Program 3 – Shape Database 3.0

The purpose of this assignment is to practice utilizing the various concepts learned in Chapters 10, 11, 12 (from CPS141) and 13, 15 and 16 of our text. Specifically, this assignment will require use of the following concepts.

- Functional Interfaces (10.11)
- Exception Handling (11.1)
- Label, Button, and TextField Controls (CPS141 - 12.2)
- Scene, VBox, HBox Controls (CPS141 - 12.3, 12.5)
- Event Handlers (CPS141 - 12.8)
- JavaFX and CSS (13.1)
- RadioButton Controls (13.2)
- ListView Controls (13.4)
- ComboBox Controls (13.5)
- Menus (13.8)
- Sorting Algorithms (16.1)
- Binary Search (16.2)

The program will implement a simple graphical user interface for the Shape database from Program 1 and will allow the user to display Shape information for all valid shape records in the database and records for specific shape types.

The Specification for Program 02 is incorporated by reference and this assignment will address only the updates required to support the new sorting features. This means that the program must work as specified in Program 02 PLUS include the new features described here.

1 User Interface Specifications

This section corresponds to Item 3 of the "Program Rubric" and specifies how the user should interact with the program. A new feature for sorting the Shape objects displayed in the Shape Selector ListView will be implemented using RadioButtons to allow the user to either sort Shapes by area either in ascending (smallest to largest) or descending (largest to smallest) order A second new feature will allow the user to search the Shapes displayed in the Shape Selector ListView by area value. The search will be conducted on the filtered list of exoplanets displayed in the center section of the application. A third and final UI update involves using 5 decimal places for all numeric values. See Figures 1 and 2 in Section 4, for examples of how these updates could look.

1.1 Sort Feature

The new Sort Feature should include the following UI Controls and Actions.

1. A "Sort by:" Label in the Left Colum between the telescope selector and the Statistics section.
2. Two RadioButtons including:
    a. "Smallest to Largest" (ascending) order – when selected causes Shapes in the ListView in the center panel to be sorted by area with the smallest area listed first.

b. "Largest to Smallest" (descending) order – when selected causes Shapes in the ListView in the center panel to be sorted by area with the largest area listed first.

3. When the program starts, the "Smallest to Largest" RadioButton should be selected and the Shapes in the ListView in the center panel should be sorted by area in ascending order.

1.2 Search Feature

The new Search Feature should include the following UI Controls and Actions.

1. A "Search by Area" Label in the Right Control Colum under the Selected Shape Section
2. A TextField control in which the user may type a name to search.
3. A "Go" Button to start the search. You are free to choose an appropriate name for this button.
4. A Label under the "Go" button in which to display the search results.
5. At program start both the search field and the results should be blank.
6. Successful search results should display the Shape information as in the Selected Shape results Section using the `toString` method of Shape.
7. Unsuccessful search results should be indicated in the search results area with an appropriate message.
8. Whenever the filtered list of Shape objects in the center section is changed, the search input and result labels should be cleared (i.e., the text set to empty strings "").

1.3 Decimal Places

Update all numeric displays to use 5 decimal places including in the ListView Shape Selector in the center panel and in the Shape class toString method. The reason for this update is to see more clearly and accurately the area values when searching.

2 Java Implementation

This section corresponds to Item 4 of the "Program Rubric" and describes the details of how the program should be implemented to achieve the User Interface experience of Section 1. Section 2 of the Program 2 assignments is incorporated by reference (i.e., your program must meet all specifications from Program 2). However, greater weight will be placed on implementation of the new sort and search features.

You are free to implement the user interface described in Section 1 in any way you choose as long as the following requirements are met:

1. The program must be written using only the controls and classes we have learned in class. Do not utilize any libraries or classes we have not covered in our course.
2. The program must be written using the specific JavaFX Controls given in Section 1.
3. The program must implement all user-interface features described in Section 1.
4. The program must be user friendly and functionally correct.

5. The program must implement and utilize any specific classes or interfaces described in the following sections as part of the overall implementation.
6. A "throws" clause must not be written in main method header. All exceptions related to reading data from the database file should be handled internally by the program.

2.1 Shape.java

Update the Shape class you created in Program 01 and reused in Program 02 according to the following specification. Make any corrections and updates that are needed to get it to work with this program. Refer to the Program 01 specifications for details on the original specification of this class. Additional specifications for Program 03 include the following:

2.1.1 Implement Comparable<Shape> Interface

Update the Shape class header to implement the `Comparable<Shape>` interface in addition to the Boundable interface it already implements.

