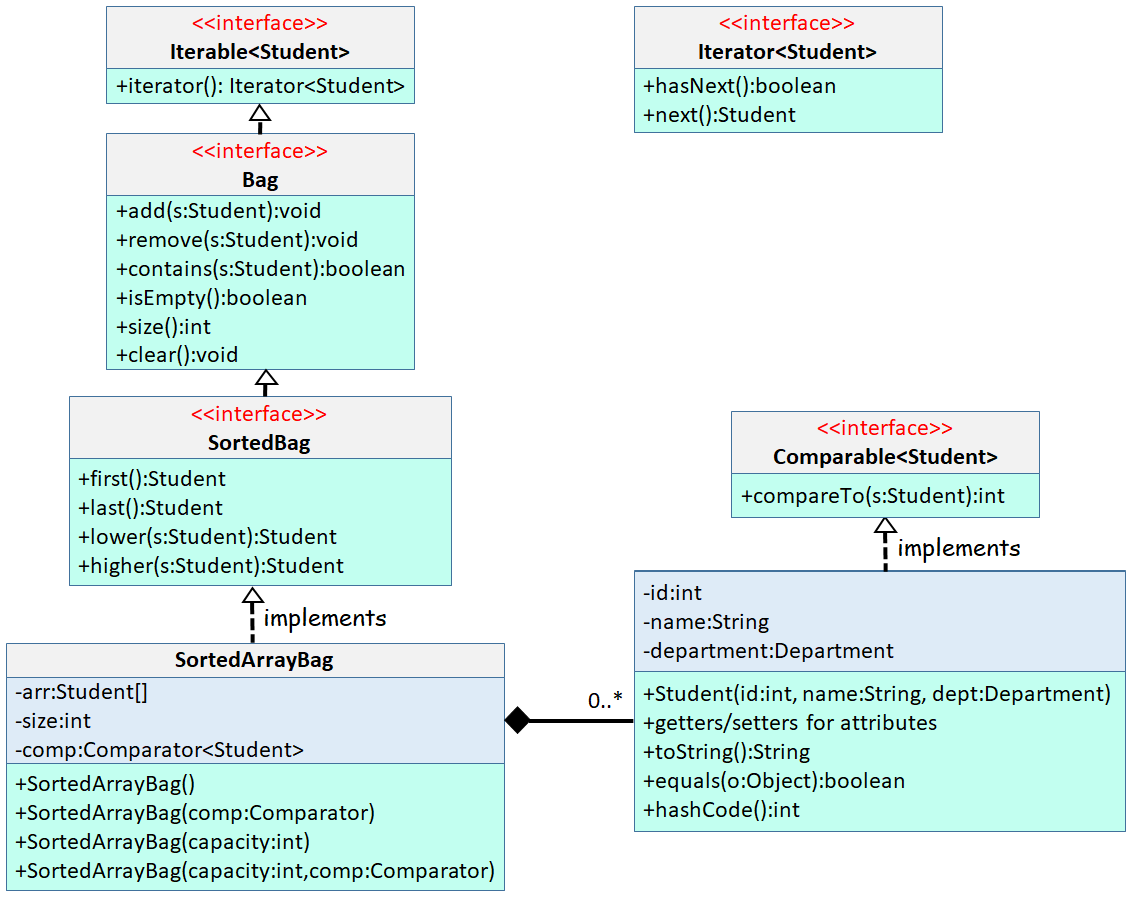
Lab 22, 23 & 24

**Rules:**

1. You are given the following files: **Department.java**, **Bag.java**, **SortedBag.java**, **BagFullException.java**, and **Test.java**, and are asked to implement the following files: **Student.java** and **SortedArrayBag.java** as described below.
2. Do not forget to take your work with you when you leave the lab by either copying your work files to your own USB flash disk, or by e-mailing them to yourself.

In this project you will extend your container (a.k.a., collection) for Student objects so that the students are stored inside the container in *some sorted order*. The new container called **SortedArrayBag** implements the **SortedBag** interface, which extends the **Bag** interface, which extends Java **Iterable** interface for data structure-free iteration over bag elements. Here is the UML class diagram of the classes you will be implementing in this Lab:



**Student Class**

You will first implement a Student class that consists of three fields: id, name and department. Department is an **enum** given to you (**Department.java**). This time, the Student class must implement Java **Comparable** interface to give a default ordering of students. By default, we want the students to be ordered in ascending of their “id”s. Here is the list of methods you must implement for Student:

|  |  |
| --- | --- |
| Student(int id, String name, Department dept) | Parametrized constructor for Student. Throw an IllegalArgument exception if id <= 0, name is null or name has less than 2 chars. |
| int getId() | Getter for id |
| Student setId(int id) | Setter for id: Sets the id to the user-supplied value, and returns a reference to the object for method chaining. If id <= 0, throw an IllegalArgumentException() |
| String getName() | Getter for name |
| Student setName(String name) | Setter for name: Sets the name to the user-supplied value, and returns a reference to the object for method chaining. If name is null or has a length < 2, throw an IllegalArgumentException() |
| Department getDepartment() | Getter for department |
| Student setDepartment(Department dept) | Setter for department |
| String toString() | Returns a nice string representation of the object |
| boolean equals(Object o) | Returns true if the student is equal to the user-supplied Object “o”. Two students are equal if their id, name and departments are the same. |
| int hashCode() | Returns the hashCode of the student. The hashCode for two equal students must be the same. |
| int compareTo(Student s) | Used to give a default ordering for students. By default, we want the students to be ordered in ascending order of their “id”s. That is, a student with a smaller “id” must come before a student with a larger “id”. |

