```
ArrayList impl List..
int[] data;
final void clear() { size = 0; data = new int[32];}

private void ensureC(int newC){
if(newC < size) return;
int[] old = data;
data = new int[newC]; System.arraycopy(old,0,data,0,size);

@Override public String toString() {
StringBuilder s = new StringBuilder("");
for (int i = 0; i < size; i++) { s.append(data[i]).append(", "); }
// LINKEDL: for (Node p = head.next; p != null; p = p.next) { s.append(p.data).append(", ");} //
s.append("size = ").append(size); return s.toString(); }

Methde : void append(List l){int n = l.size();
for(int i= 0; i<n;i++){ this.add(l.get(i));}}
```

```
class Node{
private Node next;private int data;
konstruktor kommt hier
Node head=new Node(3,null)
Liste durchlaufen:for(Node p=head,p!=null;p=p.next)
Node fuegen:p.next = new Node(5,p.next);
Node loeschen:p.next=p.next.next;
Einfuegen sortiert:
1.leere oder listenanfang OHNE Hilfsknoten (if else) MIT Hilfsknoten nur (else)
if (head==null || x <= head.data) { head =new Node(x,head);}
2.allgemein
else { Node p = head;
while(p.next != null && p.next.data < x) { p = p.next;}
p.next =new Node(x,p.next);
```

```
void add(int idx, int x) {
if (idx < 0 || idx > size) throw new IndexOutOfBoundsException();
Node p = head;
for (int i = 0; i < idx; i++){ p = p.next;}
 p.next = new Node(x,p.next); size++; }

int get(int idx) { wie oben Aber ohen letzte zeile und am ende return p.data; }

void removeElem(int x) wie oben throw new IndexOutOfBoundsException();
Node p = head;
while (o.next!=null && p.next.data!=x) { p = p.next;}
if(p.next!=null){p.next = p.next.next\\loeschen; size--; }}

void removeAlleMehrfachvorkommen(int x) {
Node p = head;
whle(p.next!=null){if(p.next.data==x)loeschen;size--; else p = p.next; }}

void append(Liste l){Node p = head;
while(p.next!=null){p=p.next}
n=l.size();
for(i=0;i<n;i++){p.next=new Node(l.get(i),null);p=p.next;size++;}}

Doppelt Verkettete :
Loeschen : p.prev.next=p.next;p.next.prev=p.prev;
Einfuegen von r NACH q: Node r =new Node(x,q.next,q) ; r.next.prev=r;q.next=r
Einfuegen von r VOR q: Node r =new Node(x,q,q.prev) ; r.prev.next=r;q.prev=r
```

```
ArrayStack implements Stack
public ArrayStack() { size = 0; data = new int [32];}
```

```
public void push(int x) { if (data.length == size) ensureC(2*size);
        data[size++] = x; }
public int top() { {if(size == 0) throw new EmptyStackException();}
return data[size-1]; }
public int pop() { {throw wie oben}
return data[--size];  }

LinkedStack imolements Stack
    public
public LinkedStack() { top = null;}
public void push(int x) { top = new Node(x,top); }
public int top() { if (top == null) {throw new EmptyStackException();}
return top.data;  }
public int pop() { throw wie oben;
int x = top.data; top = top.next; return x; }
-----
ArrayQueue implements Queue
public ArrayQueue() {
size = 0; back = DEF_CAPACITY-1; front = 0; data = new int[32]; }
void add(int x) { if (data.length == size) {ereC(2*size);}
 back = (back + 1) % data.length; data[back] = x; size++; }
int front() { if (size == 0) throw new NoSuchElementException();
return data[front]; }
 public int remove() { throw wie oben
int x = data[front];
front = (front + 1) % data.length; size--; return x; }

LinkedQueue implements Queue
public
LinkedQueue(){front = back = null;}
static private class Node { private int data; private Node next; private Node(Node p, int x)
 { data = x;next = p; } }
 void add(int x) if (front == null) front = back = new Node(x,null);
else {  back.next = new Node(x,null); back = back.next; }
 public int front() { if (front == null) throw new NoSuchElementException(); return front.data; }
 public int remove() { throw wie oben
int x = front.data;
 front = front.next;
if (front == null) {back = null; return x;} }
-------------------------------------------------------
class Dir{String name;LinkedList<Dir> dir;
...konstruktor
    public void mkdir(String n) {dir.add(new Dir(n));}
    public Dir cd(Str dir){for(Dir f:dir) {if(f.name.equals(n) return f;} return nul)}}
-----------------------------
 BinarySearch : ein Kind ermitteln :
 public int getN(){return getN1R(root)}
 private int getN1R(Node p){
    if p == null return 0;
else {int s = 0 ; if(left=null&&right!=null)||(andersrum) s++; return s +getN(left)+...right}
 }
 -----
 ITERATOR :
 implements Iterable<..>
 public Iterator iterator(){
    return new Iterator<..>(){
        private Node curr = head;
        public boolean hasNext() {return curr != null}
        public String next(){Str s = curr.nameM curr = cuu.next; retrun s;}
        public void remove(){}};}
-----------------------------
-----------------------------
```

```
***UNdirekt 1-1:
class Parkplatz {
private String name;
Konstruktor und getName;
private Auto meinAuto;
 void setAuto(Auto a) { if (a.parkt()) return;
