```
*** UNdirekt1-1:
        c l a s s  Parkplatz        {
private  String   name ;
Konstruktor      und      getName ;
private  Auto     meinAuto ;
void     setAuto ( Auto   a ){i f( a . parkt ( ) )return ;
i f {( meinAuto !=  null )//Parkplatz ist  b e r e i t s  b ese tzt return ;}
meinAuto = a ;   a . parke ( true ) ; }
void     removeAuto ()  {        { i f       ( meinAuto ==   n u l l )  //         kein      Auto      park  return ;}
meinAuto . parke ( f a l s e ) ;
meinAuto =       n u l l ;  }
publicAuto       getAuto ()       {          return    meinAuto ;}       }
        c l a s s Auto     {
private  String   name ;
Konsturkot       und      getName ;
private  booleanparkt     = f a l s e ;
public   booleanparkt () {        return    t h i s . parkt ;    }
public   void     parke ( boolean b)       {          parkt = b ;         }
** Bidirekt       1-1
        c l a s s  Parkplatz        . . . private  Auto meinAuto ;
public   void     setAuto ( Auto a )                 {          i f         ( meinAuto        !=         n u l l )
        return
i f       ( a . getParkplatz ()       !=        n u l l    && a . getParkplatz ()  !=          t h i s )  return
meinAuto = a ;   a . setParkplatz ( t h i s ) ;
void     removeAuto (){  i f         ( meinAuto ==   n u l l )  return
Auto a = meinAuto ;      meinAuto =       n u l l ;  a . removeParkplatz ()
        c l a s s Auto     .          private  Parkplatz          meinParkplatz
void     setParkplatz ( Parkplatz          p)        {          i f         ( meinParkplatz           !=         n u l l )
return ; // Auto   parkt  b e r e i t s
i f (p . getAuto () != n u l l && p . getAuto () != t h i s )       return ;// Parkplatz von   anderem Auto  be set zt
meinParkplatz   = p ;     p . setAuto ( t h i s )
void     removeParkplatz ()       {i f( meinParkplatz ==     n u l l )  eturn ;   // Auto   parkt     nicht
Parkplatz  p = meinParkplatz ;   meinParkplatz   = n u l l ;          p . removeAuto ( ) ;
-------------------------------------------------------------------------------------------------------------------------------
** Undir 1-n
        c l a s s  Parkhaus        {  //  . . .
private  List <Auto> meineAut              = new   LinkedList<Auto >();
public   void     addAuto ( Auto a ){i f( a . parkt ( ) ) return ;         //Auto  parkt      b e r e i t s
meineAutos . add ( a ) ;  a . parke ( true ) ; }
public   void     rmAu( Auto a ){i f( meineAut . remove ( a ))//Auto hat  h i e r geparkt a . parke ( false);}
public   List <Auto> getAuto ()            return    meineAutos
        c l a s s Auto {   //  . . .
private  booleanparkt     = f a l s e ;
public   booleanparkt () { return  t h i s . parkt ;}
public   void     parke ( boolean b)       { parkt   = b ;}    }
** Bidirekt       1-n
c l a s s  Parkhaus . .
private  List <Auto> meineAutos = new   LinkedList<Auto >();
public   void     addAuto ( Auto a )                 {          i f         ( meineAutos . contains ( a ))     return
i f       ( a . getParkhaus ()       !=        n u l l    && a . getParkhaus ()    !=         t h i s )  return
meineAutos . add ( a ) ;  a . setParkhaus ( t h i s ) ; }
public   void removeAuto ( Auto a )       {          i f         ( ! meineAutos . contains ( a ))   return
meineAutos . remove ( a ) ;  a . removeParkhaus ( ) ;  }
public   List <Auto> getAuto ()            {          return  meineAutos ;
c l a s s     Auto  . . .
private  Parkhaus  meinParkhaus ;
public   void     setParkhaus ( Parkhaus p)        { i f       ( meinParkhaus !=         n u l l )  return
meinParkhaus = p ;  p . addAuto ( t h i s ) ;
void     removeParkhaus () {  i f( meinParkhaus == n u l l )  return
Parkhaus p = meinParkhaus ;    meinParkhaus =           n u l l ;  p . removeAuto ( t h i s ) ;
pubic  Parkhaus  getParkhaus () {  return  meinParkhaus ;
```

