Name: **Vorname:** **Matr.-Nr.:** **Sem.:**

Fachhochschule Kaiserslautern
FB Informatik und Mikrosystemtechnik
Prof. Dr. M. **Duque-Antón**

Bachelor-Klausur im WiSe 2014/15

im Fach

Grundlagen der Informatik

Angewandte Informatik / Medieninformatik

Datum: **31. Januar 2015**

Es sind **keinerlei Hilfsmittel** zur Klausur zugelassen.

Achtung: Bitte schreiben Sie die **Lösung unter die Aufgabe in das Aufgabenblatt**. Wenn Sie mehr Platz benötigen, legen Sie **von der Aufsicht** zu erhaltende **zusätzliche Blätter** dazwischen.

Die schriftliche Prüfung besteht aus 5 Aufgaben. Schreiben Sie bitte auf **jedes Blatt Ihren Namen, Vornamen, Matrikelnummer** und Semester. Es werden nur Blätter mit Namen und Matrikelnummer korrigiert oder bewertet. Unleserliche Lösungen, Lösungen mit Bleistift oder Rotstift werden nicht korrigiert oder bewertet.

1	2	3	4	5	Summe	Note
/18	/20	/20	/22	/20	/100	

1. Aufgabe: Allgemeine Grundlagen

Kennzeichnen Sie die folgenden Aussagen durch Ankreuzen als „wahr“ oder „falsch“. Dabei wird keine Antwort: mit 0 Punkten bewertet, eine richtige Antwort mit +1 Punkt und eine falsche Antwort mit -1 Punkt. Minimal können 0 Punkte erreicht werden.

	wahr	falsch
Mit n Bit kann man genau 2^n verschiedene Zeichen codieren.	☐	☐
Die Darstellung digitaler Informationen mit Hilfe von ASCII-Zeichen ist effizienter als die entsprechende Darstellung mit Hilfe binärer Daten.	☐	☐
Die Darstellung ganzer Zahlen in der Einerkomplementdarstellung ist gut geeignet, um die gewohnte Arithmetik innerhalb eines Rechners zu realisieren.	☐	☐
String-Literale werden im String-Pool höchstens einmal angelegt und dann nicht mehr verworfen.	☐	☐
Ein Programm kann Felder erzeugen, auf die es später nicht mehr zugreifen kann.	☐	☐
In Java können Variablen auch außerhalb einer Klasse deklariert werden.	☐	☐
Auf jede Variable kann während der gesamten Programmausführung zugegriffen werden.	☐	☐
Es kann mehrere Variablen geben, die auf dasselbe Array zeigen.	☐	☐
In Java können nur gleichmäßige Arrays angelegt werden.	☐	☐
Nur bei Mehrfachvererbung darf eine Klasse mehrere Oberklassen besitzen.	☐	☐
Ein Objekt einer abgeleiteten Klasse kann immer dann verwendet werden, wenn ein Objekt einer Oberklasse erwartet wird, da die Oberklasse allgemeiner ist.	☐	☐
Der Mechanismus des Überladens betrifft nur Attribute einer Klasse.	☐	☐
Alle zu einem Array gehörenden Elemente werden immer nur im Stack abgelegt.	☐	☐
Eine abgeleitete Klasse erbt die Attribute und Methoden der Oberklasse. Es können jedoch weitere Attribute und Methoden hinzugefügt werden.	☐	☐
Beim Parameterübergabemechanismus call by value wirken sich Änderungen formaler Parameter stets auf die aktuellen Parameter aus.	☐	☐
In Java werden allen lokalen Variablen Default-Werte zugewiesen.	☐	☐
In Java ist ein zweidimensionales Array ein Referenztyp.	☐	☐
In kontextfreier Grammatik/Syntax-Diagramm darf die linke Seite einer Regel nur aus genau einem Terminal- oder Nichtterminalsymbol bestehen.	☐	☐

1.)		Summe
Punkte	/18	/18

2. Aufgabe: Zahlendarstellungen und Grammatiken

Gegenstand dieser Aufgabe ist die **Darstellung von Zahlen** innerhalb eines Rechners, die entsprechende **Umwandlung** verschiedener Basissysteme und die Erkennung von Texten auf der Basis vorgegebener **Grammatiken / EBNFs (**Extended Backus Naur Form**)**.

(a) Gegeben sei die folgende Binärzahl (Dualzahl mit der üblichen Interpretation als vorzeichenlose Zahl) $\mathbf{01011110_2}$. Wie sieht die **dezimale und die oktale Darstellung** der Zahl aus?

