### JAVA第五阶段—DAY06-JAVA案例

1. 创建栈结构，依次将五虎上将（关羽、张飞、马超、黄忠、赵云）存入栈结构中再弹出栈顶元素。

public class Demo01Stack {  
  
 public static void main(String[] args) {  
*// 创建栈结构* Stack<String> stack=new Stack<String>();  
*// 检测栈中是否是空* System.out.println(stack.empty());  
  
*// 压栈* stack.push("刘备");  
 stack.push("关羽");  
 stack.push("张飞");  
 stack.push("赵云");  
 stack.push("黄忠");  
  
*// 出栈，将栈顶元素弹出* stack.pop();  
  
*// 查看栈顶元素* System.out.println(stack.peek());  
 }  
}

1. 创建队列结构，依次装入刘备、关羽、张飞、赵云、黄忠元素。然后删除队头元素。

public class Demo02Queue {  
  
 public static void main(String[] args) {  
*// 创建队列* Queue<String> queue = new LinkedList<String>();  
*// 使用add或者offer方法添加元素* queue.add("刘备");  
 queue.add("关羽");  
 queue.add("张飞");  
 queue.offer("赵云");  
 queue.offer("黄忠");  
  
*// 删除队头元素* System.out.println(queue.poll());  
*// System.out.println(queue.remove());  
  
// 获取队头元素* System.out.println(queue.element());  
 System.out.println(queue.peek());  
  
 }  
}

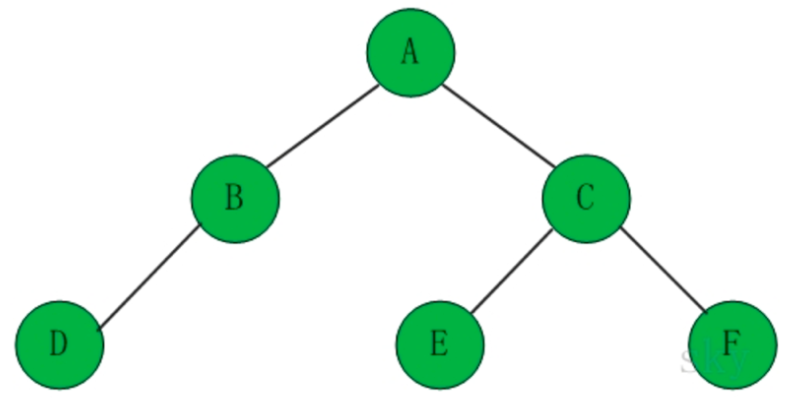
1. 自定义实现单链表结构。

*//单向链表的代码实现*public class LinkDemo {  
  
 public static void main(String[] args) {  
  
 LinkList theList = new LinkList(); *// make new list* theList.insertFirst(22);  
 theList.insertFirst(44);  
 theList.insertFirst(66);  
 theList.insertFirst(88);  
 System.out.println(theList.isEmpty());  
 theList.displayList();  
 theList.deleteFirst();  
 theList.displayList();  
  
 }  
  
}  
  
*//链表中的节点对象*class Node{  
 public int data;  
 public Node next;  
  
 public Node(int data) {  
 this.data = data;  
 }  
  
 public void display() {  
 System.out.print("{" + data + "} ");  
 }  
}  
  
class LinkList{  
  
 private Node first;  
  
 public LinkList() {  
 first = null;  
 }  
  
 public boolean isEmpty() {  
 return (first == null);  
 }  
  
 public void insertFirst(int data) { *// make new link* Node newNode = new Node(data);  
 newNode.next = first; *// newLink --> old first* first = newNode; *// first --> newLink* }  
  
 public Node deleteFirst(){ *// delete first item* if(!isEmpty()) {  
 Node temp = first; *// save reference to link* first = first.next; *// delete it: first-->old next* return temp; *// return deleted link* }  
 return null;  
  
 }  
 public void displayList() {  
 System.out.print("List (first-->last): ");  
 Node current = first; *// start at beginning of list* while (current != null) *// until end of list,* {  
 current.display(); *// print data* current = current.next; *// move to next link* }  
 System.out.println("");  
 }  
  
}

1. 练习系统提供的链表LinkedList相关API.

