بنى المعطيات

أنواع البيانات :

١. **النوع الأساسي :** (الذري , الأولي) وهي البيانات التي لا يمكن تحليلها الى مكونات أخرى من البيانات مثل ..... , integer , double , float , Boolean
٢. **النوع الفرعي :** وهي البيانات التي يمكن تحليلها الى مكونات أخرى من البيانات مثل الأغراض objects .

العمليات على البيانات :

كل نوع من البيانات يملك مجموعة من العمليات التي يمكن ان تنفذ عليه مثل integer يمكن القيام بعمليات الجمع , الطرح , الضرب ...

أيضا كل غرض object يملك مجموعة من الوظائف (العمليات) الخاصة به .

بنى المعطيات :

هي طريقة لتنظيم البيانات في الذاكرة وفق علامات محددة وعمليات محددة يمكن إنجازها على هذا التنظيم بحيث يكون استخدامها اكثر كفاءة وفعالية .

من اهم بنى المعطيات المستخدمة :

١. المصفوفات Arrays
٢. القوائم المرتبطة Linked lists
٣. المكدسات Stacks
٤. الارتال Queues
٥. الجداول Tables
٦. الأشجار Trees
٧. البيان (المخططات) Graphs

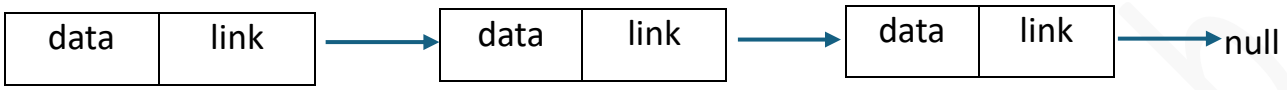
العمليات الأساسية على بنى المعطيات :

١. الإضافة لعنصر
٢. حذف عنصر
٣. الوصول الى عنصر
٤. بعض العمليات الخاصة بكل بنية تميزها عن غيرها من البنى

اولاً : القوائم المترابطة Linked lists

القوائم المترابطة هي احد اهم بنى المعطيات والتي غالبا ما تستخدم لبناء بنى معطبات أخرى ويمكن تعريفها بأنها :

سلسلة من العقد nodes حيث كل عقدة node تخزن بياناتها الخاصة مع مؤشر (عنوان) لموقع العقدة التالية أي ان العقد ترتبط ببعضها البعض لتشكل سلسلة عقد مترابطة.



العقدة الأخيرة في القائمة تشير الى null (نهاية السلسلة)

ملاحظة(١) :

على الرغم من التشابه الكبير بين المصفوفات arrays و القوائم المترابطة الا انه

١. القوائم المترابطة غير مقيدة الحجم كما في المصفوفة.
٢. عناصر القوائم المترابطة غير مفهرسة كما في المصفوفة أي يمكن حشر عنصرين عنصرين بينما المصفوفة لا يمكن ذلك . ويمكن ايضاً حذف عنصر من القائمة (الحذف والحشر بالبنية وليست بقيمة البيانات داخل البنية).
٣. **عيوب القوائم المترابطة :**

١. لا يمكن الوصول العشوائي الى عناصر القائمة ويجب في كل مرة البدء من اول عنصر للوصول الى العنصر المطلوب .

وبالتالي لا يمكن استخدامها في (البحث الثنائي).

٢. تتطلب حجم إضافي من الذاكرة من اجل تخزين العناوين المترابطة للقائمة .

ملاحظة(٢) :

عادة ما تستخدم القوائم المترابطة لبناء بنى المعطيات الأخرى مثل المكدسات والارتال والأشجار والبيان.

