1. 设计一个教师类Teacher（属于cn.net.sdkd包），要求：
2. 属性有编号（int no）、姓名（String name）、年龄（int age）、所属学院（String seminary），为这些属性设置相应的get和set方法。
3. 为Teacher类重写equals方法，要求：当两个教师对象的no相同时返回true。
4. 重写Teacher类的toString方法，通过该方法可以返回“编号为\*\*、姓名为\*\*、年龄为\*\*的\*\*学院老师”形式的字符串。
5. 由多个Teacher对象所形成的数组可以以两种方法排序（编号由低到高排序）：1）使用Arrays.sort(Object[] a)方法；2）使用Arrays.sort(Object[] a, **Comparator** c)方法。
6. 再定义一个类TeacherManagement（属于cn.sd包），提供方法search，方法可以在一组给定的教师中，根据姓名（或年龄）返回等于指定姓名（或年龄）的教师的字符串信息，信息格式为：“编号为\*\*、姓名为\*\*、年龄为\*\*的\*\*学院老师”。如果没有满足条件的教师，则返回“没有符合条件的教师”。
7. 构造main方法进行测试。

代码：

Teacher类：

package cn.net.sdkd;

public class Teacher{

private int no;

private String name;

private int age;

private String seminiary;

public Teacher(){}

public Teacher(int no,String name,int age,String seminiary)

{

this.no = no;

this.name =name;

this.age = age;

this.seminiary = seminiary;

}

public String toString()

{

return "编号为"+ this.no +"，姓名为"+ this.name +"，年龄为"+ this.age +"的"+ this.seminiary +"学院老师";

}

public boolean compareTo(Object o)

{

if (!(o instanceof Teacher)) return false;

Teacher unit = (Teacher) o;

return unit.no == no;

}

}

TeacherManagement类：

import cn.net.sdkd.Teacher;

public class TeacherManagement{

public static String search(Teacher[] teachers,String name)

{

Teacher t=null;

for(int i=0; i < teachers.length; i++)

{

if(teachers[i] != null && teachers[i].getName() == name)

{

t = teachers[i];

break;

}

}

if(t!=null) return t.toString();

else return "没有符合条件的教师";

}

public static String search(Teacher[] teachers, int age)

{

Teacher t = null;

for(int i = 0; i < teachers.length; i++)

{

if(teachers[i] != null && teachers[i].getAge() == age)

{

t = teachers[i];

break;

}

}

if(t != null) return t.toString();

else return "没有符合条件的教师";

}

}

1. 设计一个带表头的双向链表（链表中数据的具体类型可以随意），提供以下方法：

（1）insert：在某个位置插入对象；

（2）insert：在链表的最后插入对象；

（3）delete：在某个位置删除对象；

（4）delete：删除链表中与x相同的元素；

（5）size：返回当前链表中对象的个数；

（6）isEmpty：判断链表是否为空；

（7）traverse：遍历链表，打印出所有的元素；

（8）getData：取得某个位置的对象。构造main函数进行测试。

|  |
| --- |
| **import** java.util.Scanner;  **class** stack  {  **private** Object[] a;  **private** **int** left;  //初始化一个Object栈为空  stack(**int** n)  {  a = **new** Object[n];  **for** (**int** i = 0; i < n; i++) a[i] = **null**;  left = n;  }  // 判断栈是否为空  **boolean** isEmpty() { **return** left == a.length; }  //出栈，如果为空栈返回ERROR;  **void** push(Object obj)  {  **if** (left == 0)  {  System.***out***.println("ERROR!");  **return**;  }  a[left-1] = obj;  left--;  }  //入栈  Object pop()  {  Object object = a[left];  a[left] = **null**;  left++;  **return** object;  }  Object getTop()  {  **return** a[left];  }    **public** **void** bin()  {  stack st = **new** stack(100);  Scanner sc = **new** Scanner(System.***in***);  **int** num;  System.***out***.print("Please input the number:");  **while** (sc.hasNextInt())  {  num = sc.nextInt();  //将一个数模二入栈，然后按顺序出栈，得到的刚好是二进制数  **while** (num > 0)  {  st.push(num % 2);  num = num / 2;  }  System.***out***.print("The number is:");  **while** (!st.isEmpty())  {  System.***out***.print(st.getTop());  st.pop();  }  System.***out***.print("\n");  }  }  } |

