Link data structure

Linked data structure由 各个叫做nodes通过links构建而成

Links可以被看作单向箭头，从一个node指向另一个

在java里,node被看作是node class里的object

Node里的数据通过instance variable被存储

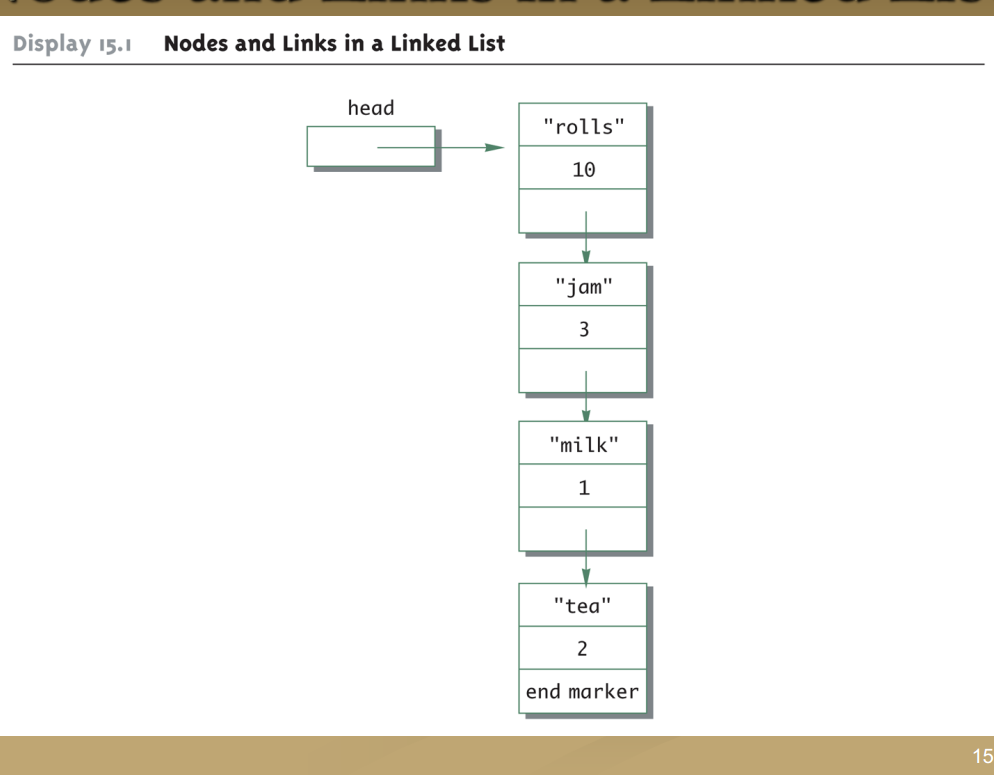
Link 被认为是reference，一个reference就是一个内存地址，被存在class type的变量里，因此,link就是node class type自己的一个instance variable

