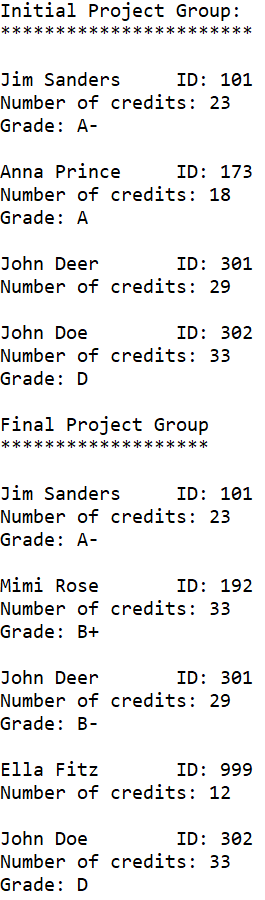
**1**



**package** Com.TSL.ProjectGroupUtilities;

/\*\*

\* **@author** EMILIA BUTU

\* version 1.0

\* since 2020-03

\*

\* Student name: Tom Lever

\* Completion date: 06/20/21

\*

\* ProjectGroupDriver.java: Demonstrates the use of a list to manage a set of objects.

\*/

**public** **class** ProjectGroupDriver

{

/\*\*

\* Creates and populates a group of students for a project.

\* It displays the list of students

\*/

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

{

ProjectGroup pg = **new** ProjectGroup();

pg.addStudent(**new** Student("101", "Jim Sanders",23, "A-"));

pg.addStudent(**new** Student("173", "Anna Prince", 18, "A"));

pg.addStudent(**new** Student("301", "John Deer", 29 ));

pg.addStudent(**new** Student("302", "John Doe", 33, "D"));

System.***out***.println("Initial Project Group: ");

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println(pg);

Student s1 = pg.find("301");

pg.addStudentAfter(s1, **new** Student("999","Ella Fitz", 12));

Student s2 = pg.find("301");

s2.setGrade("B-");

Student s3 = pg.find("173");

pg.replace(s3, **new** Student("192", "Mimi Rose",33, "B+"));

System.***out***.println("Final Project Group");

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println(pg);

}

}

package Com.TSL.ProjectGroupUtilities;

/\*\*

\* @author EMILIA BUTU

\* version 1.0

\* since 2020-03

\*

\* Student name: Tom Lever

\* Completion date: 06/20/21

\*

\* ProjectGroup.txt: template file of ProjectGroup.java

\* Student tasks: complete tasks specified in the file

\*/

import java.util.Iterator;

import java.util.LinkedList;

import java.util.List;

/\*\*

\*

\* Represents a list of students for a project group.

\*

\*/

public class ProjectGroup implements Iterable<Student>

{

// instance variable

private List<Student> list;

/\*\*

\* Constructs an initially empty list representing a group project.

\*/

public ProjectGroup()

{

//\*\*\* Task #1: add code here for the constructor

this.list = new LinkedList<Student>();

}

/\*\*

\* Adds the specified student to the end of the student list.

\*

\* @param student as the student to add

\*/

public void addStudent(Student student) {

//\*\*\* Task #2: create the body of this method

this.list.add(student);

}

/\*\*

\* Finds and returns the student matching the specified studentID

\* @param studentID, as the studentID of the target student

\* @return the student, or null if not found

\*/

public Student find(String studentID) {

//\*\*\* Task #3: fill in the body of method find

for (Student student : this.list) {

if (student.getStudentID() == studentID) {

return student;

}

}

return null;

}

/\*\*

\* Adds the specified student after the target student.

\* Does nothing if either student is null or if the target is not found.

\*

\* @param target the student after which the new student will be added

\* @param newStudent the student to add

\*/

public void addStudentAfter(Student target, Student newStudent) {

//\*\*\* Task #4: ill in the body of method addStudentAfter

if ((target == null) || (newStudent == null)) {

return;

}

int theIndexOfTheTargetStudent = this.list.indexOf(target);

if (theIndexOfTheTargetStudent == -1) {

return;

}

this.list.add(theIndexOfTheTargetStudent + 1, newStudent);

}

/\*\*

\* Replaces the specified target student with the new student. Does nothing if

\* either student is null or if the target is not found.

