420-B31

# Test 2 Review Exercises

Refer to the abstract data types, interfaces and classes from the textbook for the following questions.

## A Basic Collection Class

1. Code the **getLast()** method for the **BasicCollection** class given the following:

/\*\* Return the last element in the collection

\* @return the last element in the collection. If the collection is empty,

\* null is returned.

\*/

public E getLast(){

int index = 0;

Iterator<E> iter = list.iterator();

while (iter.hasNext()) {

Object temp = iter.next();

if (index = list.size()-1)

return temp;

index++;

}

return null;

}

1. Code the **removeLast()** method for the **BasicCollection** class given the following:

**/\*\* Removes the last element in the collection**

**\* @return true if the remove was successful; false if the collection was**

\* **empty**

**\*/**

**public boolean removeLast(){**

**int index = 0;**

**Iterator<E> iter = list.iterator();**

**while (iter.hasNext()) {**

**Object temp = iter.next();**

**if (index = list.size()-1)**

**return list.remove;**

**index++;**

**}**

**return null;**

**}**

## The List Abstract Data Type

1. Complete the following table.

| Method | Object State | Return  value |
| --- | --- | --- |
| List <String> myList = new ArrayList<String>(); | size = \_\_\_\_0  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | A new, empty arrayList |
| myList.add("A");  myList.add("B");  myList.add("C");  myList.add("D");  myList.add("E"); | size = \_\_\_\_5  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_  A, B, C, D, E |  |
| myList.add("X"); | size = \_\_\_\_6  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_  A, B, C, D, E, X |  |
| myList.add("Y"); | size = \_\_\_\_7  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_  A, B, C, D, E, X, Y |  |
| int i = myList.indexOf("D"); | size = \_\_\_\_  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | 3 |
| myList.set(i, "Z"); | size = \_\_\_\_7  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_  A, B, C, Z, E, X, Y |  |
| myList.remove(myList.size() – 2); | size = \_\_\_\_6  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_  A, B, C, D, E, Y |  |
| String temp = (String)myList.get(1); | size = \_\_\_\_  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | B |
| myList.set(1,temp.toLowerCase()); | size = \_\_\_\_6  list = \_\_\_\_\_\_\_\_\_\_\_\_\_\_  A, b, C, D, E, Y |  |

1. Code the followingmethod for the **LinkedList** class :

/\*\* Count the number of times an element occurs in the list

\* @param element - the element to count

\* @return the number of times element occurs in the list

\*/

public int frequency(E element){

int num = 0;

Iterator<E> iter = list.iterator();

while (iter.hasNext()) {

E temp = iter.next();

if (temp.equals(element)) {

num++;

}

}

return num;

}

## The Stack Abstract Data Type

1. Complete the following table.

| Method | Object State | Return value |
| --- | --- | --- |
| Stack <String> myStack = new LinkedStack<String>(); | size = \_\_\_\_  top = \_\_\_\_\_ | A new, empty Stack object |
| myStack.push("Rich");  myStack.push("Debbie");  myStack.push("Robin");  myStack.push("Dustin");  myStack.push("Philip"); | size = \_\_\_\_5  top = \_\_\_\_\_Philip |  |
| myStack.push("Jane"); | size = \_\_\_\_6  top = \_\_\_\_\_Jane |  |
| myStack.push("Joseph"); | size = \_\_\_\_7  top = \_\_\_\_\_Joseph |  |
| String top = (String) myStack.pop(); | size = \_\_\_\_6  top = \_\_\_\_\_Jane | Joseph |
| String nextTop = (String) myStack.peek(); | size = \_\_\_\_6  top = \_\_\_\_\_Jane |  |

1. Write a method that reads a line and reverses the words in the line (not the characters) using a stack. For example, given the following input:

**the quick brown fox jumps over the lazy dog.**

you should get the following output:

**dog. lazy the over jumps fox brown quick The**

**public String reverseWords(String e) {**

**String[] end = e.split(“\s”);**

**String reverse = “”;**

**for (int q = end.length-1 ; q >= 0 ; q-- ) {**

**reverse += end[q];**

**}**

**return reverse;**

**}**

1. Write a constructor for **LinkedStack** that loads the stack from an array parameter. The last array element should be at the top of the stack.

public LinkedStack(<E>[] arr) {

for (int q = 0 ; q < arr.length ; q++) {

list.push(arr[q]);

}

}

## The Queue Abstract Data Type

1. Consider the following fragment for a stack s and a queue q:

**s.push("A");**

**s.push("B");**

**s.push("C");**

**s.push("D");**

**while (!s.isEmpty())**

**{**

**String str = s.pop();**

**q.enqueue(str);**

**}**

**while (!q.isEmpty())**

**{**

**String str = q.dequeue();**

**s.push(str);**

**}**

* 1. What is stored in stack s after the first loop executes? What is stored in queue q?

Nothing is in s

Q contains D, C, B, A

* 1. What is stored in stack s after the second loop executes? What is stored in queue q?

Nothing is in q

S contains A, B, C, D

1. Assume that a queue q of capacity 6 (circular array representation) is initializes as follows:

| Method | Object State | Return value |
| --- | --- | --- |
| Queue <Character> q = new CircularQueue<Character>(6); | capacity = \_\_\_\_\_6  size = \_\_\_\_  front = \_\_\_\_\_null  rear = \_\_\_\_\_null | A new, empty Queue |
| q.enqueue(new Character('+'));  q.enqueue(new Character('\*'));  q.enqueue(new Character('-'));  q.enqueue(new Character('&'));  q.enqueue(new Character('#')); | capacity = \_\_\_\_\_6  size = \_\_\_\_5  front = \_\_\_\_\_+  rear = \_\_\_\_\_# |  |
| q.dequeue(); | capacity = \_\_\_\_\_6  size = \_\_\_\_4  front = \_\_\_\_\_\*  rear = \_\_\_\_\_# |  |
| q.enqueue(new Character('\\'));  q.enqueue(new Character('%')); | capacity = \_\_\_\_\_6  size = \_\_\_\_6  front = \_\_\_\_\_\*  rear = \_\_\_\_\_% |  |

1. Write a new queue method called **moveToRear()** that moves the element currently at the front of the queue to the rear of the queue. The element that was second in line will be the new front element. Use the **Queue** **enqueue()** and **dequeue()** methods.

Public void moveToRear() {

String srt = list.dequeue();

list.enqueue(str);

}

1. Write a new queue method called **moveToFront()** that moves the element currently at the rear of the queue to the front of the queue while the other queue elements maintain their relative positions behind the old front element. Use the **Queue** **enqueue()** and **dequeue()** methods.

public void moveToFront() {

}