|  |  |  |  |
| --- | --- | --- | --- |
| **Coventry University**  **FACULITY OF Engineering and Computing**  **Coursework Cover Sheet**  *Please ensure that you complete all relevant sections legibly*  ***First Copy:*** *Attach top copy to the front of your assignment.*  ***Second Copy:*** *Keep safety as your receipt* | | | |
| **Module Code**  **210CT** | | **Student Card ID Number**  Please print in BLOCK CAPITALS  Surname Lui  Other names Wai Ho  Signature Lui | |
| **Module Title**  **Programming, Algorithms & Data Structures** | | | |
| **Deadline date**  **26 Nov 2018** | **Actual word**  **Count NA** | | **Tutor**  **CYCheng** |
| **Coursework Title/Number**  **Programming Coursework** | | | |

This assessment is all

my own work and has not

been copied in part or in whole

from any other source, except for any

clearly marked up quotations. It

complies with the university regulations

on plagiarism, which I have read

and understood.

|  |  |  |
| --- | --- | --- |
| **FEEDBACK ON MARKED WORK:** *Lecturers will complete this section when work is marked.*  *NB. All marks notified during the year are provisional until confirmed by the end of year Assessment Board* | | |
| STRENGTHS  WEAKNESSES  ADVICE ON HOW WORK COULD BE IMPROVED AND FURTHER COMMENTS | | |
| If you require more feedback, please contact your tutor or see module web.  See assignment sheet for assessment criteria for this assignment. | **MARKED AWARDED** |  |
| Less any late penalty | **-** |
| Adjusted mark if penalty |  |

**Marker’s Signature** …………………………………………………………… **Date** ……………………………………………………….

|  |
| --- |
| Students are reminded that reference must be given for any previously published  work used to gather information to help write assignments, including internet  sources, but these sources should not be copied directly. |

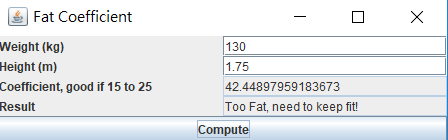
**Second Marker Additional comments Signature** …………………………………………… **Date** …………………………

Q1.

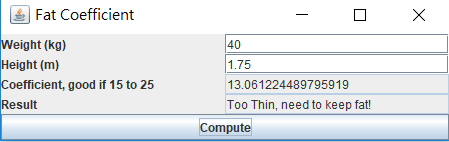
1. Write a java application which accepts the weight (kg) and height (m), then it calculates the Fat Coefficient (FC): ***FC = weight / (height \* height)*** and displays a statement in the result line according to the following rules.

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.

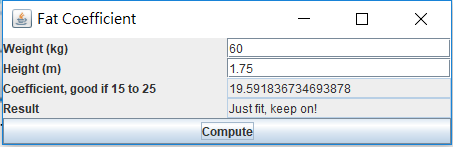
If your FC is greater than 25, the program will remind you of "Too Fat, need to keep fit!"



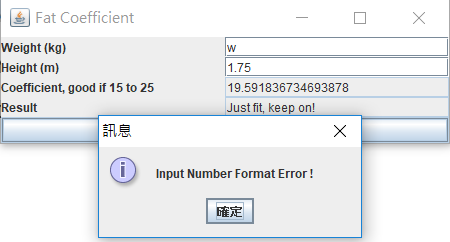
If you are less than 25, it will remind you of "Too Thin, need to keep fat!"

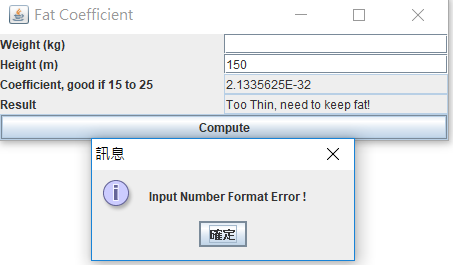


If you are at normal level, it will remind you of "Just fit, keep on!".



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();

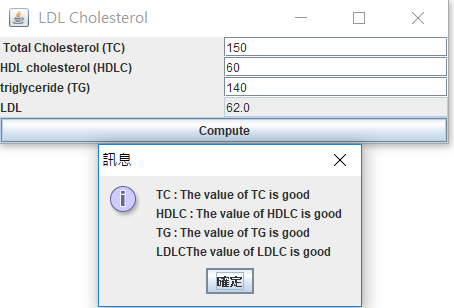
*//display the windows*  
 frame.setVisible(**true**);  
 }  
  
 **public** Fat\_Coefficient\_calculator( ) {  
 *// Panel p1 to hold labels and text fields  
 /\* create a panel, which is similar to HTML's div tag.  
 \* we can create multiple panels and specify locations in JFrame.  
 \* we can add 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 out 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 out 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.

## Based on an abstract class Shape and a concrete class Point provided during lab session, write the following classes:

1. class **Circle** which inherits from **Shape** that has

Data:

**Point center**

**double radius**

Methods: (you need to find out the signatures of the methods)

**Circle()** // initialize name, center & radius

**setRadius()**

**getRadius()**

**setCenter()**

**getCenter()**

**toString()** // return the string representation

and implements the abstract method **getArea()**

1. class **Rectangle** which also inherits from **Shape** that has

Data:

**Point topLeft**

**double length**

**double width**

Methods: (you need to find out the signatures of the methods)

**Rectangle()** // initialize name, topLeft, length &

// width

**setLength()**

**getLength()**

**setWidth()**

**getWidth()**

**setTopLeft()**

**getTopLeft()**

**toString()** // return the string representation

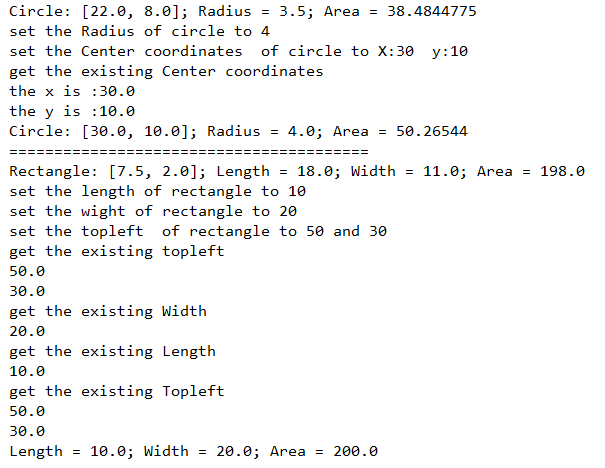
and implements the abstract method **getArea()**

In the program, I will inherit shape, and then constructor and getArea () of Override shape. Both Rectangle and Circle declare radius, length, width. These variables are defined as private. Users can only modify them by get and set method. When users create a new object, they need to input parameters (including graphic coordinates) at the same time.

an object will be new in the program, and the coordinates of the graphics will be stored in the Point object.