最简单的linked data structure就是Linked list

一个linked list由一个单一的node链组成，每一个都连到下一个通过Link

第一个Node叫做head node

最后一个node的功能是end marker，象征着结束



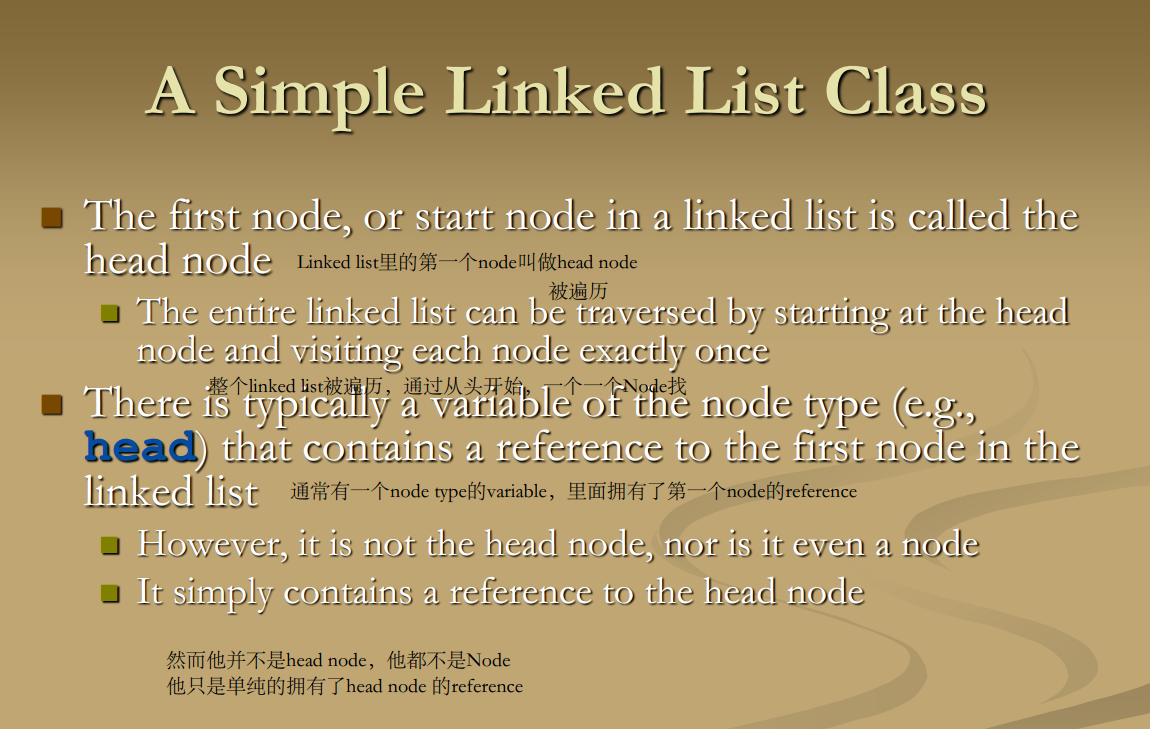
在linked list中，每个node都是node class的一个object

