学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 于承志 | **学号** | 2021901678 |
| **学院** | 信息工程学院 | **专业** | 电子信息 |

（请在下面表格本周学习情况）

|  |
| --- |
| 学习情况简述 |
| 复习了链表以及链表形式栈和队列  并学习了Java的各种修饰词和接口  练习了相应代码 |
| 本周练习过的代码 |
| 代码一：  import java.util.LinkedList;  public class linkedlist<E>{  private class Node{  public E e;  public Node next;  public Node(E e,Node next){  this.e = e;  this.next = next;  }  public Node(E e){  this(e,null);  }  public Node(){  this(null,null);  }  @Override  public String toString(){  return e.toString();  }  }  private Node dummyhead;  private int size;  public linkedlist(){  dummyhead = new Node(null,null);  size = 0;  }  public int getSize(){  return size;  }  public boolean isEmpty(){  return size == 0;  }  public void addFirst(E e ){  add(0,e);  }  public void add(int index, E e){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }   Node prev = dummyhead;  for (int i = 0; i < index ; i++) {  prev = prev.next;  }  prev.next = new Node(e,prev.next);  size ++;  }  public void addLast(E e){  add(size,e);  }  public E get(int index){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }  Node cur = dummyhead.next;  for (int i = 0; i < index; i++) {  cur = cur.next;  }  return cur.e;  }  public E getFirst(){  return get(0);  }  public E getLirst(){  return get(size - 1);  }  public void set (int index,E e){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }  Node cur = dummyhead.next;  for (int i = 0; i < index; i++) {  cur =cur.next;  }  cur.e = e;  }  public boolean contains(E e){  Node cur =dummyhead.next;  while (cur != null) {  if (cur.e.equals(e)){  return true;  }  cur = cur.next;  }  return false;  }  @Override  public String toString(){  StringBuilder res = new StringBuilder();  Node cur =dummyhead.next;  while(cur != null){  res.append(cur+"->");  cur = cur.next;  }  return res.toString();  }  public E remove(int index){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }  Node cur =dummyhead.next;  for (int i = 0; i < index-1; i++) {  cur = cur.next;  }  E e =cur.next.e;  cur.next =cur.next.next;  cur.next.next = null;  return e;   }  public E removeFirst(){  return remove(0);  }  public E removeLirst(){  return remove(size-1);  }  } |
| 代码二：栈  ackage stack;  public class Array<E> {  private E[] data;  private int size;   public int getSize() {  return size;  }   public Array(int capacity) {  data = (E[])new Object[capacity];  size = 0;  }   public Array() {  this(10);  }   public int getCapacity() {  return data.length;  }   public boolean isEmpty() {  return size == 0;  }   public void addLast(E e) {  insertArray(e, size);  }   public void addFirst(E e) {  insertArray(e, 0);  }   public void insertArray(E e, int loc) {  if (size == data.length) {  resize(2\* data.length);  }  if (loc < 0 || loc > size) {  throw new IllegalArgumentException("Insert failed. Require loc >= 0 and loc <= size");  }  for (int i = size; i > loc; i--) {  data[i] = data[i - 1];  }  data[loc] = e;  size++;  }   private void resize(int newCapacity) {  E[] newData = (E[]) new Object[newCapacity];  for (int i = 0; i < size; i++) {  newData[i] = data[i];  }  data = newData;  }   public E get(int loc) {  if (loc < 0 || loc > size) {  throw new IllegalArgumentException("Insert failed. Require loc >= 0 and loc <= size");  }  return data[loc];  }  public E getLast(){  return get(size-1);  }  public E getFirst(){  return get(0);  }  public boolean contains( E e){  for (int i= 0; i < size ; i++) {  if(data[i].equals(e) ){  return true;  }  }  return false;  }  public int find (E e){  for (int i = 0; i < size; i++) {  if(data[i].equals(e)){  return i;  }  }  return -1;  }  public E Delete( int index){  if(index < 0 || index >= size){  throw new IllegalArgumentException("Delete failed.Require index >= 0 and index < size");  }  E n = data[index];  for (int i = index+1; i < size; i++) {  data[i-1]=data[i];  }  size --;  data[size] = null ;  if(size == data.length / 4 && data.length / 2 != 0){  resize( data.length / 2);  }  return n;  }  public E DeleteFirst(){  return Delete(0);  }  public E DeleteLast(){  return Delete(size-1);  }  public void DeleteElement(E e){  int index = find(e);  if ( index != -1){  Delete(index);  }  }  @Override  public String toString() {  StringBuilder s = new StringBuilder();  s.append(String.