Testing program

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-09-20-19:50  
 \*\*/***public class** TestShape {  
 **public static void** main(String[] args) {  
  
 Circle circle;  
 Rectangle rectangle;  
  
 circle = **new** Circle(22,8,3.5);  
 rectangle = **new** Rectangle( 7.5, 2, 18 , 11);  
  
 System.***out***.print(circle.getName() + **": "** + circle.toString());  
 System.***out***.print(**"; Radius = "** + circle.get\_\_radius());  
 System.***out***.println(**"; Area = "** + circle.getArea());  
  
  
 System.***out***.println(**"set the Radius of circle to 4"**);  
 circle.setRadius(4);  
 System.***out***.println(**"set the Center coordinates of circle to X:30 y:10"**);  
 circle.setCenter(30,10);  
 System.***out***.println(**"get the existing Center coordinates"**);  
 circle.getCenter();  
  
 System.***out***.print(circle.getName() + **": "** + circle.toString());  
 System.***out***.print(**"; Radius = "** + circle.get\_\_radius());  
 System.***out***.println(**"; Area = "** + circle.getArea());  
  
  
 System.***out***.println(**"========================================"**);  
 System.***out***.print(rectangle.getName() +  
 **": "** + rectangle.toString());  
 System.***out***.print(**"; Length = "** + rectangle.getLength());  
 System.***out***.print(**"; Width = "** + rectangle.getWidth());  
 System.***out***.println(**"; Area = "** + rectangle.getArea());  
  
  
  
 System.***out***.println(**"set the length of rectangle to 10"**);  
 rectangle.setLength(10); *//new length = 10* System.***out***.println(**"set the wight of rectangle to 20"**);  
 rectangle.setWidth(20); *//new Width = 20* System.***out***.println(**"set the topleft of rectangle to 50 and 30"**);  
 rectangle.setTopleft(50.0, 30.0);  
  
 System.***out***.println(**"get the existing topleft"**);  
 rectangle.getTopleft();  
 System.***out***.println(**"get the existing Width"**);  
 System.***out***.println(rectangle.getWidth());  
 System.***out***.println(**"get the existing Length"**);  
 System.***out***.println(rectangle.getLength());;  
 System.***out***.println(**"get the existing Topleft"**);  
 rectangle.getTopleft();  
  
   
 System.***out***.print(**"Length = "** + rectangle.getLength());  
 System.***out***.print(**"; Width = "** + rectangle.getWidth());  
 System.***out***.println(**"; Area = "** + rectangle.getArea());  
  
  
 }  
}



Shape

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-09-20- 19:35  
 \*\*/  
// Definition of abstract base class Shape***public abstract class** Shape {  
  
 **protected** String **name**;  
  
 **public** Shape(String n) {  
 **name** = **new** String(n);  
 }  
  
 **public abstract double** getArea();  
  
 **public** String getName() {**return name**;}  
  
}

Rectangle.java

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-09-20- 21:39  
 \*\*/***public class** Rectangle **extends** Shape {  
 Point **topleft**;  
 **private double length**;  
 **private double width**;  
 **private** Point **shape**;  
 **public** Rectangle() { **super**(**"Rectangle"**); }  
  
 **public** Rectangle(**double** x, **double** y, **double** Length,**double** Width)  
 {  
 **super**(**"Rectangle"**);  
 **topleft** = **new** Point(x, y);  
 **this**.**length** = Length;  
 **this**.**width** = Width;  
 }  
  
 **public void** setLength(**double** length) { **this**.**length** = length; }  
  
 **public double** getLength() { **return length** ; }  
  
 **public double** getWidth() { **return width**; }  
  
 **public void** setWidth(**double** width) { **this**.**width** = width ; }  
  
 **public void** setTopleft(**double** top, **double** left) { **topleft**.setPoint(top, left) ; }  
  
 **public void** getTopleft()  
 {  
 System.***out***.println(**this**.**topleft**.getX());  
 System.***out***.println(**this**.**topleft**.getY());  
 }  
  
 **public** String toString() { **return topleft**.toString() ; }  
  
 @Override  
 **public double** getArea() { **return this**.**length**\***this**.**width** ; }  
}

Point.java

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-09-20- 19:36  
 \*\*/  
// Definition of class Point***public class** Point {  
 **private double x**, **y**; *// coordinates of the Point  
  
 // constructor* **public** Point( **double** a, **double** b ) { setPoint( a, b ); }  
  
 *// Set x and y coordinates of Point* **public void** setPoint( **double** a, **double** b ) {  
 **x** = a;  
 **y** = b;  
 }  
  
 *// get x coordinate* **public double** getX() { **return x**; }  
  
 *// get y coordinate* **public double** getY() { **return y**; }  
  
 *// convert the point into a String representation* **public** String toString()  
 { **return "["** + **x** + **", "** + **y** + **"]"**; }  
}

Circle.java

**import** javax.xml.transform.Result;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-09-20- 19:36  
 \*\*/***public class** Circle **extends** Shape{  
 **private double radius**;  
 List<String> **x\_y** = **new** ArrayList<>();  
 **private** Point **\_\_radius**;  
  
 **public** Circle(**double** x, **double** y, **double** rad)  
 {  
 *//declare the named Circle* **super**(**"Circle"**);  
 **this**.**\_\_radius** = **new** Point(x ,y);  
 **this**.**radius** = rad;  
 }  
  
 **public** String toString() { **return this**.**\_\_radius**.toString() ; }  
  
 **public void** setRadius(**double** rad ) { **this**.**radius** = rad ; }  
  
 **public double** get\_\_radius() { **return this**.**radius** ; }  
  
 **public void** setCenter(**double** x, **double** y ) { **\_\_radius**.setPoint(x,y) ; }  
  
 **public void** getCenter()  
 {  
 System.***out***.println(**"the x is :"**+ **\_\_radius**.getX());  
 System.***out***.println(**"the y is :"**+ **\_\_radius**.getY());  
 }  
  
 @Override  
 **public double** getArea() { **return** 3.14159\*(**this**.**radius** \***this**.**radius**) ; }  
}

