Algorithmen und Datenstrukturen 2 Institut für Praktische Informatik
Klausur am 01.02.2002 Gruppe Software
Univ.-Prof. Dr. Alois Ferscha

Zuname		HS	
Vorname		Sitzplatz	
Matr. Nr.		Punkte	
SKZ		Note	

Fügen Sie fehlende Teile in den folgenden Aufgaben in die umrandeten Kästen (Platzhalter) ein.

1. Operatoren

(12 = 6 * 2 Punkte)

Welche Ausgabe liefert folgendes Programmfragment?

```
void myMethod1() {
   int a = 2;
   System.out.println(a++);                    2
   System.out.println(--a);                    2
   System.out.println((++a + a-- - 1) / 2);    2
   System.out.println(a-- != 0 ? 2 : 0);       2
   System.out.println(a == 2 ? 0 : 2);         2
   System.out.println((a & 2) != 0 ? 0 : 2);   2
}
```

2. Explizite Typkonvertierung (cast) und implizite Typkonvertierung (Promotion) (6 = 3 * 2 Punkte)

Welche Zuweisungen sind erlaubt? Schreiben Sie in die Platzhalter „erlaubt“ oder „nicht erlaubt“

```
void myMethod2() {
   char a;
   short b;
   int c;
   float d;
   double e;

   d = a * (float) (b – d) + (int) d;      erlaubt

   b = a + d / (short) e;                  nicht erlaubt

   e = c * 3;                              erlaubt

}
```

3. Schleifen (10 Punkte)

Das Programmstück

```
class AsciiTabelle {

  public static void main(String[] args) {

    for (char a = ´A´; a <= ´Z´; a++) {

      if (a % 2 == 0)

        System.out.println("Zeichen " + a + " Code " +
               (int) a + " ist gerade");

      else

        System.out.println("Zeichen " + a + " Code " +
               (int) a + " ist ungerade");

    }

  }

}
```

liefert folgende Ausgabe

```
Zeichen A Code 65 ist ungerade
Zeichen B Code 66 ist gerade
...
Zeichen Z Code 90 ist gerade
```

Implementieren Sie dieses Programmstück, unter der Annahme, dass es kein for- und while-Schleifenkonstrukt gibt. Das einzige Schleifenkonstrukt, welche Ihnen zur Verfügung steht, sei das `do {...} while(...);` Konstrukt.

```
class AsciiTabelle {
  public static void main(String argv[])
  {
    char a = 'A';
    do {
      if (a % 2 == 0)
        System.out.println("Zeichen " + a + " Code " +
            (int) a + " ist gerade" );
      else
        System.out.println("Zeichen " + a + " Code " +
            (int) a + " ist ungerade" );
    } while (a++ < 'Z');
  }
}
```

4. Quersumme rekursiv (10 = 5 + 5 Punkte)

Das gegebene Programm soll die Quersumme (Ziffernsumme) *q* einer gegebenen positiven ganzen Zahl berechnen und diese zurückgeben.

Beispiel: `quersum(358) = 3 + 5 + 8 = 16`

Tip: Die Quersumme wird in diesem Algorithmus beginnend ab der Einerstelle berechnet.

```
public int quersum(int z) {
  if (z != 0)                    // Abbruchbedingung
  {
    int letzteZiffer = z % 10;
    return letzteZiffer + quersum(z / 10);
  }
  else
    return 0;
}
```