1. 设计一个带表头的双向链表（链表中数据的具体类型可以随意），提供以下方法：

（1）insert：在某个位置插入对象；

（2）insert：在链表的最后插入对象；

（2）delete：在某个位置删除对象；

（3）delete：删除链表中与x相同的元素；

（4）size：返回当前链表中对象的个数；

（5）isEmpty：判断链表是否为空；

（6）traverse：遍历链表，打印出所有的元素

（7）getData：取得某个位置的对象。构造main函数进行测试。

|  |
| --- |
| DLinkList类：  **package** 实验二;  **import** java.util.\*;  **public** **class** DLinkList {    **private** **class** Node {  **private** Node prior;  **private** Node next;  **private** Object data;  **public** Node() {  **this**.data = **null**;  **this**.prior = **null**;  **this**.next = **null**;  }  **public** Node(Object data, Node prior, Node next) {  **this**.data = data;  **this**.prior = prior;  **this**.next = next;  }  }    **private** **int** size;  **private** Node head;  **private** Node tail;    /\*  \* 初始化链表  \*/  **public** DLinkList() {  head = **new** Node(**null**, **null**, **null**);  tail = **new** Node(**null**, head, **null**);  head.next = tail;  // head=head.next;  size = 0;  }    /\*  \* 查找第index个节点并返回  \*/  **public** Node getNode(**int** index) {  **if** (index < 0 || index >= size) {  **throw** **new** IndexOutOfBoundsException();  }  Node p = head;  **for** (**int** i = 0; i < index; i++) {  p = p.next;  }  **return** p;  }  /\*  \* （1）insert：在某个位置插入对象；  \*/  **public** **int** insert(**int** index, Object data) {  **if** (index < 0 || index >= size) {  **throw** **new** IndexOutOfBoundsException();  }  **if** (index == 0) {  Node p = **new** Node(data, head, head.prior);  head.next.prior = p;  head.next = p;  size++;  **return** 1;  }  Node q = getNode(index - 1);  Node p = **new** Node(data, q, q.next);  q.next.prior = p;  q.next = p;  size++;  **return** 1;  }  /\*  \* （2）insert：在链表的最后插入对象；  \*/  **public** **int** insert(Object data) {  Node p = **new** Node(data, **null**, **null**);  tail.prior.next = p;  p.prior = tail.prior;  p.next = tail;  tail.prior = p;  size++;  **return** 1;  }  /\*  \* （2）delete：在某个位置删除对象；  \*/  **public** **int** delete(**int** index, Object data) {  **if** (index < 0 || index >= size)  **throw** **new** IndexOutOfBoundsException();  Node q = getNode(index);  q.prior.next = q.next;  q.next.prior = q.prior;  size--;  **return** 1;  }  /\*  \* （3）delete：删除链表中与x相同的元素；  \*/  **public** **int** delete(Object data) {  Node q = head;  **for** (**int** i = 0; i < size; i++) {  **if** (q.data.equals(data)) {  **this**.delete(i, data);  }  q = q.next;  }  **return** 1;  }  /\*  \* （4）size：返回当前链表中对象的个数;  \*/  **public** **int** Getsize() {  **return** size;  }  /\*  \* （5）isEmpty：判断链表是否为空；  \*/  **public** **boolean** isEmpty() {  **if** (size == 0)  **return** **true**;  **else**  **return** **false**;  }  /\*  \* （6）traverse：遍历链表，打印出所有的元素；  \*/  **public** **void** traverse() {  Node q = head;  **for** (**int** i = 0; i < size; i++) {  q = q.next;  **if** (i == 0)  System.***out***.print(q.data);  **else**  System.***out***.print(" " + q.data);  }  System.***out***.print("\n");  }  /\*  \* （7）getData：取得某个位置的对象。  \*/  **public** Object getData(**int** index) {  Node p = head;  **for** (**int** i = 0; i < index; i++) {  p = p.next;  }  **return** p.data;  }  } |

|  |
| --- |
| DLinkListMain类  **package** 实验二;  **public** **class** DLinkListMain {  **public** **static** **void** main(String[] args) {  **int** x = 0;  DLinkList list = **new** DLinkList();  System.***out***.println("获取初始链表长度：\n" + list.Getsize());  System.***out***.println("在链表依次添加元素1,3,4,5,3，并打印链表：");  **for** (**int** i = 1; i <= 5; i++) {  **if** (i != 2)  list.insert(i);  }  list.insert(3);  list.traverse();  System.***out***.println("在链表第2个位置添加元素2，并打印链表：");  list.insert(2, 2);  list.traverse();  System.***out***.println("获取链表中第5个元素：\n" + list.getData(5));  System.***out***.println("删除链表中第4个元素，并打印链表：");  list.delete(4, x);  list.traverse();  System.***out***.println("删除链表中第一个值为3的元素，并打印链表：");  list.delete(3);  list.traverse();  System.***out***.println("判断链表是否为空：\n" + list.isEmpty());  System.***out***.println("获取链表长度：\n" + list.Getsize());  }  } |