Q4

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

**addToHead(Object item)**

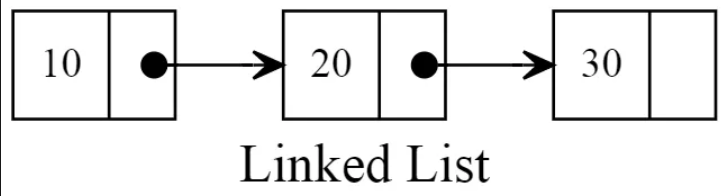
**addToTail(Object item)**

**removeFromHead()**

**removeFromTail()**

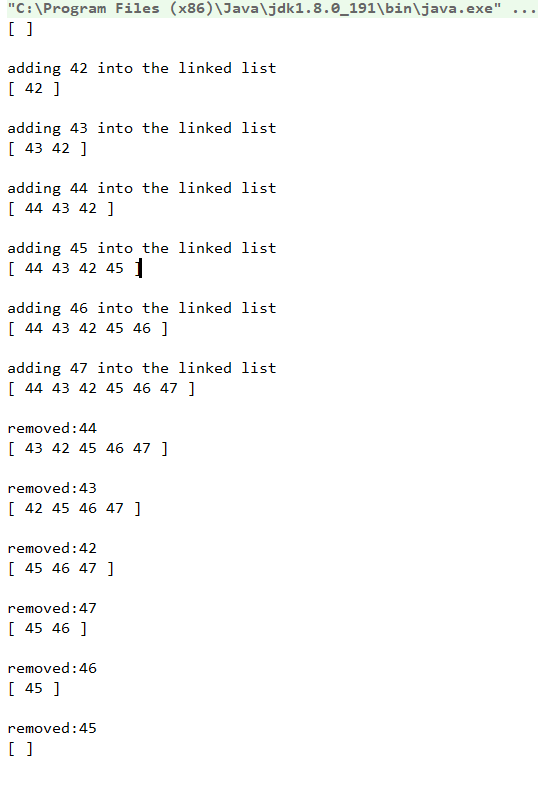
I implemented the addToHead, addToTail, removeFromHead and removeFromTail functions of linked list in this program without using

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*

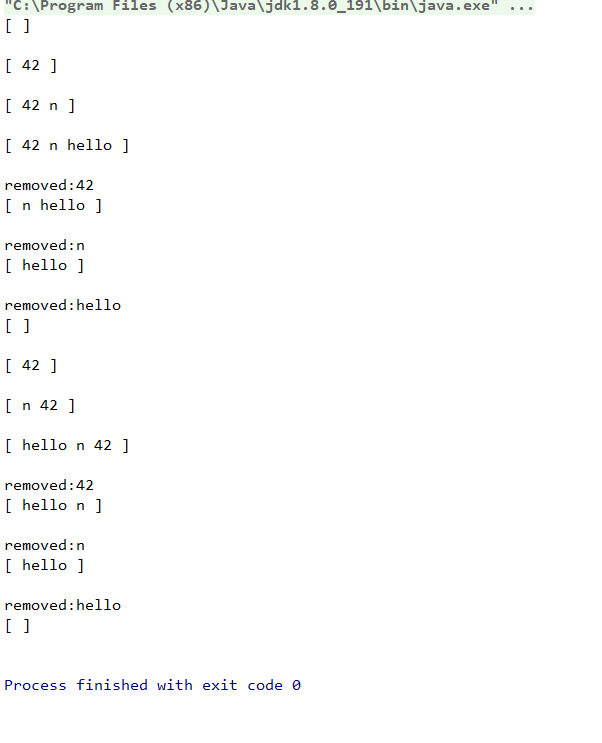
Q5

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*

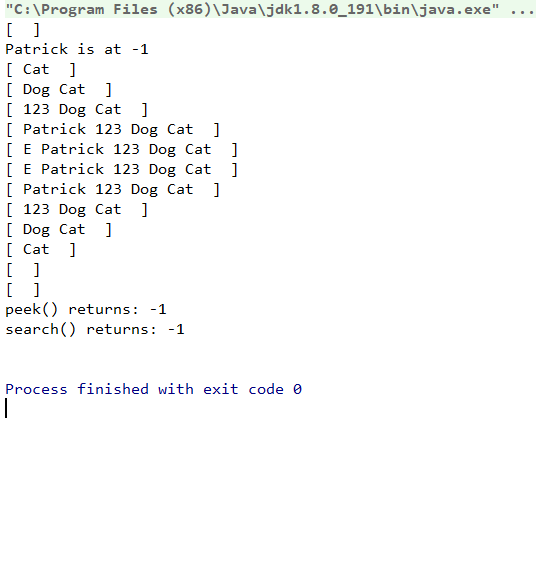
Q6

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 out error if existing length of list is greater 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

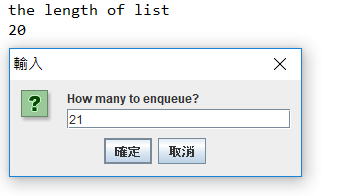
Based on the LinkedList class developed in previous week, write your own Queue class by composing a LinkedList object. The Queue class should provide methods such as constructor, Queue(), empty(), enqueue() and dequeue(). Design and write a test program to test your Queue class.

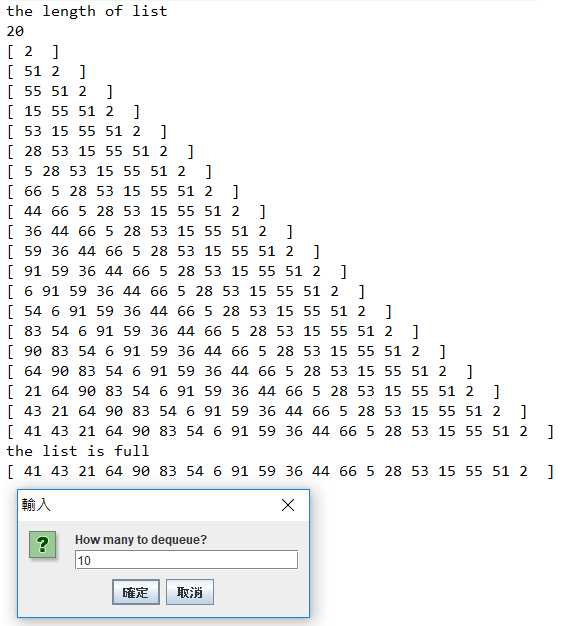
Using inherited linked list to implement Queue. Queue is a linear data structure which follows a specific order in which operations are performed. The order may be FIFO (First In First Out) or LILO (Last In Last Out). So, in the enqueue method of queue, it points to the tail of the list and insert the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

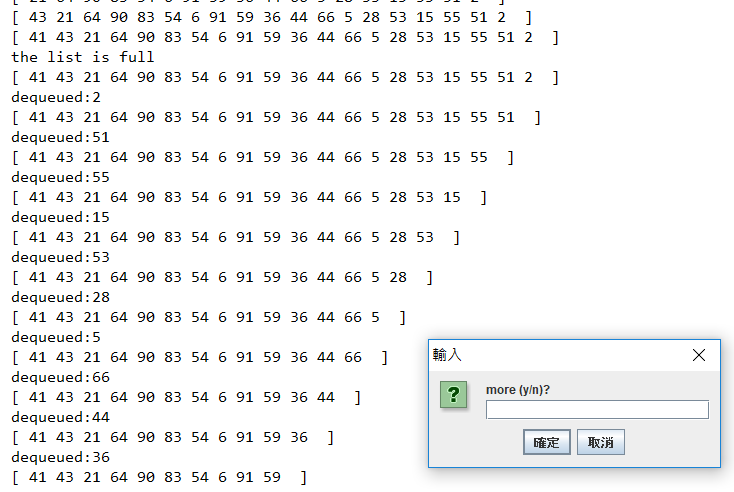
This program is the principle that allows the user to understand the queue. The length of the Queue has been defined as 20, the user can input the number of enqueues, the program will randomly generate the number and then queue. Also, the user can then input the number of elements to remove. it will 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;  
 }  
  
