Question 2: Basic Programming Problems (Class and Instance Methods and Variables)

a. Define a class called `Q2`.

Instance Variables

The class should have the following two instance variables:

(a) an `ArrayList` with the type parameter `String` called `arrayList`

(b) a `double` array (not an `ArrayList`!) called `doubleArray`

Note: You may find it necessary to add other instance variables to the class in order to complete later functionality.

Constructor

The class should have a constructor that initialises both instance variables. The constructor must accept one input parameter, an `int` which defines the initial size of the `double` array. This parameter **must** be positive, and a suitable run time exception should be thrown if it is not.

Accessor Methods

The class should also contain accessor (getter) methods for the two instance variables described above.

Encapsulation

The class must be defined with consideration given to encapsulation (so consider the visibility of instance variables, methods and constructors and the mutability of instance variables with accessor methods.

(10%)

b. Continuing to work on the class `Q2`, define as many mutator methods with the name `add` as you need to be able to accept a single parameter of each of the following types:

- `int`
- `double`
- `float`
- `String`
- `Object`

The method does not need to return any value.

Parameter Types

If the value passed as a parameter is numeric data (i.e. `int`, `double`, and `float`), this should be inserted into the `double` array. If the value passed as a parameter is not numeric data (any other type of data), this should be added to the `ArrayList`. When an `Object` is received, call its `toString` method and add the `String` to the `ArrayList`.

Numbers in Strings

If the parameter is a `String` that contains a number, you do not need to convert this value. It should be stored in the `arrayList` as a `String`.

(20%)

c. Continuing to work on the class `Q2`, define as many methods with the name `contains` as you need for the types below. All of the methods should return `true` if the passed value is present in either the `ArrayList` or `double` array, and `false` if it not present in either. The methods should be able to accept parameters of the following types:

- `int`
- `double`
- `float`
- `String`
- `Object`

(10%)

d. Continuing to work on the class `Q2`, define a `toString` method. This method must override the `toString` method inherited from the Object class. This method should return a String containing all elements from the `ArrayList` followed by all elements from the `double` array as a comma separated list.

Note

Do not return empty values in the `double` array in the `toString` method.

Example

Code:

```
Q2 test = new Q2(2);
test.add("Java");
test.add("Exam");
test.add(1);
test.add(2);
String re = test.toString();
System.out.println(re);
```

Result:

```
Java,Exam,1.0,2.0
```

(10%)

(Question Total 50%)

```
import java.util.ArrayList;
import java.util.Arrays;

public class Q2 {
    private ArrayList<String> arrayList;
    private double[] doubleArray;
    private int rearOfDoubleArray;

    public Q2(int initialSize) {
        if (initialSize <= 0) {
            throw new RuntimeException();
        }
        doubleArray = new double[initialSize];
```

```
        arrayList = new ArrayList<>();
    }

    public ArrayList<String> getArrayList() {
        return new ArrayList<>(arrayList);
    }

    public double[] getDoubleArray() {
        return Arrays.copyOf(doubleArray, rearOfDoubleArray);
    }



    public void add(Number value) {
        if(rearOfDoubleArray == doubleArray.length -1) {
            double[] newDoubleArray = new double[doubleArray.length +5];
            for (int i = 0; i < doubleArray.length; i++){
                newDoubleArray[i] = doubleArray[i];
            }
            doubleArray = newDoubleArray;
        }
        doubleArray[rearOfDoubleArray] = value.doubleValue();
        rearOfDoubleArray++;
    }


    public void add(Object value) {
        arrayList.add(value.toString());

    }

    public boolean contains(Number value) {
        for (double element : doubleArray) {
            if (element ==  value.doubleValue()) {
                return true;
            }
        }
        return false;
    }


    public boolean contains(Object value) {
        return arrayList.contains(value.toString());
    }

    public String toString() {
        String s = "";
        int count = 0;
        for (String element : arrayList){
            s += element + ",";
            count++;
        }
        for (int i = 0; i < rearOfDoubleArray; i++) {
                if (i != rearOfDoubleArray - 1) {
                    s += doubleArray[i] + ",";
                }else {
```

```
                    s += doubleArray[i];
                }
            }
            return s;
        }
}

```

Question 3: Processing Files and Objects

a. Define a class called `Q3`.

