Klausur aus Softwareentwicklung 1 Institut für Pervasive Computing

Klausur am 3.2.2005 Univ.-Prof. Dr. Alois Ferscha

Zuname		HS	
Vorname		Sitzplatz	
Matr. Nr.		Punkte	
SKZ		Note	

1.Mehrfachverzweigung (3 * 1 Punkte)

In einem `StringBuffer` sei eine Zeichenkette gespeichert.

Implementieren Sie eine Methode `static void replace(StringBuffer b)`, welche alle Vokale durch ein jeweils anderes Zeichen ersetzt.

Folgende Ersetzungstabelle soll gelten:

Vokal	a	e	i	o	u
Ersetzung	=	&	$	§	?

```
public static void replace(StringBuffer b) {
  for (int i = 0; i < b.length(); i++) {
    char replaceChar;

    switch(        b.charAt(i)        ) {

        case 'a':
          replaceChar = '=';
          break;
        case 'e':
          replaceChar = '&';
          break;
        case 'i':
          replaceChar = '$';
          break;
        case 'o':
          replaceChar = '§';
          break;
        case 'u'
          replaceChar = '?';
          break;


        default:
          replaceChar = b.charAt(i);
          break;
    }

    b.setCharAt(i, replaceChar)                    ;

  }
}
```

2. Ausdrücke (6 * 1 Punkte)

Ermitteln Sie die Ergebnisse folgender Ausdrücke und tragen Sie diese in die nebenstehenden Antwortkästchen ein.

```
class AusdruckTest {
  public static void main(String args[]) {
    int a = 2;
    int b = 3;

    System.out.println(a++ + a - --b);                                  // 2
    System.out.println(a | b);                                          // 3
    System.out.println(a & b);                                          // 2
    System.out.println(a / b + 1);                                      // 2
    System.out.println((int) (((float) a) / ((float) b) * 4.0));        // 6
    System.out.println((b * 2) % a);                                    // 1
  }
}
```

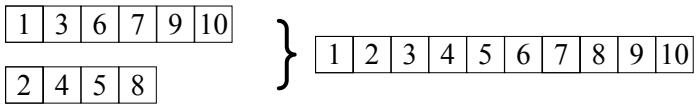
3. Iteration (4 * 1 Punkte)

Gegeben sind zwei aufsteigend sortierte Integer-Arrays `int[] a1` und `int[] a2`.

Implementieren Sie die Methode `int[] merge(int[] a1, int[] a2)`, welche die beiden Arrays zusammenführt und dieses neue, wieder aufsteigend sortierte Integer-Array zurück gibt.

Sie können davon ausgehen, dass die Zahlen die in `a1` gespeichert sind in `a2` nicht vorkommen und umgekehrt.

Beispiel:



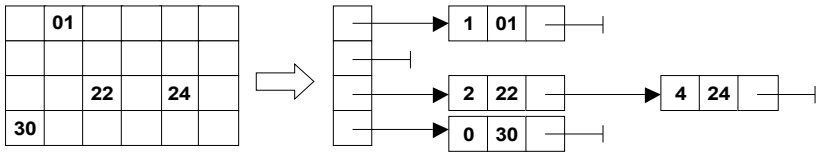

```
public static int[] merge(int[] a1, int[] a2)
{
  int[] merged = new int[a1.length + a2.length];
  int a1Index = 0;
  int a2Index = 0;

  for (int i = 0; [ i < a1.length + a2.length ]; i++) {

    if ((a2Index >= a2.length) || ((a1Index < a1.length) &&

        ( [ a1[a1Index] < a2[a2Index] ] ))) {

      merged[i] = [a1[a1Index]] ;

      a1Index++;
    } else {
      merged[i] = [a2[a2Index]] ;

      a2Index++;

    }
  }
  return merged;
}
```

4. Arrays (4 * 1 = 4 Punkte)

Gegeben sei eine große, dünn besetzte (nur wenige Elemente sind ungleich 0) Matrix `m` vom Datentyp `double[][]`.