 *//return ture if the head is null* **public boolean** empty()  
 {  
 **if** (**head** == **null**)  
 {  
 **return true** ;  
 }  
 **else** {  
 **return false**;  
 }  
 }  
   
 *//return, if existing lengrh of arrat is greater than max length   
 // else add the new element at the head* **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

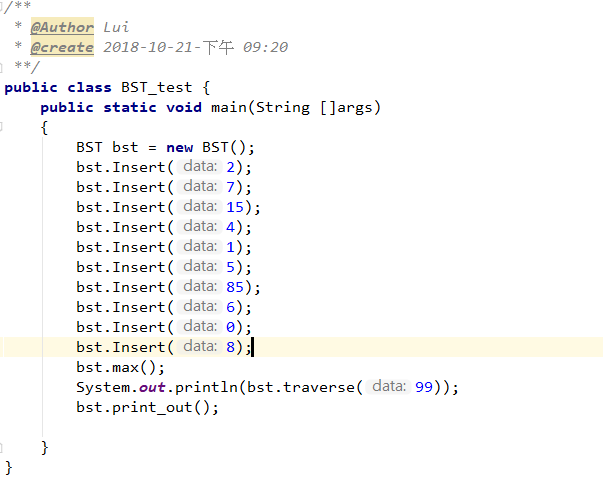
Based on the java code provided in lecture as a starting point, implement a Binary search tree class with all the necessary methods to insert, traverse and print out sequence of number. Design and test run your program by inserting the sequence [2 7 9 4 1 5 3 6 0 8] and printout the sequence again in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

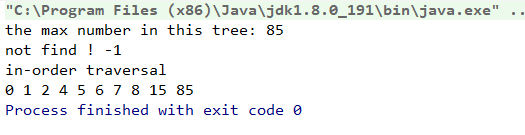
In this programme, I implement the binary search tree, there are 3 filed in each node, right, left and data. The tree is constructed by parent node, left child node and right child node. The left child node is less than parent node, the right node is greater than parent node. Each node is sorted according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

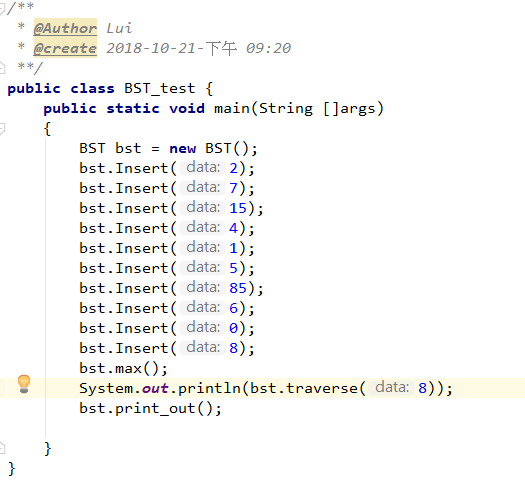
The theoretical traversal order is: the left subtree, the root, and the right subtree root are in the middle.

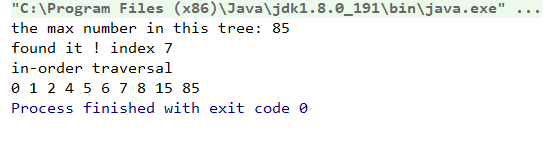
In fact, using Depth-first Search, it only changes the output order of the nodes.

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 the left node or right of current node is null it will return;* **if** (**root** == **null**)  
 {  
 **return null**;  
 }  
  
 *// if the data is small than current node, go to the left child tree* **if** (data<cur.**data** )  
 {  
 **if** (cur.**left** != **null**)  
 {  
  
 **return** traverse(data, cur.**left**);  
  
 }  
 **else** {  
 **return null**;  
 }  
 }  
*// if the data is small than current node, go to the right child tree* **else if** (data>cur.**data**)  
 {  
 **if** (cur.**right** != **null**)  
 {  
 **return** traverse(data, cur.**right**);  
 }  
 **else** {  
 **return null**;  
 }  
 }  
 *//The search succeeds, returning the address of the found node of the node* **else** {  
  
 **return** cur;  
 }  
 }  
  
 **public void** Insert(**int** data)  
 {  
 **root** = Insert(data,**root**);  
 }  
  
 **public** BinaryNode Insert(**int** data, BinaryNode bst)  
 {  
*// if the tree is null, Generate and return a binary search tree for a node* **if** (bst == **null**)  
 {  
 **root** = **new** BinaryNode(data);  
 **return root**;  
 }  
 *//Start looking for the location where you want to insert the element* **if** (data>bst.**data**)  
 {*//Recursively inserting the right subtree* bst.**right** = Insert(data, bst.**right**);  
 }  
 **if** (data<bst.**data**)  
 {  
 *//Recursively inserting the left subtree* bst.**left** = Insert(data, bst.**left**);  
 }  
 *//else x already exists, do nothing* **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)  
 { *//The maximum data usually at the right node, so keep looking to the right node.* **if** (bst.**right** != **null**)  
 {  
  
 Find\_max(bst.**right**);  
 }  
 **else** {  
 System.***out***.println(**"the max number in this tree: "** + bst.**data**);  
 **return** ;  
 }  
 }  
}

Q9

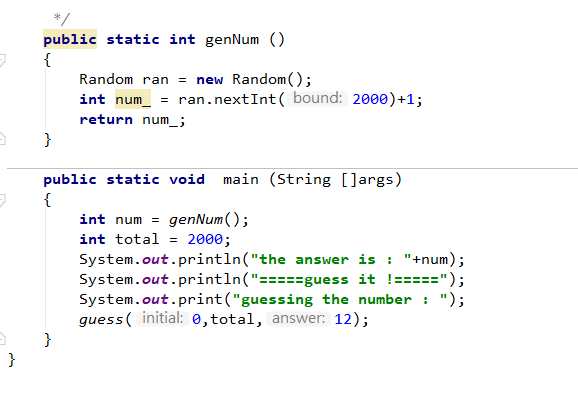
Use binary search in implementing a guessing game. One thinks of a number between 1 and 2000, the program attempts to guess the number and feedback is given whether my number is higher or lower. The program then makes a new guess and so on until it guesses the right number.

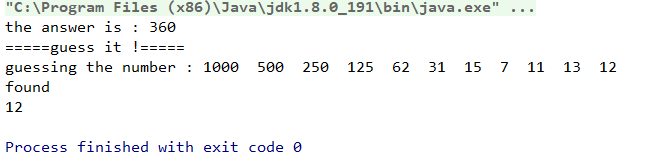
This is a guessing game as using binary search.