Data From a File

Define a class/static method named `readStudentsFile`. This methods should take a `String` as a parameter (containing a file name) and return an `ArrayList` of `Student` objects. The method should read all of the data from the file named in the parameter, use this data to create Student objects and add them to the array list in the same order as they appear in the file.

Sorting Students by Age

Define a class/static method named `sortAge`. This method should take an `ArrayList` containing `Student` objects as a parameter and sort the student objects from oldest to youngest.

Notes:

- Date values represented as `Strings` in the format YYYY-MM-DD do not need to be converted into another data type to be sorted (they can be sorted as `Strings`)
- You cannot make any changes to the Student class (these changes will be ignored when your code is being tested)

Sorting Students by Names

Define a class/static method named `sortFamilyGivenName`. This method should take an `ArrayList` containing `Student` objects as a parameter and sort the student objects alphabetically based first on their family name and then on their given name (only when family names are the same).

Finding Students

Define the class/static methods named `getMedianAgeStudent`, `getYoungestStudent`, and `getOldestStudent`. The methods should all take an `ArrayList` containing `Student` objects as a parameter and return the `Student` object that is the median age, youngest or oldest respectively.

问题3:处理文件和对象

a.定义一个名为Q3的类。

文件中的数据

定义一个名为readStudentsFile的类/静态方法。这个方法应该接受一个String作为参数(包含一个文件名)，并返回一个学生对象的数组列表。该方法应该从参数中指定的文件中读取所有数据，使用这些数据创建Student对象，并按照它们在文件中出现的顺序将它们添加到数组列表中。

按年龄分类学生

定义一个名为sortAge的类/静态方法。这个方法应该接受一个包含学生对象的ArrayList作为参数，并将学生对象从最老到排序
最小的。
注:

- 以“YYYY-MM-DD”格式的字符串表示的日期值不需要
  转换为另一种要排序的数据类型(它们可以排序为string)
- 您不能对Student类进行任何更改(这些更改将在测试代码时被忽略)。

按学生姓名排序

定义一个名为sortFamilyGivenName的类/静态方法。这个方法应该接受一个包含学生对象的ArrayList作为参数，并首先根据他们的姓氏，然后根据他们的名字对学生对象进行排序(只有当姓氏相同时)。

找到学生

定义名为getMedianAgeStudent、getYoungestStudent和getOldestStudent的类/静态方法。这些方法都应该接受一个包含学生对象的ArrayList作为参数，并分别返回年龄中位数、最年轻或最年长的学生对象。

```java
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

public class Q3 {
    public static ArrayList<Student> readStudentsFile(String fileName) {
        ArrayList<Student> students = new ArrayList<Student>();
        try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
            String line;

            while ((line = br.readLine()) != null) {
                line = line.trim();
                String[] parts = line.split(",");
                Student s = new Student(parts[1], parts[0], parts[2]);
                students.add(s);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return students;
```

```
    }

    public static void sortAge(ArrayList<Student> students) {
        Collections.sort(students, (a, b) -> {
            return a.getDateOfBirth().compareTo(b.getDateOfBirth());
        });
    }

    public static void sortFamilyGivenName(ArrayList<Student> students) {
        Collections.sort(students, (a, b) -> {
            int comparison = a.getFamilyName().compareTo(b.getFamilyName());
            if (comparison == 0) {
                return a.getGivenName().compareTo(b.getGivenName());
            }
            return comparison;
        });
    }

    public static Student getMedianAgeStudent(ArrayList<Student> students) {
        sortAge(students);
        int medIndex = students.size() / 2;
        return students.get(medIndex);
    }

    public static Student getYoungestStudent(ArrayList<Student> students) {
        sortAge(students);
        int youngestIndex = students.size() - 1;
        return students.get(youngestIndex);
    }

    public static Student getOldestStudent(ArrayList<Student> students) {
        sortAge(students);
        int oldestIndex = 0;
        return students.get(oldestIndex);
    }
}

```

Hint:

按照年龄大小排序 `Collection.sort()`

```java
List<Student> list = new ArrayList<Student>();
list.add(new Student("Sean", 36));
list.add(new Student("Abey", 37));
list.add(new Student("Vivek", 18));
list.add(new Student("Anca", 21));
System.out.println(list);
Collections.sort(list, (a, b) -> { return Integer.compare(b.
↪  age, a.age); });
System.out.println(list);
```

```
[Sean (36), Abey (37), Vivek (18), Anca (21)]
[Abey (37), Sean (36), Anca (21), Vivek (18)]
```