بناء قائمة مترابطة باستخدام java

كما ذكرنا كل عقدة (node) من القائمة تملك مؤشر (عنوان) يشير الى العقدة التالية والعقدة الأخيرة تشير الى null

نبدأ ببناء الصف linked list والذي يتضمن الصف الداخلي الساكن node الذي يمثل العقدة والمؤلف من عنصرين بيانيان وهما value تمثل البيانات المخزنة في العقدة و next تمثل المؤشر الذي سوف يشير الى العقدة التالية بالإضافة الى الصف الداخلي node يملك الصف linked list عضو بياني من نوع node اسمه first يدل على العنصر الأول في القائمة ويملك ايضاً الطرق method او العمليات التالية :

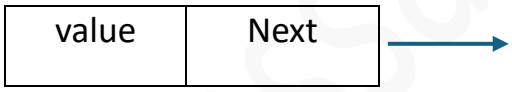
١. طريقة إضافة عنصر من الامام
٢. طريقة إضافة عنصر من الخلف
٣. طريقة حذف عنصر من الامام
٤. طريقة حذف عنصر من الخلف
٥. طريقة طباعة القائمة
٦. طريقة main()

وبالتالي تكون الشيفرة للبرنامج بالشكل التالي :

١) الصف الداخلي Node :

```
Static class node {
Int value ;
Node next ;
Public node (int v ) {
Value = v ;
  }
}
```

يمثل هذا الصف عقدة تملك قيمة صحيحة ومؤشر يشير الى العقدة التالية (يخزن فيه عنوان العقدة التالية)



٢) الخاصية First :

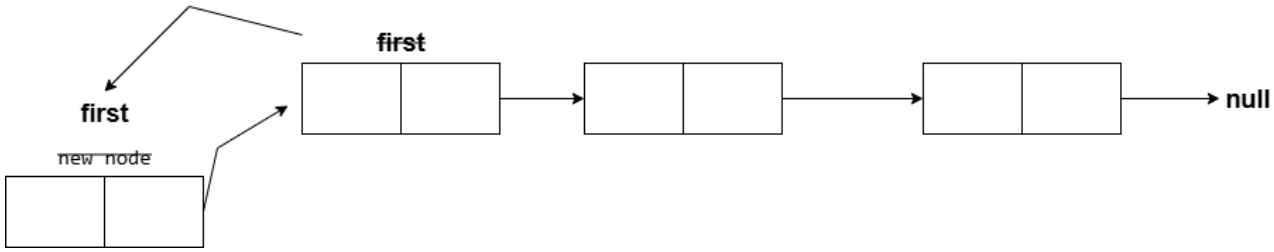
احدى خصائص الصف الخارجي linked list والذي يعرف عقدة تمثل (تشير) الى العقدة الأولى في القائمة .

٣) الدالة AddAtFront :

هذه الدالة مهمتها إضافة عنصر من الامام للقائمة أي قبل العنصر الأول first في القائمة .

```
Public void AddAtFront(Node node) {
Node.next=first ;
First = node;
}
```

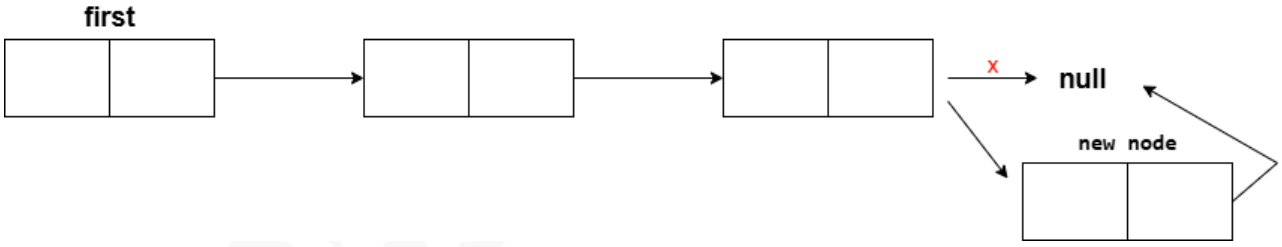
أي اسناد العقدة الأولى (المؤشر) للعقدة الجديدة والاشارة الى العقدة الجديدة على انها العقدة الأولى



٤) الدالة AddAtEnd :

