rectangle r : rectangles) {
            System.out.println("Rectangle area: " +
r.calculateArea());
        }

        Rectangle[] rectangleArray = rectangles.toArray(new
Rectangle[0]);
        sorter.sort(rectangleArray);

        System.out.println("\nAfter sorting Rectangles:");
        for (Rectangle r : rectangleArray) {
            System.out.println("Rectangle area: " +
r.calculateArea());
        }
    }
}
```