Java Fundamentals Homework

# The Objective

The objective of this homework is to empower your knowledge and understanding of Java Collections framework. You will be using Collection interfaces and their implementation for solving a wide range of technical tasks at Endava.

Writing your own collection implementation is the best way to learn quickly what collections are capable of. You will obtain a very good practice with java interfaces, nested classes, generics, flow control, data types and other.

# The assignment

Each intern has the assignment to write an implementation of java collection similar to the well-known archetypes of java collections framework by implementing one of the interfaces: List, Set, Map, Deque.

|  |  |  |
| --- | --- | --- |
| **Person** | **Interface** | **Implementation** |
| Alexandr Medvedev | Deque | ArrayDeque |
| Alexandru Chetroi | List | ArrayList |
| Alexandru M Popa | Map | HashMap |
| Alexei Lucas | Set | TreeSet |
| Andrei Haveriuc | Map | TreeMap |
| Dmitrii Severin | List | ArrayList |
| Ion Stavila | Map | HashMap |
| Maxim Colesnic | Set | TreeSet |
| Nicolae Nicolaev | Deque | LinkedList |
| Nicolae Spinu | Set | HashSet |
| Pavel Panasenco | List | LinkedList |
| Roman Petras | Map | TreeMap |
| Sergiu Siminiuc | List | LinkedList |
| Vasilii Samota | Map | HashMap |
| Veaceslav Canna | Set | HashSet |
| Victor Ciobanu | Map | TreeMap |
| Vitalie Roibu | Deque | ArrayDeque |
| Xenia Sapojnic | Set | HashSet |

## Details

You will create a collection that implements a specific interface. Your collection needs to internally function in a manner similar to the well-known implementations. For example: HashMap and ArrayList will internally have an array to store data. The data stored in your collection will be a Student object. Maps will store Student as key, and an arbitrary Object as a value (your choice, can be an Integer with value of Student’s age).

## The Student class

Your collection will store objects of Student class inside. You might need to make some adjustments for this class to properly work with your collection. Consider making the necessary adjustments.

package com.endava.internship;  
  
import java.time.LocalDate;  
  
public class Student //*TODO consider implementing any interfaces necessary for your collection*{  
 private String name;  
 private LocalDate dateOfBirth;  
 private String details;  
  
 public Student(String name, LocalDate dateOfBirth, String details) {  
 this.name = name;  
 this.dateOfBirth = dateOfBirth;  
 this.details = details;  
 }  
  
 public String getName() { return name; }  
  
 public LocalDate getDateOfBirth() { return dateOfBirth; }  
  
 public String getDetails() { return details; }  
  
 /\**TODO consider overriding any methods for this object to function properly within a collection:  
 1. A student is considered unique by a combination of their name and dateOfBirth  
 2. Student names are sorted alphabetically, if two students have the same name, then the older one is  
 placed before the younger student in an ordered student list.*  
 \*/  
}

# Collections overview and recommendations for homework

After creating the Student class in your project, you will declare your collection similar to this:

public class StudentList implements List<Student> {

In InteliJ idea you can generate automatically method declarations when implementing an interface with this shortcut: **CTRL+I**. Select all the methods available to implement. You do not need to override the interface default methods like Collection.stream(). Ignore interface default methods.

## List

public class StudentList implements List<Student> {  
  
 @Override  
 public int size() {  
 //*TODO* return 0;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean contains(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public Iterator<Student> iterator() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Object[] toArray() {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public <T> T[] toArray(T[] ts) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean add(Student student) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean remove(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean containsAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean addAll(Collection<? extends Student> collection) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean addAll(int i, Collection<? extends Student> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean removeAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean retainAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public void clear() {  
 //*TODO* }  
  
 @Override  
 public Student get(int i) {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student set(int i, Student student) {  
 //*TODO* return null;  
 }  
  
 @Override  
 public void add(int i, Student student) {  
 //*TODO* }  
  
 @Override  
 public Student remove(int i) {  
 //*TODO* return null;  
 }  
  
 @Override  
 public int indexOf(Object o) {  
 //*TODO* return 0;  
 }  
  
 @Override  
 public int lastIndexOf(Object o) {  
 //*TODO* return 0;  
 }  
  
 @Override  
 public ListIterator<Student> listIterator() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public ListIterator<Student> listIterator(int i) {  
 //*TODO* return null;  
 }  
  
 @Override  
 public List<Student> subList(int i, int i1) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
}

### ArrayList

The ArrayList should start with an internal array of length 10 unless specified differently in constructor. Once you add more elements than the internal array length, make a new array double the size, and copy the elements from the smaller array into it.

### LinkedList

The Linked List needs to be double linked. LinkedList implements both List and Deque intefaces. For this situation implement only the List portion.