\*

\* @param target the student to be replaced

\* @param newStudent the new student to add

\*/

public void replace(Student target, Student newStudent) {

//\*\*\* Task #5: fill in the body of method replace

if ((target == null) || (newStudent == null)) {

return;

}

int theIndexOfTheTargetStudent = this.list.indexOf(target);

if (theIndexOfTheTargetStudent == -1) {

return;

}

this.list.set(theIndexOfTheTargetStudent, newStudent);

}

/\*\*

\* Creates and returns a string representation of this ProjectGroup object.

\*

\* @return a string representation of the ProjectGroup object

\*/

public String toString() {

//\*\*\* Task #6: fill in the body of the method toString()

String theRepresentationOfThisProjectGroup = "";

for (Student student : this.list) {

theRepresentationOfThisProjectGroup += student + "\n";

}

return theRepresentationOfThisProjectGroup;

}

/\*\*

\* Returns an iterator for this Program of Study.

\*

\* @return an iterator for the Program of Study

\*/

public Iterator<Student> iterator()

{

return list.iterator();

}

}

**package** Com.TSL.ProjectGroupUtilities;

/\*\*

\* Program: Student.java

\* Scope: Defines a class Student, representing a student inrolled in CSC 202.

\*

\*/

**public** **class** Student

{

//instance variables

**private** String studentID;

// this variable represents credits taken so far, and helps classification of student as freshman, sophomore, and so on.

**private** String name;

**private** **int** number;

**private** String grade;

/\*\*

\* Constructs the student with the specified information.

\*

\* **@param** studentID - the studentID of the student

\* **@param** number - the number of credits student has taken so far

\* **@param** name - the name of the student

\* **@param** grade the grade received for a specific assignment

\*/

**public** Student(String studentID, String name,**int** number, String grade)

{

**this**.studentID = studentID;

**this**.name = name;

**this**.number = number;

**if** (grade == **null**)

**this**.grade = "";

**else**

**this**.grade = grade;

}

/\*\*

\* Constructs the course with the specified information, with no grade

\* established, because the student has not submitted the assignment.

\*

\* **@param** studentID the studentID of the student

\* **@param** number the number of credits the student has taken so far

\* **@param** name the name of the course

\*/

**public** Student(String studentID, String name, **int** number)

{

**this**(studentID, name, number,"");

}

// accessors

/\*\*

\* Returns the studentID of the student.

\*

\* **@return** the studentID of the student

\*/

**public** String getStudentID()

{

**return** studentID;

}

/\*\*

\* Returns the name of this student.

\*

\* **@return** the name of the student

\*/

**public** String getName()

{

**return** name;

}

/\*\*

\* Returns the number of credits student has taken so far.

\*

\* **@return** the number of credits student has taken so far

\*/

**public** **int** getNumber()

{

**return** number;

}

/\*\*

\* Returns the grade for this student for a specific assignment

\*

\* **@return** the grade for the assignment

\*/

**public** String getGrade()

{

**return** grade;

}

// mutators

/\*\*

\* Sets the grade for this course to the one specified.

\*

\* **@param** grade the new grade for the course

\*/

**public** **void** setGrade(String grade)

{

**this**.grade = grade;

}

/\*\*

\* Returns true if this has received a grade for the assignment so far

\*

\* **@return** true if the student has a grade so far and false otherwise

\*/

**public** **boolean** graded()

{

**return** !grade.equals("");

}

/\*\*

\* Determines if this student is equal to the one specified, based on the

\* student (studentID and number).

\*

\* **@return** true if this student is equal to the parameter

\*/

**public** **boolean** equals(Object other)

{

**boolean** result = **false**;

**if** (other **instanceof** Student)

{

Student otherStudent = (Student) other;

**if** (studentID.equals(otherStudent.getStudentID()) &&

number == otherStudent.getNumber())

result = **true**;

}

**return** result;

}

/\*\*

\* Creates and returns a string representation of this course.

\*

\* **@return** a string representation of the course

\*/

**public** String toString()

{

String result ="\n" + name + "\tID: "+ studentID + "\nNumber of credits: " + number;

**if** (!grade.equals(""))

result += "\nGrade: " + grade ;

**return** result;

}

}

**2**

**package** Com.TSL.NonZeroListUtilities;

/\*\*

\* **@author** YINGJIN CUI

\* version 1.0

\* since 2020-03

\*

\* NonzeroListDriver.java: The driver program for NonzeroList.java

\*/

**public** **class** NonzeroListDriver{

**public** **static** **void** main(String[] args){

NonzeroList list=**new** NonzeroList(5);