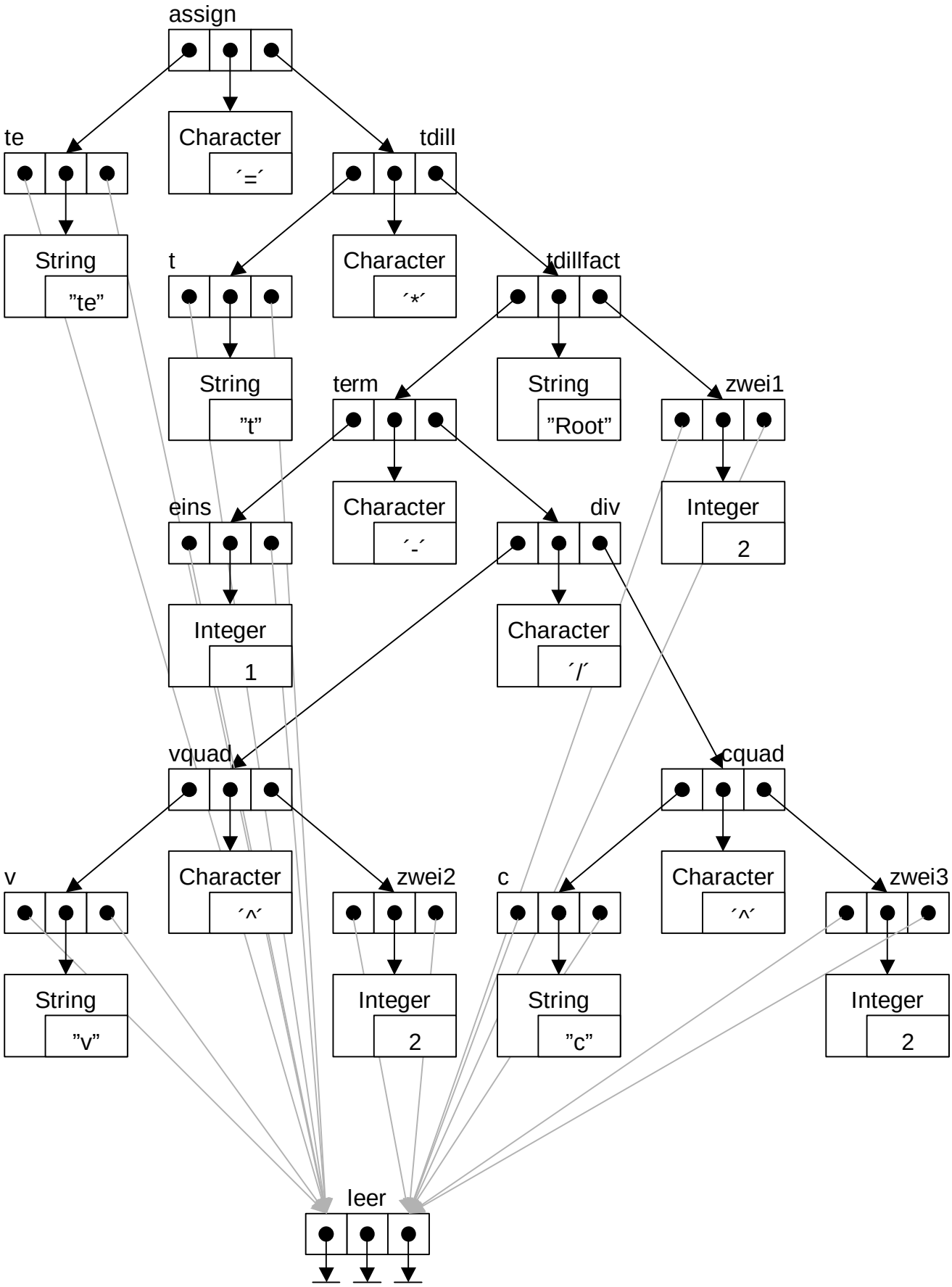
5. Konstruktoren und Verweise (20 Punkte)

Eine Klasse für Bäume von Objekten sei wie folgt festgelegt:

```
public class Baum {
  Object inhalt;
  Baum links, rechts;
  public final static Baum LEER= new Baum();

  public Baum() {
    inhalt= links= rechts= null;
  }

  public Baum(Object x) {
    this(LEER, x, LEER);
  }

  public Baum(Baum l, Object x, Baum r) {
    inhalt= x; links= l; rechts= r;
  }
}
```

Zeichnen Sie die Verweisstruktur, wie Sie durch folgende Aufrufe erzeugt wird. Verwenden Sie dazu die in der Vorlesung eingeführte Notation.

```
Baum zwei3  =  new Baum(new Integer(2));
Baum c      =  new Baum(new String(("c"));
Baum cquad  =  new Baum(c, new Character('^'), zwei3);
Baum zwei2  =  new Baum(new Integer(2));
Baum v      =  new Baum(new String(("v"));
Baum vquad  =  new Baum(v, new Character('^'), zwei2);
Baum div    =  new Baum(vquad, new Character('/'), cquad);
Baum eins   =  new Baum(new Integer(1));
Baum term   =  new Baum(eins, new Character('-'), div);
Baum zwei1  =  new Baum(new Integer(2));
Baum dillfact= new Baum(term, new String("Root"), zwei1);
Baum t      =  new Baum(new String("t"));
Baum tdill  =  new Baum(t, new Character('*'), dillfact);
Baum te     =  new Baum(new String("te"));
Baum assign =  new Baum(te, new Character('='), tdill);
```


assign
te
Character
´=´
tdill
String
”te”
t
Character
´*´
tdillfact
String
”t”
term
String
”Root”
zwei1
eins
Character
´-´
div
Integer
2
Integer
1
Character
´/´
vquad
cquad
v
Character
´^´
zwei2
c
Character
´^´
zwei3
String
”v”
Integer
2
String
”c”
Integer
2
leer