** Bidirekt n-m
c l a s s Student { // . .
private List <Vorlesung> meineVor = new LinkedList<Vorlesung >();
public void addVorlesung (Vorlesung v) {
i f (! meineVorl . contains (v)) { meineVorl . add (v) ; v . addStudent (t h i s) ; } }
public void removeVor (Vorlesung v) { i f (meineVorl . remove (v)) v . removeSt (t h i s) ; }
public List <Vorlesung> getVorlesungen () { return meineVorl ; }
c l a s s Vorlesung { // . .
private List <Student> meineSt = new LinkedList<Student >()
public void addStudent (Student s){ i f (! meineSt . contains (s)){ meineSt . add (s) ; s . addVorlesung (t h i s) ; }
public void removeStudent (Student s) { i f (meineSt . remove (s))s . removeVorlesung (t h i s) ; }
public List <Student> getStudenten () { return meineStudenten ;

HauufigstesWort :
TreeMap<String , Integer > h = new . . . . ; auf n u l l pruefen h-put (w, h . get (w)+1)}
f o r (String w: w o r t l i s t e){ i f (! h . containsKey (w)) h . put (w, 1) e l s e
i n t max = 0 ; String l ;
f o r (Map . Entry<St , Inte> e : h . entrySet ( )) { i f (e . getValue () > max) l = e . getKey ( ) ;
max = e . getValue ( ) ; }return l ;

L i s t e add woerter s o r t i e r t klausur 13/14:
i f (head == n u l l) head = newNode(null , w) ;
else if (w. compareTo (head . wort) < 0) head = newNode(head , w) ;
else if (w. equals (head . wort)) head . h++;
else {Node p = head ;
while (p . next != n u l l&& w. compareTo (p . next . wort) >0)p = p . next ;
if (p . next != n u l l&& w. equals (p . next . wort)) p . next . h++;
else p . next = new Node(p . next , w)

MESSEN in Worstcase :
n log n -> (n*vergrößerung)*log(n*vergrößerung) *msec durch n log n
n^2 -> (vergrößerung^2)*msec

MEHRFACHES löschen : Node p = head.n; while(p.n != null) if (p.data == p.n.d) p.n = p.n.n; else p = p.n;

Doppelt verkettete : löschen p.prev.next = p.next; p.next.prev = p.prev;
Einfügen von r NACH q : Node r = new Node(x,q.next,q);r.next.prev=r;q.next=r;
Einfügen von r VOR q : Node r = new Node(x,q,q.prev);r.prev.next=r;q.prev=r;
Einfügen nach head : Node r = new Node();r.data = 5;r.next = head.next;r.prev = head;
head.next = r;r.next.prev = r;

---

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

---

k <: S <: Comparable<S> <: Comparable<? super S> <: Comparable<? super K> For die Sort methode

---

kleinset Elem in Binary : if (p == null) return null;if (p.left == null)p = p.right;
else p.left = delMinR(p.left);return p; GRÖßte -> p.right p.left vertauschen

---

Anzahl Blätter Binär private static int numberOfLeaves(Node p) { if (p == null) return 0; else if (p.next == null && p.right == null) return 1; else return numberOfLeaves(p.left)+numberOfLeaves(p.right);

INSERT Sortiert Aufsteigend :Node p = head;while (p.next != null && p.next.data < x)
p = p.next;p.next = new Node(p.next,x);

MAX löschen if (head.next == null)throw new NullPointerException();
Node p = head;while (p.next.next != null)p = p.next;
int r = p.next.data;p.next = null;return r;

Falls A <: B, dann folgt
C<? extends A> <: C<? extends B> (Kovarianz) und
C<? super B> <: C<? extends A> (Kontravarianz)

Contains absteigend Sortiert Verkette :
Node p = head.next;while (p!= null && p.d >x)p = p.next;
if(p==null || p.d < x) false;else true;

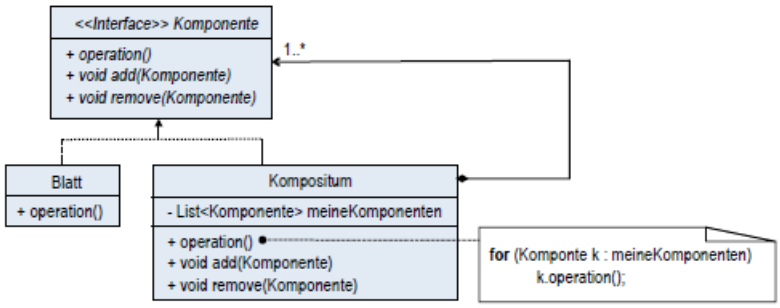
---

remove : gleich while wie oben;if(p != null && p.d==x) p=p.next;

