Course: **Object-Oriented Programming** 22-Jan-2022 **(Fall 2022)**

Resource Person: **Dr. Muhammad Faheem**

ASSIGNMENT–1 (Double Dimensional Arrays, Sorting and Search)

Total Points: 30
Submission Due: Friday Jan 28, 2022
(Google Classroom Course Page)

Instructions: Please Read Carefully!

- **This is an individual assignment. Everyone is expected to complete the given assignment on their own, without seeking any help from any website or any other individual.** There will be strict penalties for any work found copied from any source and the university policy on plagiarism will be strictly enforced.
- You are expected to submit this assignment as:
    - Create a single **.java** file solution of the each assignment question. It means there must be two files. The name of the **.java** file should be as mentioned in the assignment question.
- Assignment is to be submitted via Google Classroom.
    - You should already have created your account on Google Classroom as per my earlier email. If not, then follow the link in that email to create your account.
    - Submit your assignment on or before due date. **No late submissions will be possible**.

Question 1: [10]

You are given a .java code file named "**Q1.java**". Your task is to read the comments in that file and the instructions in this document and complete the required code against each method.

Also, you will need to write code in the **main** method and test all your methods.

Method Details

The details of the methods are as follows:

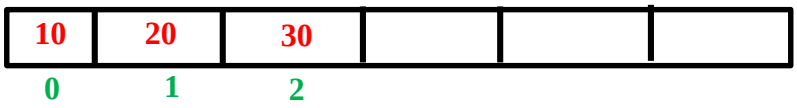
1. **printIntArray** Method **[1]**

   This method prints the values of the argument int array.

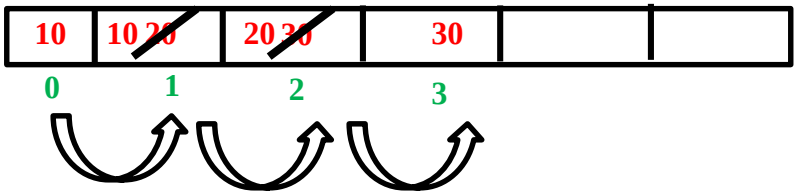
2. **insertKey** Method **[2.5]**

   This method inserts the given key value at index 0 in the array. If the array already has a value at index 0, it will shift the values to the right and make space for the new key.

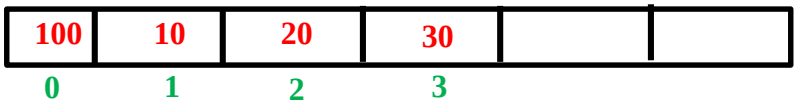
   For example, let us say that we have the following array, and we need to insert a new key **100** at index **0**:



   To insert the new key **100** at index **0**, we will need to make room for this value by shifting all the existing values one place to the right as given below:

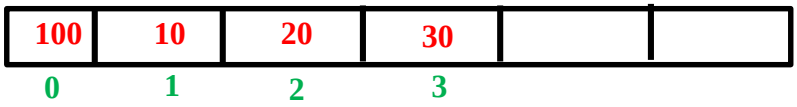


   Now we have shifted all values one place to the right, we can now safely insert the new key **100** at index **0,** and the new array now becomes**:**

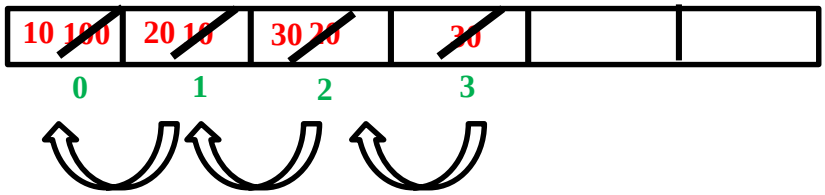


3. **removeKey** Method **[2.5]**

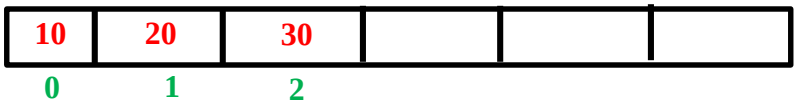
   This method performs the opposite function of the above method and removes and returns the key that exists at index **0** in an array. For removal, this method shifts all values to one place to the left. Suppose we have the following array:



To remove the first key, we will first store that value and then shift all values one place to the left as given below:



The resulting array after removal would be as given below:



4. **copyArray** Method **[2]**

   This method copies all values from the argument array to a new array. It then returns the newly copied array.

5. **arraysEqual** Method **[2]**

This method returns **true** if both the argument arrays have the exact same values. Otherwise, returns **false.**

Question: 2

Program Name: MyBookList.java [20]

You are being asked to write a program which will keep the list of books. The program will keep record for the book's ***`title, price,`*** and ***`pages.`*** You are required to use the ***`two-dimensional String array`*** for this purpose.

The array would have ***`n rows and 3 columns`*** where **n** is the size entered by user. The first, second, and third columns would be used for storing a book's **title, price** and **pages** respectively. We will also treat **price** and **pages** as a **String.** This array should be of this form:

Introduction to AI	**1200.00**	**450**
Reinforcement Learning	**1300.99**	**1000**
Deep Learning: Practical	**1575.60**	**850**

The program when run should do the following in sequence:

1. Ask the user for the ***`size`*** of the array, that is number of rows, and validate the size to ensure that it is greater than or equal to 5.
2. Display the menu as follows:
   1. Press 0 to Exit

2. Press 1 to Add a New Book
3. Press 2 to Search by Title
4. Press 3 to Search by Price
5. Press 4 to Sort by Title
6. Press 5 to Sort by Pages
7. Press 6 to Display the Most Expensive Book
8. Press 7 to Display All Books

3. The choice entered by user must be between **0** to **7**. If the user enters any other number, ask them to enter the choice again.