*format*("Array: size = %d, capacity = %d\n", size, data.length));  s.append('[');  for (int i = 0; i < size; i++) {  s.append(data[i]);  if (i != size - 1) {  s.append(',');  }  }  s.append(']');  return s.toString();  } }  package stack;  public class Arraystack<E> implements Stack<E> {   Array<E> array;  public Arraystack( int capacity){  array = new Array<>(capacity);  }  public Arraystack(){  array = new Array<>();  }   @Override  public int getsize() {  return array.getSize();  }   @Override  public boolean isEmpty() {  return array.isEmpty();  }  public int getCapacity(){  return array.getCapacity();  }  @Override  public void push(E e) {  array.addLast(e);  }   @Override  public E pop() {  return array.DeleteLast();  }   @Override  public E peek() {  return array.getLast();  }  @Override  public String toString(){  StringBuilder res = new StringBuilder();  res.append("Stack:");  res.append("[");  for (int i = 0; i < array.getSize(); i++) {  res.append(array.get(i));  if (i != array.getSize()-1){  res.append(",");  }  }  res.append("] top");  return res.toString();  } }  package stack;  public class LinkedList<E>{  private class Node{  public E e;  public Node next;  public Node(E e,Node next){  this.e = e;  this.next = next;  }  public Node(E e){  this(e,null);  }  public Node(){  this(null,null);  }  @Override  public String toString(){  return e.toString();  }  }  private Node dummyhead;  private int size;  public LinkedList(){  dummyhead = new Node(null,null);  size = 0;  }  public int getSize(){  return size;  }  public boolean isEmpty(){  return size == 0;  }  public void addFirst(E e ){  add(0,e);  }  public void add(int index, E e){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }   Node prev = dummyhead;  for (int i = 0; i < index ; i++) {  prev = prev.next;  }  prev.next = new Node(e,prev.next);  size ++;  }  public void addLast(E e){  add(size,e);  }  public E get(int index){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }  Node cur = dummyhead.next;  for (int i = 0; i < index; i++) {  cur = cur.next;  }  return cur.e;  }  public E getFirst(){  return get(0);  }  public E getLirst(){  return get(size - 1);  }  public void set (int index,E e){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }  Node cur = dummyhead.next;  for (int i = 0; i < index; i++) {  cur =cur.next;  }  cur.e = e;  }  public boolean contains(E e){  Node cur =dummyhead.next;  while (cur != null) {  if (cur.e.equals(e)){  return true;  }  cur = cur.next;  }  return false;  }  @Override  public String toString(){  StringBuilder res = new StringBuilder();  Node cur =dummyhead.next;  while(cur != null){  res.append(cur+"->");  cur = cur.next;  }  return res.toString();  }  public E remove(int index){  if (index < 0||index > size){  throw new IllegalArgumentException("Add failed.Illegal index.");  }  Node cur =dummyhead.next;  for (int i = 0; i < index-1; i++) {  cur = cur.next;  }  E e =cur.next.e;  cur.next =cur.next.next;  cur.next.next = null;  return e;   }  public E removeFirst(){  return remove(0);  }  public E removeLirst(){  return remove(size-1);  }  }  package stack;  public class LinkedListStack<E> implements Stack<E> {   private LinkedList<E> list;   public LinkedListStack(){  list = new LinkedList<>();  }   @Override  public int getsize() {  return list.getSize();  }   @Override  public boolean isEmpty() {  return list.isEmpty();  }   @Override  public void push(E e) {  list.addFirst(e);  }   @Override  public E pop() {  return list.removeFirst();  }   @Override  public E peek() {  return list.getFirst();  }  @Override  public String toString(){  StringBuilder res = new StringBuilder();  res.append("Stack: top ");  res.append("list");  return res.toString();  } }  package stack;  public interface Stack<E> {  int getsize();  boolean isEmpty();  void push( E e );  E pop(); //取出元素  E peek(); //栈顶元素  } |