Gesucht ist eine Methode `List[] convert(double[][] m)`, welche als Ergebnis ein Array von Listen liefert (für jede Zeile der Matrix eine lineare Liste), sodass in diesen Listen nur diejenigen Elemente der Matrix zusammen mit ihren Spaltennummern vorkommen die ungleich 0 sind.

Beispiel:




```
public class Node {
  int index;
  double value;
  Node next;

  public Node(int index, double value) {
    this.index = index;
    this.value = value;
    next = null;
  }
}

public class List {
  Node head;

  public List() {
    head = null;
  }

  public void append(Node n) {
    Node cur = head;
    if (cur != null) {
      while(cur.next != null) { cur = cur.next; }
      cur.next = n;
    } else {
      head = n;
    }
  }

  public static List[] convert(double[][] m) {
    List[] l = new List[m.length];

    for (int i = 0; i < l.length; i++) { l[i] = new List(); }

    for (int row = 0; row < m.length ; row++) {

      for (int col = 0; col < m[row].length ; col++) {

        if (m[row][col] != 0.0) {

          l[row].append(new Node( col , m[row][col] ));
        }
      }
    }
    return l;
  }
}
```

5.Listen (6 * 0.5 = 3 Punkte)

Gegeben ist eine Liste `List` welche ganze Zahlen verwaltet. Gesucht sind die beiden Methoden `int sumPositiveNumbers()` und `int sumNegativNumbers(). int sumPositiveNumbers()` summiert alle in der Liste vorkommenden positiven Zahlen und gibt das Ergebins zurück und `int sumNegativeNumbers()` summiert alle in der Liste vorkommenden negativen Zahlen und gibt ebenfalls das Ergebnis zurück.

Sie können davon ausgehen, dass alle anderen zum Aufbau der Liste notwendigen Methoden implementiert sind.

```
class Node {
  private int value;
  private Node next;

  public Node(Node prevNode, int value) {
    if (prevNode != null) {
      prevNode.next = this;
    }
    this.next = null;
    this.value = value;
  }

  public int getValue() {
    return value;
  }

  public Node getNext() {
    return next;
  }
}

public class List {
  Node head = null;

  public List() {...}

  public int sumPositiveNumbers() {
    int sum = 0;
    Node actNode;

    for (actNode = head; actNode != null; actNode = actNode.getNext() ) {
      if ( actNode.getValue() > 0 ) {
        sum += actNode.getValue() ;
      }
    }
    return sum;
  }

  public int sumNegativeNumbers() {
    int sum = 0;
    Node actNode;

    for (actNode = head; actNode != null; actNode = actNode.getNext() ) {
      if ( actNode.getValue() < 0 ) {
        sum += actNode.getValue() ;
      }
    }
    return sum;
  }
}
```

6. Vererbung (6 * 1 + 6 * 0.5 = 9 Punkte)

Regelmäßige geometrische Figuren sind solche, bei denen alle Seiten gleich lang sind. Beispiele dafür sind das gleichseitige Dreieck, das Quadrat und das regelmäßige Sechseck.

Leiten Sie von der abstrakten Klasse `Figur` drei Subklassen für die konkreten regelmäßigen Figuren gleichseitiges Dreieck, Quadrat und regelmäßiges Sechseck ab und implementieren Sie jeweils den Konstruktor und die beiden abstrakten Methoden `double flaeche()` und `double umfang()`. Beachten Sie, dass die Variable `a` in der Klasse Figur `private` ist, weshalb Sie nicht direkt aus den Subklassen auf diese Variable zugreifen können.

