Name: _____ Matrikel-Nr.: _____

Dr. Jochen Hoenicke
Alexander Nutz

Probeklausur zur Vorlesung
Einführung in die Informatik
Sommersemester 2014

Die Klausur besteht aus diesem Deckblatt und elf Blättern mit den Aufgaben, sowie einem Blatt mit einem Ausschnitt aus der Dokumentation der Java Standardbibliothek. Auf Anfrage erhalten Sie zusätzliches Papier. Tragen Sie auf jedem Blatt bitte Ihren Namen und Ihre Matrikelnummer ein.

Zur Bearbeitung haben Sie 120 Minuten Zeit. Es sind keine Hilfsmittel zugelassen. Neben jeder Aufgabe steht die Zahl der durch ihre Bearbeitung ereichbaren Punkte. Am Ende der Klausur finden Sie ein Blatt mit einigen Klassen und Methoden der Java Standardbibliothek. Diese dürfen Sie in der Klausur benutzen, andere nicht.

Falls Sie eine Aufgabe mehrmals bearbeiten, machen Sie bitte kenntlich, welche Lösung bewertet werden soll. Falls Sie eine Aufgabe nicht direkt unter der Aufgabenstellung bearbeiten, machen Sie das bitte bei der Aufgabenstellung kenntlich.

Aufgabe	Erreichte Punkte
1. Einfaches Programmieren	von 15
2. Java Syntax	von 7
3. Programmieren/Binäre Suchbäume	von 20
4. Programme verstehen (1)	von 8
5. Programme verstehen (2)	von 8
6. Klassendiagramm/Vererbung	von 15
7. Generics	von 7
7. $\mathcal{O}$-Notation/Komplexität	von 10
8. Verschiedenes	von 20
9. Testen	von 10
Summe	von 120

Name: _____ Matrikel-Nr.: _____

1. Aufgabe (Einfaches Programmieren) **15**

(a) Schreiben Sie eine Methode, die ein Array von ints als Eingabe nimmt und deren Summe zurückliefert.

(b) Schreiben Sie eine Methode, die eine ArrayList von Integern als Eingabe nimmt und zwei ArrayLists von Integern (in einem Array) zurückliefert. Dabei sollen in der einen genau die geraden Zahlen aus der Eingabeliste enthalten sein, in der andern die ungeraden.

(c) Die Exponentialfunktion zur basis e kann man definieren als

$$exp(x) = \sum_{n=0}^{\infty} x^n/n!$$

Schreiben sie eine Methode die die Exponentialfunktion mit einer gegebenen Genauigkeit (die Grenze, bis zu der die Summe laufen soll als `int`) ausrechnet.

Lösungsskizze

(a)
```
int sumup(int[] a) {
  int result = 0;
  for (int i = 0; i < a.length; i++)
    result += a[i]
  return result;
}
```

(b)
```
ArrayList<Integer>[] split(ArrayList<Integer> inputList) {
  ArrayList<Integer> evenNumbers = new ArrayList<Integer>();
  ArrayList<Integer> oddNumbers = new ArrayList<Integer>();
  for (int i = 0; i < inputList.size(); i++) {
    if (inputList.get(i) % 2 ==0)
      evenNumbers.add(inputList.get(i));
    else
      oddNumbers.add(inputList.get(i));
  }
  return new ArrayList<Integer>[] { evenNumbers, oddNumbers};
}
```

(c)
```
double exp(double x, int n) {
  double result = 0.0;
  long currentFac = 1;
  double currentPower = 1;
  for (int i = 1; i <= n; i++) {
    result += currentPower/currentFac;
    currentFac *= i;
    currentPower *= x;
```