2.1.2 Implement the compareTo method required by Comparable <Shape> interface

```
+compareTo(other : Shape) : int
```

Write the required compareTo method using the following algorithm:

```
Algorithm compareTo(other)
If Math.abs(this.area() – other.area()) <= 0.00001
       return 0
endIf
return Double.compare(this.area(), other.area())
```

2.2 Sorter Class

Create a Sorter class with a single `public static` method called `sort` with the method header shown below. Implement this method using your choice of one of the four sorting algorithms we covered in class including, Bubble Sort, Insertion Sort, Quicksort, or Selection Sort. NOTE: The book uses int[] array types. See the discussion on page 1033 on how to use Comparable[] array types.

```
public static void sort(Comparable[] array)
```

NOTE: IGNORE ANY COMPILER WARNINGS THIS METHOD HEADER GENERATES. WE WILL DEAL WITH A WAY TO PREVENT THEM IN A FUTURE ASSIGNMENT. DO NOT USE ANY CORRECTIONS SUGGESTED BY ECLIPSE FOR THIS METHOD HEADER.

2.3 Searcher Class

Write a Searcher helper class with a single public static method called search using the method header below. Implement the search method using either the recursive BinarySearch algorithm (Section 15.4) or the iterative BinarySearch algorithm (Section 16.2).

```
public static int search(Comparable[] array, Comparable toFind)
```

NOTE: IGNORE ANY COMPILER WARNINGS THIS METHOD HEADER GENERATES. WE WILL DEAL WITH A WAY TO PREVENT THEM IN A FUTURE ASSIGNMENT. DO NOT USE ANY CORRECTIONS SUGGESTED BY ECLIPSE FOR THIS METHOD HEADER.

2.4 Program03 - Main Class File

Use the file you wrote for Program 2 as the starting point for this assignment. Make any improvements or corrections that are needed to get Program 2 working properly. Include the following items in addition to any specific methods or items required in Program 3.

2.4.1 Sort Controls

Complete the following for the new Sort Controls.

1. Declare and initialize two RadioButton instances plus an appropriate Label instance for use in the Left Control Column of the interface. The first RadioButton should correspond to sorting in ascending (smallest to largest) order, and the second should correspond to sorting in descending (largest to smallest). The Label should indicate the purpose of these controls to the user. The variable names I used were `sortNormalRadioButton` and `sortReverseRadioButton`.

2. Use a ToggleGroup to ensure only one option is selected at a time.

3. Add the three new controls to the Left Control Column container.

4. For the Sort Ascending RadioButton, set an event handler that simply calls the handleShapeFilter.update() method.

5. For the Sort Descending RadioButton, set an event handler that simply calls the handleShapeFilter.update() method.

2.4.2 Sort private static method

Add the following method to your Main class after the loadData method. This method takes an ArrayList<Boundable> object and returns another one that is a sorted version of the first. It also takes a "reverseOrder" argument, which if set to true, will cause the array to be sorted in reverse of the natural order.

```
// NEW FOR PROGRAM 3 - Method for sorting and array of Boundable objects, forward or reverse
private static ArrayList<Boundable> sort(ArrayList<Boundable> unsorted, boolean reverseOrder) {

	// Convert the ArrayList to a regular Java array
	Shape[] sorted = new Shape[unsorted.size()];

	// Copy all of the the items in the the array
	for (int i=0; i<sorted.length; i++) {
		sorted[i] = (Shape)unsorted.get(i);
	}

	// Sort the Java array using the preferred method.
```

```
        Sorter.sort(sorted);

        // Created the final result array
        ArrayList<Boundable> result = new ArrayList<Boundable>(sorted.length);

        if (reverseOrder) {
                // Copy elements in reverse order
                for (int i=sorted.length-1; i>=0; i--) {
                        result.add(sorted[i]);
                }
        } else {
                // Copy elements in natural order
                for (int i=0; i<sorted.length; i++) {
                        result.add(sorted[i]);
                }
        }

        return result;
}
```

2.4.3 UpdateHandler - `handleShapeFilter`

Inside the handleShapeFilter Lambda expression, use the following code to filter and sort the array of shapes. This line of code wraps the original filter code inside a call to the sort method created above.

```
// Get the filtered array and keep for future selection.
// NEW FOR PROGRAM 3 - use the sort method to sort the filtered values.
shapesFiltered = sort(filter.filter(shapes,predicate),sortReverseRadioButton.isSelected());
```

2.4.4 Shape Search Controls

Complete the following to implement the new Sort feature.