public class LinkDemo02 {  
  
 public static void main(String[] args) {  
  
 LinkedList<String> list = new LinkedList<String>();  
*// 添加元素* list.add("100");  
 list.add("200");  
 list.add("300");  
 list.add("400");  
 list.add("500");  
*// 位置2的地方插入* list.add(2, "150");  
 System.out.println(list);  
*// 获取索引3位置的元素* System.out.println(list.get(3));  
*// 删除元素* System.out.println(list.remove(2));  
 System.out.println(list);  
  
*// 获取首节点* System.out.println("链表的第一个元素是：" + list.getFirst());  
*// 获取尾节点* System.out.println("链表的最后一个元素是：" + list.getLast());  
 }  
}

1. 自定义实现2叉树结构，并按成下图的存储结构。



**public class** TreeListDemo01 {  
  
 **public static void** main(String[] args) {  
 BinaryTreeNode a = **new** BinaryTreeNode(**'A'**);  
 BinaryTreeNode b = **new** BinaryTreeNode(**'B'**);  
 BinaryTreeNode c = **new** BinaryTreeNode(**'C'**);  
 BinaryTreeNode d = **new** BinaryTreeNode(**'D'**);  
 BinaryTreeNode e = **new** BinaryTreeNode(**'E'**);  
 BinaryTreeNode f = **new** BinaryTreeNode(**'F'**);  
  
 a.setLeftChirld(b);  
 a.setRightChirld(c);  
 b.setLeftChirld(d);  
 c.setLeftChirld(e);  
 c.setRightChirld(f);  
  
 BinaryTree tree = **new** BinaryTree(a);  
*// 先序遍历* tree.preOrderTreeNode(tree.getRoot());  
*// 中序遍历* tree.inOrderTreeNode(tree.getRoot());  
*// 后序遍历* tree.postOrderTreeNode(tree.getRoot());  
 }  
  
}  
*/\*\*  
 \* 二叉树节点  
 \*/***class** BinaryTreeNode {  
 **private char** data; *//数据* **private** BinaryTreeNode leftChirld; *//左孩子* **private** BinaryTreeNode rightChirld; *//右孩子* **public** BinaryTreeNode(**char** data) {  
 **this**.data = data;  
 }  
  
 **public char** getData() {  
 **return** data;  
 }  
 **public void** setData(**char** data) {  
 **this**.data = data;  
 }  
 **public** BinaryTreeNode getLeftChirld() {  
 **return** leftChirld;  
 }  
 **public void** setLeftChirld(BinaryTreeNode leftChirld) {  
 **this**.leftChirld = leftChirld;  
 }  
 **public** BinaryTreeNode getRightChirld() {  
 **return** rightChirld;  
 }  
 **public void** setRightChirld(BinaryTreeNode rightChirld) {  
 **this**.rightChirld = rightChirld;  
 }  
  
 @Override  
 **public** String toString() {  
 **return " {"** + data + **'}'**;  
 }  
}  
  
*/\*\*  
 \* 二叉树  
 \*/***class** BinaryTree {  
 **private** BinaryTreeNode root;  
  
 *//初始化二叉树* **public** BinaryTree(){}  
  
 **public** BinaryTree(BinaryTreeNode root){  
 **this**.root = root;  
 }  
  
 **public void** setRoot(BinaryTreeNode root){  
 **this**.root = root;  
 }  
  
 **public** BinaryTreeNode getRoot(){  
 **return** root;  
 }  
  
 *// 先序遍历* **public void** preOrderTreeNode(BinaryTreeNode node){  
 **if**(node == **null**){  
 **return** ;  
 }  
 System.out.println(node);  
 preOrderTreeNode(node.getLeftChirld());  
 preOrderTreeNode(node.getRightChirld());  
 }  
  
 *// 中序遍历* **public void** inOrderTreeNode(BinaryTreeNode node){  
 **if**(node == **null**){  
 **return** ;  
 }  
 inOrderTreeNode(node.getLeftChirld());  
 System.out.println(node);  
 inOrderTreeNode(node.getRightChirld());  
 }  
  
 *// 后序遍历* **public void** postOrderTreeNode(BinaryTreeNode node){  
 **if**(node == **null**){  
 **return** ;  
 }  
 postOrderTreeNode(node.getLeftChirld());  
 postOrderTreeNode(node.getRightChirld());  
 System.out.println(node);  
 }  
}