Name: _____ Matrikel-Nr.: _____

```
    }
    return result;
  }
```

Name: _____ Matrikel-Nr.: _____

2. Aufgabe (Java Syntax) **7**

Folgendes Programm kompiliert nicht. Benennen Sie die Fehler, die dies verhindern. Für „Fehler“, die keine sind, gibt es Punktabzug. Die minimale Punktzahl für diese Aufgabe ist 0. Unseres Wissens und nach unserer Zählung gibt es neun Fehler in diesem Programm. Wenn Sie sieben korrekt benennen, gibt es schon die volle Punktzahl – geben Sie trotzdem alle an, die Sie finden, für den Fall, dass die Zählweisen verschieden sind.

```
1  public class {
2    public static main(String args) {
3      Car car = new Car();
4      Road road = new Road();
5      if (car.speed > Road.speedLimit) {
6        System.out.println('Car was too fast!');
7        System.out.println('Speed limit is ' + road.speedLimit + '!');
8      }
9    }
10 }
11
12 public class Car {
13   private int speed
14
15   public Car(double speed) {
16     this.speed = speed;
17   }
18 }
19
20 public class Road {
21   public double speedLimit = 50;
22 }
```

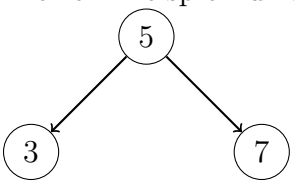
Lösungsskizze

- Zl. 1: Klassenname fehlt
- Zl. 2: Rückgabetyp fehlt (void)
- Zl. 2: main nimmt ein Array von Strings
- Zl. 3: Car hat keinen Konstruktor ohne Parameter
- Zl. 5: car.speed ist private
- Zl. 5: Road.speedLimit ist nicht static
- Zl. 6/7: Falsche Anführungszeichen
- Zl. 13: Semikolon fehlt
- Zl. 16: Zuweisung eines doubles in einen int

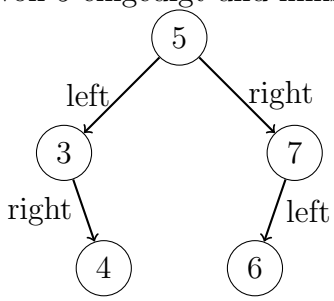
Name: _____ Matrikel-Nr.: _____

3. Aufgabe (Programmieren/Binäre Suchbäume) **20**

In dieser Aufgabe beschäftigen wir uns mit *binären Suchbäumen*. Ein Binärbaum besteht aus *Knoten*. Ein Knoten besteht aus einem Element und zwei weiteren Knoten (den Kindern), genannt `left` und `right`. Ein Knoten kann auch leer sein, er enthält also keinen Inhalt und hat keine Kind-Knoten. Bei einem *Suchbaum* sind die Elemente zudem geordnet: In dem linken Kind eines Knoten folgen nur noch Elemente, welche kleiner sind als das Element des Knotens und im rechten Kind nur Elemente welche größer sind als das Element des Knotens.
Hier ein Beispiel für einen sortierten Baum, der `int`-Werte enthält.



Fügen wir jetzt das Element 4 ein, muss dieser links von der 5 eingefügt werden und dann rechts von der 3. Wird danach das Element 6 eingefügt, so muss dieses rechts von 5 eingeüfgt und links von 7, wir erhalten also den folgenden Baum:



Folgende Klasse modelliert also einen Knoten im Baum (und somit auch den Baum, der diesen Knoten zur Wurzel hat). Ein leerer Baum wird durch `null` repräsentiert.

```
public class BinTreeNode {
    BinTreeNode left;
    BinTreeNode right;
    int elem;

    public BinTreeNode(BinTreeNode left, BinTreeNode right, int elem) {
        this.left = left;
        this.right = right;
        this.elem = elem;
    }

    public BinTreeNode(int elem) {
        this(null, null, elem);
    }

    public boolean contains(int obj) {
        if (this.elem == obj) {
            return true;
```

Name: _____ Matrikel-Nr.: _____

```
        } else if (obj < this.elem) {
            if (this.left != null)
              return contains(this.left, obj);
            else
              return false;
        } else {
            // obj > this.elem
            if (this.right != null)
              return contains(this.right, obj);
            else
              return false;
        }
    }

    public int size() {
        // Implementieren Sie diese Methode
    }

    public void printOrdered() {
        // Implementieren Sie diese Methode
    }

    public void insert(int obj) {
        // Implementieren Sie diese Methode
    }
}
```

(a) Schreiben Sie eine Methode `size()`, die berechnet, wie viele Knoten der Baum enthält.