*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*

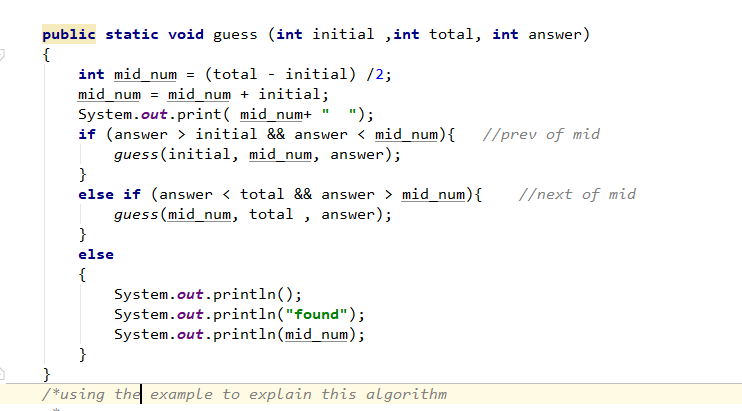
*keep recursive still midden number == answer*

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.

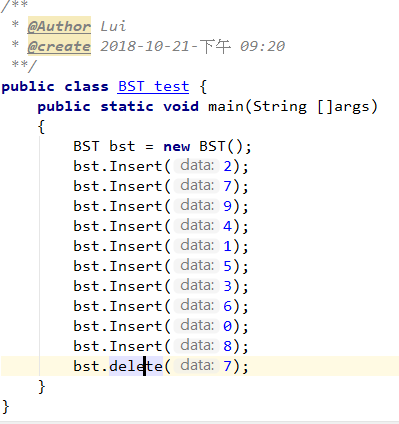
There are two situations in this package.

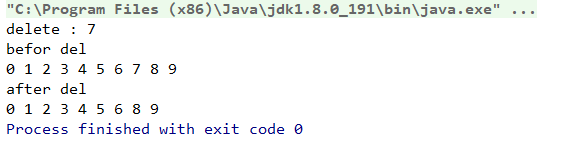
There are no child nodes to delete nodes: just delete nodes from the tree.

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** )  
 {  
 *// if the data is small than current node, go to the left child tree* **return** position\_Find(data, cur.**left**);  
 }  
 **else if** (data>cur.**data**)  
 {  
 *// if the data is small than current node, go to the right child tree* **return** position\_Find(data, cur.**right**);  
 }  
 **else** {  
 *//The search succeeds, returning the address of the found node of the node* **return** cur;  
 }  
 }  
  
 **public void** Insert(**int** data)  
 {  
 **root** = Insert(data,**root**);  
 }  
  
 **public** BinaryNode Insert(**int** data, BinaryNode bst)  
 {  
  
 **if** (bst == **null**)  
 {  
 *// if the tree is null, Generate and return a binary search tree for a node* **root** = **new** BinaryNode(data);  
 **return root**;  
 }  
 *//Start looking for the location where you want to insert the element* **if** (data>bst.**data**)  
 {  
 *//Recursively inserting the right subtree* bst.**right** = Insert(data, bst.**right**);  
 }  
 **if** (data<bst.**data**)  
 {  
 *//Recursively inserting the left subtree* bst.**left** = Insert(data, bst.**left**);  
 }  
 *//else x already exists, do nothing* **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)  
 {  
 *//Start from the left of the tree, to the middle end,  
 // and finally to the right.* **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);*//Left subtree recursively deleted* **return** bst;  
 }  
 **if** (data>bst.**data**)  
 {  
  
 bst.**right** = delete(bst.**right**,data);*//right subtree recursively deleted* **return** bst;  
 }  
 **else** {  
 *//Successfully found the node to delete  
  
  
 //A deleted node has only one node or no node* **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;  
  
 }  
 *//The nodes that are deleted have nodes on the left and right* **else if** (bst.**left** != **null** && bst.**right** != **null**)  
 {  
 *//Find the smallest element in the right subtree to fill the deleted node* tmp = Find\_min(bst.**right**);  
 bst.**data** = tmp.**data**;  
 *//Delete the smallest element in the right subtree of the deleted node* bst.**right** = delete(bst.**right**, tmp.**data**);  
  
 }  
  
 }**return** bst;  
  
 }  
  
 **public** BinaryNode Find\_min(BinaryNode bst)  
 {  
 *//The minimum data usually at the left node, so keep looking to the left node.* **while** (bst.**left**!=**null**)  
 {  
 bst = bst.**left**;  
 }  
 **return** bst;  
 }  
}

Q11

Manually arrange the sequence [30 13 22 90] in ascending order using merge sort, bubble sort and selection sort, showing at each step the new configuration of the sequence. How many comparisons and how many element moves were used by each method? Which is the best performing method for sorting this array of integers? Which would be the worst arrangement of this sequence?

This program implements merge sort, bubble sort and selection sort algorithm to sort the 30,13,22,90

Bubble sort:

Sort by comparing the size of two adjacent numbers, if they are in the wrong order, they will exchange adjacent elements repeatedly

Selection sort:

I use two arrays in my program, one is sorted array and the other is unsorted array. The number of iterations is determined by the length of the unordered array. Each iteration selects the smallest element from the unordered array and moves it to the sorted array.

merge sort:

Merge sort is to separate and merge the input arrays.

For example, if you enter 1, 2, 3, 4, I will divide the array into left and right arrays until the left and right arrays are divided into only one element. Return the remaining element because of the return and then merge the returned left and right arrays in size. In a recursive way, the input arrays can be output as sorted arrays.

In each sorting program, I record the input arrays with moving\_time and comparison\_time , how many comparisons and how many element moves they need . For example, record the positions of exchange array elements in bubbles as moving\_time and compare the sizes of adjacent elements as comparison\_time.