**Bag Interface**

**Bag** is an interface that extends Java **Iterable** interface so that we can iterate over the elements of a Bag in a data structure independent manner. In addition to the iterator() method inherited from Java Iterable interface, here is the list of methods you must implement in the **Bag** interface.

|  |  |
| --- | --- |
| void add(Student s) | Adds the given student to the Bag. Duplicates are allowed. |
| void remove(Student s) | Removes the student from the Bag if it exists. If the bag does not contain “s”, do nothing |
| boolean contains(Student s) | Returns true if the bag contains “s”, false otherwise |
| boolean isEmpty() | Returns true if the Bag is empty, i.e., there are no students in the bag |
| int size() | Returns the number of students in the bag |
| void clear() | Removes all students from the bag and makes it empty |

As you can see, **Bag** interface defines what is called an **Abstract Data Type (ADT)**: We know how the operations should work, but we do not know how they should be implemented. Usually, there is more than one way to implement the ADT operations.

**SortedBag Interface**

**SortedBag** is an interface that extends **Bag** interface so that we can store the student objects in *some sorted order* inside the container. It has the following 4 additional methods:

|  |  |
| --- | --- |
| Student first() | Return the first student in sorted order. If the container is empty, return null |
| Student last() | Return the last student in sorted order. If the container is empty, return null |
| Student lower(Student s) | Returns the student that comes **before** the user-supplied student “s” in sorted order. For example, assume that we are using the default ordering with respect to “id”, and the container has students with ids 3 and 5. If we call lower with a student “s” whose id is 4, then the student that comes **before** “s” is the one with “id” 3. If there is NO student that comes **before** the user-supplied student “s”, then return null |
| Student higher(Student s) | Returns the student that comes **after** the user-supplied student “s” in sorted order. For example, assume that we are using the default ordering with respect to “id”, and the container has students with ids 3 and 5. If we call higher with a student “s” whose id is 4, then the student that comes **after** “s” is the one with “id” 5. If there is NO student that comes **after** the user-supplied student “s”, then return null |

**SortedArrayBag class**

In this Lab you will be using an array to implement the **SortedBag** operations, therefore the name **SortedArrayBag**. The idea is to allocate an array of size “capacity”, which is a user-supplied value >= 4. In order to keep track of the number of students in the Bag, **SortedArrayBag** class has an attribute named “size”, initially 0, which keeps track of the current number of students in the Bag. SortedArrayBag can have an external Java **Comparator** passed in during object creation. If that is the case, then you must use this comparator to order the students inside the array. If the user does not supply any custom comparator objects, then you must use the default comparator of the Student objects, which orders the students in ascending order of their “id”s. Here is how you should be implementing each method in **SortedArrayBag**:

|  |  |
| --- | --- |
| SortedArrayBag() | No args constructor: Create an array of capacity 4. Since the user does NOT pass a custom comparator, use the default Student ordering. |
| SortedArrayBag(Comparator comp) | Create an array of capacity 4. Use the user-supplied comparator to order the students inside the array. |
| ArrayBag(int capacity) | If the user-supplied capacity value < 4, set it to 4. Since the user does NOT pass a custom comparator, use the default Student ordering. |
| ArrayBag(int capacity, Comparator comp) | If the user-supplied capacity value < 4, set it to 4. Use the user-supplied comparator to order the students inside the array. |
| void add(Student s) | If the array is already full, throw a BagFullException exception, and do not add the student to the bag. Otherwise, find the location inside the array where this student must be inserted using the current ordering criteria. Then shift all elements that come after this student one position to the right to open up a cell to insert the new student. Finally, put the new student in the opened location. Increment size by 1. |
| void remove(Student s) | Find the index where the student is located. If the student does not exists in the array, do nothing. Otherwise, move all students that come after this student one position to the left. Decrement size by 1. |
| boolean contains(Student s) | Return true if the user-supplied student exists in the array. |
| boolean isEmpty() | Return true is size == 0 |
| int size() | Return the size of the array, i.e., the current number of students in the array |
| void clear() | Clean references to all students, and set size to 0 |
| Iterator<Student> iterator | Return a Java Iterator so that we can iterate over the elements of the ArrayBag. The methods you have to implement in Java Iterator interface are shown in the UML diagram. Specifically, you must implement hasNext() and next() methods. |

**Test class**

To test your **Student** and **OrderedArrayBag** classes, we have given you a test code Test.java that creates a number of students, creates a **SortedArrayBag**, and tests all methods of **SortedArrayBag** using the created students. The test code prints your Lab grade at the end. You are advised to implement your own test code. When grading, we may use a different Test. Make sure that your code works under all circumstances.

Lab Work Submission:

* You can continue to work on this lab after our lab class, on your own, at home.
* Submit your lab work via Blackboard on or before: **Wednesday, November 29, 2023, 11:59pm**.
* The only accepted submission method!
* Once you submit your assignment you will not be able to resubmit it!
* Make absolutely sure the Java files you want to submit are the Java files you want graded.
* You will not be able to submit your lab work under any circumstances once **Lab22** disappears at **12:00 a.m.** on **Thursday, November 30, 2023**.
* There will be **NO** exceptions to these rules!
* To submit your lab work, upload **Student.java & SortedArrayBag.java** files (**with .java extension**) you did for this lab to the **Lab22** assignment in the **Labs** tab in your Lab section’s presence in Blackboard.
* Then, make sure you click the **Submit** button to submit your lab work.
* This lab is worth **20 points**.