(b) Schreiben Sie eine Methode `printOrdered()` welche die Elemente des Baums sortiert ausgibt, das kleinste Element zuerst. Bei obigem Baum sollte die Ausgabe also sein

```
3
4
5
6
7
```

Tipp: Für einen Knoten jeweils zuerst die Elemente des linken Kindes sortiert ausgeben, dann das Element und dann die Elemente des rechten Kindes.

(c) Schreiben Sie eine Methode `insert(int obj)`, welche die Zahl `obj` in den Baum einfügt und dabei die Suchbaumeigenschaft erhält. Ist die Zahl bereits im Baum enthalten, soll nichts eingefügt werden.

Name: _____ Matrikel-Nr.: _____

Lösungsskizze

```
public int size() {
  int sizeLeft = 0;
  if (left != null)
    sizeLeft = left.size();
  int sizeRight = 0;
   if (right != null)
    sizeRight = right.size();
  return sizeLeft + sizeRight + 1;
}

public void printOrdered() {
   if (left != null)
     left.printOrdered();
   System.out.println(elem);
   if (right != null)
     right.printOrdered();
}

public void insert(int obj) {
  if (obj < elem) {
    if (left != null)
      left.insert(obj);
    else
      left = new BinTreeNode(null, null, obj);
  } else if (obj > elem) {
    if (right != null)
      right.insert(obj);
    else
      right = new BinTreeNode(null, null, obj);
  } else {
    //obj == elem -> do nothing
  }
}
```

Name: _____ Matrikel-Nr.: _____

4. Aufgabe (Programme verstehen (1)) **8**

Betrachten Sie folgendes Programm. Was ist seine Ausgabe?

```
public class MyClass {
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    C c = new C();

    System.out.println(a.echo("1"));
    System.out.println(b.echo("2"));
    System.out.println(c.echo("3"));
    System.out.println(b.echo(1));
    System.out.println(c.echo(2));
  }
}

class A {
  public String echo(String s) {
    return "A echoes: " + s;
  }

}

class B extends A {
  public String echo(Integer i) {
    return "B echoes: " + i;
  }
}

class C extends B {
  public String echo(String i) {
    return "C echoes: " + i;
  }
}
```

.....Lösungsskizze .....

```
A echoes 1
A echoes 2
C echoes 3
B echoes 1
B echoes 2
```

Name: _____ Matrikel-Nr.: _____

5. Aufgabe (Programme verstehen (2)) **8**

Betrachten Sie folgendes Programm. Was ist seine Ausgabe?

```
import java.util.ArrayList;

public class MyClass {
  public static void main(String[] args) {
    ArrayList<Integer> al1 = new ArrayList<>();
    ArrayList<Integer> al2 = new ArrayList<>();
    for (int i = 0; i < 5; i++) {
      al1.add(i);
      if (i != 0)
        al2.add(i);
    }
    makeAtLeast5big(al1);
    makeAtLeast5big(al2);

    System.out.println(al1);
    System.out.println(al2);
  }

  static void makeAtLeast5big(ArrayList<Integer> list) {
    if (list.size() < 5) {
      ArrayList<Integer> al = new ArrayList<Integer>();
      al.addAll(list);
      for (int i = 0; i < 5; i++)
        al.add(0);
      list = al;
    }
    // otherwise do nothing
  }
}
```

.....Lösungsskizze .....