6. Dynamische Datenstrukturen (20 = 4 * 5 Punkte)

Doppelt verkettete Liste

Die Klasse Liste sei wie folgt implementiert. Erweitern Sie die Implementierung um eine Methode *remove*, welche ein Element (das erste, wenn mehrere dieser Elemente in der Liste enthalten sind) aus der Liste entfernt und die Liste wieder konsistent verkettet. Die Methode *remove* gibt *true* zurück, wenn das Element gelöscht wurde und *false*, falls das Element, da nicht in der Liste, nicht gelöscht wurde. (Beachten Sie die beiden Sonderfälle: Erstes Element löschen bzw. letztes Element löschen.)

```
public class List {

  private class Node {
    Object element;
    Node next;
    Node prev;
  }

  private Node head = null;
  private Node tail = null;

  public List()
  {
    head = null;
    tail = null;
  }

  public boolean empty()
  {
     return head == null;
  }

  public boolean insertTail(Object object)
  {
    Node n = new Node();
    n.element = object;
    n.next = null ;
    n.prev = tail;

    if (tail != null)
      tail.next = n;

    if (head == null)
      head = tail;

    tail = n;
  }
```

```
public boolean remove(Object x)
  {
    Node n;

    for (n = head; n != null; n = n.next) {
      if (n.element.equals(x)) {
        if (n != tail)
          n.next.prev = n.prev;
        else
          tail = n.prev;

        if (n != head)
          n.prev.next = n.next;
        else
          head = n.next;
        return true;
      }
    }
    return false;
}
```

7. Wrapper (8 = 4 * 2 Punkte)

Gegeben ist die doppelt verkettete Liste von Beispiel 6 und weiters die Methode

```
public class List {
  .
  .
  Object getLast() {
    If (tail != null)
      return tail.element;
    else
      return null;
  }
  .
  .
}
```

Implementieren Sie in der folgenden Klasse die beiden Methoden push() und pop(), sodass diese Klasse die Funktionalität eines Stack (Stapelspeicher = last in – first out (lifo)) bekommt:

push() fügt ein Objekt am Ende der Liste ein.

pop() holt das Objekt am Ende der Liste und löscht dieses aus der Liste.

```
class myStack {
  List l;
  myStack() {
    l = new List();
  }

  void push(Object o) {
    l.insertTail(o);
  }

  Object pop() {
    Object o;
    o = l.getLast();
    l.remove(o);    // Elemente in der Liste sind disjunkt
    return o;
  }
}
```

8. Abstrakte Klassen und Vererbung (24 = 3 * (5) + 5 + 2 * 2 Punkte)

Gegeben sei eine abstrakte Klasse *Geometrie*, die die abstrakte Methode *umfang*() zur Umfangsberechnung bereitstellt.

```
abstract class Geometrie {

  public abstract double umfang();

}
```

a) Implementieren Sie die erbenden Klassen (Konstruktor und Methoden) Rechteck, Kreis und Quadrat, sodass eine Verwendung im Sinne der Aufrufe unten möglich wird.

```
class Rechteck extends Geometrie {
  double laenge, breite;

  Rechteck (double laenge, breite) {
    this.laenge = laenge;
    this.breite = breite;
  }

  public double umfang(){
    return(2 * laenge + 2 * breite);
  }
}

class Kreis extends Geometrie {
  double radius;

  Kreis (double radius) {
    this.radius = radius;
  }

  public double flaeche(){
    return(2 * radius * 3.14159);
  }
}
```

```
class Quadrat extends Geometrie {
  double seite;

  Quadrat (double seite) {
    this.seite = seite;
  }

  public double flaeche(){
    return(4 * seite);
  }
}

...
Geometrie[] tabelle = new Geometrie[5];
tabelle[0] = new Kreis(1.0);
tabelle[1] = new Rechteck(3.0, 5.2);
tabelle[2] = new Quadrat(4.0);
tabelle[3] = new Kreis(0.0);
tabelle[4] = new Rechteck(1.0, 2.5);
double gesamtumfang = 0.0;
for (int i = 0; i < tabelle.length; i++) {
  gesamtumfang += tabelle[i].umfang();
}
System.out.println("Gesamtumfang:  " + gesamtumfang);
...
```

b) Welche Ausgabe produziert das obige Programm?

```
Gesamtumfang:     45,683
```

c) Von welchen Klassen würden Sie die beiden Geometrien **Raute** und **Parallelogramm** ableiten, um einen möglichstgroßen Teil (in Bezug auf die Umfangsberechnung) des schon bereitgestellten Quellcodes wiederzuverwenden.
Hinweis: Definition Raute: Vier gleichlange Seiten - rechter Winkel.
Definition Parallelogramm: Zwei mal zwei gleichlange Seiten - rechter Winkel.

```
Raute ableiten von:              Quadrat


Parallelogramm ableiten von:     Rechteck
```