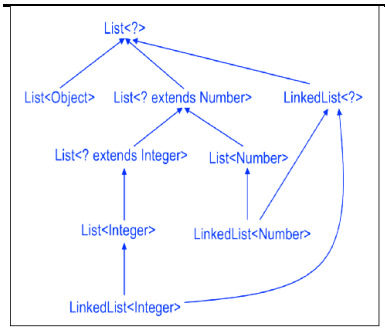
LISTE und MAP zusammen :Map<String, Integer> anzVorlJeDozent = new TreeMap<String, Integer>();
for (LV lv : lvList) { if (!anzVorlJeDozent.containsKey(lv.dozent))
anzVorlJeDozent.put(lv.dozent,1);else anzVorlJeDozent.put(lv.dozent, anzVorlJeDozent.get(lv.dozent)+1);}
List<String> flDoz = new LinkedList<String>();; for (String doz : anzVorlJeDozent.keySet())
if (anzVorlJeDozent.get(doz) > 3) flDoz.add(doz); return flDoz;

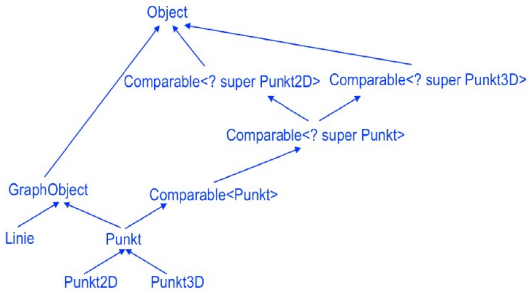
ADD und GET Liste : public List<Integer> getList() {
List<Integer> l = new LinkedList<Integer>();addListR(l, root);return l;}
private void addListR(List<Integer> l, Node p) {
if (p != null) {addListR(l, p.left);l.add(p.data);addListR(l, p.right);}

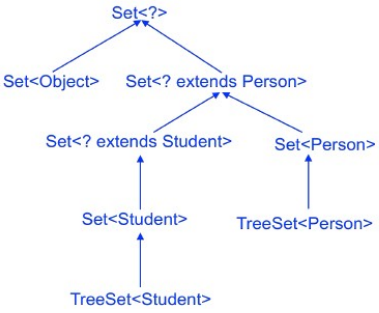
Binär REKURSIV:
-public boolean contains(int x) {return containsR(x,root);}
-private boolean containsR(int x, Node p) {if (p == null)return false;
else if (x < p.data)return containsR(x, p.left);
else if (x > p.data)return containsR(x, p.right);
elsereturn true;
-public void insert(int x) {root = insertR(x,root);}
-private Node insertR(int x, Node p) {if (p == null)p = new Node(x);
else if (x < p.data)p.left = insertR(x,p.left);
else if (x > p.data)p.right = insertR(x,p.right);return p;}
private int getMin(Node p) {assert (p != null);while(p.left != null)p = p.left;return p.data;



Dictionary:
Map<s,Set<s>> m;
insert(string dt, string engl){if(m.get(dt)==null) m.put(dt,new TreeSet<s>()) else m.get(dt).add(engl)}
get(string s){m.get(s)}
search(string s){List<string> n; for(String d : m.keySet()) if(d.contains(s)) n.add(d) return n;}








```
public boolean equals(BinarySearchTree bst) {
   if (bst == null)
      return false;
   else
      return equalsR(root, bst.root);
}

private static boolean equalsR(Node p, Node q) {
   if (p == null && q == null)
      return true;
   else if (p == null && q != null)
      return false;
   else if (p != null && q == null)
      return false;
   else
      return p.data == q.data && equalsR(p.left,q.left)
                              && equalsR(p.right,q.right);
}
```

```
private void moveToEnd(Node p) {
     assert p != null && p.next != null;

     // Knoten ist schon am Ende:
     if (p.next == end) return;

     // p.next aushängen:
     Node q = p.next;
     p.next = p.next.next;

     // Am Ende einhängen:
     end.next = q;
     q.next = null;
     end = q;
}


public String get(int key) {
     Node p = begin;
     while (p.next != null && p.next.key != key)
          p = p.next;
     if (p.next == null) return null;
     moveToEnd(p);
     return end.value;
}


public void set(int key, String value) {

     Node p = begin;
     while (p.next != null && p.next.key != key)
          p = p.next;
     if (p.next == null) {
          // Neuer Knoten:
          end.next = new Node(key,value,null);
          end = end.next;
          if (size == MAXSIZE)
               begin.next = begin.next.next;
          else
               size++;
     }
     else {
          // Schluessel existiert bereits:
          moveToEnd(p);
          end.value = value;
     }

}
```