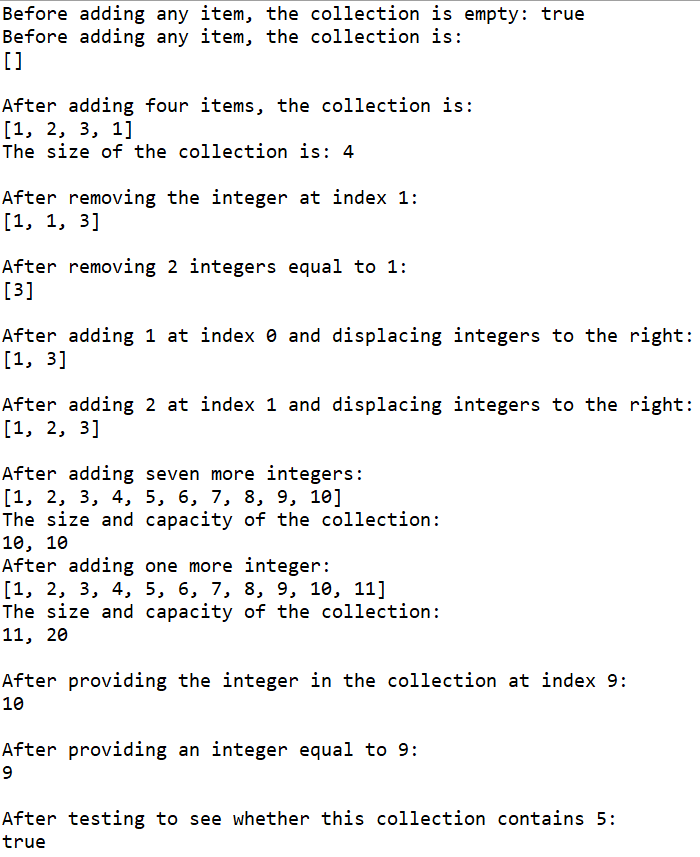
**1**



**package** Com.TSL.CollectionAbstractDataTypeUtilities;

/\*\*

\* CollectionTTest encapsulates a method that tests all of the methods of the CollectionT class.

\*

\* **@author** Tom Lever

\* **@version** 1.0

\* **@since** 06/14/21

\*/

**public** **class** CollectionTTest

{

/\*\*

\* main tests all of the methods of the CollectionT class.

\*

\* **@param** args

\*/

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

{

CollectionT<Integer> theCollectionOfIntegers = **new** CollectionT<Integer>(10);

System.***out***.println("Before adding any item, the collection is empty: " + theCollectionOfIntegers.isEmpty());

System.***out***.println("Before adding any item, the collection is:");

theCollectionOfIntegers.print();

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

System.***out***.println("After adding four items, the collection is:");

**for** (**int** i = 0; i < 3; i++) {

theCollectionOfIntegers.add(i + 1);

}

theCollectionOfIntegers.add(1);

theCollectionOfIntegers.print();

System.***out***.println("The size of the collection is: " + theCollectionOfIntegers.size() + "\n");

System.***out***.println("After removing the integer at index 1:");

theCollectionOfIntegers.remove(1);

theCollectionOfIntegers.print();

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

**int** theNumberOfIntegersRemoved = theCollectionOfIntegers.remove((Integer) 1);

System.***out***.println("After removing " + theNumberOfIntegersRemoved + " integers equal to 1:");

theCollectionOfIntegers.print();

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

// At this point, for theCollectionOfIntegers, the following have been tested:

// - CollectionT(int size)

// - add(T ele)

// - remove(int index)

// - remove(T ele)

System.***out***.println("After adding 1 at index 0 and displacing integers to the right:");

theCollectionOfIntegers.add(1, 0);

theCollectionOfIntegers.print();

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

System.***out***.println("After adding 2 at index 1 and displacing integers to the right:");

theCollectionOfIntegers.add(2, 1);

theCollectionOfIntegers.print();

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

System.***out***.println("After adding seven more integers:");

**for** (**int** i = 4; i <= 10; i++) {

theCollectionOfIntegers.add(i);

}

theCollectionOfIntegers.print();

System.***out***.println("The size and capacity of the collection:");

System.***out***.println(theCollectionOfIntegers.size() + ", " + theCollectionOfIntegers.providesItsCapacity());