## Set

public class StudentSet implements Set<Student> {  
 @Override  
 public int size() {  
 //*TODO* return 0;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean contains(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public Iterator<Student> iterator() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Object[] toArray() {  
 //*TODO* return new Object[0];  
 }  
  
 @Override  
 public <T> T[] toArray(T[] ts) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean add(Student student) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean remove(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean containsAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean addAll(Collection<? extends Student> collection) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean retainAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean removeAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public void clear() {  
 //*TODO* }  
}

### HashSet

You are allowed to solve hashCode collisions in whatever way you like, You can even use a standard LinkedList for that.

### TreeSet

TreeSet internals are similar to TreeMap internals. Most important thing about your TreeSet implementation: **IT’S OK FOR YOUR TREES IMPLEMENTATION TO BE UNBALANCED.** Unbalanced means that the tree will not rebalance itself with every object insertion (the RED-BLACK trees do that). This means that if you insert items in sorted order, the tree will not be efficient at searching (it will essentially be a LinkedList).

Self-Balancing Trees (like Red-Black) are out of scope fort this homework.

## Deque

public class StudentDeque implements Deque<Student> {  
 @Override  
 public void addFirst(Student student) {  
 //*TODO* }  
  
 @Override  
 public void addLast(Student student) {  
 //*TODO* }  
  
 @Override  
 public boolean offerFirst(Student student) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean offerLast(Student student) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public Student removeFirst() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student removeLast() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student pollFirst() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student pollLast() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student getFirst() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student getLast() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student peekFirst() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student peekLast() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public boolean removeFirstOccurrence(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean removeLastOccurrence(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean add(Student student) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean offer(Student student) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public Student remove() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student poll() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student element() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Student peek() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public void push(Student student) {  
 //*TODO* }  
  
 @Override  
 public Student pop() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public boolean remove(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean contains(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public int size() {  
 //*TODO* return 0;  
 }  
  
 @Override  
 public Iterator<Student> iterator() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Iterator<Student> descendingIterator() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 //*TODO* return false;  
 }  
  
 @Override  
 public Object[] toArray() {  
 //*TODO* return new Object[0];  
 }  
  
 @Override  
 public <T> T[] toArray(T[] ts) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();   
 }  
  
 @Override  
 public boolean containsAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean addAll(Collection<? extends Student> collection) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean removeAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public boolean retainAll(Collection<?> collection) {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public void clear() {  
 //*TODO* }  
}

### ArrayDeque

ArrayDeque behaves like a Queue and Stack combined, you can add/read/delete elements at the beginning and end. Internally you will use an array to store data.

### LinkedList

LinkedList implements both List and Deque intefaces. For this situation implement only the Deque portion.

## Map

The Student class will act as **Key** for your map. You can store anything you like as a **Value.**

public class StudentMap implements Map<Student, Integer> {  
 @Override  
 public int size() {  
 //*TODO* return 0;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean containsKey(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public boolean containsValue(Object o) {  
 //*TODO* return false;  
 }  
  
 @Override  
 public Integer get(Object o) {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Integer put(Student student, Integer integer) {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Integer remove(Object o) {  
 //*TODO* return null;  
 }  
  
 @Override  
 public void putAll(Map<? extends Student, ? extends Integer> map) {  
 //*TODO* }  
  
 @Override  
 public void clear() {  
 //*TODO* }  
  
 @Override  
 public Set<Student> keySet() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Collection<Integer> values() {  
 //*TODO* return null;  
 }  
  
 @Override  
 public Set<Entry<Student, Integer>> entrySet() {  
 //Ignore this for homework  
 throw new UnsupportedOperationException();  
 }  
}

### HashMap

You are allowed to solve hashCode collisions in whatever way you like. You can even use a standard LinkedList for that.

### TreeMap

TreeMap internals are similar to TreeSet internals. Most important thing about your TreeMap implementation: **IT’S OK FOR YOUR TREES IMPLEMENTATION TO BE UNBALANCED.** Unbalanced means that the tree will not rebalance itself with every object insertion (the RED-BLACK trees do that). This means that if you insert items in sorted order, the tree will not be efficient at searching (it will essentially be a LinkedList).

Self-Balancing Trees (like Red-Black) are out of scope fort this homework.

# Bonus

## Generify your collection

If you managed to implement the task alone without any help and wish to push the bounds a bit further, consider submitting a second Pull Request with a Generic version of your collection. Remove any traces of Student class from your collection. Make it accept a type parameter <T>. (or <K,V> if it is a map)

Some collections are a bit easier to generify, others more difficult. If your collection stores data internally in an array you will need to investigate the problems related to arrays not mixing well with generic types.

## Implement all the methods

Implement methods with UnsupportedOperationException.