The testing program:

**import** com.sun.scenario.effect.Merge;  
  
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-11-25-下午 04:07  
 \*\*/***public class** Easy\_Q11 {  
  
  
  
 **public static void** main(String []args)  
 {  
 **int** [] arr = {30,13,22,90 };  
 System.***out***.println(**"Bubble\_sort"**);  
 bubble bubble = **new** bubble(arr);  
 **int** bubble\_Comparison\_time = bubble.getComparison\_time();  
 **int** bubble\_getMoving\_time\_time = bubble.getMoving\_time();  
  
 System.***out***.println(**"=================================================="**);  
 System.***out***.println(**"Merge\_sort"**);  
 Merges Merges = **new** Merges(arr);  
 **int** merge\_Moving\_time = Merges.getMerge\_count();  
 **int** merge\_Comparison\_time = Merges.getComparison\_count();  
 System.***out***.println(**"=================================================="**);  
 System.***out***.println(**"Selection\_sort"**);  
 Selection Selection = **new** Selection(arr);  
 **int** Selection\_Comparison\_time = Selection.getComparison\_count();  
 **int** Selection\_Moving\_time = Selection.getCount();  
  
 System.***out***.println(**"=================================================="**);  
 System.***out***.println(**"The comparison time"**);  
 System.***out***.println(**"bubble sort "** + bubble\_Comparison\_time);  
 System.***out***.println(**"Selection sort "** + Selection\_Comparison\_time);  
 System.***out***.println(**"merge sort "** + merge\_Comparison\_time);  
 System.***out***.println();;  
 System.***out***.println(**"The Moving\_time"**);  
 System.***out***.println(**"bubble sort "** + bubble\_getMoving\_time\_time);  
 System.***out***.println(**"Selection sort "** + Selection\_Moving\_time);  
 System.***out***.println(**"merge sort "** + merge\_Moving\_time);  
  
 System.***out***.println();  
 System.***out***.println(**"Which is the best performing method for sorting this array of integers? Which would be the worst arrangement of this sequence?"**);  
 **for** (**int** i : arr)  
 System.***out***.print(i+**"\t"**);  
  
 System.***out***.println();  
  
 System.***out***.println();  
 **int** total\_bubble = bubble\_Comparison\_time+bubble\_getMoving\_time\_time;  
 **int** total\_selection = Selection\_Moving\_time + Selection\_Comparison\_time;  
 **int** total\_merge = merge\_Comparison\_time+ merge\_Moving\_time;  
 **int**[] good\_worst\_time = **new int**[3];  
 good\_worst\_time[0] = total\_bubble;  
 good\_worst\_time[1] = total\_selection;  
 good\_worst\_time[2] = total\_merge;  
 **int** i = *getMax*(good\_worst\_time);  
 **if** (i == total\_bubble)  
 System.***out***.println(**"bubble\_sort is the best"**);  
 **else if** (i == total\_selection)  
 System.***out***.println(**"selection\_sort is the best"**);  
 **else** System.***out***.println(**"merge is the best"**);  
  
 **int** j = *getMin*(good\_worst\_time);  
  
 **if** (j == total\_bubble)  
 System.***out***.println(**"bubble\_sort is the worst"**);  
 **else if** (j == total\_selection)  
 System.***out***.println(**"selection\_sort is the worst"**);  
 **else** System.***out***.println(**"merge is the worst"**);  
  
  
  
  
  
 }  
 **public static int** getMin(**int**[] inputArray){  
 **int** minValue = inputArray[0];  
 **for**(**int** i=1;i<inputArray.**length**;i++){  
 **if**(inputArray[i] < minValue){  
 minValue = inputArray[i];  
 }  
 }  
 **return** minValue;  
 }  
 **public static int** getMax(**int**[] inputArray){  
 **int** maxValue = inputArray[0];  
 **for**(**int** i=1;i < inputArray.**length**;i++){  
 **if**(inputArray[i] > maxValue){  
 maxValue = inputArray[i];  
 }  
 }  
 **return** maxValue;  
 }  
}

The Test output:

Bubble\_sort

comparison time of this round 2

element moving time of this round 1

13 30 22 90

comparison time of this round 6

element moving time of this round 2

13 22 30 90

==================================================

Merge\_sort

element comparison time of this round : 1

element moving time of this round: 7

13 0

element comparison time of this round : 2

element moving time of this round: 8

13 22

element comparison time of this round : 3

element moving time of this round: 11

30 0

element comparison time of this round : 4

element moving time of this round: 12

30 90

element comparison time of this round : 5

element moving time of this round: 13

13 0 0 0

element comparison time of this round : 6

element moving time of this round: 14

13 22 0 0

element comparison time of this round : 7

element moving time of this round: 15

13 22 30 0

element comparison time of this round : 8

element moving time of this round: 16

13 22 30 90

totally spend 16round

totally spend 8round

==================================================

Selection\_sort

element comparison time of this round 4

element moving time of this round 1

unsort list

22 30 90

sorted list

13

======================================

element comparison time of this round 7

element moving time of this round 2

unsort list

30 90

sorted list

13 22

======================================

element comparison time of this round 9

element moving time of this round 3

unsort list

90

sorted list

13 22 30

======================================

element comparison time of this round 10

element moving time of this round 4

unsort list

sorted list

13 22 30 90

======================================

==================================================

The comparison time

bubble sort 9

Selection sort 10

merge sort 8

The Moving\_time

bubble sort 2

Selection sort 4

merge sort 16

Which is the best performing method for sorting this array of integers? Which would be the worst arrangement of this sequence?

13 22 30 90

merge is the best

bubble\_sort is the worst

Process finished with exit code 0

The bubble sort program

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-11-25-下午 05:16  
 \*\*/***public class** bubble {  
  
 **int moving\_time** = 0;  
 **int comparison\_time** = 0;  
 **public** bubble(**int**[] arr)  
 {  
 bubble(arr);  
 }  
 **public void** bubble(**int**[] arr)  
 {  
 **for** (**int** i = 0; i < arr.**length**-1; i++){  
  
 **for**(**int** j = i ; j < arr.**length**; j++ ){  
  
 **comparison\_time**+= 1;  
 *// repeatedly swapping the adjacent elements if they are in wrong order.* **if** (arr[i] > arr[j]){  
 **int** tmp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = tmp;  
 **moving\_time**+=1;  
 System.***out***.println(**"comparison time of this round "**+**comparison\_time**);  
 System.***out***.println(**"element moving time of this round "**+**moving\_time**);  
 **for** (**int** k = 0; k < arr.**length**; k++){  
 System.***out***.print(arr[k]);  
 System.***out***.print(**"\t"**);  
 }  
 System.***out***.println();  
  
 }  
 }  
 }  
 }  
  
 **public int** getMoving\_time()  
 {  
 **return moving\_time**;  
 }  
 **public int** getComparison\_time()  
 {  
 **return comparison\_time**;  
 }  
}

The Selection sort program:

**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-11-25-下午 04:46  
 \*\*/***public class** Selection {  
  
 **int count** = 0;  
 **int comparison\_count** = 0;  
 **public** Selection(**int**[] arr)  
 {  
 List<Integer> sort = **new** ArrayList<Integer>();  
 List<Integer> sorted = **new** ArrayList<Integer>();  
  
 **for** (**int** i = 0; i<arr.**length**;i++)  
 {  
 sort.add(arr[i]);  
 }  
 sort(sort,sorted);  
 }  
 **public void** sort(List<Integer> sort, List<Integer> sorted)  
 {  
 **int** index ;  
 **while**(sort.size()!=0) {  
 **int** min = sort.get(0);  
 index = 0;  
 *// find the minimum element in the array and add to sorted list* **for** (**int** i = 0; i < sort.size(); i++) {  
 **int** cur = sort.get(i);  
 **comparison\_count** += 1;  
 **if** (cur < min) {  
 min = cur;  
 index = i;  
 }  
 }  
 sorted.add(min);  
 sort.remove(index);  
 **count** += 1;  
  