if {(meinAuto != null) // Parkplatz ist bereits besetzt return;}
meinAuto = a; a.parke(true);}
 void removeAuto() { {if (meinAuto == null) // kein Auto park return;}
meinAuto.parke(false);
meinAuto = null; }
publicAuto getAuto() { return meinAuto;} }

 public class Auto {
private String name;
Konsturkot und getName;
private boolean parkt = false;
public boolean parkt() { return this.parkt; }
public void parke(boolean b) { parkt = b; }

**Bidirekt 1-1
class Parkplatz ...private Auto meinAuto;
public void setAuto(Auto a) { if (meinAuto != null)  return
if (a.getParkplatz() != null && a.getParkplatz() != this) return
meinAuto = a; a.setParkplatz(this);
void removeAuto(){ if (meinAuto == null) return
Auto a = meinAuto; meinAuto = null; a.removeParkplatz()

class Auto . private Parkplatz meinParkplatz
void setParkplatz(Parkplatz p) { if (meinParkplatz != null) return; // Auto parkt bereits
if (p.getAuto() != null && p.getAuto() != this) return;// Parkplatz von anderem Auto besetzt
meinParkplatz = p; p.setAuto(this)
void removeParkplatz() { if (meinParkplatz == null) eturn; // Auto parkt nicht
Parkplatz p = meinParkplatz; meinParkplatz = null; p.removeAuto();
.....................
**Undir 1-n
class Parkhaus { // ...
private List<Auto> meineAut  = new LinkedList<Auto>();
public void addAuto(Auto a) { if (a.parkt()) return; // Auto parkt bereits
meineAutos.add(a); a.parke(true);}
public void rmAu(Auto a) { if (meineAut.remove(a)) // Auto hat hier geparkt a.parke(false); }
public List<Auto> getAuto() return meineAutos

public class Auto { // ...
private boolean parkt = false;
public boolean parkt() {return this.parkt;}
public void parke(boolean b) {parkt = b;} }

**Bidirekt 1-n
class Parkhaus..
private List<Auto> meineAutos = new LinkedList<Auto>();
public void addAuto(Auto a) { if (meineAutos.contains(a)) return
if (a.getParkhaus() != null && a.getParkhaus() != this) retur
meineAutos.add(a); a.setParkhaus(this);}
public void
removeAuto(Auto a) { if (!meineAutos.contains(a)) return
meineAutos.remove(a); a.removeParkhaus(); }
public List<Auto> getAuto() { return meineAutos;

class Auto ...
private Parkhaus meinParkhaus;
public void setParkhaus(Parkhaus p) {if (meinParkhaus != null) return
```

```
meinParkhaus = p; p.addAuto(this);
void removeParkhaus() { if (meinParkhaus == null) return
Parkhaus p = meinParkhaus; meinParkhaus = null; p.removeAuto(this);
pubic Parkhaus getParkhaus() { return meinParkhaus;
```

```
**Bidirekt n-m
 public class Student { // ..
private List<Vorlesung> meineVor  = new LinkedList<Vorlesung>();
public void addVorlesung(Vorlesung v) {
if (!meineVorl.contains(v)) { meineVorl.add(v); v.addStudent(this); } }
public void removeVor(Vorlesung v) { if (meineVorl.remove(v)) v.removeSt(this); }
publicList<Vorlesung> getVorlesungen() { return meineVorl; }

blic class Vorlesung { // ..
private List<Student> meineSt  = new LinkedList<Student>()
public void addStudent(Student s){if(!meineSt.contains(s)){meineSt.add(s); s.addVorlesung(this);}}
public void removeStudent(Student s) { if (meineSt.remove(s)) s.removeVorlesung(this); }
public List<Student> getStudenten() { return meineStudenten;
```

```
HauufigstesWort :
TreeMap<String,Integer> h = new ....; auf null pruefen
for(String w: wortliste){if(!h.containsKey(w)) h.put(w,1) else h-put(w,h.get(w)+1)}
int max = 0 ; String l;
for(Map.Entry<St,Inte> e : h.entrySet()){if(e.getValue() > max) l = e.getKey();
max = e.getValue();} return l;

Liste add woerter sortiert klausur 13/14:
if(head == null)head = newNode(null, w);
elseif (w.compareTo(head.wort) < 0)head = newNode(head, w);
elseif (w.equals(head.wort))head.h++;
else {Node p = head;
while(p.next != null&& w.compareTo(p.next.wort) >0)p = p.next;
if(p.next != null&& w.equals(p.next.wort))p.next.h++;
elsep.next = new Node(p.next, w)

class Spiel implements Comparable<Spiel>{ public int compareTo(Spiel s) {
if(this.getAnzSp() < s.getAnzSp()) return - 1; else if == return 0; else return 1; }
```

MESSEN in Worstcase :
Suchbaum $n \log n$
$O(n \log n)$ $n = 1000; 2msec \rightarrow \frac{T(10^6)}{T(10^3)} = \frac{10^6 * \log 10^6}{10^3 * \log 10^3} = 2000\ dann * msec$
$O(n^2)$ $T(10^6) = T(10^4) * (10^2)^2$
PRIMZAHL Bestcase : n ist gerade $\rightarrow T(n) = O(1)$ konstante
Worstcase : n ist prim $\rightarrow T(n) = O(\sqrt{n})$
MAX linear $T(n) = O(n)$

```
int max(int[] a) {int n = a[0]; for(i=1;i<n;i++) if(a[i]>n) n=a[i]; return n; }
```

Sortieren durch Einfügen insertionSort: bestcase $O(n)$ worstcae n^2
Sortieren dur Auswählen selectionSort: quadratisch n^2
durch Vertauschen also bubble sort : wie durch Einfügen
Quicksort : $n \log n$ best; n^2 worst
Mergesort : $n \log n$
Binärsuchbaum : Worst-Case : $O(n)$; Average : $\log n$

PreOrder wurzel links rechts; PostOrder : links rechts wurzel; inOrder :links wurzel rechts