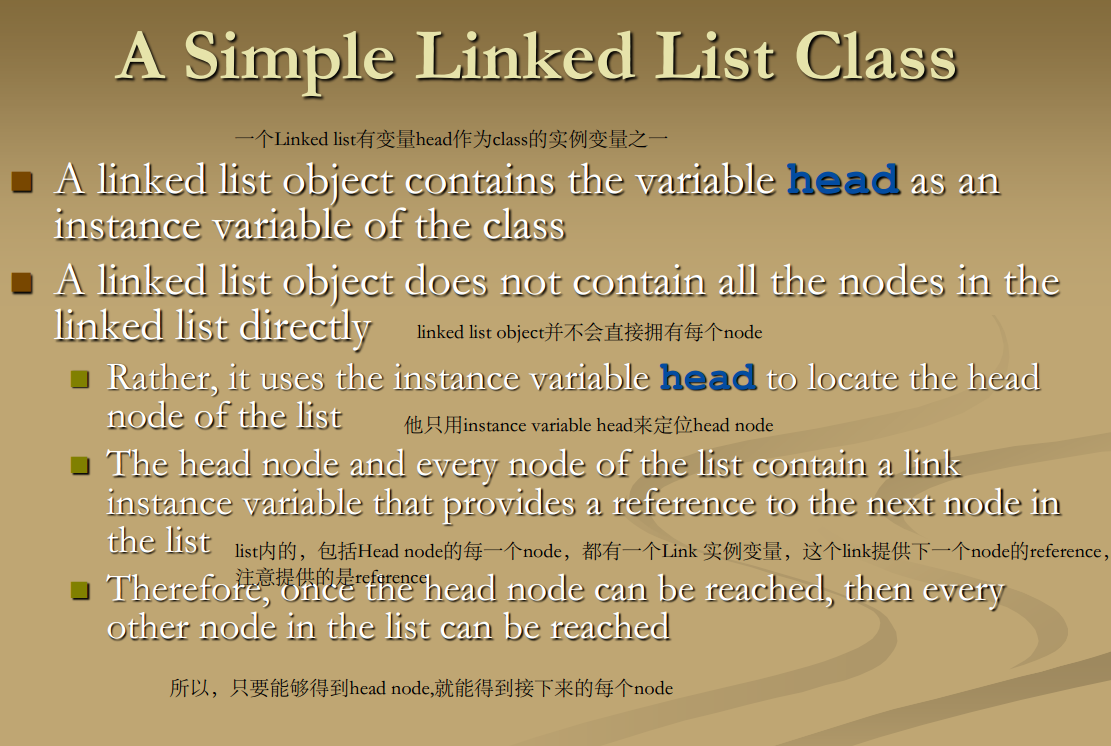
注意每个node被看成一个拥有一个或多个data的box

每个node拥有data和指向另一个node的 link，一个data以实例变量的形式存储在node中

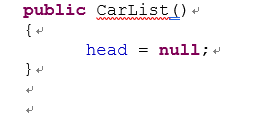
Data代表着node “box”里存储的信息

Link被看做node的reference  
link被看做指向他们link的node的箭头