 System.***out***.println(**"element comparison time of this round "**+**comparison\_count**);  
 System.***out***.println(**"element moving time of this round "**+**count**);  
 System.***out***.println(**"unsort list"**);  
 **for** (**int** i = 0; i < sort.size(); i++) {  
 System.***out***.print(sort.get(i));  
 System.***out***.print(**'\t'**);  
 }  
 System.***out***.println();  
 System.***out***.println(**"sorted list"**);  
 **for** (**int** i = 0; i < sorted.size(); i++) {  
 System.***out***.print(sorted.get(i));  
 System.***out***.print(**'\t'**);  
 }  
 System.***out***.println();  
 System.***out***.println(**"======================================"**);  
 }  
 }  
 **public int** getCount()  
 {  
 **return count**;  
 }  
 **public int** getComparison\_count()  
 {  
 **return comparison\_count**;  
 }  
}

The Merge sort program:

*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-11-25-下午 04:35  
 \*\*/***public class** Merges {  
 **int merge\_count** = 0;  
  
 **int comparison\_count** = 0;  
 **public** Merges(**int**[] arr) {  
 Megre(arr);  
 }  
  
 **public int**[] Megre(**int**[] array)  
 {  
 *// return , stop iteration if the length of array only one* **if** (array.**length** == 1)  
 {  
 **return** array;  
 }  
 *// find the midpoint of array* **int** midpoint = array.**length**/2;  
 **int**[] left = **new int**[midpoint];  
 **int** [] right;  
 *// If the result of dividing 2 is an integer,  
 // then the array can be assigned the same number.* **if** (array.**length** %2 == 0)  
 {  
 right = **new int** [midpoint];  
  
 }  
 *//If the result of dividing 2 is a decimal, then the distribution cannot be evenly distributed.  
 // The right array has one more than the left array.* **else** {  
 right = **new int** [midpoint+1];  
 }  
 *//Split the array into left and right* **for** (**int** i = 0; i < midpoint ; i++)  
 {  
 **merge\_count** += 1;  
 left[i] = array[i];  
  
 }  
 **for** (**int** i = 0; i < right.**length** ; i++)  
 {  
 **merge\_count** += 1;  
 right[i] = array[midpoint+i];  
  
 }  
 **int**[] result = **new int** [array.**length**];  
 *//Recursive method,  
 // the elements in the array are separated ...* left = Megre(left);  
 right = Megre(right);  
 *//Compare two arrays for merging and sorting* result = Megre(left, right);  
  
 **return** result;  
 }  
 **public int**[] Megre(**int**[] left, **int**[]right)  
 {  
 **int**[] result = **new int**[left.**length**+right.**length**];  
 **int** leftPoint = 0;  
 **int** rightPoint = 0;  
 **int** resultPoint = 0;  
 **while** (leftPoint < left.**length** || rightPoint < right.**length**)  
 {  
 **comparison\_count** += 1;  
 **if** (leftPoint < left.**length** && rightPoint < right.**length**)  
 {  
 **if** (left[leftPoint] > right[rightPoint])  
 {  
 result[resultPoint] = right[rightPoint];  
 resultPoint++;  
 rightPoint++;  
 }  
 **else** {  
 result[resultPoint] = left[leftPoint];  
 resultPoint++;  
 leftPoint++;  
 }  
 }  
 **else if** (leftPoint < left.**length**)  
 {  
 result[resultPoint] = left[leftPoint];  
 resultPoint++;  
 leftPoint++;  
  
 }**else if** (rightPoint < right.**length**)  
 {  
 result[resultPoint] = right[rightPoint];  
 resultPoint++;  
 rightPoint++;  
  
 }  
 **merge\_count** += 1;  
 System.***out***.println(**"element comparison time of this round : "**+**comparison\_count**);  
 System.***out***.println(**"element moving time of this round: "**+**merge\_count**);  
 **for** (**int** i : result)  
 {  
 System.***out***.print(i);  
 System.***out***.print(**"\t"**);  
 }  
 System.***out***.println();  
 }  
  
 **return** result;  
 }  
 **public int** getMerge\_count(){  
 System.***out***.println(**"totally spend "**+ **merge\_count** +**"round"**);  
 **return merge\_count**;  
 }  
 **public int** getComparison\_count(){  
 System.***out***.println(**"totally spend "**+ **comparison\_count** +**"round"**);  
 **return comparison\_count**;  
 }  
  
}

Q12

Implement a java program to sort the same sequence above with quick sort method

In this program, by implementing Balance Quick Sort as sorting algorithm, I select the middle element from the sequence as pivot, which is larger than pivot and smaller than pivot. The left and right sides of pivot are regarded as two arrays, and repeat the above actions until one or zero elements remain in the sequence, then the smallest element in the current array will be found, then one by one to complete the sorting

Quick sort

**import** com.sun.corba.se.impl.oa.toa.TOA;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-11-14-上午 09:22  
 \*\*/***public class** quick\_sort {  
  
 **public static** List<Integer> quick\_sort(List<Integer> array) {  
 **if** (array.size() < 2)  
 **return** array;  
 *// use the mid of the array to pivot* **int** pivot = array.get(array.size()/2);  
 array.remove(array.size()/2);  
 List<Integer> less = **new** ArrayList<Integer>();  
 List<Integer> great = **new** ArrayList<Integer>();  
 List<Integer> total = **new** ArrayList<Integer>();  
  
 *//Based on pivot to allocate array elements, if the number is less than pivot, it is placed in less array  
 // , and if it is larger than pivot, it is placed in greater array.* **for** (**int** i = 0 ; i < array.size(); i++)  
 {  
 **if** (array.get(i)<pivot)  
 {  
 less.add(array.get(i));  
 }  
 **else** {  
 great.add(array.get(i));  
 }  
 }  
 *//Repeat the above actions recursively until there is only one less array left* List<Integer>less\_result = *quick\_sort*(less);  
  
 *//By recursion, we find the smallest value and then merge it with pivot.* **for** (**int** i : less\_result)  
 {  
 total.add(i);  
 }  
 total.add(pivot);  
 *//Use the greater array to recurse again, repeat the above actions, spell and merge* List<Integer> great\_result = *quick\_sort*(great);  
 **for** (**int** i : great\_result)  
 {  
 total.add(i);  
 }  
 **return** total;  
 }  
 **public static void** main(String []args)  
 {  
 List<Integer> list = **new** ArrayList<Integer>();  
 **int**[] array = {30,13,22,90,50};  
 **for** (**int** i = 0 ; i < array.**length**; i++)  
 list.add(array[i]);  
 List<Integer> result = *quick\_sort*(list);  
  
 **for** (**int** i : result)  
 {  
 System.***out***.println(i);  
 }  
 *//quick\_sort(array);* }  
}

Q13

Implement an unweighted, undirected graph structure in java language. Implement the BFS or DFS traversal for this graph and test run your program with your own data

I use adjacency matrices to demonstrate undirected graphs, which are undirected graphs represented by adjacency matrices.

