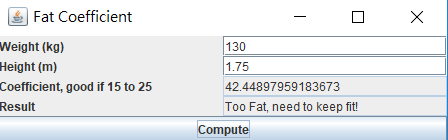
Q1.

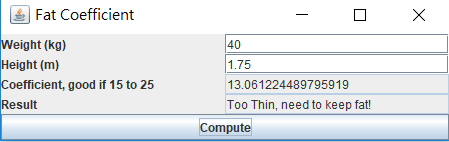
1. Write a java application which accepts the weight (kg) and height (m), then it calculates the Fat Coefficient (FC): ***FC = weight / (height \* height)*** and displays a statement in the result line according to the following rules.

I use Swing to design a graphical user interface, which makes it easy for users to use. This program is a Fat Coefficient calculator. Users can calculate your Fat Coefficient value for you by input height and weight data. Moreover, the program will remind you of your physical condition based on your FC value.

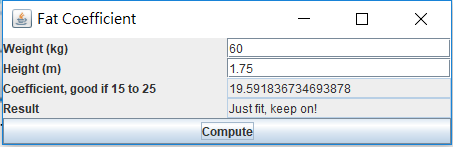
If your FC is greater than 25, the program will remind you of "Too Fat, need to keep fit!"



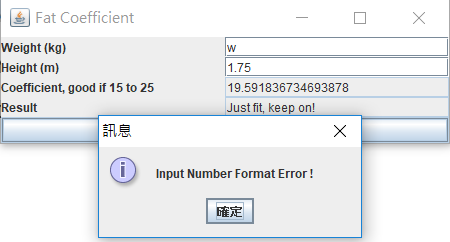
If you are less than 25, it will remind you of "Too Thin, need to keep fat!"

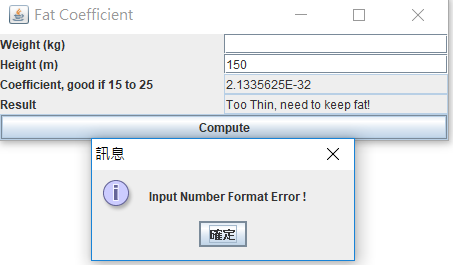


If you are at normal level, it will remind you of "Just fit, keep on!".



Users may be unfamiliar with the program, or inadvertently cause input errors. Data, such as typing in English, will pop up the error, "Input Number Format Error!" or "input error".





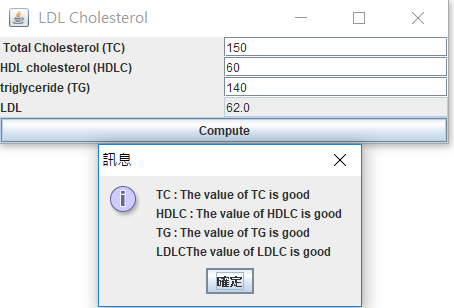
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-10-18-下午 08:20  
 \*\*/***import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** javax.swing.\*;  
  
**public class** Fat\_Coefficient\_calculator **extends** JFrame **implements** ActionListener {  
 *//create TextField and button for user input and calculate* **private** JTextField jtfWeight = **new** JTextField(10);  
 **private** JTextField jtfHeight = **new** JTextField(10);  
 **private** JTextField jtfCofficient = **new** JTextField(20);  
 **private** JTextField jtfResult = **new** JTextField(20);  
 **private** JButton jbCompute = **new** JButton(**"Compute"**);  
  
 *// Main method* **public static void** main(String[ ] args) {  
 *// GUI* Fat\_Coefficient\_calculator frame = **new** Fat\_Coefficient\_calculator( );  
 *//create and set up the title of the window* frame.setTitle(**"Fat Coefficient"**);  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); frame.pack();

*//display the windows*  
 frame.setVisible(**true**);  
 }  
  
 **public** Fat\_Coefficient\_calculator( ) {  
 *// Panel p1 to hold labels and text fields  
 /\* create a panel, which is similar to HTML's div tag.  
 \* we can create multiple panels and specify locations in JFrame.  
 \* we can add text fields, buttons and other components in the panel.  
 \*/* JPanel p1 = **new** JPanel( );  
 p1.setLayout(**new** GridLayout(4, 2));  
 p1.add(**new** JLabel(**"Weight (kg)"**));  
 p1.add(jtfWeight);  
 p1.add(**new** JLabel(**"Height (m)"**));  
 p1.add(jtfHeight);  
 p1.add(**new** JLabel(**"Coefficient, good if 15 to 25"**));  
 p1.add(jtfCofficient);  
 jtfCofficient.setEditable(**false**);  
 p1.add(**new** JLabel(**"Result"**));  
 p1.add(jtfResult);  
 jtfResult.setEditable(**false**);  
  
 *// Add p1 to the frame* **this**.getContentPane().setLayout(**new** BorderLayout());  
 **this**.getContentPane().add(p1,BorderLayout.CENTER);  
 **this**.getContentPane().add(jbCompute,BorderLayout.SOUTH);  
  
 *// Register listener* jbCompute.addActionListener(**this**);  
 }  
  
 **public void** actionPerformed(ActionEvent e) {  
 *//When the listener the user press the button* **if** (e.getSource() == jbCompute) {  
 **try** {  
 */\*calculate the FC  
 \* got the heigh and weight but they are String type, so convert to the Double to calculate  
 \* base on the FC, display the suggestion message  
 \* \*/* Double Height = Double.parseDouble(**this**.jtfHeight.getText());  
 Double Weight = Double.parseDouble(**this**.jtfWeight.getText());  
 Double FC = Weight / (Height \* Height);  
 String result = Double.toString(FC);  
 jtfCofficient.setText(result);  
 **if** ( FC > 25)  
 {  
 jtfResult.setText(**"Too Fat, need to keep fit!"**);  
 }  
 **else if** (FC < 15)  
 {  
 jtfResult.setText(**"Too Thin, need to keep fat!"**);  
 }  
 **else** {  
 jtfResult.setText(**"Just fit, keep on!"**);  
 }  
 *//when input data is wrong the application will raise up the error message to the user* }**catch** (NumberFormatException e1)  
 {  
 JOptionPane.showMessageDialog(**null**,**"Input Number Format Error !"**);  
 }**catch** (Exception e2)  
 {  
 JOptionPane.showMessageDialog(**null**,**"Input Error!"**);  
 }  
*//write your code here* } *// end of if* } *// end of ActionPerformed*} *// end Fat\_Coefficient\_calculator*

Q2. Total Cholesterol (TC), HDL cholesterol (HDLC), and triglyceride (TG) levels are counted directly from a blood sample. LDL cholesterol (LDLC) is calculated by using the formula:

I use Swing to design a graphical user interface, which makes it easy for users to use. This program is a Total Cholesterol calculator. Users can calculate your LDL cholesterol value for you by input your TC, HDLC and TG data.

Base on the Cholesterol Guide lines, the program will remind you of your physical condition based on your value

.