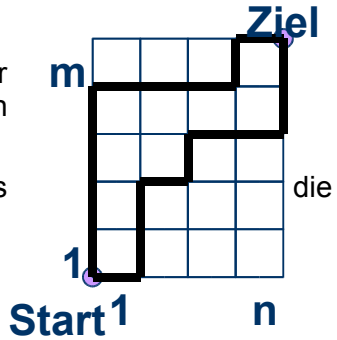
Figur	*Fläche*	*Umfang*
gleichseitiges Dreieck	$\frac{\sqrt{3}}{4} \cdot a^2$	$3 \cdot a$
Quadrat	a^2	$4 \cdot a$
regelmäßiges Sechseck	$\frac{3 \cdot \sqrt{3}}{2} a^2$	$6 \cdot a$

```
abstract class Figur {
  private double a;

  public Figur(double a) {
    this.a = a;
  }

  double getA() {
    return a;
  }

  abstract double flaeche();
  abstract double umfang();
}

class Quadrat extends Figur {
  public Quadrat(double a) {
    super(a);
  }

  public double flaeche() {
    return getA() * getA();
  }

  public double umfang() {
    return 4 * getA()
  }
}
```

```
class GleichseitigesDreieck [extends Figur] {
  public GleichseitigesDreieck(double a) {
    [super(a)] ;
  }
  public double flaeche() {
    [return Math.sqrt(3) * getA() * getA() / 4] ;
  }
  public double umfang() {
    [return 3 * getA()] ;
  }
}
class RegelmaessigesSechseck [extends Figur] {
  public RegelmaessigesSechseck(double a) {
    [super(a)] ;
  }
  public double flaeche() {
    [return 3 * Math.sqrt(3) * getA() * getA() / 2] ;
  }
  public double umfang() {
    [return 6 * getA()] ;
  }
}
```

7. Rekursion (5 * 1 = 5 Punkte)

Gegeben sei ein Feld von $m \times n$ Einheitsquadraten.



Gesucht ist die Anzahl der kürzesten Wege längs der Seiten der Einheitsquadrate, die von der linken unteren Ecke zur rechten oberen Ecke führen.

Vervollständigen Sie die nachfolgende rekursive Methode so, dass die Anzahl der kürzesten Wege ermittelt wird.

```
public static int numShortestWays(int m, int n) {
  if (m == 1 || n == 1) {
    return [n + m] ;
  }
  return numShortestWays( [m - 1] , [n] ) +
         numShortestWays( [m] , [n - 1] );
}
```

8.Exception (6 * 1 = 6 Punkte)

Die Methode `public static int teileGanzzahlig(int a, int b)` ist Teil einer Arithmetik-Klasse und soll die beiden übergebenen Argumente als Zähler und Nenner eines Bruchs a/b interpretieren und versuchen a ganzzahlig durch b zu dividieren.

Hierbei sind zwei Fehlerfälle zu unterscheiden:

1. Der Nenne b ist 0 (division by zero) und
2. a läßt sich nicht ohne Rest durch b teilen (not divisible without remainder).

Vervollständigen Sie folgendes Programmfragment so, dass für beide Fehlerfälle jeweils eine Exception signalisiert wird. Im Fall 1 soll das eine `DivisionByZeroException` und im Fall 2 eine `NotDivisibleWithoutReminderException` sein.

```
public class DivisionByZeroException [extends Exception] {}

public class NotDivisibleWithoutReminderException [extends Exception] {
  private int a;
  private int b;

  public NotDivisibleWithoutReminderException(int a, int b) {
    this.a = a; this.b = b;
  }
  public int getZaehler() { return a; }
  public int getNenner() { return b; }
}

public class Arithmetik {
  public static int teileGanzzahlig(int a, int b) throws
        DivisionByZeroException,
        NotDivisibleWithoutReminderException {
    if (b == 0) {
      throw [new DivisionByZeroException()];
    }
    if (a % b != 0) {
      throw [new NotDivisibleWithoutReminderException(a, b)];
    }
    return a / b;
  }
}

public class Arithmetik Programm {
  public static void main(String args[]) {
    try {
      System.out.println(Arithmetik .teileGanzzahlig(4,3));
    } catch ( [DivisionByZeroException dze] ) {
      System.out.println("Division durch 0 aufgetreten");
    } catch ( [NotDivisibleWithoutReminderException ndwr] ) {
      System.out.println(ndwr.getZaehler() +
             " laesst sich nicht ganzzahlig durch " + ndwr.getNenner() +
             " dividieren");
    }
  }
}
```