1. Declare and initialize the following JavaFX controls
    a. Label for the "Search by Area" text
    b. TextField for the name search user input, called `shapeSearchTextField`
    c. Button to execute the search using the name entered by the user called `shapeSearchButton`.
    d. Label to display the search results called `shapeSearchLabel`.

2. Add the four new controls in an appropriate location in the Right Control Column container.

3. The search TextField (1.b) and the search results Label (1.d) must be cleared (i.e., .setText("")) each time your filtered array of Exoplanet objects is updated. Write the code to clear these fields in your "handleShapeFilter" UpdateHandler.

2.4.5 Shape Search Button Handler

Use the following code, or modify to match your variable names, to handle the search feature event when the user presses the search button. This code will call the Sorter.sort method and Searcher.seach methods that you wrote above. Place this code after the other setOnAction handlers in your code in the getShapeInfoScene method.

```
// NEW FOR PROGRAM 3 - add search button handler
// search button handler
shapeSearchButton.setOnAction(event -> {

	// 1. Get the area value to find.
	double areaToFind = Double.parseDouble(shapeSearchTextField.getText());

	// 2. Create an instance of a Comparable<Shape> with the given area
	Shape objectToFind = new Shape(ShapeType.ABSTRACT) {

		@Override
		public double perimeter() {
			// value doesn't matter since we're not searching for perimeter, return 0.
			return 0;
		}

		@Override
		public double area() {
			return areaToFind; // return the area we want to find
		}

	};

	// 3. Create a regular Java array of Shape objects the same size as shapesFiltered
	Shape[] searchArray = new Shape[this.shapesFiltered.size()];

	// 4. Copy all of the the records in shapesFiltered into the search array
	for (int i=0; i<searchArray.length; i++) {
		searchArray[i] = (Shape)this.shapesFiltered.get(i);
	}

	// 5. Sort the searchArray using the preferred method
	Sorter.sort(searchArray);

	// 6. Search for the value using the Searcher
	int found = Searcher.search(searchArray, objectToFind);

	// 7. if the found index is less than 0 then display an appropriate not found message
	if (found < 0) {
		shapeSearchLabel.setText("Not found");
	}

	// 8. Otherwise display the shape at the found index of searchArray
	else {
		shapeSearchLabel.setText(searchArray[found].toString());
	}
});
```

3 Deliverables

The deliverables for this project will consist of the following .java and .css files.

1. Program03.java – or whatever you name your main program.
2. Shape.java – Your updated Shape class file
3. Sorter.java - Your new Sorter class file
4. Searcher.java – Your new Searcher class file
5. Include all files from Program 2 with any updates or corrections you may have written.
6. Optionally, if you do not include the files from Program 2, I will reuse the ones you previously submitted. Do not use this option if updates or corrections were needed.

DO NOT SEND THE .class FILES. I CANNOT USE THEM.

Materials that are Provided

Use the materials already provided with Programs 01 and 02. No additional materials are provided for Program 03.

The due date/time is:
Saturday, August 5, 2023, at 11:59PM

4 Sample Output

The screenshots below demonstrate some possible examples of the program user interface.