تعمل هذه الدالة على إضافة العقدة الجديدة الممررة لها الى نهاية القائمة الموجودة .

```
Public void AddAtEnd (Node node) {
if( first == null)
First = node;
else{
node ptr =first ;
while(ptr.next != null )
ptr =ptr.next;
ptr.next = node;
  }
}
```

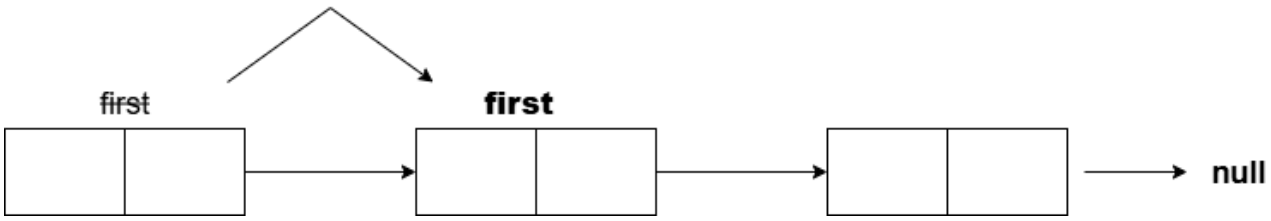


٥) الدالة removeFront :

تعمل هذه الدالة على حذف اول عنصر من القائمة .

```
Public void removeFront() {
first = first.next;
}
```

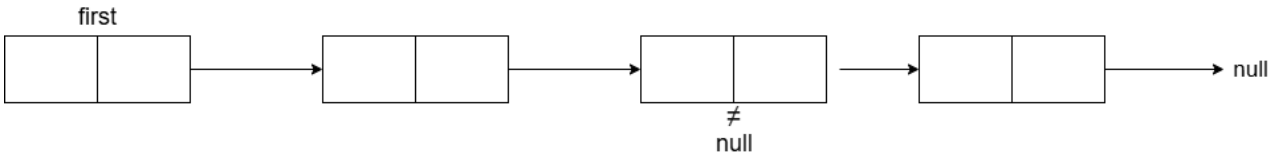
ويتم ذلك بتجاهل العنصر الأول وتسمية الذي يليه بالأول .



٦) الدالة removeEnd :

تعمل هذه الدالة على حذف العنصر الأخير من القائمة ويتم ذلك بالذهاب الى العنصر الأخير اولاً وتجاهله .

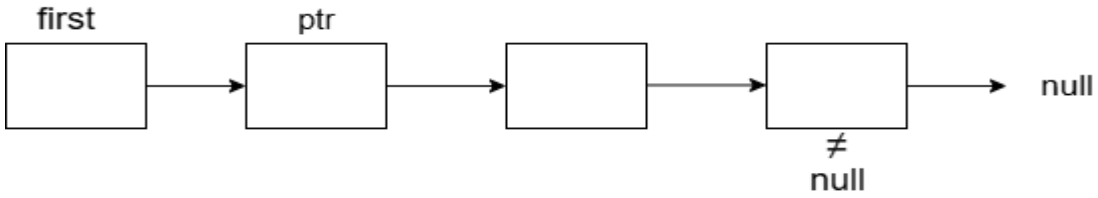
```
Public void removeEnd () {
if( first.next == null)
First = null;
else{
node ptr =first ;
while(ptr.next.next != null )
ptr =ptr.next;
ptr.next = null;
  }
}
```