System.***out***.println("After adding one more integer:");

theCollectionOfIntegers.add(11);

theCollectionOfIntegers.print();

System.***out***.println("The size and capacity of the collection:");

System.***out***.println(theCollectionOfIntegers.size() + ", " + theCollectionOfIntegers.providesItsCapacity() + "\n");

System.***out***.println("After providing the integer in the collection at index 9:");

System.***out***.println(theCollectionOfIntegers.get(9) + "\n");

System.***out***.println("After providing an integer equal to 9:");

System.***out***.println(theCollectionOfIntegers.get((Integer) 9) + "\n");

System.***out***.println("After testing to see whether this collection contains 5:");

System.***out***.println(theCollectionOfIntegers.contains(5));

}

}

**package** Com.TSL.CollectionAbstractDataTypeUtilities;

/\*\*

\* **@author** Yingjin Cui

\* version 1.0

\* since 2020-03-30

\*

\* Student name: Tom Lever

\* Completion date: 06/14/21

\*

\* CollectionT.txt: the template file of CollectionT.java

\* Student tasks: implement tasks #1, #2 and #3 as specified in this file

\*/

**public** **class** CollectionT<T> **implements** CollectionInterface<T>{

**private** T[] data;

**private** **int** count = 0;

/\*\*

\* CollectionT(int size) is the one-parameter constructor for CollectionT, which sets this collection's data

\* reference to a new collection of <size> elements of type T.

\*

\* **@param** size

\*/

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

data = (T[]) (**new** Object[size]);

}

**public** **void** add(T ele){

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

/\*

Requirements: if the collection is full then double the array size before adding operation

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

\*/

**if** (isFull()) {

enlargeTheArrayOfData();

}

data[count] = ele;

count++;

}

/\*\*

\* add(T theElementToAdd, int theIndexAtWhichToAddTheElement) enlarges the array of data of this collection if the

\* array is full, shifts elements at and to the right of a provided index to the right, inserts a provided element

\* at the provided index, and increments the running number of elements in the array. If the index at which to

\* add the element is invalid, add throws an invalid index exception.

\*/

**public** **void** add(T theElementToAdd, **int** theIndexAtWhichToAddTheElement) {

**if** (theIndexAtWhichToAddTheElement < 0) {

**throw** **new** AnInvalidIndexException(

"Exception: add with an invalid index at which to add an element was requested."

);

}

**if** (theIndexAtWhichToAddTheElement > size()) {

add(theElementToAdd);

}

**if** (isFull()) {

enlargeTheArrayOfData();

}

**for** (**int** i = size(); i > theIndexAtWhichToAddTheElement; i--) {

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

}

**this**.data[theIndexAtWhichToAddTheElement] = theElementToAdd;

count++;

}

/\*\*

\* enlargeTheArrayOfData transfers this collection's data to an array with twice the capacity of the previous array.

\*/

**public** **void** enlargeTheArrayOfData() {

T[] theEnlargedArray = (T[]) **new** Object[2 \* **this**.data.length];

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

theEnlargedArray[i] = **this**.data[i];

}

**for** (**int** i = **this**.data.length; i < theEnlargedArray.length; i++) {

theEnlargedArray[i] = **null**;

}

**this**.data = theEnlargedArray;

}

**public** **boolean** remove(**int** index) {

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

/\*

Requirements: Removes the element at index position, if index<size() returns true after removing, otherwise, returns false

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

\*/

**if** ((index >= 0) && (index < size())) {

**this**.data[index] = **this**.data[size() - 1];

**this**.data[size() - 1] = **null**;

count--;

**return** **true**;

}

**return** **false**;

}

**public** **int** remove(T ele) {

// \*\*\* Student task #3 \*\*\*

/\*

Requirements:

Removes all elements from this collection such that each of them equals to ele [equals(ele) returns true]

and returns the number of elements being removed.

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

\*/

**int** theNumberOfElementsRemoved = 0;

**int** theIndexOfTheElementAfterTheLastElementChecked = 0;

**while** (theIndexOfTheElementAfterTheLastElementChecked < size()) {

**if** (**this**.data[theIndexOfTheElementAfterTheLastElementChecked].equals(ele)) {

remove(theIndexOfTheElementAfterTheLastElementChecked);

theNumberOfElementsRemoved++;

}

**else** {

theIndexOfTheElementAfterTheLastElementChecked++;

}

}

**return** theNumberOfElementsRemoved;

}

/\*\*

\* get(T ele) provides an element equal to the provided element.

