# Lab 2: Comparing Objects, Searching and Sorting

## Background

In this assignment, you'll explore ways to search and sort objects which usually requires the ability to compare 2 objects against each other:

- The **Comparable** interface provides a default way to compare two objects of the same type, by implementing the compareTo method.
- The **Comparator** interface allows you to define multiple ways to compare objects, by implementing the compare method.

**Note:** The compareTo method should only be used to sort instances of the same class, while the compare method can be used to compare instances of different classes, or to impose various other comparison strategies.

## Instructions

1. This lab can be completed individually or in a group (there is no size limit for the group), complete the three exercises below using the Lab2-StartingCode provided by your instructor.
   - If working in a group, only one submission is required – **all group members' names must be included in the comments of the submission.**
     - All group members will receive the same grade.
     - It is your responsibility to manage the communication, cooperation and contribution of all group members.
2. See the *Marking Criteria* section below for details on how you will be assessed.
   - If there are more than one submission, both individually or across all group members, only the latest submission before the deadline will be accepted.
3. Submit your completed **zipped** exported Eclipse project to Brightspace by the posted due date.

### Exercise 1

1. Implement a class **Student** with the fields **name** and **age**.
2. Implement the Comparable and Comparator interfaces to compare students based on their name and age.
   - The compareTo method, defined by the Comparable interface, compares students based on their name.

- The compare method, defined by the Comparator interface, compares students based on their age. **If the ages are equal**, it then compares based on their name.
- **Note:** The Student class implements the Comparable interface, while the Comparator interface will be implemented as an external class to allow for the two different ways to compare students.

3. To test the implementation, use the list of Student objects provided in the exercise1 package and sort it using the Collections.sort method.
4. To ensure the correct functionality, display the list of Student objects with both its name and age before and after it has been sorted.

## Exercise 2

Implement a static method binarySearch that takes the sorted list of integer objects provided in the exercise2 package and an integer target that is prompted from the user when the program first runs, and returns the index of the target in the list if it exists, or -1 if it doesn't.

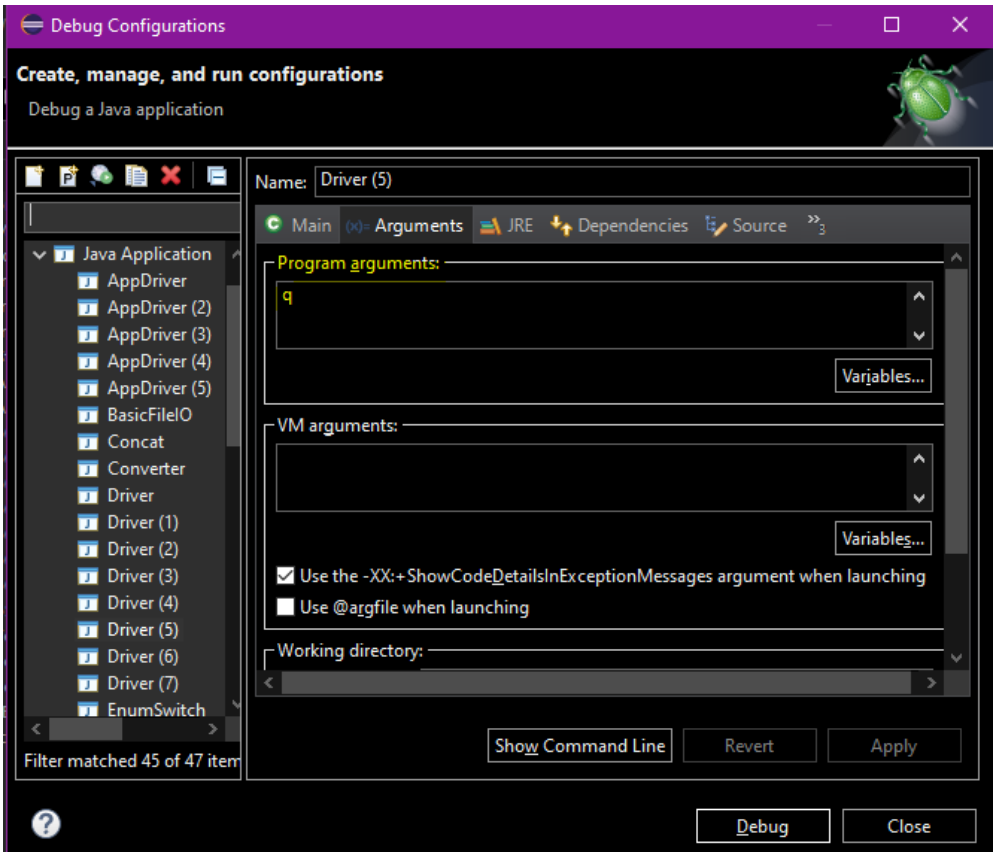
**Note:** You must develop this algorithm yourself, do NOT use any of the binarySearch() method from the Java library!

## Exercise 3

Write a program in Java that sorts an array of integers using **ONE** of following four different sorting algorithms (Bubble Sort, Insertion Sort, Selection Sort or QuickSort).

- Your program should read the choice of sorting algorithm from the command line that is passed into the main method to choose a sorting algorithm:
    - The valid inputs would be the characters: **b**, **i**, **s** or **q**
    - You can assume that only one single character will be passed into your main method through the command line
        - No error checking is needed for this exercise

  You can test this command line in Eclipse using the "Run Configurations" tool under the "Arguments" tab:




- The selected algorithm then sorts the array of integers in **descending order** created in the starting code.
    - The array should be ordered from largest to smallest after the sort
        - Index 0 will the largest integer and index n-1 is the smallest integer.
    - Sorting in ascending order and then reversing the array is NOT acceptable!
- Since your program will only have one algorithm implemented, the other 3 will simply do nothing!

**Note:** If more than one algorithm is implemented, you will not receive extra marks and it is up to the instructor's discretion which will be graded!

- Before and after sorting using any of the sorting algorithms, the program should output the contents of the array in the form of a string.

## Marking Criteria

| **Criteria** | **Missing** (0%) | **Needs Improvement** (0-50%) | **Good** (51-75%) | **Excellent** (76-100%) | **Marks** |
|---|---|---|---|---|---|
| **Exercise 1** | Not submitted | Significant components are missing | Not all components function as expected | All components are fully functional | **/6** |
| **Exercise 2** | Not submitted | Significant components are missing | Not all components function as expected | All components are fully functional | **/4** |
| **Exercise 3** | Not submitted | Significant components are missing | Not all components function as expected | All components are fully functional | **/10** |
| | | | | **Total** | **/20** |