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-10-18-下午 08:29  
 \*\*/***import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** javax.swing.\*;  
**public class** LDL\_Cholesterol\_Calculator **extends** JFrame **implements** ActionListener {  
 *//create TextField and button for user input and calculate* **private** JTextField **jtfTotal\_Cholestero** = **new** JTextField(10);  
 **private** JTextField **jtfHDL\_cholesterol** = **new** JTextField(10);  
 **private** JTextField **jtftriglyceride** = **new** JTextField(10);  
 **private** JTextField **jtfResult** = **new** JTextField(20);  
 **private** JButton **jbCompute** = **new** JButton(**"Compute"**);  
 *// Main method* **public static void** main(String[ ] args) {  
 LDL\_Cholesterol\_Calculator frame = **new** LDL\_Cholesterol\_Calculator( );  
 frame.pack();  
 *//create and set up the title of the window* frame.setTitle(**" LDL Cholesterol"**);  
 frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 *//display the windows* frame.setVisible(**true**);  
 }  
 **public** LDL\_Cholesterol\_Calculator( ) {  
 *// Panel p1 to hold labels and text fields  
 /\* create a panel, which is similar to HTML's div tag.  
 \* we can create multiple panels and specify locations in JFrame.  
 \* we can add text fields, buttons and other components in the panel.  
 \*/* JPanel p1 = **new** JPanel( );  
 p1.setLayout(**new** GridLayout(4, 2));  
 p1.add(**new** JLabel(**" Total Cholesterol (TC)"**));  
 p1.add(**jtfTotal\_Cholestero**);  
 p1.add(**new** JLabel(**"HDL cholesterol (HDLC)"**));  
 p1.add(**jtfHDL\_cholesterol**);  
 p1.add(**new** JLabel(**"triglyceride (TG)"**));  
 p1.add(**jtftriglyceride**);  
 p1.add(**new** JLabel(**"LDL"**));  
 p1.add(**jtfResult**);  
 **jtfResult**.setEditable(**false**);  
 *// Add p1 to the frame* **this**.getContentPane().setLayout(**new** BorderLayout());  
 **this**.getContentPane().add(p1,BorderLayout.***CENTER***);  
 **this**.getContentPane().add(**jbCompute**,BorderLayout.***SOUTH***);  
 *// Register listener* **jbCompute**.addActionListener(**this**);  
 }  
 **public void** actionPerformed(ActionEvent e) {  
 *//When the listener the user press the button* **if** (e.getSource() == **jbCompute**) {  
 **try** {  
 */\*calculate the FC  
 \* got the TC, HDLC and TG but they are String type, so convert to the Double to calculate  
 \* base on these vale and the LDLC, display the suggestion message  
 \* \*/* String TC\_massage = **""**;  
 String HDLC\_massage = **""**;  
 String TG\_massage = **""**;  
 String LDLC\_massage = **""**;  
 Double TC = Double.*parseDouble*(**this**.**jtfTotal\_Cholestero**.getText());  
 Double HDLC = Double.*parseDouble*(**this**.**jtfHDL\_cholesterol**.getText());  
 Double TG = Double.*parseDouble*(**this**.**jtftriglyceride**.getText());  
 Double LDLC = TC - HDLC - (TG / 5);  
 String result = Double.*toString*(LDLC);  
 **jtfResult**.setText(result);  
 **if** (TC < 200)  
 TC\_massage += **"The value of TC is good"**;  
 **else if** (TC>130&&TC<239)  
 TC\_massage += **"be careful"**;  
 **else** TC\_massage += **"The value is too high. Please go to the hospital quickly."**;  
 **if** (HDLC > 40)  
 HDLC\_massage += **"The value of HDLC is good"**;  
 **else** HDLC\_massage += **"The value is too high. Please go to the hospital quickly."**;  
 **if** (TG < 150)  
 TG\_massage += **"The value of TG is good"**;  
 **else if** (TG>150&&TG<199)  
 TG\_massage += **"be careful"**;  
 **else** TG\_massage += **"The value is too high. Please go to the hospital quickly."**;  
 **if** (LDLC < 130)  
 LDLC\_massage += **"The value of LDLC is good"**;  
 **else if** (TG>130&&TG<159)  
 LDLC\_massage += **"be careful"**;  
 **else** LDLC\_massage += **"The value is too high. Please go to the hospital quickly."**;  
 JOptionPane.*showMessageDialog*(**null**,**"TC : "**+TC\_massage +**"\n"**+**"HDLC : "**+HDLC\_massage+**"\n"**+**"TG : "**+TG\_massage +**"\n"**+**"LDLC"**+LDLC\_massage);  
  
 *//when input data is wrong the application will raise up the error message to the user* }**catch** (NumberFormatException e1)  
 {  
 JOptionPane.*showMessageDialog*(**null**,**"Input Number Format Error !"**);  
 }**catch** (Exception e2)  
 {  
 JOptionPane.*showMessageDialog*(**null**,**"Input Error!"**);  
 }  
*//write your code here* } *// end of if* } *// end of ActionPerformed*} *// end LDL\_Cholesterol\_Calculator*

Q3.

Q4

**Based on the java code given during lecture & lab session, write the following LinkedList methods:**

**addToHead(Object item)**

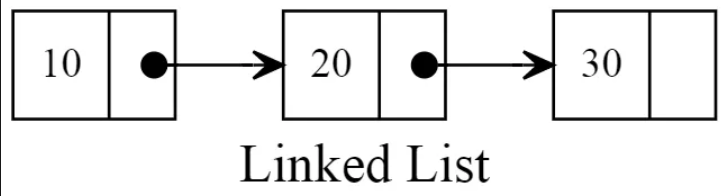
**addToTail(Object item)**

**removeFromHead()**

**removeFromTail()**

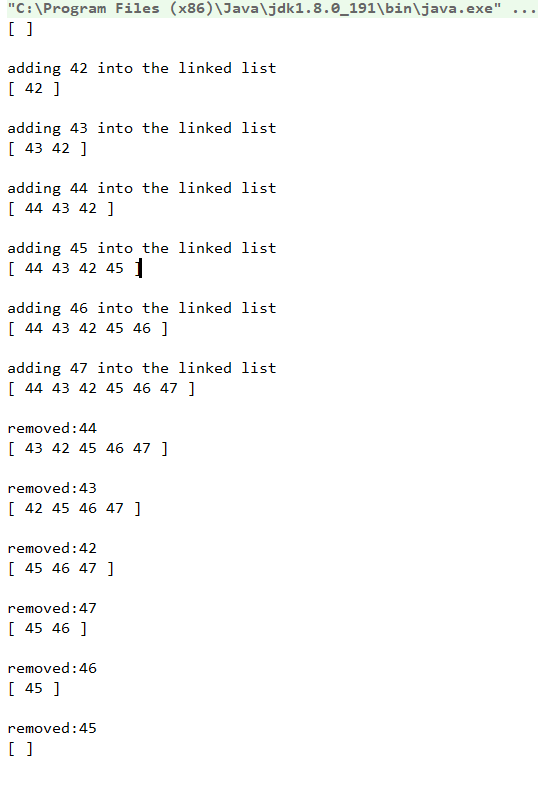
I implemented the addToHead, addToTail, removeFromHead and removeFromTail functions of linked list in this program without using

library. Build a node. and connecting each Node to form a linear data structure.



Test

**import** java.util.LinkedList;  
  
*/\*\*  
 \* Title: TestLinkedList.java  
 \* Description: A simple test drive program for the LinkedList class  
 \* Company: ICT HKIVE(TY)  
 \** ***@author*** *\*/***public class** TestLinkedList {  
 **public static void** main (String args[]) {  
 public\_linked\_lisk s = **new** public\_linked\_lisk();  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"adding 42 into the linked list "**);  
 s.addToHead (**new** Integer(42));  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"adding 43 into the linked list "**);  
 s.addToHead (**new** Integer(43));  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"adding 44 into the linked list "**);  
 s.addToHead (**new** Integer(44));  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"adding 45 into the linked list "**);  
 s.addToTail (**new** Integer(45));  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"adding 46 into the linked list "**);  
 s.addToTail (**new** Integer(46));  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"adding 47 into the linked list "**);  
 s.addToTail (**new** Integer(47));  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"removed:"** + s.removeFromHead());  
  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"removed:"** + s.removeFromHead());  
  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"removed:"** + s.removeFromHead());  
  
 System.***out***.println(s);  
 System.***out***.println();  
  
 System.***out***.println(**"removed:"** + s.removeFromTail());  
  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"removed:"** + s.removeFromTail());  
  
 System.***out***.println(s);  
 System.***out***.println();  
 System.***out***.println(**"removed:"** + s.removeFromTail());  
  
 System.***out***.println(s);  
 System.***out***.println();  
  