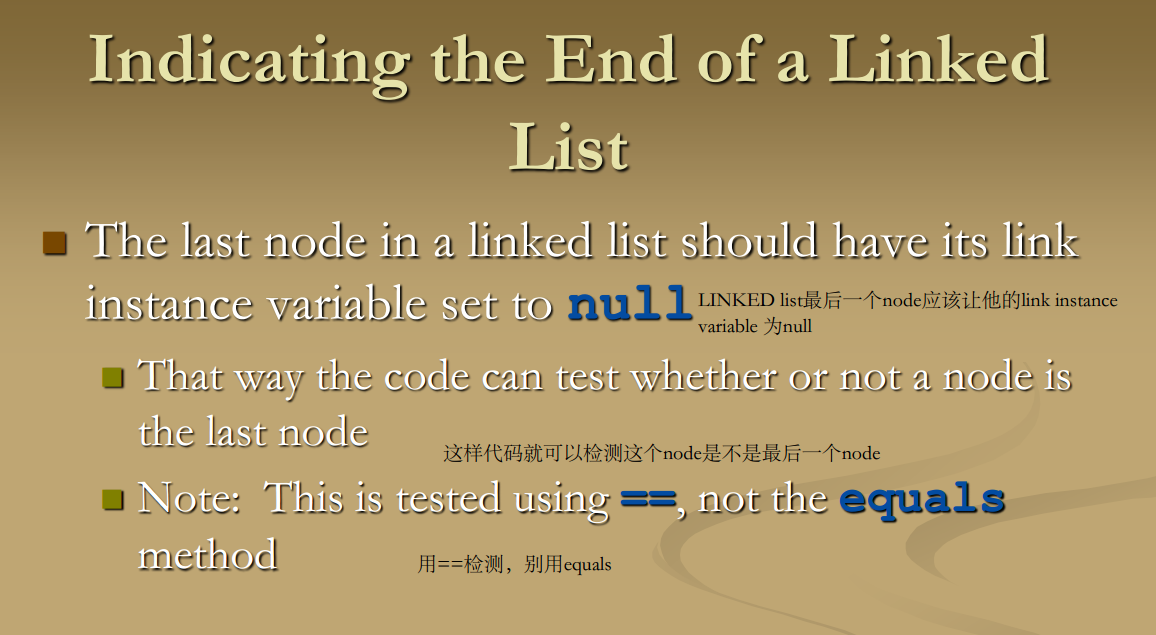




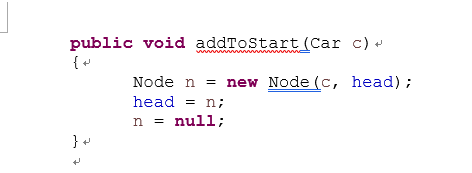








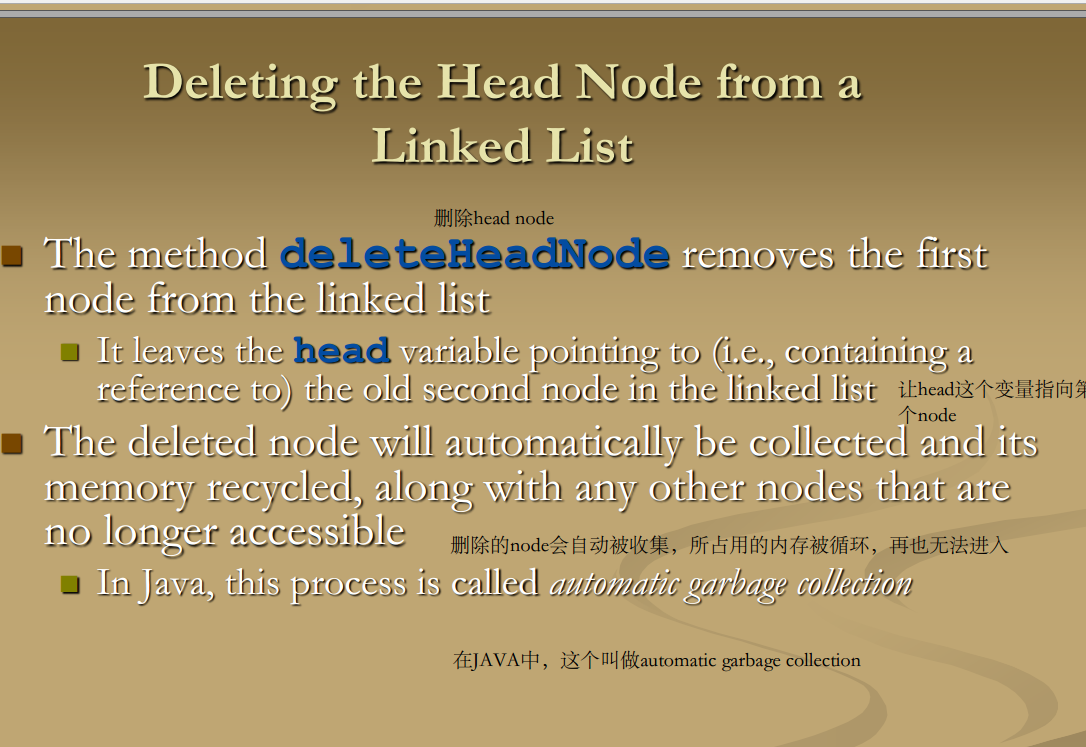
 然后t1一直往下，直到他=null

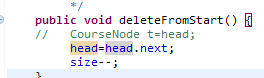


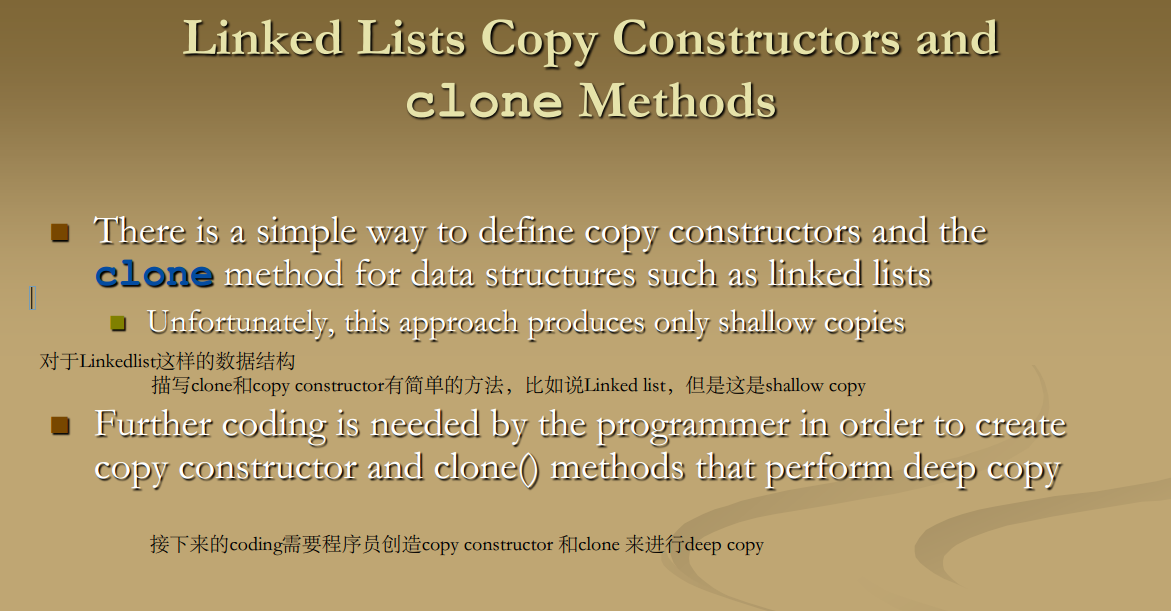
head=new Node(I,head);