System.***out***.println("Before adding any data, call list.isEmpty() retrurns: "+list.isEmpty()+ " list.size() returns: "+ list.size());

list.add(2);

list.add(12);

list.add(5);

list.add(15);

list.add(9);

list.print();

System.***out***.println("The list is full: "+ list.isFull());

System.***out***.println("Try to add more data to the list:");

list.add(7);

list.add(200);

list.removeData(89);

list.removeData(5);

System.***out***.println("After removing 5.");

list.print();

System.***out***.println("size="+list.size());

System.***out***.println("Try to add 0");

list.add(0);

list.print();

System.***out***.println("Use loop to delete all data from the list");

**int**[] theListAsAnArray = {2, 12, 9, 15};

**for** (**int** i = list.size() - 1; i >= 0; i--) {

list.removeData(theListAsAnArray[i]);

System.***out***.print("Removed right-most element from the list; list is\n\t");

list.print();

}

}

}

**package** Com.TSL.NonZeroListUtilities;

/\*\*

\* **@author** YINGJIN CUI

\* version 1.0

\* since 2020-03

\*

\* Student name: Tom Lever

\* Completion date: 06/20/21

\*

\* NonzeroList represents the structure of an array-based list of non-zero integers.

\*/

**public** **class** NonzeroList{

**private** **int**[] data;

**private** **int** index; //the location where new data will be added to the array.

**private** **int** numberOfElements;

/\*\*

\* NonzeroList(int cap) is the one-parameter constructor for NonzeroList, which initializes this list's array of

\* integers to a new array with the provided capacity, initializes this list's location for adding new integers to

\* zero, and initializes this list's number of integers to 0.

\*

\* **@param** cap

\*/

**public** NonzeroList(**int** cap) {

data = **new** **int**[cap];

index=numberOfElements=0;

}

**public** **void** add(**int** num){

// \*\*\* Student task #1 \*\*\*

/\*

Requirements:

-if num is zero, print message saying that zero is not allowed in a nonzerolist

-if it is full, display message "The NonzeroList is full."

- Else, add num to this list's array of integers, and increment index and numberOfElements.

\*\*\* Enter your code below \*\*\*

\*/

**if** (num == 0) {

System.***out***.println("zero is not allowed in a nonzerolist.");

**return**;

}

**if** (isFull()) {

System.***out***.println("The NonzeroList is full.");

**return**;

}

**this**.data[**this**.index] = num;

**this**.index++;

**this**.numberOfElements++;

}

**public** **void** removeData(**int** target) {

// \*\*\* Student task #2 \*\*\*

/\*

Requirements:

-Remove the first occurrence of the target value in the NonzeroList

-Move the last item in the NonzeroList to the above position.

-You may shift after the removed item but it's not sufficient. The big-O of Moving the last item to the

removed target location if O(1), while complexity of shifting algorithm is O(N)-

Can you figure out why?

-If the target value does not exist, print message: "Target value does not exist."

\*\*\* Enter your code below \*\*\*

\*/

**for** (**int** i = 0; i < **this**.numberOfElements; i++) {

**if** (**this**.data[i] == target) {

**this**.data[i] = **this**.data[**this**.numberOfElements - 1];

**this**.data[**this**.numberOfElements - 1] = 0;

**this**.index--;

**this**.numberOfElements--;

**return**;

}

}

System.***out***.println("Target value " + target + " does not exist.");

}

**public** **int** indexOf(**int** target){ // return the index of the first occurrence of target in the data array

**for**(**int** i=0; i<numberOfElements; i++){

**if**(data[i]==target)

**return** i;

}

**return** -1; // not found

}

/\*\*

\* isFull indicates whether or not this list is full.

\* **@return**

\*/

**public** **boolean** isFull(){

**return** numberOfElements==data.length;

}

/\*\*

\* isEmpty indicates whether or not this list is empty.

\* **@return**

\*/

**public** **boolean** isEmpty(){

**return** numberOfElements==0;

}

/\*\*

\* size provides the number of elements in this list.

\* **@return**

\*/

**public** **int** size(){

**return** numberOfElements;

}

/\*\*

\* print outputs a representation of this list to the standard output stream.

\*/

**public** **void** print(){

System.***out***.print("[");

**for**(**int** i=0; i<numberOfElements; i++){

System.***out***.print(data[i]);

**if**(i<numberOfElements-1)

System.***out***.print(", ");

}

System.***out***.println("]");

}

}