 }  
} *// class TestLinkedList*



Linked List.java

**import** java.util.List;  
  
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-10-20-下午 07:35  
 \*\*/  
//Node constructor***class** ListNode {  
 **public** Object data;  
 **public** ListNode next;  
 **public** ListNode(Object o)  
 {  
 data = o;  
 }  
} *// class ListNode***class** EmptyListException **extends** RuntimeException {  
 **public** EmptyListException () { **super** (**"List is empty"**); }  
} *// class EmptyListException***class** public\_linked\_lisk {  
 **public** ListNode head;  
 **public** ListNode tail;  
 **public boolean** isEmpty() { **return** head == **null**; }  
 **public void** addToHead(Object item)  
 {  
 ListNode new\_node = **new** ListNode(item);  
 **if** (head == **null**)  
 {  
 head = new\_node;  
 **return**;  
 }  
 */\*  
 \* if the linked list is null ,the head address will be point to new\_node address  
 \*/* new\_node.next = head; *//the new node point the next to the old head of the linked list as a new head node* head = new\_node;  
 **return**;  
 }  
 **public void** addToTail(Object item)  
 {  
 ListNode new\_node = **new** ListNode(item);  
 **if** (head == **null**)  
 {  
 head = new\_node;  
 **return**;  
 }  
 */\*  
 \* if the linked list is null ,the head address will be point to new\_node address  
 \*/* ListNode cur = head; *// declare the variable current, and point to the first node address* ListNode prv = cur; *//declare the prv and point to the prv of the cur node* cur = cur.next; *//point the cur address to front of the prv node* **while** (cur != **null**)  
 {  
 cur = cur.next;  
 prv = prv.next;  
 }  
 */\*  
 \* get out of the loop when the cur node is null  
 \*/* prv.next = new\_node;  
 **return**;  
 }  
 **public** Object removeFromHead()  
 {  
 **if** (head == **null**)  
 {  
 **return** -1;  
 }  
 */\*  
 \* if the head is null , just return  
 \*/* ListNode cur = **head**;  
 **head** = cur.**next**; *//declare the cur and point to head, then point the head to the next of current node* **return** cur.**data**;  
 }  
 **public** Object removeFromTail()  
 {  
 **if** (**head** == **null**)  
 {  
 **return** -1;  
 }  
 */\*  
 \* if the head is null , just return  
 \*/* ListNode cur = **head**;  
 **if** (cur.**next** == **null**)  
 {  
  
 Object tmp = cur.**data**;  
 **head** = **null**;  
 **return** tmp;  
 }  
 */\*  
 \* if the second node is null , point the head node to null  
 \*/* **while** (cur.**next**.**next** != **null**)  
 {  
 cur = cur.**next**;  
  
 }  
 */\*  
 \* get out of the loop when the next node of next node of current node is null  
 \*/* Object tmp = cur.next.data;  
 cur.next = cur.next.next; *//the current node will point to the null for remove the tail node* **return** tmp;  
 }  
 **public** String toString () {  
 String s = **"[ "**;  
 ListNode current = head;  
 **while** (current != **null**) {  
 s += current.data + **" "**;  
 current = current.next;  
 }  
 **return** s + **"]"**;  
 }  
  
  
} *// class LinkedList*

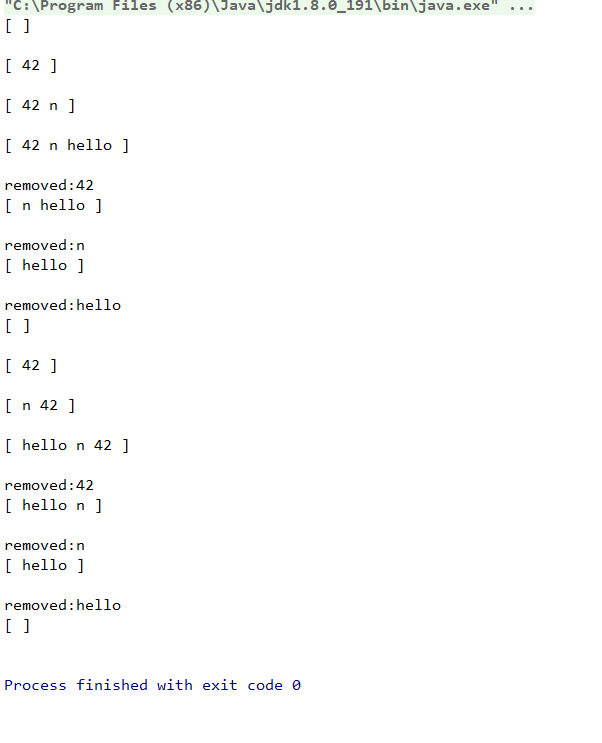
Q5

The design of the above given ListNode class violates the principle of information hiding. Modify the above classes so that the principle of information hiding is enforced

In programming, the process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity. One of the chief mechanisms for hiding information is encapsulation. In order not to allow users to change the data of Node at will, we need to setting variables to private, users can not change data outside the class, so I build the interface, users can only change the data through the method I set up.

Testing

*/\*\*  
 \* Title: TestLinkedList.java  
 \* Description: A simple test drive program for the LinkedList class  
 \* Company: ICT HKIVE(TY)  
 \** ***@author*** *\*/***public class** TestLinkedList {  
 **public static void** main (String args[]) {  
 LinkedList s = **new** LinkedList();  
 System.***out***.println(s);  
 System.***out***.println();  
  
 s.addToTail (**new** Integer(42));  
 System.***out***.println(s);  
 System.***out***.println();  
 s.addToTail (**new** Character(**'n'**));  
 System.***out***.println(s);  
 System.***out***.println();  
 s.addToTail (**new** String(**"hello"**));  
 System.***out***.println(s);  
 System.***out***.println();  
 **while** (!s.isEmpty()) {  
 System.***out***.println(**"removed:"** + s.removeFromHead());  
 System.***out***.println(s);  
 System.***out***.println();  
 }  
  
 s.addToHead (**new** Integer(42));  
 System.***out***.println(s);  
 System.***out***.println();  
 s.addToHead (**new** Character(**'n'**));  
 System.***out***.println(s);  
 System.***out***.println();  
 s.addToHead (**new** String(**"hello"**));  
 System.***out***.println(s);  
 System.***out***.println();  
 **while** (!s.isEmpty()) {  
 System.***out***.println(**"removed:"** + s.removeFromTail());  
 System.***out***.println(s);  
 System.***out***.println();  
 }  
 }  
} *// class TestLinkedList*



Linked List.java in Q5

**import** java.security.PublicKey;  
  
**class** ListNode {  
 **private** Object **data**;  
 **private** ListNode **next**;  
 **public** ListNode(Object o) { **data** = o; **next** = **null**; }  
 **public** ListNode(Object o, ListNode nextNode)  
 { **data** = o; **next** = nextNode; }  
 **public** Object getData() { **return data**; }  
 **public** ListNode getNext() { **return next**; }  
 **public void** setData(Object o, ListNode curNode) { curNode.**data** = o; }  
 **public void** setNext(ListNode nextNode, ListNode curNode) { curNode.**next** = nextNode; }  
} *// class ListNode***class** EmptyListException **extends** RuntimeException {  
 **public** EmptyListException () { **super** (**"List is empty"**); }  
} *// class EmptyListException***class** LinkedList {  
 **private** ListNode **head**;  
 **private** ListNode **tail**;  
 **public** LinkedList() { **head** = **tail** = **null**; }  
 **public boolean** isEmpty() { **return head** == **null**; }  
 **public void** addToHead(Object item) {  
 ListNode new\_node = **new** ListNode(item);  
 **if** (isEmpty())  
 {  
 **this**.**head** = new\_node;  
 **this**.**tail** = new\_node;  
 }  
 **else** {  
 ListNode cur = **this**.**head**;  
 ListNode tmp = cur;  
 ListNode node= **new** ListNode(new\_node.getData(),tmp);  
  
 **this**.**head** = node;  
 }  
 }  
 **public void** addToTail(Object item) {  
 ListNode new\_node = **new** ListNode(item);  
 **if** (isEmpty())  
 {  
 **this**.**head** = new\_node;  
 **this**.**tail** = new\_node;  
 }  
 **else** {  
 ListNode cur = **this**.**head**;  
 **while** (cur.getNext() != **null**)  
 {  
 cur = cur.getNext() ;  
 }  
 cur.setNext(new\_node,cur);  
  
 **this**.**tail** = new\_node;  
 }  
 }  
 **public** Object removeFromHead() **throws** EmptyListException {  
 **if** ( isEmpty() ) *// throw exception if List is empty* **throw new** EmptyListException();  
  