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4]
```

Name: _____ Matrikel-Nr.: _____

6. Aufgabe (Klassendiagramm/Vererbung) **15**

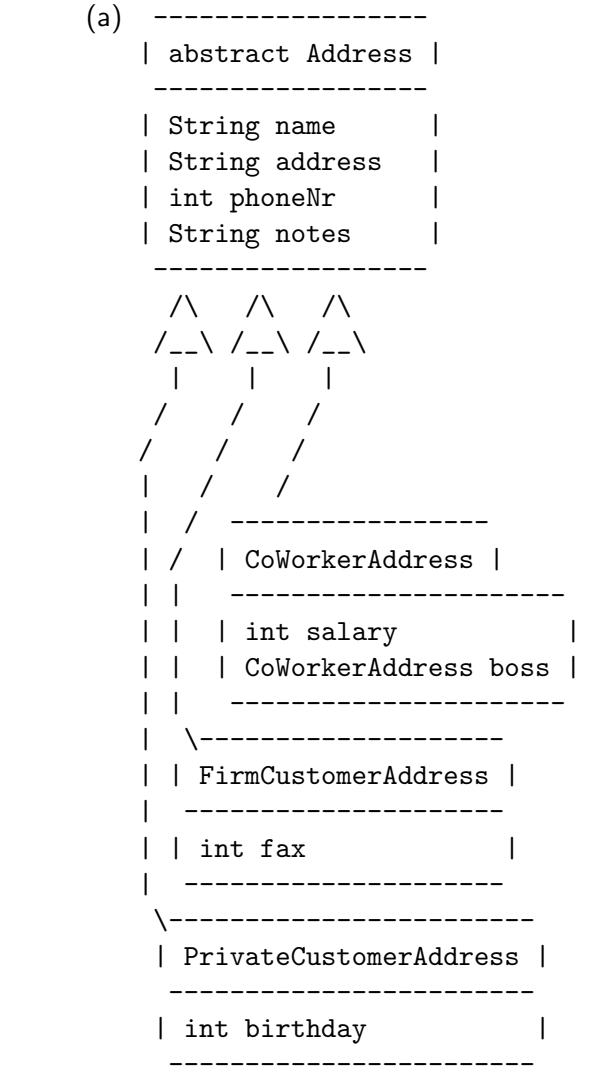
Angenommen, Sie wollen für eine Firma eine Adressverwaltung programmieren. Gespeichert werden sollen die Adressen von Mitarbeitern, Firmenkunden und Privatkunden. Alle drei haben einen Namen (`String`), eine Adresse (`String`), eine Telefonnummer (`int`) und ein Feld für zusätzliche Notizen (`String`).
Zusätzlich dazu hat ein Mitarbeiter eine Gehaltsstufe (`int`), und einen Vorgesetzten (auch vom Typ Mitarbeiter).
Ein Firmenkunde hat außerdem eine Faxnummer (`int`), ein Privatkunde stattdessen einen Geburtstag (`int`, damit man ihm eine Glückwunschkarte schreiben kann..).

(a) Zeichnen Sie ein Klassendiagramm für die beschriebenen Adresstypen, so dass bei einer Optimierung redundanter Code vermieden wird.

(b) Implementieren Sie das Klassendiagramm in Java.

Lösungsskizze

(a)



(CoWorkerAddress.boss ließe sich auch als Assoziation darstellen)

Name: _____ Matrikel-Nr.: _____

(b)
```
abstract class Address {
  String name;
  String address;
  int phoneNr;
  String notes;
}
class CoWorkerAddress extends Address {
 int salary;
 CoWorkerAddress boss;
}
class FirmCustomerAddress extends Address {
 int fax;
}
class PrivateCustomerAddress extends Address {
 int birthday;
}
```

Name: _____ Matrikel-Nr.: _____

7. Aufgabe (Generics) **7**

Folgendes Programm sortiert ein Array von Integern. (Es handelt sich um BubbleSort, das Sie eventuelle schon aus der Übung kennen.) Verändern Sie das Programm so, dass es Arrays eines beliebigen Objekttyps `T` sortiert, solange dieser das interface Comparable implementiert. Benutzen Sie dabei Generics, so dass ein Benutzer der Klasse mithilfe des Typsystems sicherstellen kann, dass eine `BubbleSort`-Instanz nur Arrays mit zum Beispiel ausschließlich `String`-Objekten sortieren kann.
Notationsvorschlag: Wenn Sie eine Zeile ersetzen möchten, streichen Sie einfach die alte durch und schreiben Sie ihre Zeile darüber.