新的node把老的head作为下一个node，然后新的node赋给head，

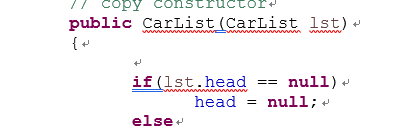
而老的第一个head是null，所以最后一个node永远是Null

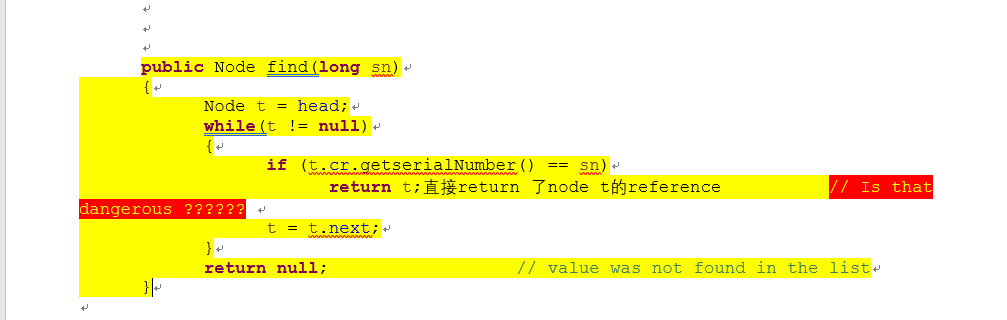


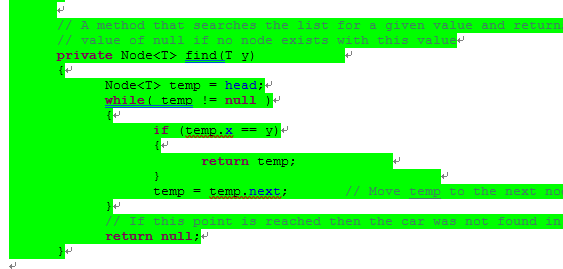






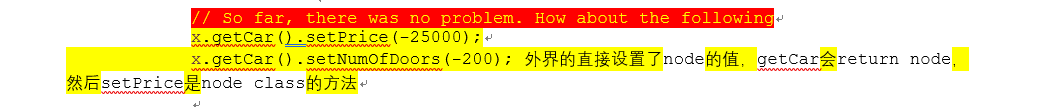
.