 ListNode cur = **this**.**head**;  
 **if** (**this**.**head** == **this**.**tail**)  
 {  
 **this**.**head** = **this**.**tail** = **null**;  
 **return** cur.getData();  
 }  
 **if** (**this**.**head** == **null**)  
 {  
 System.***out***.println(**"head is null"**);  
 **return this**.**head**;  
 }  
  
 ListNode tmp = cur.getNext() ;  
 **this**.**head** = tmp;  
  
  
 **return** cur.getData();  
  
 }  
 **public** Object removeFromTail() **throws** EmptyListException {  
 **if** ( isEmpty() ) *// throw exception if List is empty* **throw new** EmptyListException();  
 ListNode prev = **this**.**head**;  
 ListNode cur = prev.getNext() ;  
  
 **if** (**this**.**head** == **this**.**tail**)  
 {  
 ListNode head = **this**.**head**;  
 **this**.**head** = **this**.**tail** = **null**;  
 **return** head.getData();  
 }  
 **while** (cur.getNext() != **null**)  
 {  
 prev = prev.getNext() ;  
 cur = cur.getNext() ;  
 }  
 prev.setNext(**null**,prev);  
 **this**.**tail** = prev;  
 **return** cur.getData();  
 }  
 **public** String toString () {  
 String s = **"[ "**;  
 ListNode current = **head**;  
 **while** (current != **null**) {  
 s += current.getData() + **" "**;  
 current = current.getNext() ;  
 }  
 **return** s + **"]"**;  
 }  
} *// class LinkedList*

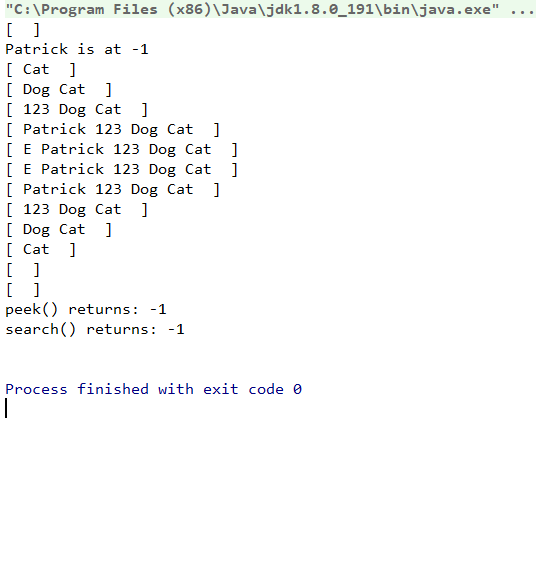
Q6

Based on the LinkedList class developed in previous week, write your own ListStack class by inheriting the LinkedList class. Your ListStack class should provide the same methods of Java API Stack. The methods are constructor(s) ListStack, empty(), push(), pop(), peek() and search(). You should read the Java API doc for the Stack class. Design and write a test program to test your ListStack class.

Using inherited linked list to implement stack. Stack is a linear data structure which follows a specific order in which operations are performed. The order may be LIFO (Last In First Out) or FILO (First In Last Out). So, in the pop method of stack, it points to the head of the list. The push method inserts the data into the head node of the list.

The Testing

*/\*\*  
 \* Title: TestStack.java  
 \* Description: Test driver for Stack class  
 \*  
 \* Company: ICT HKIVE(TY)  
 \** ***@author*** *Patrick Tong  
 \*/***import** java.util.Stack;  
**import** java.util.Iterator;  
  
**public class** TestStack {  
 **public static void** main (String args[]) **throws** ExceptionQueueFull {  
 ListStack s = **new** ListStack(5);  
 System.***out***.println(s);  
 System.***out***.println(**"Patrick is at "** + s.search(**"Patrick"**));  
 s.push(**"Cat"**);  
 System.***out***.println(s);  
 s.push(**"Dog"**);  
 System.***out***.println(s);  
 s.push(**new** Integer(123));  
 System.***out***.println(s);  
 s.push(**"Patrick"**);  
 System.***out***.println(s);  
 s.push(**new** Character(**'E'**));  
 System.***out***.println(s);  
 s.push(**new** Double(789.123));  
 System.***out***.println(s);  
 s.pop();  
 System.***out***.println(s);  
 s.pop();  
 System.***out***.println(s);  
 s.pop();  
 System.out.println(s);  
 s.pop();  
 System.out.println(s);  
 s.pop();  
 System.out.println(s);  
 s.pop();  
 System.out.println(s);  
 System.out.println(**"peek() returns: "** + s.peek());  
 System.out.println(**"search() returns: "** + s.search(**"123"**));  
  
  
 System.out.println();  
 }  
  
} *// class TestStack*



Linked List .java in Q6

*/\*\*  
 \* Title: LinkedList.java  
 \* Description: Contains classes for linked list  
 \* Company: ICT HKIVE(TY)  
 \** ***@author*** *\*  
 \* (for Lab03)  
 \*/***class** ListNode {  
  
 **private** Object **data**;  
 **private** ListNode **next**;  
  
 **public** ListNode(Object o) { **data** = o; **next** = **null**; }  
 **public** ListNode(Object o, ListNode nextNode)  
 { **data** = o; **next** = nextNode; }  
  
 **public** Object getData() { **return data**; }  
 **public void** setData(Object o) { **data** = o; }  
  
 **public** ListNode getNext() { **return next**; }  
 **public void** setNext(ListNode next) { **this**.**next** = next; }  
  
} *// class ListNode***class** EmptyListException **extends** RuntimeException {  
 **public** EmptyListException ()  
 { **super**(**"List is empty"**); }  
} *// class EmptyListException***public class** LinkedList {  
  
 **protected** ListNode **head**; *// <== chnage to protected for inheriting* **protected** ListNode **tail**; *// <== change to protected for inheriting* **protected int length**; *// the length of the list <== chnage to protected for inheriting* **public** LinkedList() {  
 **head** = **tail** = **null**;  
 **length** = 0;  
 }  
  
 **public boolean** isEmpty() { **return head** == **null**; }  
  
 **public void** addToHead(Object item) {  
 **if** (isEmpty())  
 **head** = **tail** = **new** ListNode(item);  
 **else  
 head** = **new** ListNode(item, **head**);  
 **length**++;  
 }  
  
 **public void** addToTail(Object item) {  
 **if** (isEmpty())  
 **head** = **tail** = **new** ListNode(item);  
 **else** {  
 **tail**.setNext(**new** ListNode(item));  
 **tail** = **tail**.getNext();  
 }  
 **length**++;  
 }  
  
 **public** Object removeFromHead() **throws** EmptyListException {  
 Object item = **null**;  
 **if** (isEmpty()) {  
  
 **return null**;  
 *//throw new EmptyListException();* }  
 item = **head**.getData();  
 **if** (**head** == **tail**)  
 **head** = **tail** = **null**;  
 **else  
 head** = **head**.getNext();  
 **length**--;  
 **return** item;  
 }  
  
 **public** Object removeFromTail() **throws** EmptyListException {  
 Object item = **null**;  
 **if** (isEmpty())  
 **throw new** EmptyListException();  
 item = **tail**.getData();  
 **if** (**head** == **tail**)  
 **head** = **tail** = **null**;  
 **else** {  
 ListNode current = **head**;  
 **while** (current.getNext() != **tail**)  
 current = current.getNext();  
 **tail** = current;  
 current.setNext(**null**);  
 }  
 **length**--;  
 **return** item;  
 }  
  
 **public** String toString() {  
 String str = **"[ "**;  
 ListNode current = **head**;  
 **while** (current != **null**) {  
 str = str + current.getData() + **" "**;  
 current = current.getNext();  
 }  
 **return** str + **" ]"**;  
 }  
  