Ptr =first

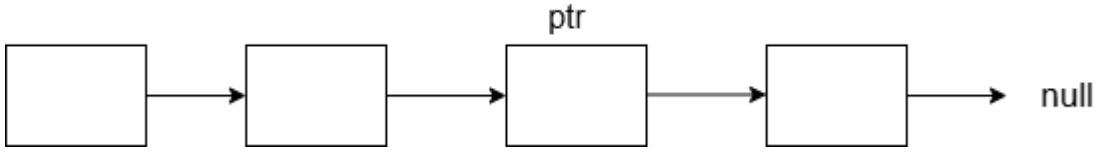
ptr.next.next ≠ null

ptr = ptr.next



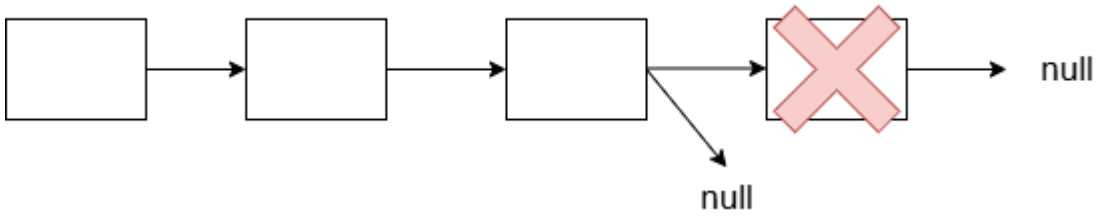
ptr.next.next ≠ null

ptr = ptr.next



ptr.next.next = null

ptr.next=null



٧) الدالة print() :

تعمل هذه الدالة على طباعة القائمة التي حصلنا عليها .

```
Public void print() {
Node ptr = first ;
While(ptr != null){
System.out.print(ptr.value + " ➔");
Ptr=ptr.next ;
  }
}
```

تعمل الدالة من خلال حلقة للمرور على كل عقدة وطباعة قيمتها حتى نهاية القائمة .

٨) الدالة main() :

تعمل الدالة الأساسية على توليد كائن (غرض) object من الصف linked list واستخدامه لبناء قائمة .

```
Public static void main(string [] args) {
Linkedlist L = new linkedlist ();
L.AddAtFront (new node(1));
L.AddAtFront (new node(2));
L.AddAtFront t(new node(3));
L.print();
L.removeFront();
L.print();
L.removeEnd();
L.print();
}
```

C:/Users/PCE1/Documents/NetBeansProjects/LinkedList/src/linkedlist/LinkedList.java

```
package linkedlist;
public class LinkedList {
    //.....................................................
    static class Node {
        int value;
        Node next;
        public Node(int v){
            value = v;
        }
    }
    //.....................................................
    Node first = null;
    //.....................................................
    public void addAtFront(Node node){
        node.next = first;
        first=node;
    }
    //.....................................................
    public void addAtEnd(Node node){
        if(first == null)
            first =node;
        else
        {
            Node ptr = first;
            while (ptr.next != null)
                ptr= ptr.next;
            ptr.next = node;
        }
    }
    //.....................................................
    public void removeFront(){
        first = first.next;
    }
    //.....................................................
    public void removeEnd(){
        if(first.next == null)
            first =null;
        else
        {
            Node ptr = first;
            while (ptr.next.next != null)
                ptr= ptr.next;
            ptr.next = null;
        }
```

1.1 of 2

2021.06.06 12:22:20

C:/Users/PCE1/Documents/NetBeansProjects/LinkedList/src/linkedlist/LinkedList.java

```
    }
    //.....................................................
    public void print(){
        Node ptr = first;
        while(ptr != null){
            System.out.print(ptr.value + "==>");
            ptr=ptr.next;
        }
        System.out.println();
    }
    //.....................................................
    public static void main(String[] args) {
        LinkedList L =new LinkedList ();
        L.addAtFront(new Node(1));
        L.addAtFront(new Node(2));
        L.addAtFront(new Node(3));
        L.addAtFront(new Node(4));
        L.addAtFront(new Node(5));
        L.print();
        L.removeFront();
        L.print();
        ////////////////////////////////////////////
        LinkedList L1 =new LinkedList ();
        L1.addAtEnd(new Node(1));
        L1.addAtEnd(new Node(2));
        L1.addAtEnd(new Node(3));
        L1.addAtEnd(new Node(4));
        L1.addAtEnd(new Node(5));
        L1.addAtEnd(new Node(6));
        L1.print();
        L1.removeEnd();
        L1.print();
    }
}
```

2.1 of 2

2021.06.06 12:22:20