4．利用二维数组（double[]）实现一个矩阵类：Matrix。要求提供以下方法：

（1）set（int row, int col, double value）：将第row行第col列的元素赋值为value

（2）get（int row，int col）：取第row行第col列的元素；

（3）width()：返回矩阵的列数；

（4）height()：返回矩阵的行数；

（5）Matrix add（Matrix b）：返回当前矩阵与矩阵b相加后的矩阵；

（6）Matrix multiply（Matrix b）：返回当前矩阵与矩阵b相乘后的矩阵。

（7）Matrix transpose()：返回当前矩阵的转置矩阵；

（8）getMax()：返回矩阵中的最大值及其所在行和列；

（9）print()：以行和列的形式打印出当前矩阵。

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Matrix ma = new Matrix(3,3);  Matrix mb = new Matrix(3,3);  Matrix mc = new Matrix(2,4);  Matrix md = new Matrix(4,2);  for(int i = 0; i < 3; i++) {  for(int j = 0; j < 3; j++) {  ma.set(i, j, 1.0);  }  }  for(int i = 0; i < 3; i++)  {  for(int j = 0; j < 3; j++) {  mb.set(i, j, 2.0);  }  }  for(int i = 0; i < 2; i++)  {  for(int j = 0; j < 4; j++) {  mc.set(i, j, (double)(i+1)\*(j+1));  }  }  for(int i = 0; i < 4; i++)  {  for(int j = 0; j < 2; j++) {  md.set(i, j, (double)i+j+1);  }  }  System.out.println("矩阵3：");  mc.print();  mc.getMax();  }  } |

|  |
| --- |
| public class Matrix {  double array[][];  private int height; //行  private int width; //列  Matrix(int h,int w){  this.height = h;  this.width = w;  array = new double[h][w];  }  //set（int row, int col, double value）：将第row行第col列的元素赋值为value；  public void set(int row,int vol,double value) {  array[row][vol] = value;  }  //get（int row，int col）：取第row行第col列的元素  public double get(int row,int vol) {  return array[row][vol];  }  //height()：返回矩阵的行数  public int height() {  return height;  }  //width()：返回矩阵的列数  public int width() {  return width;  }  //Matrix add（Matrix b）：返回当前矩阵与矩阵b相加后的矩阵  public Matrix add(Matrix b) {  if(b.height() == height && b.width() == width) {  for(int i = 0; i < height; i++) {  for(int j = 0; j < width; j++) {  array[i][j] = array[i][j] + b.get(i, j);  }  }  }  return this;  }  //Matrix multiply（Matrix b）：返回当前矩阵与矩阵b相乘后的矩阵  public Matrix multiply(Matrix b) {  Matrix c = new Matrix(b.width, this.height);  if(this.height == b.width) {  int tmp = 0;  for(int i = 0; i < this.height; i++) {  for(int j = 0; j < b.width; j++) {  for(int k = 0; k < this.width; k++) {  tmp += this.get(i,k)\*b.get(k, j);  c.set(i, j, tmp);  }  tmp = 0;  }  }  }  return c;  }  //Matrix transpose()：返回当前矩阵的转置矩阵  public void Matrixtranspose() {  double[][] b = new double[width][height];  System.out.println(height + "-----------------" + width);  for(int i = 0; i < height; i++) {  for(int j = 0; j < width; j++) {  b[j][i] = array[i][j];  }  }  for(int i = 0; i < width; i++) {  for(int j = 0; j < height; j++) {  System.out.print(b[i][j] + "\t");  }  System.out.println();  }  }    //getMax()：返回矩阵中的最大值及其所在行和列  public void getMax() {  double max = 0;  int x=0,y=0;  for(int i = 0; i < this.height; i++) {  for(int j = 0; j < this.width; j++) {  if(array[i][j] > max)  {  max = array[i][j];  x = i+1;  y = j+1;  }  }  }  System.out.println("最大值为"+max+" 位于第"+x+"行第"+y+"列");    }    //print()：以行和列的形式打印出当前矩阵  public void print() {  for(int i = 0; i < this.height; i++) {  for(int j = 0; j < this.width; j++) {  System.out.print(array[i][j]+"\t");  }  System.out.println();  }  }  } |