 **public int** count() {  
 **return length**;  
 }  
  
 **public** Object remove(**int** n) {  
 Object item = **null**;  
 **if** (n <= **length**) { *// make sure there is nth node to remove  
 // special treatment for first and last nodes* **if** (n == 1) **return** removeFromHead();  
 **if** (n == **length**) **return** removeFromTail();  
 *// removal of nth node which has nodes in front and behind* ListNode current = **head**;  
 ListNode previous = **null**;  
 **for** (**int** i = 1; i < n; i++) { *// current will point to nth node* previous = current;  
 current = current.getNext();  
 }  
 *// data to be returned* item = current.getData();  
 *// remove the node by adjusting two pointers (object reference)* previous.setNext(current.getNext());  
 }  
 **length**--;  
 **return** item;  
 }  
  
 **public void** add(**int** n, Object item) {  
 *// special treatment for insert as first node* **if** (n == 1) {  
 addToHead(item);  
 **return**;  
 }  
 *// special treatment for insert as last node* **if** (n > **length**) {  
 addToTail(item);  
 **return**;  
 }  
 *// locate the n-1th node* ListNode current = **head**;  
 **for** (**int** i = 1; i < n-1; i++) *// current will point to n-1th node* current = current.getNext();  
 *// create new node and insert at nth position* current.setNext(**new** ListNode(item, current.getNext()));  
 **length**++;  
 }  
  
 **public int** search (Object o)  
 {  
 ListNode cur = **head**;  
 **if** (cur == **null**)  
 {  
 **return** -1;  
 }  
 **int** count = 1;  
 **while** (cur != **null**)  
 {  
 String a = cur.getData().toString();  
 String b = o.toString();  
 **if** (a.equals(b))  
 *// == both object point to same memory location  
 // equal compare the value of the object* {  
 **return** count;  
 }  
 count += 1;  
 cur = cur.getNext();  
 }  
  
 System.***out***.println(**"null"**);  
 **return** -1;  
 }  
  
 *//return the head if the head is not null* **public** Object get\_head() {  
 **if** (**head** == **null**)  
 {  
 **return** -1;  
 }  
 **return head**.getData();  
 }  
  
} *// class LinkedList*

Linked Stack.java in Q6

*/\*\*  
 \* Title: ListStack.java  
 \* Description: A Stack class extended from LinkedList class  
 \* Company: ICT HKIVE(TY)  
 \** ***@author*** *Patrick Tong  
 \*/***import** java.util.EmptyStackException;  
**class** StackNode {  
  
 **private** Object **data**;  
 **private** StackNode **next**;  
  
 **public** StackNode(Object o) { **data** = o; **next** = **null**; }  
 **public** StackNode(Object o, StackNode nextNode)  
 { **data** = o; **next** = nextNode; }  
  
 **public** Object getData() { **return data**; }  
 **public void** setData(Object o) { **data** = o; }  
  
 **public** StackNode getNext() { **return next**; }  
 **public void** setNext(StackNode next) { **this**.**next** = next; }  
  
}  
**class** ExceptionQueueFull **extends** Exception {  
  
 *// Constructor* **public** ExceptionQueueFull() {  
  
 }  
  
 *// Constructor with parameters* **public** ExceptionQueueFull(String mag) {  
 System.***out***.println(mag);  
 }  
}  
**public class** ListStack **extends** LinkedList {  
 **private static int** *max\_length*;  
 **private static int** *cur\_length*;  
  
  
 **public** ListStack(**int** max\_length) { *// <== constructor, different from ListStackComp.java* **this**.*max\_length* = max\_length;  
 *cur\_length* = 1;  
 }  
  
 **public boolean** empty() {  
 **return** isEmpty();  
 }  
  
 **public** Object push(Object item) **throws** ExceptionQueueFull {  
 *//raise up error if existing length of list is grater than max length* **if** (*cur\_length* > *max\_length*)  
 {  
 **return null**;  
 *//throw new ExceptionQueueFull("stack is empty");* }  
 *cur\_length* += 1;  
 addToHead(item);  
 **return** item;  
 }  
  
 **public** Object pop() {  
 *cur\_length* -= 1;  
 ListNode cur = **head**;  
 *// pop out the head node, but if the list is null , if will raise error* **if** (empty()==**true**)  
 {  
 *//the removeFromHead method extended from the list* **return** removeFromHead();  
 }  
 **else** {  
 ListNode tmp = cur;  
 removeFromHead();  
 **return** tmp.getData();  
  
 }  
 }  
  
 **public** Object peek() {  
 **return** get\_head();  
 }  
}

Q7

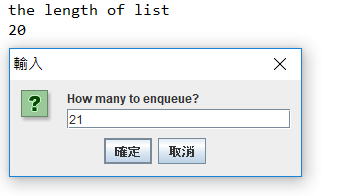
Based on the LinkedList class developed in previous week, write your own Queue class by composing a LinkedList object. The Queue class should provide methods such as constructor, Queue(), empty(), enqueue() and dequeue(). Design and write a test program to test your Queue class.

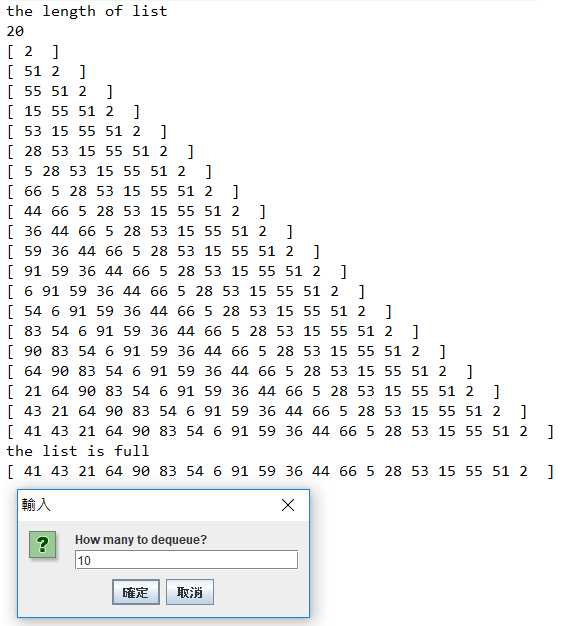
Using inherited linked list to implement Queue. Queue is a linear data structure which follows a specific order in which operations are performed. The order may be FIFO (First In First Out) or LILO (Last In Last Out). So, in the enqueue method of queue, it points to the tail of the list and insert the data to it. The dequeue method which will point to the head of the list and remove the data.

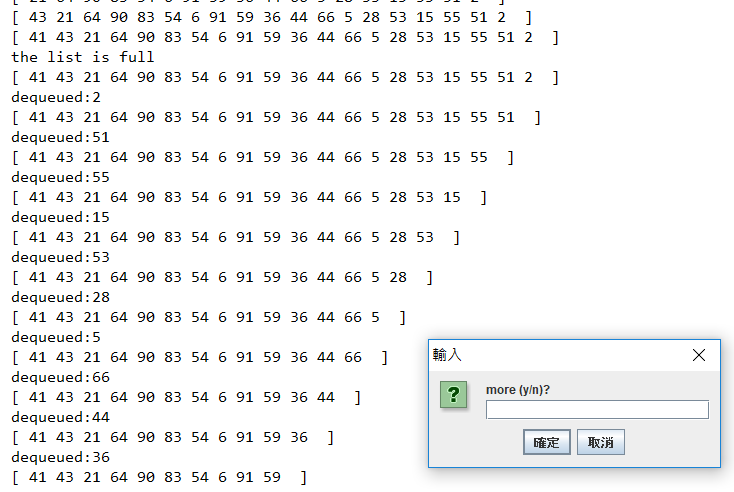
This program is the principle that allows the user to understand the queue. The length of the Queue has been defined as 20, the user can input the number of enqueues, the program will randomly generate the number and then queue. Also, the user can then input the number of elements to remove. it will show the process

Test

*/\*\*  
 \* Title: TestQueue2.java  
 \* Description: A test drive program for the Queue class  
 \* Used SimpleInput class methods for inputs  
 \* Company: ICT HKIVE(TY)  
 \** ***@author*** *Patrick Tong  
 \*/***import** miscLib.SimpleInput;  
**import** miscLib.GenLib;  
  