(b) Betrachten Sie den **Java-Datentyp short**. Wir nehmen an, dass die beiden Zahlen $\mathbf{35_{10}}$ und $\mathbf{-50_{10}}$ jeweils in einer short-Variablen gespeichert sind. Wie sieht die entsprechende interne **binäre Darstellung der beiden Zahlen** aus?

(c) Bestimmen Sie eine Grammatik G = (N, T, P, S) mit T = {0, 1} so, dass die erzeugte Sprache L(G) die Menge aller durch 8_{10} teilbaren Dualzahlen darstellt. L(G) soll insbesondere die Darstellung der Zahl mit dem Wert 0 enthalten, führende (binäre) Nullen sind nicht erlaubt. Wählen Sie N und P möglichst minimal!

2.)	a	b	c	Summe
Punkte	/6	/8	/6	/20

3. Aufgabe: Java-Programmierung

Gegenstand dieser Aufgabe ist die Programmierung mit Hilfe der **Programmiersprache Java**, insbesondere die Verwendung von **Rekursion und Iteration** und die entsprechenden Komplexitäten.

(a) Gegeben sei die folgende Klasse WasWohlTest. Was wird beim Aufruf java WasWohlTest auf dem Bildschirm ausgegeben?

```
public class WasWohlTest  {

    static int waswohl (int n) {
        if (n < 2)   return n;
        return (3 * waswohl (n-1)  - 2 * waswohl (n-2) );
    } // waswohl

    public static void main (String [ ] args) {
        for (int i = 2; i < 7; i++) {
                System.out.println (i + " : " + waswohl (i) );
        }

    } // main

} // WasWohlTest
```

(b) Im folgenden wird zu einer in Java vorgegebenen Methode die funktionale bzw. mathematische Beschreibung gesucht. Als Beispiel wird zunächst die Java-Methode **methode1** vorgegeben:

```
static double methode1 (double n) {
      if (n >= 0) return n;
      else return -n;
}  // methode1
```

In diesem Fall wäre die korrekte Antwort: Die Methode **methode1** liefert den Betrag einer beliebigen Gleitkommazahl zurück.

Die Aufgabe lautet nun, für die folgende Java-Methode **methode2** die entsprechende Funktionalität bzw. die entsprechende mathematische Funktion aus dem Java-Programm „herauszulesen“:

```
static boolean methode2 (int n) {
      if (n == 1) return false;
      for (int i = 2; i < n; i++) {
         if ((n % i) == 0) return false;
      }
      return true;
}  // methode2
```

(c) Wie sieht die Rechen-Komplexität der Algorithmen aus Teilaufgabe (a) und Teilaufgabe (b) im worst-case aus?

3.)	a	b	c	Summe
Punkte	/8	/6	/6	/20

4. Aufgabe: Java-Programmierung

Gegenstand dieser Aufgabe ist die Programmierung mit Hilfe der Programmiersprache Java, insbesondere die Verwendung interner Datenstrukturen und der Umgang mit Ein- und Ausgabe-Methoden in Java.

(a) In dieser Teilaufgabe soll ein klassischer Würfel implementiert werden. Die entsprechende Implementierung in der Java-Methode **wurf** soll also die Zahlen 1 bis 6 jeweils mit gleicher Wahrscheinlichkeit zurück liefern.
Insgesamt soll die Klasse **TestWuerfel** implementiert werden, welche neben der Methode **wurf** natürlich auch die **main-Methode** enthält.
In der main-Methode soll der Würfel genau eine Million Mal aufgerufen werden. Die entsprechenden Würfel-Ergebnisse sollen im Sinne der relativen Häufigkeiten in einer internen Datenstruktur verwaltet und am Ende der main-Methode auf der Standard-Ausgabe ausgegeben werden.
Bitte beachten Sie, dass die Methode wurf als static deklariert ist. Daher müssen auch die entsprechenden internen **Datenstrukturen als Klassenvariablen** eingeführt werden.
Hinweis: Bitte beachten Sie, dass auf dem Bildschirm die Zahlen 1 bis 6 (und nicht 0 bis 5) erwartet werden. Eine korrekte Ausgabe sieht beispielsweise wie folgt aus:

```
Zahl 1:  0.1665858
Zahl 2:  0.1667806
Zahl 3:  0.1668205
Zahl 4:  0.1665142
Zahl 5:  0.1664964
Zahl 6:  0.1668025
```

```
public class TestWuerfel {

        static int wurf ( ) {

        } // wurf

        public static void main (String args [])  {

        } // main

} //  TestWuerfel
```