4. The program should display the menu after every selection. For example, if user press **4**, the program should display the menu again after sorting the books by name.

5. There should be a method to perform each of the tasks above.

The details for each menu item are as follows:

- **Add a New Book**

  Asks the user to enter the **title, price,** and **pages** for the book, then add the book into the array.

  Hint: You need to keep the record that how many books have been added to the array. For this purpose, you can use an integer variable which will work as a *counter*.

  You may use the following header for this method:

  ```
  static boolean add(String[][] books, int currentSize)
  ```

  Note that the return type of the method is **boolean**, it means that if **currentSize** is less than the length of the array, then a new book can be added and the method should return **true** after adding the book to the array, or it will return **false** if the array is full.

  You should display **"The list is full. No more books can be added.",** if the list gets full.

- **Search by Title**

  Ask the user for the title (name) he/she wants to search for, then call the method which searches the array for the title and return the index where the title of the book is found. Returns -1 otherwise.

  Hint: The **title** is stored in the **first** column, so you only need to search in every first column of the array only.

  You may use the following header for this method:

  ```
  static int searchByTitle(String[][] books, int currentSize,
                                                  String key)
  ```

- **Search by Price**

  Ask the user for the price he/she wants to search for, then call the method which searches the array for the price and return the index where the price of the book is found. Returns -1 otherwise.

  Hint: The **price** is stored in the **second** column, so you only need to search in every second column of the array only.

You may use the following header for this method:

```
static int searchByPrice(String[][] books, int currentSize,
                                                String key)
```

- **Sort by Title**

  Sort the array by the **title** in ***ascending order***.

  NOTE: Sort by Title does not mean that you sort the titles only, the price and pages associated with each title must also be sorted. For example, consider the following list of books:

  Introduction to AI 1200.00 450

  Reinforcement Learning 1300.99 1000

  Deep Learning: Practical 1575.60 850

  When we sort this list by title, it will be changed to the following:

  Deep Learning: Pratical 1575.60 850

  Introduction to AI 1200.00 450

  Reinforcement Learning 1300.99 1000

  You can see that the positions of price and pages will also change along with the title.

  You may use the following header for this method:

```
static void sortByTitle(String[][] books, int currentSize)
```

- **Sort by Pages**

  Sort the array by pages of the book in ***ascending order***.

  Hint: Sort by Pages does not mean that you sort the pages only, the title, and price associated with each pages must also be sorted. For example, consider the following list of books:

  Introduction to AI 1200.00 450

  Reinforcement Learning 1300.99 1000

  Deep Learning: Practical 1575.60 850

  When we sort this list by pages, it will be changed to the following:

  Introduction to AI 1200.00 450

  Deep Learning: Practical 1575.60 850

  Reinforcement Learning 1300.99 1000

You can see that the positions of title and price will also be changed along with pages.

You may use the following header for this method:

```
static void sortByPages(String[][] books, int currentSize)
```

- **Display the Most Expensive Book**

  Displays the most expensive book from the list. For example, consider the following list of books:

  Introduction to AI 1200.00 450

  Reinforcement Learning 1300.99 1000

  Deep Learning: Practical 1575.60 850

  When we select the option, it displays the following output:

  Deep Learning: Practical 1575.60 850

  You may use the following header for this method:

```
static void showMostExpensive(String[][] books, int currentSize)
```

- **Display All Books**

  Displays all the books as following:

Title	Price	Pages
Introduction to AI	**1200.00**	**450**
Reinforcement Learning	**1300.99**	**1000**
Deep Learning: Practical	**1575.60**	**850**

  You may use the following header for this method:

```
static void displayAllBooks(String[][] books, int currentSize)
```

Useful String Methods:

You may find the following **String** methods useful. They will help you in searching and sorting.

NOTE: You can see the code in the **`StringMethodExamples.java`** file for examples.

- ***`equals`*** method

  The String **equals** method takes another String value as an argument and compares both Strings (caller and argument) based on their Unicode values. It returns **true** if the caller String is equal to argument String, or **false** otherwise.

  For example:

```
String str = "Java";
boolean result = str.equals("Horse");
```

  The value of the result will be **false** because "Java" is not equal to "Horse".

  Take another example:

```
String str = "Java";
boolean result = str.equals("Java");
```

  The value of the result will be **true** because "Java" is equal to "Java".

- ***`compareTo`*** method

  The String **compareTo** method takes another String value as an argument and compares both Strings (caller and argument) based on their Unicode values. The method does the following:

  `Returns 0` if the values of both caller and argument String objects is same.

  `Returns a positive value` if the value of caller String object is greater than the value of argument String object.

  `Returns a negative value` if the value of caller String object is less than the value of argument String object.

  Examples:

```
String s1 = "java";
String s2 = "java";
String s3 = "nava";
String s4 = "jasa";

//Returns 0, because both are equal
int result = s1.compareTo(s2);
System.out.println(result);

//Returns -4, because "j" is 4 times lower than "n"
result = s1.compareTo(s3);
System.out.println(result);

//Returns 3 because "v" is 3 times greater than "s"
result = s1.compareTo(s4);
System.out.println(result);
```