**public class** TestQueue {  
  
 **public static void** main(String[] args) **throws** ExceptionQueueFull {  
 ListQueue q = **new** ListQueue(20);  
*// ArrayQueue q = new ArrayQueue(20);* **byte** barray[] = **new byte**[20];  
 **int** n;  
 **do** {  
 n = SimpleInput.*getInteger*(**"How many to enqueue? "**);  
 **for** (**int** i = 0; i < n; i++) {  
 q.enqueue(**new** Integer(GenLib.*genInt*(1, 100)));  
 System.***out***.println(q);  
 }  
 n = SimpleInput.*getInteger*(**"How many to dequeue? "**);  
 **for** (**int** i = 0; i < n; i++) {  
 **if** (q.empty()) {  
 System.***out***.println(**"Queue is now empty, dequeue ignored."**);  
 **break**;  
 } **else** {  
 System.***out***.println(**"dequeued:"**+q.dequeue());  
 System.***out***.println(q);  
 }  
 }  
 } **while** (SimpleInput.*getChar*(**"more (y/n)? "**) == **'y'**);  
  
 System.***out***.println(**"Bye now."**);  
 System.*exit*(0);  
 }  
  
} *// clas TestQueue*







ListQueue.java in Q7

**import** sun.invoke.empty.Empty;  
  
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-10-05-上午 11:16  
 \*\*/***class** ExceptionQueueFull **extends** Exception {  
  
 *// Constructor* **public** ExceptionQueueFull() {  
  
 }  
  
 *// Constructor with parameters* **public** ExceptionQueueFull(String mag) {  
 System.***out***.println(mag);  
 }  
}  
  
**public class** ListQueue **extends** LinkedList{  
 *//existing length of queue* **private static int** *length\_of\_array*;  
 *//max length of queue* **private static int** *max\_length*;  
 *//declare the Length* **public** ListQueue(**int** capacity)  
 {  
 System.***out***.println(**"the length of list"**);  
 System.***out***.println(capacity);  
 *// why capacity - 1 because the list judge by length\_of\_array > max\_length  
 max\_length* = capacity-1;  
 *length\_of\_array* = 0;  
 }  
  
 **public boolean** empty()  
 {  
 **if** (**head** == **null**)  
 {  
 **return true** ;  
 }  
 **else** {  
 **return false**;  
 }  
 }  
 **public void** enqueue (Object o)**throws** ExceptionQueueFull  
 {  
  
 **if** (*length\_of\_array* > *max\_length*)  
 {  
 System.***out***.println(**"the list is full"**);  
 **return** ;  
 }  
 *length\_of\_array* += 1;  
 addToHead(o);  
 }  
 **public** Object dequeue()  
 {  
 *length\_of\_array* -= 1;  
 ListNode cur = **head**;  
 *// cur is store the tmp data to return which data is remove* **while** (cur.getNext() != **null**)  
 {  
 cur = cur.getNext();  
 }  
 removeFromTail();  
 **return** cur.getData();  
 }  
}

LinkedList.java in Q7

*/\*\*  
 \* Title: LinkedList.java  
 \* Description: Contains classes for linked list  
 \* Company: ICT HKIVE(TY)  
 \** ***@author*** *\*  
 \* (for Lab03)  
 \*/***class** ListNode {  
  
 **private** Object **data**;  
 **private** ListNode **next**;  
  
 **public** ListNode(Object o) { **data** = o; **next** = **null**; }  
 **public** ListNode(Object o, ListNode nextNode)  
 { **data** = o; **next** = nextNode; }  
  
 **public** Object getData() { **return data**; }  
 **public void** setData(Object o) { **data** = o; }  
  
 **public** ListNode getNext() { **return next**; }  
 **public void** setNext(ListNode next) { **this**.**next** = next; }  
  
} *// class ListNode***class** EmptyListException **extends** RuntimeException {  
 **public** EmptyListException ()  
 { **super**(**"List is empty"**); }  
} *// class EmptyListException***public class** LinkedList {  
  
 **protected** ListNode **head**; *// <== chnage to protected for inheriting* **protected** ListNode **tail**; *// <== change to protected for inheriting* **protected int length**; *// the length of the list <== chnage to protected for inheriting* **public** LinkedList() {  
 **head** = **tail** = **null**;  
 **length** = 0;  
 }  
  
 **public boolean** isEmpty() { **return head** == **null**; }  
  
 **public void** addToHead(Object item) {  
 **if** (isEmpty())  
 **head** = **tail** = **new** ListNode(item);  
 **else  
 head** = **new** ListNode(item, **head**);  
 **length**++;  
 }  
  
 **public void** addToTail(Object item) {  
 **if** (isEmpty())  
 **head** = **tail** = **new** ListNode(item);  
 **else** {  
 **tail**.setNext(**new** ListNode(item));  
 **tail** = **tail**.getNext();  
 }  
 **length**++;  
 }  
  
 **public** Object removeFromHead() **throws** EmptyListException {  
 Object item = **null**;  
 **if** (isEmpty())  
 **throw new** EmptyListException();  
 item = **head**.getData();  
 **if** (**head** == **tail**)  
 **head** = **tail** = **null**;  
 **else  
 head** = **head**.getNext();  
 **length**--;  
 **return** item;  
 }  
  
 **public** Object removeFromTail() **throws** EmptyListException {  
 Object item = **null**;  
 **if** (isEmpty())  
 **throw new** EmptyListException();  
 item = **tail**.getData();  
 **if** (**head** == **tail**)  
 **head** = **tail** = **null**;  
 **else** {  
 ListNode current = **head**;  
 **while** (current.getNext() != **tail**)  
 current = current.getNext();  
 **tail** = current;  
 current.setNext(**null**);  
 }  
 **length**--;  
 **return** item;  
 }  
  
 **public** String toString() {  
 String str = **"[ "**;  
 ListNode current = **head**;  
 **while** (current != **null**) {  
 str = str + current.getData() + **" "**;  
 current = current.getNext();  
 }  
 **return** str + **" ]"**;  
 }  
  
 **public int** count() {  
 **return length**;  
 }  
  
 **public** Object remove(**int** n) {  
 Object item = **null**;  
 **if** (n <= **length**) { *// make sure there is nth node to remove  
 // special treatment for first and last nodes* **if** (n == 1) **return** removeFromHead();  
 **if** (n == **length**) **return** removeFromTail();  
 *// removal of nth node which has nodes in front and behind* ListNode current = **head**;  
 ListNode previous = **null**;  
 **for** (**int** i = 1; i < n; i++) { *// current will point to nth node* previous = current;  
 current = current.getNext();  
 }  
 *// data to be returned* item = current.getData();  
 *// remove the node by adjusting two pointers (object reference)* previous.setNext(current.getNext());  
 }  
 **length**--;  
 **return** item;  
 }  
  
 **public void** add(**int** n, Object item) {  
 *// special treatment for insert as first node* **if** (n == 1) {  
 addToHead(item);  
 **return**;  
 }  
 *// special treatment for insert as last node* **if** (n > **length**) {  
 addToTail(item);  
 **return**;  
 }  
 *// locate the n-1th node* ListNode current = **head**;  
 **for** (**int** i = 1; i < n-1; i++) *// current will point to n-1th node* current = current.getNext();  
 *// create new node and insert at nth position* current.setNext(**new** ListNode(item, current.getNext()));  
 **length**++;  
 }  
  
 **public** Object get(**int** n) {  
 *// n is too big, no item can be returned* **if** (**length** < n) **return null**;  
 *// locate the nth node* ListNode current = **head**;  
 **for** (**int** i = 1; i < n; i++)  
 current = current.getNext();  
 **return** current.getData();  
 }  
  
} *// class LinkedList*

Q8

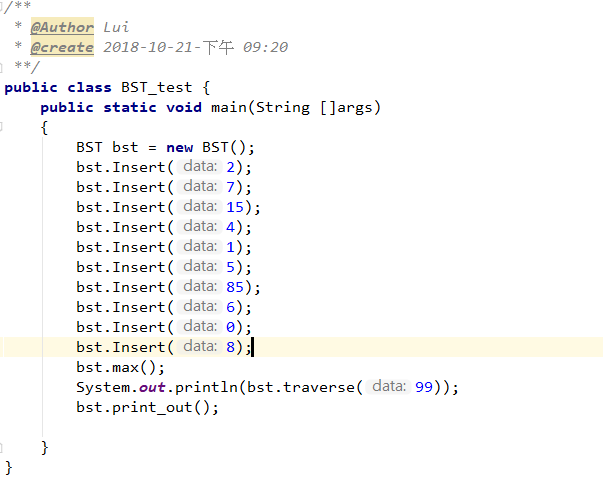
Based on the java code provided in lecture as a starting point, implement a Binary search tree class with all the necessary methods to insert, traverse and print out sequence of number. Design and test run your program by inserting the sequence [2 7 9 4 1 5 3 6 0 8] and printout the sequence again in ascending order.