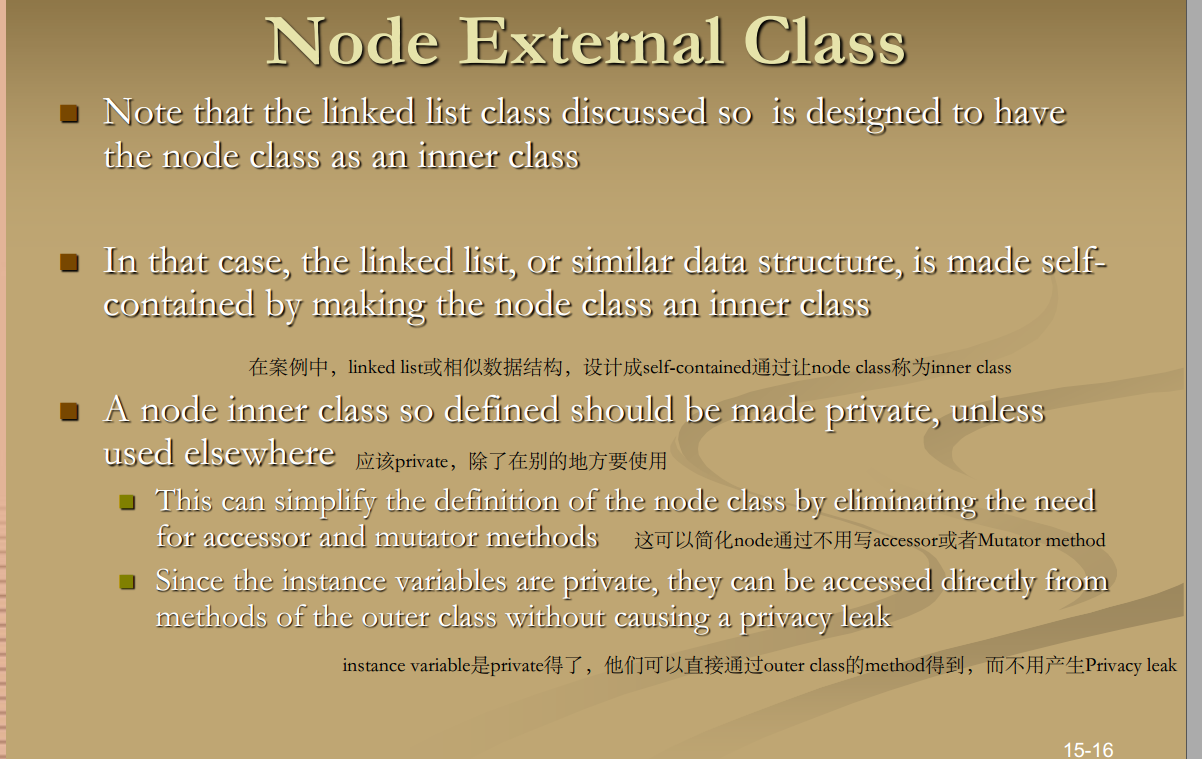




标准find，要创建一个标准temp，然后head赋值，



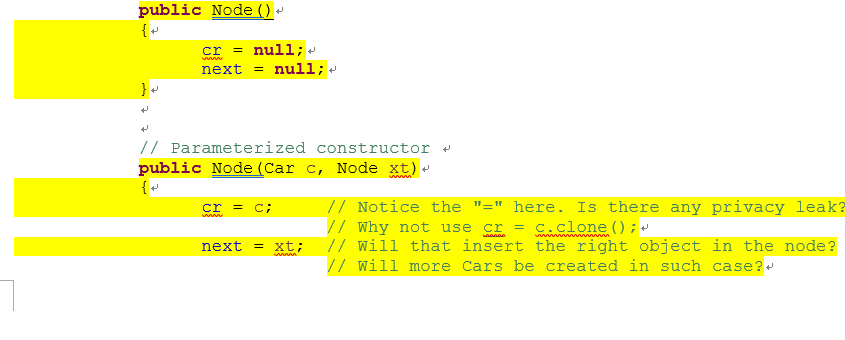


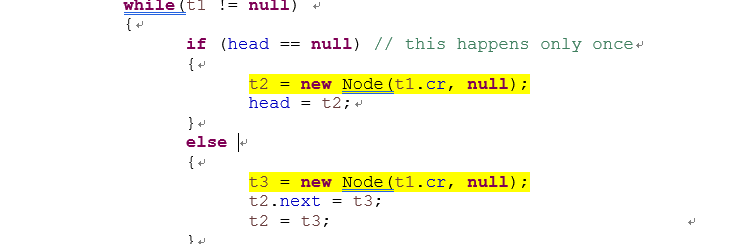


一个标准private构成

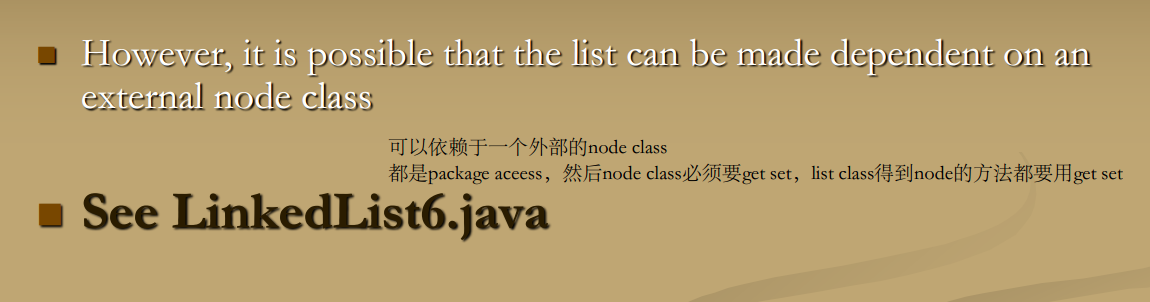
**private** Car cr;基础data

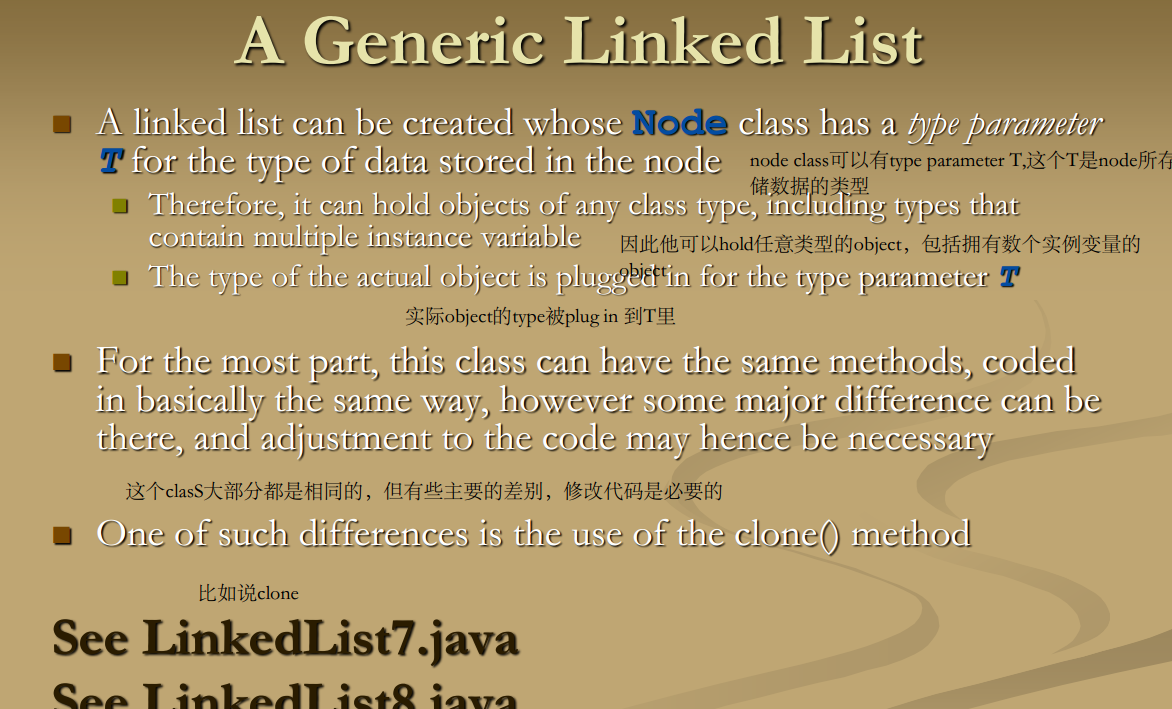
**private** Node next;next link





外界可以直接提取他的cr系数，不用担心privacy leak





**标准的copy**

**public** List(List<T> cl)

{

**if**(cl == **null**) **throw** **new** NullPointerException();

**if** (cl.head == **null**)

head = **null**;

**else**

{

// Call our copyList() method to copy the list

head = copyList(cl.head);

}

}

**public** List<T> clone()

{

**try**

{

List<T> newCL = (List<T>)**super**.clone();

**if** (head == **null**)

newCL.head = **null**;

**else**

// Call our copyList() method to copy the list

newCL.head = copyList(head);

**return** newCL;

}

**catch** (CloneNotSupportedException e)

{ //This should not happen

**return** **null**;

}

}

// Given a Node pointer that is not null, this method create and return

// a deep copy of this list pointed by this pointer

**private** Node<T> copyList(Node<T> cn1)

{

Node<T> temp = cn1;

Node<T> newHead = **null**;

Node<T> end = **null**; // This pointer will always point at the end of the new list

// while it is being created (growing)

newHead = **new** Node<T>((T)(temp.x).clone(), **null**);

end = newHead;

temp = temp.next;

**while** (temp != **null**)

{

