THE OPEN UNIVERSITY OF SRI LANKA
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
BACHELOR OF SOFTWARE ENGINEERING / BACHELOR OF TECHNOLOGY
ECX4235 / ECX4265 – DATA STRUCTURES AND ALGORITHMS

PRACTICAL SESSION

Scenario 1 - A queue in a supermarket.

Scenario 2 - A stack of books

Scenario 3 – Set of people with different heights

Identify the actions can be performed for each scenario given above and write simple java programs to implement the specified actions using your data structure knowledge. The underlined words depict the data structures.

Example:
For scenario 1, a customer joins the queue; a customer leaves the queue etc…

For scenario 2, the largest book at the bottom need to be retrieved, the smallest book at the top need to be retrieved etc…

For scenario 3, use binary sort, bubble sort to sort people according to their heights, how each person can be linked with their position numbers using a linked list etc…

You can get hands on experience in implementing different data structures in Java by attempting the following tasks before moving to the given question.

Task 1: Get familiar with writing a simple java program

```
/* The students are expected to get an understanding about writing classes, methods using
appropriate access specifiers and data types*/

//Class declaration
public class HelloWorld {
//Method declaration
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Task 2: Get familiar with writing a simple java interface

```
/*Java Interface example. This Java Interface example describes how interface is defined
and being used in Java language.

 Syntax of defining java interface is,
<Modifier> interface <interface-name> {
 //members and methods ()
}*/
//declare an interface
interface IntExample{
 /* Syntax to declare method in java interface is,
 <modifier> <return-type> methodName(<optional-parameters>);
 IMPORTANT : Methods declared in the interface are implicitly public and abstract.*/
  public void sayHello();
  }
}
/* Classes are extended while interfaces are implemented. To implement an interface use
implements keyword.
IMPORTANT : A class can extend only one other class, while it can implement n number
of interfaces. */
public class JavaInterfaceExample implements IntExample{
/* We have to define the method declared in implemented interface, or else we have to
declare the implementing class as abstract class. */
 public void sayHello(){
    System.out.println("Hello Visitor !");
  }
 public static void main(String args[]){
  //create object of the class
  JavaInterfaceExample javaInterfaceExample = new JavaInterfaceExample();
  //invoke sayHello(), declared in IntExample interface.
    javaInterfaceExample.sayHello();
  }
}
```

Task 3: Get familiar with writing a ***Linked list*** in Java

/* This class implements a linked list that can contain any type of object that implements the nested Linkable interface. Note that the methods are all synchronized, so that it can safely be used by multiple threads at the same time.*/

```
public class LinkedList {

/* This interface defines the methods required by any object that can be linked into a linked
list. */

public interface Linkable {
    public Linkable getNext();             // Returns the next element in the list
  public void setNext(Linkable node);      // Sets the next element in the list
  }
 // This class has a default constructor: public LinkedList() {}
 /* This is the only field of the class. It holds the head of the list */
  Linkable head;
 /* Return the first node in the list */
  public synchronized Linkable getHead() {
    return head;
  }
 /* Insert a node at the beginning of the list */
  public synchronized void insertAtHead(Linkable node) {
    node.setNext(head);
    head = node;
  }
 /* Insert a node at the end of the list */
 public synchronized void insertAtTail(Linkable node) {
    if (head == null)
      head = node;
    else {
      Linkable p, q;
      for (p = head; (q = p.getNext()) != null; p = q)
        /* no body */;
      p.setNext(node);
    }
  }
 /* Remove and return the node at the head of the list */
 public synchronized Linkable removeFromHead() {
    Linkable node = head;
    if (node != null) {
      head = node.getNext();
      node.setNext(null);
    }
    return node;
  }
 /* Remove and return the node at the end of the list */
 public synchronized Linkable removeFromTail() {
    if (head == null)
      return null;
    Linkable p = head, q = null, next = head.getNext();
    if (next == null) {
      head = null;
      return p;
    }
    while ((next = p.getNext()) != null) {
      q = p;
      p = next;
    }
    q.setNext(null);
    return p;
  }
```

/*Remove a node matching the specified node from the list. Use equals() instead of == to test for a matched node.*/