(b) Der euklidische Abstand ist der Abstandsbegriff der euklidischen Geometrie. Anschaulich ist mit Hilfe des euklidischen Abstands die Länge einer Strecke gemeint, welche diese zwei Punkte miteinander verbindet.
Zur Lösung der folgenden Aufgabenstellung ist dieses Hintergrundwissen ohne Bedeutung. Vielmehr ist **nur** die Definition des **euklidischen Abstands D** zweier Vektoren $A = (a_1, a_2, \ldots, a_n)$ und $B = (b_1, b_2, \ldots, b_n)$ relevant:

$$\mathbf{D\ (A,B) = \sqrt{(a_1 - b_1)^2 + \ldots + (a_n - b_n)^2}}$$

In dieser **Aufgabe soll die vorgegebene Java-Klasse TestAbstand derart ergänzt werden**, dass **zwei beliebige Vektoren A und B gleicher Dimension beim Aufruf des Java-Programms eingelesen werden** und als **Ergebnis der euklidische Abstand auf dem Bildschirm ausgegeben** wird**.**

Die Vektoren A und B sind jeweils in einer eigenen ASCII-Datei abgelegt in der aus der Vorlesung bekannten üblichen Variante: Zunächst ist die Länge des Vektors aufgeführt, gefolgt von den eigentlichen Werten. Die Längenangabe soll als Integer, die einzelnen Werte als Double-Werte interpretiert werden.

Bitte **vervollständigen Sie die beiden Methoden einlesenVector** und **berechneEuklidAbstand**. Die beiden anderen Methoden, **testCompa und main dürfen nicht verändert** werden.

Hinweis: Zur Berechnung der Quadratzahl und der Wurzel einer Zahl können die Methoden **static double pow (double a, double b)** bzw. **static double sqrt (double a)** der statischen Klasse **Math** aus dem Standard Java-Package (lang) verwendet werden. Die Methode pow erwartet im ersten Parameter a die Basis und im zweiten Parameter b den Exponenten.

```
import java.util.Scanner;
import java.io.*;

public class TestAbstand  {
    static private double [ ] vectorA, vectorB;

    private static double [ ] einlesenVector (String datei) throws IOException {
    . . .
    } // einlesenVector

    private static boolean testCompa (double [ ] A, double [ ] B)  {
       if (A.length == B.length) return true;
       else return false;
    } // testCompa

    private static void berechneEuklidAbstand (double [ ] A, double [ ] B)  {
    . . .
    } // berechneEuklidAbstand

    public static void main (String args [ ])  throws IOException {
            if (args.length != 2) {
                    System.out.print ("Korrekter Aufruf sieht wie folgt aus >>>>> ");
                    System.out.println ("java A.txt B.txt  ");
                    return;
            } // if
            vectorA = einlesenVector (args[0]);
            vectorB = einlesenVector (args[1]);
            if ( !testCompa (vectorA, vectorB)) return;
            berechneEuklidAbstand (vectorA, vectorB);
    } // main

} // TestAbstand
```

```
private static double [ ] einlesenVector (String datei) throws IOException {
```

} **// einlesenVector**

```
private static void berechneEuklidAbstand (double [ ] A, double [ ] B)  {
```

} **// berechneEuklidAbstand**

4.)	a	b	Summe
Punkte	/10	/12	/22

5. Aufgabe: Objektorientierung

(a) Mit Hilfe von Java soll ein Wörterbuch als Klasse Dictionary mit der gängigen Bedeutung implementiert werden. Das Wörterbuch soll also Wort-Paare enthalten, um beispielsweise für ein vorgegebenes deutsches Wort das entsprechende spanische Wort zurückzugeben.
Dazu sind die beiden folgenden Methoden in Java zu realisieren: Mit Hilfe der Methode **void insert (String in, String out)** werden die Strings *in* und *out* in das Wörterbuch eingetragen. Mit Hilfe der Methode **String lookup (String in)** wird der String *in* im Wörterbuch gesucht und der entsprechende Wert zurückgegeben. Falls String *in* nicht im Wörterbuch enthalten ist, dann soll die Null-Referenz zurückgegeben werden.
Bei der Realisierung soll darauf geachtet werden, dass auf die internen Strukturen **von Außen nicht zugegriffen werden** kann. Ein Objekt der Klasse Dictionary soll nur über die beiden Methoden insert und lookup ansprechbar sein. Die Klasse Dictonary soll **genau einen Konstruktor mit einem Parameter** anbieten, der zum Zeitpunkt der Objekt-Generierung die gewünschte interne Größe des Wörterbuchs aufnimmt.