```
class BubbleSort {

  static void sort(int[] a) {

    boolean swapped = false;

    int j = a.length;

    do {

      swapped = false;

      for (i = 0; i < j - 1; i++) {

        if (a[i] > a[i+1]) {

          int tmp = a[i+1];

          a[i+1] = a[i];

          a[i] = tmp;

          swapped = true;

        }

      }

      j--;

    } while (swapped);

  }
}
```

.....Lösungsskizze .....

Name: _____ Matrikel-Nr.: _____

```
class BubbleSort<T extends Comparable> {

  static void sort(T[] a) {
    boolean swapped = false;
    int j = a.length;
    do {
      swapped = false;
      for (i = 0; i < j - 1; i++) {
        if (a[i].compareTo(a[i+1]) > 0)  {
          T tmp = a[i+1];
          a[i+1] = a[i];
          a[i] = tmp;
          swapped = true;
        }
      }
      j--;
    } while (swapped);
  }
}
```

Name: _____ Matrikel-Nr.: _____

8. Aufgabe (O-Notation/Komplexität) **10**

(a) Betrachten Sie folgendes Programm und geben Sie seine Laufzeit und die jeder Anweisung (Statements) und jedes Blocks (z.B. Schleife, Methode) an (in $\mathcal{O}$-Notation).

```
int[] m(int[] a) {
  int n = a.length;
  for (int i = 0; i < 1000; i++) {
    for (int j = 0; j < i; j++) {
      a[i] = j;
    }
  }

  for (int j = 0; j < n; j++) {
    for (int i = 0; i < Math.pow(2,n); i++) {
       a[j]++;
    }
  }
}
```

(b) Was ist die Laufzeit der beiden folgenden Methoden in $\mathcal{O}$-Notation (für positive Eingaben)? Begründen Sie Ihre Antwort kurz.

```
boolean even(int i) {
  if (i == 0)
    return true;
  else
    return odd(i - 1);
}

boolean odd(int i) {
  if (i == 0)
    return false;
  else
    return even(i - 1);
}
```

.....Lösungsskizze .....

(a)
```
int[] m(int[] a) {
  int n = a.length; //O(1)
  // O(1) i ist konstant, d.h. nicht von der Eingabe abhängig
  for (int i = 0; i < 1000; i++) {
    //O(1) -- j ist durch i begrenzt, i ist konstant
```

Name: _____ Matrikel-Nr.: _____