```
  public synchronized void remove(Linkable node) {
    if (head == null)
      return;
    if (node.equals(head)) {
      head = head.getNext();
      return;
    }
    Linkable p = head, q = null;
    while ((q = p.getNext()) != null) {
      if (node.equals(q)) {
        p.setNext(q.getNext());
        return;
      }
      p = q;
    }
  }
```

/*This is a test class that implements the Linkable interface */

```
 static class LinkableInteger implements Linkable {
    int i;             // The data contained in the node

  Linkable next;   // A reference to the next node in the list
  public LinkableInteger(int i) {
      this.i = i;
  } // Constructor
    public Linkable getNext() {
      return next;
    } // Part of Linkable
    public void setNext(Linkable node) {
      next = node;
    } // Linkable
    public String toString() {
      return i + "";
    } // For easy printing
public boolean equals(Object o) { // For comparison
      if (this == o)
        return true;
      if (!(o instanceof LinkableInteger))
        return false;
      if (((LinkableInteger) o).i == this.i)
        return true;
      return false;
    }
  }
```

/* The test program. Insert some nodes, remove some nodes, then printout all elements in t
he list. It should print out the numbers 4, 6, 3, 1, and 5*/

```
  public static void main(String[] args) {
    LinkedList ll = new LinkedList();             // Create a list
    ll.insertAtHead(new LinkableInteger(1));      // Insert some stuff
    ll.insertAtHead(new LinkableInteger(2));
    ll.insertAtHead(new LinkableInteger(3));
    ll.insertAtHead(new LinkableInteger(4));
    ll.insertAtTail(new LinkableInteger(5));
    ll.insertAtTail(new LinkableInteger(6));
    System.out.println(ll.removeFromHead());      // Remove and print a node
    System.out.println(ll.removeFromTail());      // Remove and print again
  ll.remove(new LinkableInteger(2));              // Remove another one
  // Now print out the contents of the list.
   for (Linkable l = ll.getHead(); l != null; l = l.getNext())
      System.out.println(l);
  }
}
```

<u>Task 4: Get familiar with writing a simple ***Queue*** in Java</u>

```
public class Queue {
  private int maxSize;
  private long[] queArray;
  private int front;
  private int rear;
  private int nItems;

  public Queue(int s) {
    maxSize = s;
    queArray = new long[maxSize];
    front = 0;
    rear = -1;
    nItems = 0;
  }
   //   put item at end of a queue
   public void insert(long j) {
    if (rear == maxSize - 1)          // deal with wraparound
      rear = -1;
    queArray[++rear] = j;             // increment rear and insert
    nItems++;
  }
   //   take item from front of queue
   public long remove() {
    long temp = queArray[front++];    // get value and increment front
      if (front == maxSize)           // deal with wraparound
         front = 0;
    nItems--;       // one less item
     return temp;
  }
   public long peekFront() {
    return queArray[front];
  }
  public boolean isEmpty() {
    return (nItems == 0);
  }

  public boolean isFull() {
    return (nItems == maxSize);
  }
  public int size() {
    return nItems;
  }
  public static void main(String[] args) {
    Queue theQueue = new Queue(5);  // queue holds 5 items

    theQueue.insert(10);
    theQueue.insert(20);
    theQueue.insert(30);
    theQueue.insert(40);
    theQueue.remove();
    theQueue.remove();
    theQueue.remove();
    theQueue.insert(50);
    theQueue.insert(60);  //   (wraps around)
      theQueue.insert(70);
    theQueue.insert(80);

    while (!theQueue.isEmpty()) {
      long n = theQueue.remove();  // (40, 50, 60, 70, 80)
        System.out.print(n);
      System.out.print(" ");
    }
    System.out.println("");
  }
}
```

Task 5: Get familiar with writing a simple ***Stack*** in Java