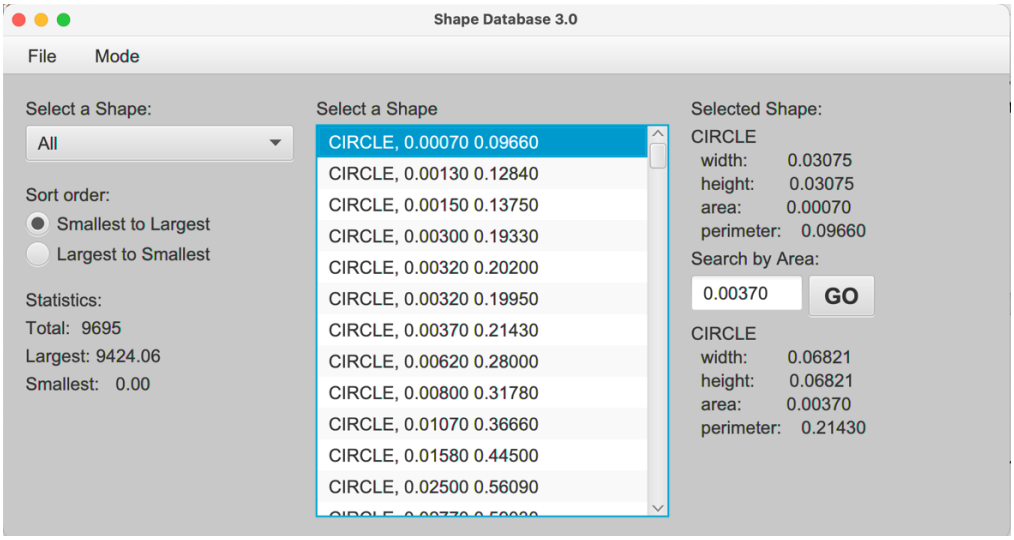

Figure 1, Shape Database 3.0 - Light Mode

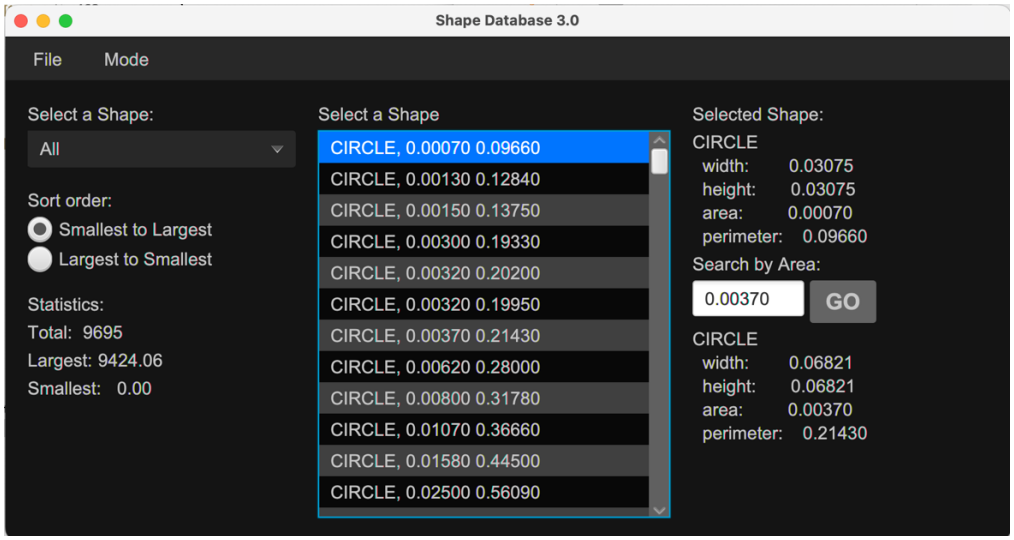

Figure 2, Shape Database 3.0 - Dark Mode

5 General Submission Policy

You may submit your code multiple times up until the due date/time. Any submission after the due date/time will be subject to the late assignment policy described in the course Syllabus. In all cases the last code you submit will be used for the grade.

6 General Requirements

All code you write for this course should meet the following general requirements:

1. All code should be your own work and not copied from another student, the internet, or any solution guide for the textbook.

2. Code from the following sources may be used or modified to complete the project:
    a. Any code provided by me in the class notes.
    b. Any code provided by the textbook itself (excluding any solution guides).
    c. Any code written by you for this or another course.

3. All Java classes should have a documentation comment including, author, date, course (i.e., CPS142-Summer 2023), and a purpose or brief description statement.

4. All methods must have properly formatted and descriptive documentation comments. The only exception is methods that override well documented interface or superclass methods. Private methods must also be commented.

5. Any method with more than just a few lines of code should have additional single-line comments describing important parts of the algorithm.

6. The use of break, continue, or exit statements inside loops is not acceptable in the implementation of any class. If found points will be subtracted from Item 5 of the Programming Rubric.

NOTE: The above requirements should be followed only where applicable.

7 Program Rubric

Your grade for this assignment will be computed using the following rubric.

Item	Description	Percent of Grade
1	Compilation: Compiles with no syntax errors.	10
2	Execution: Runs with no crashes or runtime errors.	10
3	External Operation: Written to meet all requirements for input and output as specified or implied by the problem.	20
4	Internal Operation: Written to meet all requirements for internal operation as specified as specified or implied by the problem statement.	40
5	Coding Style: Written and formatted in accordance with best practices presented in class (indentation, class names, variable names, method names, etc.).	5
6	Project Comment: Contains a **documentation comment** with the assignment name, author, date, class (i.e., CPS142-Summer 2023), and purpose/description at the top of each file.	5
7	Documentation: All methods, classes, and interfaces have **documentation comments**. Methods that override well-documented interface method and the main method are exempt.	5
8	Comments: Includes sufficient other regular comments to describe important parts of the code.	5
	Total	**100**