```
      for (int j = 0; j < i; j++) {
        a[i] = j; //O(1)
      }
    }

    for (int j = 0; j < n; j++) { //O($n \cdot 2^n$)
      for (int i = 0; i < Math.pow(2,n); i++) { //O($2^n$)
         a[j]++; //O(1)
      }
    }
  }
```

(b) Beide laufen in $\mathcal{O}(i)$: vor jedem Aufruf wird i dekrementiert, bei 0 ist die Rekursion zuende.

Name: _____ Matrikel-Nr.: _____

9. Aufgabe (Verschiedenes) **20**

(a) beantworten Sie folgende Fragen einfach mit ja/nein

(i) Gilt $5n + 5 \in \mathcal{O}(2n)$?

(ii) Gilt $5n^2 + 5 \in \mathcal{O}(2n)$?

(iii) Gilt $5n^2 + 5 \in \mathcal{O}(2^n)$?

(iv) Gilt $\mathcal{O}(n) = \mathcal{O}(2n)$?

(v) Gilt $\mathcal{O}(n^2) = \mathcal{O}(n^3)$?

(vi) Gilt $\mathcal{O}(2^n) = \mathcal{O}(3^n)$?

(b) Laut Definition aus der Vorlesung muss ein Algorithmus *finit* sein. Was bedeutet das?

(c) Welche Folgen hat es, wenn man eine Variable als `static` deklariert?

(d) Wie unterscheidet sich eine abstrakte Klasse von einem interface?

(e) Was ist der Unterschied zwischen den Typen `int` und einem `Integer` in Java?

(f) Was ist der Unterschied den Ausdrücken `i++` und `++i` in Java?

.....Lösungsskizze .....

(a) (i) ja

(ii) nein

(iii) ja

(iv) ja

(v) nein

(vi) nein

(b) Es bedeutet, dass er eine endliche Beschreibung haben muss, z.B. ein Java-Programm endlicher Länge.

(c) Es wird nur eine instanz der Variablen angelegt. Sie wird vor der Ausführung des restlichen Programms initialisiert. Sie existiert unabhängig von allen Instanzen (Objekten) der Klasse. Wird Sie geändert gilt die Änderung dementsprechend über alle Instanzen hinweg.

(d) Die abstrakte Klasse kann Felder haben und sie darf Implementierungen von Methoden enthalten.

(e) Integer ist die sogenante Wrapper-Klasse für int. Ein Integer ist ein Objekt, auf dem man z.B. Methoden aufrufen kann. In der Integer-Variablen ist eine Referenz auf das Objekt gespeichert (vgl. Call-by-value..), in einer int-Variablen ist einfach nur der Wert der Zahl gespeichert.

(f) Ihr Effekt ist der selbe, aber `i++` wertet zu `i` aus, `++i` zu `i+1`.

Name: _____ Matrikel-Nr.: _____

10. Aufgabe (Tests – H) **10**

Nehmen Sie an, Sie sollen ein Programm schreiben, das, gegeben den Quelltext eines anderen Programms P als String und einer Eingabe e dafür, auch als String, `true` ausgibt, falls P auf e irgendwann ein Ergebnis liefert und stoppt, und `false` ausgibt, falls es nie zum Ende kommt (d.h. unendlich lange in einer Schleife bleibt, oder eine unendliche Rekursion durchführt).
Geben Sie zwei Tests mit dem Erwartungswert `true` zwei mit dem Erwartungswert `false` an, mit denen Sie P testen können, sollten Sie es eines Tages mal geschrieben haben.

.....Lösungsskizze .....

Mögliche Tests wären:
Für den Erwartungswert `false`:

```
public class DoesNeverStop {
 public static void main(String[] args) {
  int i = 0;
  while (true) {
   i++;
  }
 }
}

public class DoesNotStopOnEmptyInput {
 public static void main(String[] args) {
  int i = 0;
  try {
    i = Integer.parseInt(args[0]);
  } catch (Exception e) {
    while (true) {
      i++;
    }
  }
 }
}
```

1. Testfall: P = DoesNeverStop, e = ""
2. Testfall: P = DoesNotStopOnEmptyinput, e = ""

Für den Erwartungswert `true`:

```
public class StopsAlways {
 public static void main(String[] args) {
  try {

  }
  int i = 0;
 }
}
```

Name: _____ Matrikel-Nr.: _____

1. Testfall: P = StopsAlways, e = ""
2. Testfall: P = DoesNotStopOnEmptyinput, e = "1"

Hinweis: Das hier zu schreibende bzw. zu testende Programm ist deshalb interessant, weil seine Existenz theoretisch unmöglich ist (vgl. Wikipedia „Halteproblem“). In der Klausur wird kein so esoterisches Programm vorkommen.

Name: _____ Matrikel-Nr.: _____

Ausschnitt aus der Dokumentation zur Java Standardbibliothek (sinngemäß zitiert):

Klasse `ArrayList<T>`

- `add(T element)`

  Fügt element am ende der Liste ein.

- `remove(int index)`

  Entfernt das Element an Stelle `index` aus der Liste.

Interface `Comparable<T>`

- `int compareTo(T other)`

  Liefert einen Wert größer als 0, falls der Aufrufer größer als der Parameter ist, 0 falls sie gleich sind, einen negativen Wert sonst.

Klasse `Math`

- `static double pow(double a, double b)`

  Liefert a^b zurück.

Klasse `System`

- Statisches Feld `out` hat Methode `void println(String)`

  Schreibt den angegebenen String auf die Textkonsole.

Klasse `Integer`

- Integer parseInt(String s)

  Falls s eine Ganzahl (die nicht zu groß ist) enthält, wird diese zurückgeliefert. Sonst wird eine Exception geworfen.