Take a example :

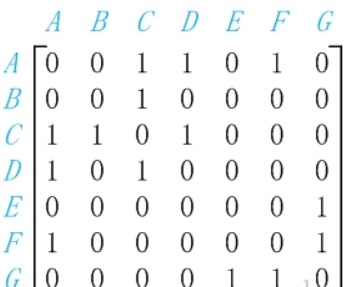


Figure G1 above contains seven vertices of "A, B, C, D, E, F, G" and seven edges of "(A, C), (A, D), (A, F), (B, C), (C, D), (E, G), (F, G)". Since this is an undirected graph, edges (A, C) and edges (C, A) are the same edges; when enumerating edges here, they are listed in alphabetical order.

The matrix on the right side of the figure above is a schematic diagram of the adjacency matrix of G1 in memory. A[i][j]=1 denotes that the first vertex and the second vertex are adjacent points, A[i][j]=0 denotes that they are not adjacent points, while A[i][j] denotes the value of the j column in row i; for example, A[1,2]=1 denotes that the first vertex (vertex B) and the second vertex (C) are adjacent points.

In the program, I create a matrix with known data.

Matrix UDG in the program is the structure corresponding to the adjacency matrix. MVexs is used to save vertices, and mMatrix is a two-dimensional array for storing matrix information. For example, mMatrix [i] [j]= 1 means that vertex i (i.e. mVexs [i]) and vertex J (i.e. mVexs [j]) are adjacent points, and mMatrix [i] [j]= 0 means that they are not adjacent points.

Program

**import** java.util.\*;  
  
*/\*\*  
 \** ***@Author*** *Lui  
 \** ***@create*** *2018-11-20-下午 09:53  
 \*\*/***public class** adjanency {  
  
 **public static class** MatrixUDG {  
 **private char**[] **mVexs**;  
 **private int**[][] **mMatrix**;  
  
 */\*  
 \* vexs = Vertex array  
 \* edges = edges array  
 \*  
 \* \*/* **public int**[][] MatrixUDG(**char**[] vexs, **char**[][] edges) {  
 *//initial "length of Vertex and Edges* **int** vlen = vexs.**length**;  
 **int** elen = edges.**length**;  
  
  
 *//Initialize the vertex to the length of the input vexs  
 // Copy one more to mVexs* **mVexs** = **new char**[vlen];  
 **for** (**int** i = 0; i < **mVexs**.**length**; i++)  
 **mVexs**[i] = vexs[i];  
 **for** (**char** i : **mVexs**) {  
 *//System.out.println(i);* }  
 *//Initialize edge* **mMatrix** = **new int**[vlen][vlen];  
 **for** (**int** i = 0; i < elen; i++) {  
 *// read the starting vex and ending ves of the edge* **int** p1 = getPosition(edges[i][0]);  
  
 *//System.out.println(p1);* **int** p2 = getPosition(edges[i][1]);  
 *//System.out.println(p2);* **mMatrix**[p1][p2] = 1;  
 **mMatrix**[p2][p1] = 1;  
 }  
 **return mMatrix**;  
 }  
  
 **private int** getPosition(**char** ch) {  
 **for** (**int** i = 0; i < **mVexs**.**length**; i++) {  
 *//System.out.println(mVexs[i]);* **if** (**mVexs**[i] == ch)  
 **return** i;  
 }  
 **return** -1;  
 }  
  
 **public void** print() {  
 System.***out***.printf(**"Martix Graph:\n"**);  
 **for** (**int** i = 0; i < **mVexs**.**length**; i++) {  
 **for** (**int** j = 0; j < **mVexs**.**length**; j++)  
 System.***out***.printf(**"%d "**, **mMatrix**[i][j]);  
 System.***out***.printf(**"\n"**);  
 }  
 }  
  
 **public static boolean** contains(**int**[] visited, **int** item) {  
 **if** (visited == **null**) {  
 System.***out***.println(**"null"**);  
 **return false**;  
 }  
 *//System.out.println(visited);* **for** (**int** n : visited) {  
 *//System.out.println(n);* **if** (item == n) {  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 **public void** DFS()  
 {  
  
 Stack <Integer> stack=**new** Stack<Integer>();  
 **int** []visited=**new int**[**mVexs**.**length**];  
  
 **int** unvisited=getUnvisited(visited);  
  
 **while**(unvisited>=0)  
 {  
 *//start with starting node and push to stack* visited[unvisited]=1;  
 stack.push(unvisited);  
  
 System.***out***.print(**mVexs**[unvisited]+**" "**);  
 **while**(!stack.isEmpty())  
 {  
 *//find the top of stack but don’t pop* **int** index=stack.peek();  
  
 *//find which mVexs not be visited by iteration* **boolean** found=**false**;  
 **for**(**int** i=0;i<**mVexs**.**length**;i++)  
 {  
 **if**(index!=i&&visited[i]==0&&**mMatrix**[index][i]>0)  
 {  
 visited[i]=1;  
 stack.push(i);  
 *//output the iteration element* System.***out***.print(**mVexs**[i]+**" "**);  
 found=**true**;  
 **break**;  
 }  
 }  
 **if**(!found)  
 {  
 *//pop the element at the stack* stack.pop();  
 }  
 }  
 *// find the node which was not be visited* unvisited=getUnvisited(visited);  
 }  
  
 }  
 **private int** getUnvisited(**int**[] visited) {  
 **int** index=-1;  
 **for**(**int** i=0;i<visited.**length**;i++)  
 {  
 *// the node of i not be visited* **if**(visited[i]==0)  
 {  
 index=i;  
 **break**;  
 }  
 }  
  
 **return** index;  
 }  
 }  
  
 **public static void** main(String[] args) {  
 **char**[] vexs = {**'A'**, **'B'**, **'C'**, **'D'**, **'E'**, **'F'**, **'G'**};  
 **char**[][] edges = **new char**[][]{  
 {**'A'**, **'C'**},  
 {**'A'**, **'D'**},  
 {**'A'**, **'F'**},  
 {**'B'**, **'C'**},  
 {**'C'**, **'D'**},  
 {**'E'**, **'G'**},  
 {**'F'**, **'G'**}};  
  
 MatrixUDG pG;  
  
 pG = **new** MatrixUDG();  
 **int**[][] Matrix = pG.MatrixUDG(vexs, edges);  
 **int**[] array = **null**;  
 pG.DFS();  
 System.***out***.println();  
 pG.print(); *// print out the matrix* }  
}