(b) Gegenstand dieser Aufgabe ist der geschickte Umgang mit Klassenvariablen und Instanzvariablen. **Programmieren Sie eine Java-Klasse Beleg**, deren Objektinstanzen bei der Generierung (Konstruktor) automatisch eine **eindeutige Objekt-individuelle Belegnummer** erhalten.

```
import java.text.SimpleDateFormat;
import java.util.Date;

class Beleg{
      ..

      Beleg() {
         ..

      } // Beleg

      int nummer () {
         ..
      } // nummer

} // Beleg


public class Test {

      public static void main (String args[]) {
         Beleg b1 = new Beleg ();
         Beleg b2 = new Beleg ();

         System.out.println(b1.nummer());    // 150131000
         System.out.println(b2.nummer());    // 150131001
      } // main

} // Test
```

Die individuelle Belegnummer pro Objektinstanz soll also als Integer codiert werden, welche aus genau 9 Ziffern bestehen soll. Dabei stellen die ersten 6 Ziffern das aktuelle Datum dar und die letzten drei Ziffern eine laufende Nummer, die mit 000 beginnen soll. Von links nach rechts betrachtet, besteht das aktuelle Datum aus jeweils zwei Ziffern für das Jahr, den Monat und den Tag.

Ein Beispiel für eine Belegnummer stellt die Nummer **`150131001`** dar. Es handelt sich also um den 2ten Beleg, der am heutigen Tag, dem 31. Januar 2015 generiert wurde. Falls das Test-Programm morgen, Sonntag, den 1. Februar aufgerufen wird, dann liefert die **Methode nummer** für den ersten (morgen) generierten Beleg die Nummer: **`150201000.`**

Es wird angenommen, dass pro Tag maximal 1000 Belege zu erstellen sind, so dass die 3 Ziffern für die laufende Belegnummer als ausreichend betrachtet werden können.

Auf der folgenden Seite sind einige Hinweise zur Lösung der Aufgabe aufgeführt!

Bei der Objektgenerierung muss daher das aktuelle Datum herangezogen werden. **Zur einfachen Generierung der Datums-Information soll die Klasse SimpleDateFormat und die Klasse Date verwendet werden**. Mit Hilfe der **Klasse Date** wird das **aktuelle Datum** erzeugt, welches alle relevanten Datums-Daten enthält. Mit Hilfe der **Klasse SimpleDateFormat** kann die komplexe Datums-Information **nach Wunsch angepaßt** werden. Die folgenden Beispiele veranschaulichen den Umgang mit den beiden Klassen.

```
SimpleDateFormat format1 = new SimpleDateFormat ("dd.MMMM yyyy");
String date = format1.format (new Date ());
System.out.println (date);   // 13.Januar 2015

SimpleDateFormat format2 = new SimpleDateFormat ("dd.MM  yy");
date = format2.format (new Date ());
System.out.println (date);   // 13.01  15

SimpleDateFormat format3 = new SimpleDateFormat ("ddMMyy");
date = format3.format (new Date ());
System.out.println (date);   // 130115
```

Mit Hilfe der statischen Methode static int parseInt (String s) kann ein String in ein Integer-Wert umgewandelt werden und anschließend als Integer verwaltet bzw. manipuliert werden. Die Methode parseInt wird von der Klasse Integer angeboten, die im Default Package java.lang enthalten ist.

Im folgenden wird ein Programmgerüst angeboten, welches korrekt aufgefüllt werden soll. Dazu müssen die entsprechenden **Klassen- und Instanzvariablen** vorgesehen und der **Code** für den **Konstruktor und** die **Methode nummer vervollständigt** werden. Der Konstruktor sorgt dafür, dass bei der Generierung des Objekts die Belegnummer korrekt gebildet wird, die Methode nummer liefert diese Nummer zurück.

```
import java.text.SimpleDateFormat;
import java.util.Date;

class Beleg{

    Beleg() {

    } // Beleg

    int nummer () {

    } // nummer

} // Beleg
```

5.)	a	b	Summe
Punkte	/10	/10	/20