end.next = **new** Node<T>((T)(temp.x).clone(), **null**);

end = end.next;

temp = temp.next;

}

// Now the entire list is created, just return its head pointer

**return** newHead;

}

标准过程

**class** ExitListener **implements** ActionListener

{

**public** **void** actionPerformed(ActionEvent e)

{

System.*exit*(0);

}

}

**public** **class** Swing1{

**public** **static** **final** **int** *BUTTON\_WIDTH* = 300;

**public** **static** **final** **int** *BUTTON\_HIGHT* = 200;

**public** **static** **void** main(String[] args)

{

JFrame myWindow = **new** JFrame("Welcome to Swing");

程序命名

myWindow.setSize(*BUTTON\_WIDTH*, *BUTTON\_HIGHT*);设置了size

// Disable the default close button of the frame (this is the X button usually located

// on the top right corner of the frame)

myWindow.setDefaultCloseOperation(JFrame.*DO\_NOTHING\_ON\_CLOSE*);//加了这一行以后，单击X不会关掉窗口

JButton exitButton = **new** JButton("Click Here to Terminate the Program.");

创造一个JBUTTON

exitButton.setBackground(**new** Color(200,150,150));

设置背景颜色

// Now create the listener, which will actually handle the event

ExitListener endLsnr = **new** ExitListener();

设置单机exitbutton以后的listener，这个是exitlistener所以会exit

// Relate the button to that listener

exitButton.addActionListener(endLsnr);

对button增加这个listener

// Add the button to the frame

myWindow.add(exitButton);

把这个button加入mywindow

// Finally, show the frame after everything is ready

myWindow.setVisible(**true**);//让他可视化，如果不加这一行，不会出现窗口