\*/

**public** T get(T ele){

**int** index = indexOf(ele);

**if**(index >=0){

**return** data[index];

}**else**{

**return** **null**;

}

}

/\*\*

\* get(int theIndexOfTheElementToGet) provides the element at the provided index, or throws an invalid index

\* exception if the provided index is invalid.

\*/

**public** T get(**int** theIndexOfTheElementToGet) {

**if** (theIndexOfTheElementToGet < 0 || theIndexOfTheElementToGet >= size()) {

**return** **null**;

}

**return** **this**.data[theIndexOfTheElementToGet];

}

/\*\*

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

\*/

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

**return** count == data.length;

}

/\*\*

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

\*/

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

**return** count == 0;

}

/\*\*

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

\*/

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

**return** count;

}

/\*\*

\* indexOf provides the index of the left-most element equal to a provided element, or provides -1 if no such

\* element exists.

\*/

**public** **int** indexOf(T ele){

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

**if**(data[i].equals(ele)){

**return** i;

}

}

**return** -1;

}

/\*\*

\* contains indicates whether or not there exists an element in this collection that is equal to a provided

\* element.

\*/

**public** **boolean** contains(T ele){

**return** indexOf(ele) != -1;

}

/\*\*

\* print provides a representation of this collection to the standard output stream.

\*/

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

String tmp="";

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

tmp += data[i].toString()+", ";

}

**if**(tmp.length()>0){

tmp = tmp.substring(0, tmp.length() -2);

}

System.***out***.println("["+tmp+"]");

}

/\*\*

\* providesItsCapacity provides the capacity of the array of elements of this collection.

\* **@return**

\*/

**public** **int** providesItsCapacity() {

**return** **this**.data.length;

}

}

**package** Com.TSL.CollectionAbstractDataTypeUtilities;

/\*\*

\* **@author** YINGJIN CUI

\* version 1.0

\* since 2020-03-30

\*

\* CollectionInterface.java -

\* Note: Generic type is used to allow you to write a general, generic methods that works

\* with different types-allowing for code re-use.

\* A collection allows addition, removal, and access of elements.

\* In this implemention, null is not allowed, but duplicate elements are permitted.

\*/

**public** **interface** CollectionInterface<T>{

**void** add(T ele);

//Add an element to the collection

**void** add(T ele, **int** index);

//Add an element in the index location

//if index>number of element of the collection, then append the ele to the collection

T get(T ele);

// Returns an element e from this collection such that e.equals(ele).

// If no such element exists, returns null.

T get(**int** index);

// Returns the (index+1)th element in the collection.

// If index<0 or index>= the number of elements in this collection, returns null.

**boolean** contains(T ele);

// Returns true if this collection contains an element e such that

// e.equals(ele) returns true; otherwise returns false.

**int** remove(T ele);

// Removes all elements from this collection such that each of them equals to ele [equals(ele) returns true]

// and returns the number of elements being removed.

**boolean** remove(**int** index);

// Removes the element at index position, if index<size() returns true after removing, otherwise, returns false

**int** indexOf(T ele);

// returns index of the first element e such that e.equals(ele) returns true

// If no such element exists, returns -1

**boolean** isFull();

// Returns true if this collection is full; otherwise, returns false.

**boolean** isEmpty();

// Returns true if this collection is empty; otherwise, returns false.

**int** size();

// Returns the number of elements in this collection.

**void** print();

// Print all elements

}

**package** Com.TSL.CollectionAbstractDataTypeUtilities;

/\*\*

\* AnInvalidIndexException represents an exception that occurs when a get method with an invalid index is requested.

\*

\* **@author** Tom Lever

\* **@version** 1.0

\* **@since** 06/09/21

\*/

**public** **class** AnInvalidIndexException **extends** RuntimeException

{

/\*\*

\* AnInvalidIndexException() is a conventional zero-parameter constructor for AnInvalidIndexException, which calls

\* Exception's zero-parameter constructor.

\*/

**public** AnInvalidIndexException()

{

**super**();

}

/\*\*

\* AnInvalidIndexException(String message) is a one-parameter constructor for AnInvalidIndexException, which passes

\* argument message to Exception's one-parameter constructor.

\* **@param** message

\*/

**public** AnInvalidIndexException(String message)

{

**super**(message);

}

}