```
  public class MyStack {
  private int maxSize;
  private long[] stackArray;
  private int top;

  public MyStack(int s) {
    maxSize = s;
    stackArray = new long[maxSize];
    top = -1;
  }
  public void push(long j) {
    stackArray[++top] = j;
  }
  public long pop() {
    return stackArray[top--];
  }
  public long peek() {
    return stackArray[top];
  }
  public boolean isEmpty() {
    return (top == -1);
  }
  public boolean isFull() {
    return (top == maxSize - 1);
  }
  public static void main(String[] args) {
    MyStack theStack = new MyStack(10);  // make new stack
      theStack.push(20);
    theStack.push(40);
    theStack.push(60);
    theStack.push(80);

    while (!theStack.isEmpty()) {
      long value = theStack.pop();
      System.out.print(value);
      System.out.print(" ");
    }
    System.out.println("");
  }
}
```

Task 6: Get familiar with writing a method for ***Selection Sort*** in Java

```
public int[] selectionSort(int[] data){
  int lenD = data.length;
  int j = 0;
  int tmp = 0;
  for(int i=0;i<lenD;i++){
    j = i;
    for(int k = i;k<lenD;k++){
      if(data[j]>data[k]){
        j = k;
      }
    }
    tmp = data[i];
    data[i] = data[j];
    data[j] = tmp;
  }
  return data; }
```

Task 7: Get familiar with writing a method for ***Insertion Sort*** in Java

```
public int[] InsertionSort(int[] data){
  int len = data.length;
  int key = 0;
  int i = 0;
  for(int j = 1;j<len;j++){
    key = data[j];
    i = j-1;
    while(i>=0 && data[i]>key){
      data[i+1] = data[i];
      i = i-1;
      data[i+1]=key;
    }
  }
  return data;
}
```

Task 8: Get familiar with writing a method for ***Bubble Sort*** in Java

```
public int[] bubbleSort(int[] data){
  int lenD = data.length;
  int tmp = 0;
  for(int i = 0;i<lenD;i++){
    for(int j = (lenD-1);j>=(i+1);j--){
      if(data[j]<data[j-1]){
        tmp = data[j];
        data[j]=data[j-1];
        data[j-1]=tmp;
      }
    }
  }
  return data;
}
```

Task 9: Get familiar with writing a method for ***Quick Sort*** in Java

```
public int[] quickSort(int[] data){
int lenD = data.length;
int pivot = 0;
int ind = lenD/2;
int i,j = 0,k = 0;
if(lenD<2){
  return data;
}
else{
  int[] L = new int[lenD];
  int[] R = new int[lenD];
  int[] sorted = new int[lenD];
  pivot = data[ind];
  for(i=0;i<lenD;i++){
    if(i!=ind){
      if(data[i]<pivot){
        L[j] = data[i];
        j++;
      }
      else{
        R[k] = data[i];
        k++;
      }
    }
  }
  int[] sortedL = new int[j];
  int[] sortedR = new int[k];
  System.arraycopy(L, 0, sortedL, 0, j);
  System.arraycopy(R, 0, sortedR, 0, k);
  sortedL = quickSort(sortedL);
  sortedR = quickSort(sortedR);
  System.arraycopy(sortedL, 0, sorted, 0, j);
  sorted[j] = pivot;
  System.arraycopy(sortedR, 0, sorted, j+1, k);
  return sorted;
  }
}
```

Task 10: Get familiar with writing a method for ***Merge Sort*** in Java

```
public int[] mergeSort(int[] data){
  int lenD = data.length;
  if(lenD<=1){
    return data;
  }
  else{
    int[] sorted = new int[lenD];
    int middle = lenD/2;
    int rem = lenD-middle;
    int[] L = new int[middle];
    int[] R = new int[rem];
    System.arraycopy(data, 0, L, 0, middle);
    System.arraycopy(data, middle, R, 0, rem);
    L = this.mergeSort(L);
    R = this.mergeSort(R);
    sorted = merge(L, R);
    return sorted;
  }
}
public int[] merge(int[] L, int[] R){
  int lenL = L.length;
  int lenR = R.length;
  int[] merged = new int[lenL+lenR];
  int i = 0;
  int j = 0;
  while(i<lenL||j<lenR){
    if(i<lenL & j<lenR){
      if(L[i]<=R[j]){
        merged[i+j] = L[i];
        i++;
      }
      else{
        merged[i+j] = R[j];
        j++;
      }
    }
    else if(i<lenL){
      merged[i+j] = L[i];
      i++;
    }
    else if(j<lenR){
      merged[i+j] = R[j];
      j++;
     }
   }
   return merged;
}
```