Java Utility and Data Structure
Introduction

In the last module, we introduced arrays, ArrayList, Stack, Queue and LinkedList data structure. This module we learn how to use Java utility library (java.util) to implement these data structures. The java.util package is very powerful that contains classes that implement most typical data structures such as; Array List, Linked List, Stack and Queue. You have learnt ArrayList class in last week. We will focus on the other data structures in this module.

Objectives

- To learn how to use the LinkedList class provided in the standard library
- To be able to use *iterators* to traverse linked lists
- To learn how to use Stack class provided in the standard library
- To know how to implement Queue by using existing java classes

LinkedList class

The LinkedList class, defined in java.util, provides methods to manipulate a linked list data structure. The methods allow you to access the first and the last element (E is the element type of LinkedList):

- void addFirst (E obj)
- void addLast (E obj)
- E getFirst( )
- E getLast( )
- E removeFirst ( )
- E removeLast ( )

Other common used methods:

- int size( )
- boolean add (E obj)
- void add (int i, E obj)
- Object set (int i, E obj)
- Object get (int i);

It needs to be mentioned that you need to use a *list iterator* to access elements inside a linked list.

You obtain a list iterator with the listIterator method of the LinkedList class:

e.g

```java
LinkedList<String> employeeName = new LinkedList<String>( );
ListIterator<String> myIterator = employeeName.listIterator( );
```
Now, you are able to traverse list elements by using the myIterator reference. The most common methods are `next()` and `hasNext()`, please refer to page 744, 745 for a further understanding. An example for LinkedList is given below, you should inspect this program as well.

```java
//Demonstrate LinkedList
import java.util.*;
class LinkedListDemo
{
    public static void main(String[] args)
    {
        //create a LinkedList
        LinkedList<String> ll = new LinkedList<String> ();
        //add elements to the linked list
        ll.add("F");
        ll.add("B");
        ll.add("D");
        ll.add("E");
        ll.add("C");
        ll.addLast("Z");
        ll.addFirst("A");
        ll.add(1, "A2");

        System.out.println("Original contents of ll: "+ll);

        //remove elements from the linked list
        ll.remove("F");
        ll.remove(2);

        System.out.println("Contents of ll after deletion: "+ll);

        //remove first and last elements
        ll.removeFirst();
        ll.removeLast();

        System.out.println("ll after deleting first and last: "+ll);

        //get and set a value
        Object val = ll.get(2);
        ll.set(2, (String) val+" Changed");

        System.out.println("ll after change: "+ll);
    }
}
```
Stack class

Stack class defined in java.util provides the following common methods.

- void push(E obj)
- E pop()
- int size()
- boolean empty()
- E peek()

Please go to java API documentation to check above methods. Pay attention to the difference between method pop() and peek().

Queue

A queue is similar to a stack, except that you add items to one end of the queue (the tail) and remove them from the other end of the queue (the head). There is no Queue class defined in java.util. However, it can be simply implemented by a linked list. For example, the methods addLast() and removeFirst() can be used to implement enqueue and dequeue operations. There is an example of this on page 764 of the text.

Remark

As you can see, LinkedList, ArrayList, and Stack classes defined in java.util library provide most operations on these data structures. A queue structure can be implemented by the LinkedList. These classes and methods are ready to use. For an application programmer, in most cases, you do not need to worry about how these methods are implemented in the library. An application programmer’s main concern is to understand:

1. What are these data structures?
2. Where the classes for these data structure are defined in Java?
3. How do you use the classes and methods in your applications?
Now, you should be able to answer the above questions.

Reading
Text book:
Chapter 20: An Introduction to Data Structure

Review questions
Review exercises:
Programming exercises:
Page 770: Exercise P20.12

Stack exercise: Describe the output of the following series of stack operations:
push(5) → push(3) → pop() → push(2) → push(8) → pop() → pop() → push(9) → push(1)
→ pop() → push(7) → push(6) → pop() → pop() → push(4) → pop() → pop().
Write a program to test your answer.

Queue exercise: Describe the output of the following series of stack operations:
enqueue(5) → enqueue (3) → dequeue() → enqueue (2) → enqueue (8) → dequeue () →
dequeue () → enqueue (9) → enqueue (1) → dequeue () → enqueue (7) → enqueue (6) →
dequeue () → dequeue () → enqueue (4) → dequeue () → dequeue ()
Write a program to test your answer.

References
Solution to Programming Exercise

P20-12

ExP20_12/ExP20_12.java
/**
   * A tester program for the Stack class.
   */
public class ExP20_12
{
    public static void main(String[] args)
    {
        Stack s = new Stack();
        s.push("A");
        s.push("B");
        s.push("C");

        // the following loop prints C, B, A
        while (s.size() > 0)
            System.out.println(s.pop());
    }
}

ExP20_12/Stack.java
import java.util.LinkedList;
/**
   * A Stack using a LinkedList to store elements.
   */
public class Stack
{
    /**
     * Construct a Stack object.
     */
    public Stack()
    {
        data = new LinkedList();
    }

    /**
     * Pushes an item onto the top of this stack.
     * @param item the item to be pushed onto this stack.
     * @return the item argument.
     */
    public Object push(Object item)
    {
        data.addFirst(item);
        return item;
    }

    /**
     * Removes the object at the top of this stack and returns that
     * object as the value of this function.
     * @return The object at the top of this stack.
     */
public Object pop()
{
    return data.removeFirst();
}

/**
 * Looks at the object at the top of this stack without removing it
 * from the stack.
 * @return the object at the top of this stack
 */
public Object peek()
{
    return data.getFirst();
}

/**
 * Tests if this stack is empty.
 * @return true if and only if this stack contains
 * no items, false otherwise.
 */
public boolean empty()
{
    return size() == 0;
}

/**
 * Gets the number of items in the stack.
 * @return the size
 */
public int size()
{
    return data.size();
}

private LinkedList data;