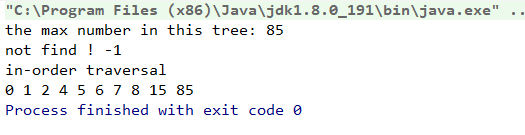
In this programme, I implement the binary search tree, there are 3 filed in each node, right, left and data. The tree is constructed by parent node, left child node and right child node. The left child node is less than parent node, the right node is greater than parent node. Each node is sorted according to this rule to form a binary tree

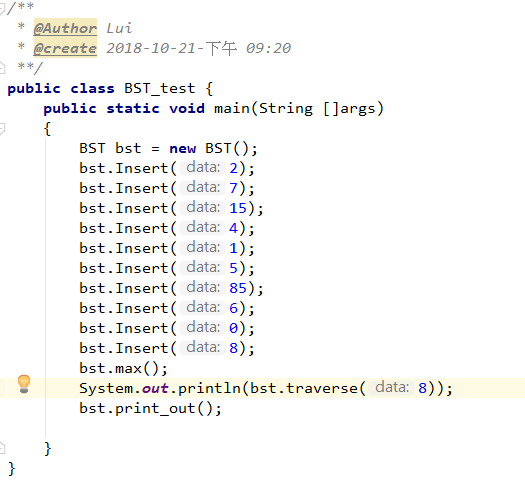
The theoretical traversal order is: the left subtree, the root, and the right subtree root are in the middle.

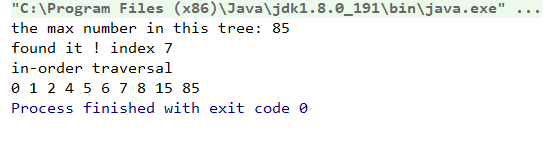
In fact, using Depth-first Search, it only changes the output order of the nodes.

Test









BST.java */\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-10-21-下午 08:57  
 \*\*/***class** BinaryNode  
{  
 **public int data**;  
 **public** BinaryNode **left**;  
 **public** BinaryNode **right**;  
 **public** BinaryNode(**int** data)  
 {  
 **this**.**data** = data;  
 **left** = **right** = **null**;  
 }  
 **public** BinaryNode()  
 {  
 **return**;  
 }  
}  
**public class** BST {  
 BinaryNode **root** ;  
 **public int** traverse(**int** data)  
 {  
 BinaryNode result;  
 result = traverse(data,**root**);  
 **if** (result != **null**) {  
 System.***out***.print(**"found it ! index "**);  
 **return** result.**data** - 1;  
 }  
 **else** {  
  
 System.***out***.print(**"not find ! "**);  
 **return** -1;  
 }  
 }  
 **public** BinaryNode traverse(**int** data, BinaryNode bst)  
 {  
 BinaryNode cur = bst;  
  
 **if** (**root** == **null**)  
 {  
 **return null**;  
 }  
  
 *// if the left node or right of current node is null it will return;* **if** (data<cur.**data** )  
 {  
 **if** (cur.**left** != **null**)  
 {  
  
 **return** traverse(data, cur.**left**);  
  
 }  
 **else** {  
 **return null**;  
 }  
 }  
  
 **else if** (data>cur.**data**)  
 {  
 **if** (cur.**right** != **null**)  
 {  
 **return** traverse(data, cur.**right**);  
 }  
 **else** {  
 **return null**;  
 }  
 }  
 **else** {  
  
 **return** cur;  
 }  
 }  
  
 **public void** Insert(**int** data)  
 {  
 **root** = Insert(data,**root**);  
 }  
  
 **public** BinaryNode Insert(**int** data, BinaryNode bst)  
 {  
  
 *// The left child node is less than parent node, the right node is greater than parent node* **if** (bst == **null**)  
 {  
 **root** = **new** BinaryNode(data);  
 **return root**;  
 }  
 **if** (data>bst.**data**)  
 {  
 bst.**right** = Insert(data, bst.**right**);  
 }  
 **if** (data<bst.**data**)  
 {  
 bst.**left** = Insert(data, bst.**left**);  
 }  
 **return** bst;  
 }  
  
 **public void** print\_out ()  
 {  
 *//System.out.println("Pre-order traversal");  
 //preorder(root);  
 //System.out.println();* System.***out***.println(**"in-order traversal"**);  
 inorder(**root**);  
 *//System.out.println();  
 //System.out.println("pro-order traversal");  
 //postorder(root);* }  
 **public** BinaryNode preorder (BinaryNode bst)  
 {  
 **if** (bst == **null**)  
 {  
 *//System.out.println("visited");* **return null**;  
 }  
 System.***out***.print(bst.**data** + **" "**);  
 preorder(bst.**left**);  
 preorder(bst.**right**);  
 **return** bst;  
 }  
 **public** BinaryNode inorder (BinaryNode bst)  
 {  
 **if** (bst == **null**)  
 {  
 *//System.out.println("visited");* **return null**;  
 }  
 inorder(bst.**left**);  
 System.***out***.print(bst.**data** + **" "**);  
 inorder(bst.**right**);  
  
 **return** bst;  
 }  
 **public** BinaryNode postorder (BinaryNode bst)  
 {  
 **if** (bst == **null**)  
 {  
 *//System.out.println("visited");* **return null**;  
 }  
 postorder(bst.**left**);  
 postorder(bst.**right**);  
 System.***out***.print(bst.**data** + **" "**);  
 **return** bst;  
 }  
 **public void** max()  
 {  
 Find\_max(**root**);  
 }  
 **public void** Find\_max (BinaryNode bst)  
 {  
 **if** (bst.**right** != **null**)  
 {  
  
 Find\_max(bst.**right**);  
 }  
 **else** {  
 System.***out***.println(**"the max number in this tree: "** + bst.**data**);  
 **return** ;  
 }  
 }  
}

Q9

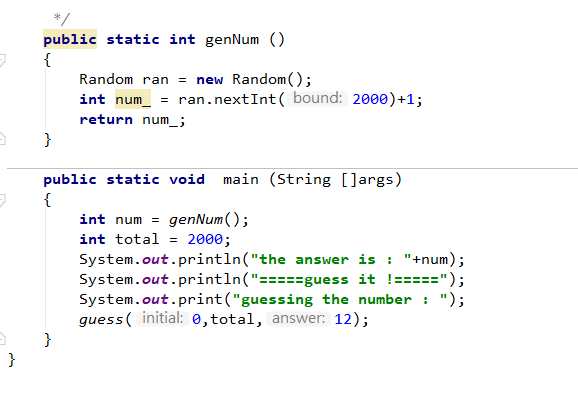
Use binary search in implementing a guessing game. One thinks of a number between 1 and 2000, the program attempts to guess the number and feedback is given whether my number is higher or lower. The program then makes a new guess and so on until it guesses the right number.

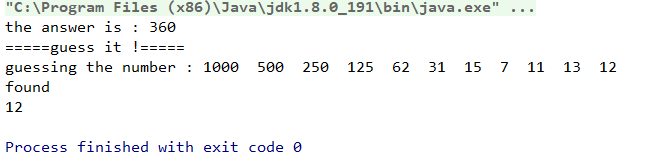
This is a guessing game as using binary search.

