C S 272/463 Introduction to data structures
Fall 2019
Lab 3: Collection class

1 Learning objectives

Objective 1 (collection), Objective 5, Objective 6, Objective 7

2 Requirements

In lab1, you use a fixed-length array to store all the employees. This approach has one disadvantage: when the array is full, we cannot add more employees. In this lab, based on what we have learned in class, you are asked to design a data structure `EmployeeSet`, which can be used to store all the employee information. For employees, you are required to use your `Employee` class definition in Lab2

You are required to design a data structure `EmployeeSet` to store employees and support basic operations.

This `EmployeeSet` data structure should implement the functionality of a collection whose space can grow automatically. This collection data structure does NOT allow you to store the same employee multiple times. Two employees are considered same if they have the same employee nos. I.e., if the collection already contains one `Employee` object `o1`, you cannot add another `Employee` object `o2` whose employee no is the same as `o1`'s employee no.

The class `EmployeeSet` should define proper instance variables and implement the following methods.

1 (5 pts) This class should include proper instance variables to keep all the distinct employee objects and the actual number of employees.

2 (5 pts) A no-argument constructor, which initializes an `EmployeeSet` instance, sets its capacity to 10, and creates an array to store 10 `Employee` objects.

```
public EmployeeSet()
```

3 (10 pts) A copy constructor, which creates a new `EmployeeSet` instance and copies the content of the given object to the new instance.
The precondition is that `obj` should NOT be `null` and should be an instance of `EmployeeSet`.

```
public EmployeeSet(Object obj)
```

4 (5 pts) The following method which returns the actual number of elements in this collection.

```
public int size()
```

5 (5 pts) The following method which returns the capacity of this collection instance.

```
public int capacity()
```

6 (10 pts) A method which adds one given `Employee` object to the first available space of the employee array in this `EmployeeSet` instance. When the collection space is sufficient to hold the new employee, this employee object can be directly added to the collection. Otherwise, you need to double the space of the instance array by implementing a method `ensureCapacity` (defined below).
The precondition is that the employee object `a` should NOT be `null`.

```
public void add(Employee a)
```

7 (10 pts) A method to check whether this collection contains the input parameter.

- If `a` is a `null` object, directly return false.
- If `a` is not a `null` object,
  - if this collection contains an employee object, which equals to `a`, then return true;
  - Otherwise, return false.

```
public boolean contains(Employee a)
```

8 (10 pts) A method to remove from the collection the employee with the given employee no `eno`.

```
public boolean remove(int eno)
```

9 (10 pts) The following method which guarantees the capacity of the collection.
If this collection's capacity is smaller than the input parameter, this method sets the capacity to `minimumCapacity` and enlarges the array to hold `minimumCapacity` objects;
Otherwise, this collection is left unchanged.
The precondition is that the input parameter `minimumCapacity` should be positive.

```
private void ensureCapacity(int minimumCapacity)
```

10 (15 pts) A method which adds one employee object to this `EmployeeSet` instance such that the objects in the employee array are in ascending order of employee nos. When the collection space is sufficient to hold the new employee, this employee object can be directly added to the collection. Otherwise, you need to double the space of the instance array by calling the method `ensureCapacity`.
The preconditions are that

- the employee object `a` should NOT be `null`, and
- the objects in the collection's employee array are already in ascending order of employee nos.

```
public void addOrdered(Employee a)
```

11 (15 pts) `main()` method to thoroughly test your code.
Design test cases, put them in your main method, run your program through the test cases. One test case need to read all the employee information from the data file (`https://www.cs.nmsu.edu/~hcao/teaching/cs272/lab/lab1/core_dataset.csv`) and add those employees to an employee set. You should not use an array with fixed size to keep the employee information as you did in Lab 1. You cannot use any Java built-in collections (e.g., ArrayList) either.

```
public static void main(String[] args)
```

3 Note

- **Specifications** for all your classes and methods:
  Please properly explain (1) the functionality of the methods, (2) the parameters, (3) the return values, (4) the pre-conditions if there is any;
  Please use inline comments, meaningful variable names, indentation, formatting, and whitespace throughout your program to improve its readability.
- You can (but are not required to) design and implement other facilitating methods (E.g., other get and set methods, toString method) to finish the implementation of the required methods.

4 Submission

Submit through canvas a zipped file containing your java file(s) (not `.class` files).

5 Grading Criteria

(1) The score allocation is beside the questions.

(2) Please make sure that you test your code **thoroughly** by considering all possible test cases. Your code may be tested using more test cases.