| 代码三：  public class ArrayQueue<E> implements Queue<E>{  private Array<E> array;   public ArrayQueue(int capacity){  array = new Array<>(capacity);  }  public ArrayQueue(){  array = new Array<>();  }   @Override  public int getsize() {  return array.getSize();  }   @Override  public boolean isEmpty() {  return array.isEmpty();  }   public int getcapapcity(){  return array.getCapacity();  }  @Override  public void enqueue(E e) {  array.addLast(e);  }   @Override  public E dequeue() {  return array.DeleteFirst();  }   @Override  public E getfront() {  return array.getFirst();  }  @Override  public String toString(){  StringBuilder res = new StringBuilder();  res.append("Queue:");  res.append("front [");  for (int i = 0; i < array.getSize(); i++) {  res.append(array.get(i));  if (i != array.getSize()-1){  res.append(",");  }  }  res.append("] tail");  return res.toString();  }  }  public class LinkedListQueue<E> implements Queue<E> {  @Override  public int getsize() {  return size;  }   @Override  public boolean isEmpty() {  return size == 0;  }   @Override  public void enqueue(E e) {  if (tail == null){  tail = new Node(e);  head = tail;  }else {  tail.next = new Node(e);  tail = tail.next;  }  size ++;  }   @Override  public E dequeue() {  if (isEmpty()){  throw new IllegalArgumentException("Cannot dequeue from an empty queue.");  }  Node retNode = head;  head = head.next;  retNode.next = null;  if (head == null){  tail = null;  }  size --;  return retNode.e ;  }  @Override  public String toString(){  StringBuilder res = new StringBuilder();  res.append("Queue: front ");  Node cur =head;  while(cur != null){  res.append(cur+"->");  cur = cur.next;  }  res.append("Null tail");  return res.toString();  }  @Override  public E getfront() {  if (isEmpty()){  throw new IllegalArgumentException("Queue is empty.");  }  return head.e;  }   private class Node{  public E e;  public Node next;  public Node(E e,Node next){  this.e = e;  this.next = next;  }  public Node(E e){  this(e,null);  }  public Node(){  this(null,null);  }  @Override  public String toString(){  return e.toString();  }  }  private Node head, tail;  private int size;  public LinkedListQueue(){  head = null;  tail = null;  size = 0;  } }  public class LoopQueue<E> implements Queue<E> {  private E[] data;  private int front,tail;  private int size;  public LoopQueue(int capacity) {  data = (E[])new Object[capacity+1];  front = 0;  tail = 0;  size = 0;  }  public LoopQueue(){  this(10);  }  public int getCapacity(){  return data.length-1;  }  @Override  public int getsize() {  return size;  }   @Override  public boolean isEmpty() {  return front == tail;  }   @Override  public void enqueue(E e) {  if ((tail + 1)% data.length == front)  resize(getCapacity()\*2);  data[tail] = e;  tail = (tail + 1) % data.length;  size ++;  }   private void resize(int Newcapacity) {  E[] NewData = (E[])new Object[ Newcapacity + 1 ];  for (int i = 0; i < size; i++) {  NewData[i] = data[(i+front) % data.length];  }  data =NewData;  front = 0;  tail = size;  }   @Override  public E dequeue() {  if (isEmpty()){  throw new IllegalArgumentException("Cannot dequeue from an empty queue.");  }  E rat =data[front];  front = (front + 1) % data.length;  size -- ;  if(size == getCapacity() / 4 && getCapacity()!= 0){  resize(getCapacity() / 2);  }  return rat;  }   @Override  public E getfront() {  if (isEmpty()){  throw new IllegalArgumentException("Cannot dequeue from an empty queue.");  }  return data[front];  }  @Override  public String toString(){  StringBuilder s = new StringBuilder();  s.append(String.*format*("Queue: size = %d, capacity = %d\n", size, getCapacity()));  s.append("front [");  for (int i = front; i != tail; i=(i+1)&data.length) {  s.append(data[i]);  if (i != size - 1) {  s.append(',');  }  }  s.append("] tail");  return s.toString();  } }  public interface Queue<E> {  int getsize();  boolean isEmpty();  void enqueue(E e);  E dequeue();  E getfront();  } |