*I will using the example to explain this algorithm  
the programme will start with initial = 0, total = 2000  
calculate midden number when you enter this method  
mid\_num = ((total - initial ) / 2 ) + initial  
when answer is 800  
(2000-0)/ 2 = 1000 + 0 = 1000 ==> midden number  
if answer : 800 < midden number :1000  
guess(initial: 0, midden number : 1000, answer: 800)  
it will recursive the initial , midden number and answer  
when the answer is less than midden number  
(1000-0) / 2 = 500 + 0 = 500  
if 800 > 500:  
guess(midden number: 500 , total: 1000, answer: 800)  
it will recursive the midden number , total and answer  
when the answer is greater than midden number*

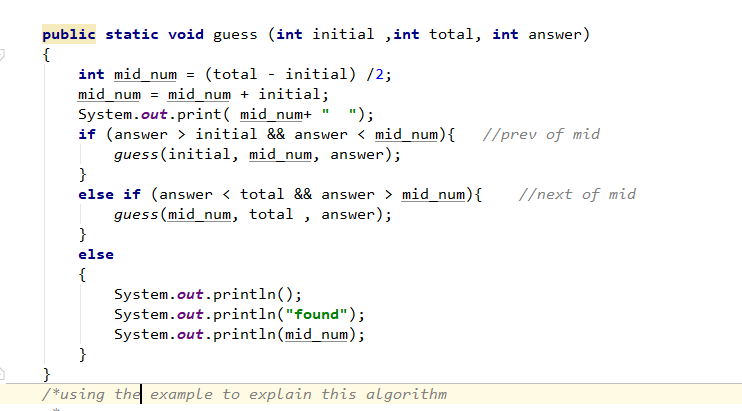
*keep recursive still midden number == answer*

Test





Guess.java



Q10

Implement a function that deletes a node in a binary search tree. Test run your program by deleting a node in the previous sequence and print out the sequence again.

To delete nodes in the binary search tree, we must keep the ordered sequence of nodes.

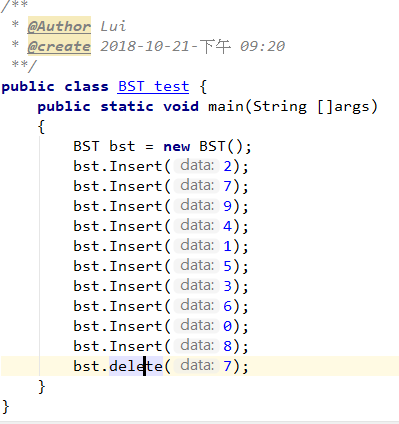
There are two situations in this package.

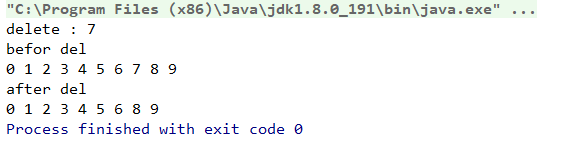
There are no child nodes to delete nodes: just delete nodes from the tree.

The node to be deleted has one child node: if the only node is left subtree, delete the node and find the greater element in the left subtree to replace the node to be deleted. if the only node is right subtree, delete the node and find the least element in the left subtree to replace the node to be deleted.

The node to be deleted has two child node : delete the node and find the least element in the right subtree to replace the node to be deleted.

Test





BST.java in Q10

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-10-21-下午 08:57  
 \*\*/***class** BinaryNode  
{  
 **public int data**;  
 **public** BinaryNode **left**;  
 **public** BinaryNode **right**;  
 **public** BinaryNode(**int** data)  
 {  
 **this**.**data** = data;  
 **left** = **right** = **null**;  
 }  
 **public** BinaryNode()  
 {  
 **return**;  
 }  
}  
  
**public class** BST {  
 BinaryNode **root** ;  
 **public int** position\_Find(**int** data)  
 {  
 BinaryNode result;  
 result = position\_Find(data,**root**);  
 System.***out***.print(**"found it ! "**);  
 **return** result.**data**;  
 }  
 **public** BinaryNode position\_Find(**int** data, BinaryNode bst)  
 {  
 BinaryNode cur = bst;  
  
 **if** (**root** == **null**)  
 {  
 **return null**;  
 }  
  
 **if** (data<cur.**data** )  
 {  
 **return** position\_Find(data, cur.**left**);  
 }  
 **else if** (data>cur.**data**)  
 {  
  
 **return** position\_Find(data, cur.**right**);  
 }  
 **else** {  
  
 **return** cur;  
 }  
 }  
  
 **public void** Insert(**int** data)  
 {  
 **root** = Insert(data,**root**);  
 }  
  
 **public** BinaryNode Insert(**int** data, BinaryNode bst)  
 {  
  
 **if** (bst == **null**)  
 {  
 **root** = **new** BinaryNode(data);  
 **return root**;  
 }  
 **if** (data>bst.**data**)  
 {  
 bst.**right** = Insert(data, bst.**right**);  
 }  
 **if** (data<bst.**data**)  
 {  
 bst.**left** = Insert(data, bst.**left**);  
 }  
 **return** bst;  
 }  
  
 **public void** traversal ()  
 {  
 *//System.out.println("Pre-order traversal");  
 //preorder(root);  
 //System.out.println();* System.***out***.println(**"in-order traversal"**);  
 inorder(**root**);  
 *//System.out.println();  
 //System.out.println("pro-order traversal");  
 //postorder(root);* }  
 **public** BinaryNode preorder (BinaryNode bst)  
 {  
 **if** (bst == **null**)  
 {  
 *//System.out.println("visited");* **return null**;  
 }  
 System.***out***.print(bst.**data** + **" "**);  
 preorder(bst.**left**);  
 preorder(bst.**right**);  
 **return** bst;  
 }  
 **public** BinaryNode inorder (BinaryNode bst)  
 {  
 **if** (bst == **null**)  
 {  
 *//System.out.println("visited");* **return null**;  
 }  
 inorder(bst.**left**);  
 System.***out***.print(bst.**data** + **" "**);  
 inorder(bst.**right**);  
  
 **return** bst;  
 }  
 **public** BinaryNode postorder (BinaryNode bst)  
 {  
 **if** (bst == **null**)  
 {  
 *//System.out.println("visited");* **return null**;  
 }  
 postorder(bst.**left**);  
 postorder(bst.**right**);  
 System.***out***.print(bst.**data** + **" "**);  
 **return** bst;  
 }  
 */\*public void max()  
 {  
 Find\_max(root);  
 }  
 public void Find\_max (BinaryNode bst)  
 {  
 if (bst.right != null)  
 {  
  
 Find\_max(bst.right);  
 }  
 else  
 {  
  
 System.out.println(bst.data);  
 return ;  
 }  
 }\*/* **public void** delete (**int** data)  
 {  
 System.***out***.println(**"delete : "**+ data);  
 System.***out***.println(**"befor del"**);  
 inorder(**root**);  
 delete(**root** , data);  
 System.***out***.println();  
 System.***out***.println(**"after del"**);  
 inorder(**root**);  
 }  
 **public** BinaryNode delete (BinaryNode bst, **int** data)  
 {  
 BinaryNode tmp;  
 **if** (data<bst.**data**)  
 {  
 bst.**left** = delete(bst.**left**, data);  
 **return** bst;  
 }  
 **if** (data>bst.**data**)  
 {  
  
 bst.**right** = delete(bst.**right**,data);  
 **return** bst;  
 }  
 **else** {  
 **if** (bst.**left** == **null** && bst.**right** == **null**) {  
 **return null**;  
 }  
 **else if** (bst.**right** == **null** && bst.**left**!=**null**)  
 {  
 bst = bst.**left**;  
 **return** bst;  
  
 }  
 **else if** (bst.**left** == **null** && bst.**right**!=**null**)  
 {  
 bst = bst.**right**;  
 **return** bst;  
  
 }  
 **else if** (bst.**left** != **null** && bst.**right** != **null**)  
 {  
 tmp = Find\_min(bst.**right**);  
 bst.**data** = tmp.**data**;  
 bst.**right** = delete(bst.**right**, tmp.**data**);  
  
 }  
  
 }**return** bst;  
  
 }  
  
 **public** BinaryNode Find\_min(BinaryNode bst)  
 {  
 **while** (bst.**left**!=**null**)  
 {  
 bst = bst.**left**;  
 }  
 **return** bst;  
 }  
}