TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Angewandte Informatik / Kooperative Systeme WS 14/15
Einführung in die Informatik I **Klausur**
Prof. Dr. A. Brüggemann-Klein, R. Palenta, A. Herz, Dr. A. Reuss 14.2.2015

Name	Vorname	Studiengang	Matrikelnummer
Hörsaal	Reihe	Sitzplatz	Unterschrift

Allgemeine Hinweise:

- **Bitte füllen Sie die oben angegebenen Felder vollständig aus und unterschreiben Sie!**
- **Schreiben Sie nicht mit Bleistift oder in roter/grüner Farbe!**
- **Es sind keine Hilfsmittel zugelassen. Verwenden Sie nur die bereitgestellten Klausurbögen und einen dokumentenechten blauen oder schwarzen Stift!**
- **Was nicht bewertet werden soll, kennzeichnen Sie bitte eindeutig durch Durchstreichen.**
- **Die Arbeitszeit beträgt 120 Minuten.**
- **Prüfen Sie, ob Sie alle 12 Seiten erhalten haben.**
- **In dieser Klausur können Sie insgesamt 82 Punkte erreichen. Zum Bestehen werden 33 Punkte benötigt.**
- **Falls Ihnen der Platz bei einer Aufgabe nicht ausreicht, verwenden Sie die leere Rückseite der Aufgabe.**

vorzeitige Abgabe um: ..... Hörsaal verlassen von ..... bis .....

1	2	3	4	5	6	7	8	9	$\sum$

.....
Erstkorrektur

.....
Zweitkorrektur

Aufgabe 1 [8 Punkte] **Einfache Datentypen**

Ergänzen Sie im folgenden Java-Code die korrekten Typbezeichner (z.B. `int`), damit der Code vom Compiler akzeptiert und übersetzt wird. Welche Ausgabe liefert der Code bei seiner Ausführung?

```
int x = 5;
x = 7 - x / (int)(2+0.3);
System.out.println(x);

          y = x + "4";
System.out.println(y);
System.out.println(x + y);
          xnull = x + 0;
System.out.println(xnull + "0");

double string = Math.max(x,19); // Maximum von x und 19
System.out.println(string);
System.out.println(string + x);
System.out.println(string + y);
System.out.println(string + 0);
```

Ausgabe:

Aufgabe 2 [10 Punkte] **Summe iterativ und rekursiv**

Betrachten Sie die Methode `SumIt`:

```
public static int sumIt(int n) {
   int sum, i;
   sum = 0;
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   return sum;
}
```

Welche Zahl Z wird in Abhängigkeit der Eingabe n zurückgegeben? Geben Sie eine Formel für Z an.

$Z =$

Geben Sie den Kontrollflussgraphen zum Programm `SumIt` an.

Wandeln Sie die iterative Variante `SumIt` um in eine rekursive Variante mit der Signatur `public static int sumRec(int n)`, die exakt die selbe Berechnung durchführt:

```
public static int sumRec(int n) {




}
```

Aufgabe 3 [10 Punkte] **Lotto**

a) Schreiben Sie eine Methode `public static int[] ziehung()`, die sechs unterschiedliche(!) Zahlen aus 1 bis 49 zufällig erzeugt und diese in einem `int`-Array zurückgibt.
b) Schreiben Sie eine Methode `public static int[] schnitt(int[] ziehung, int[] tipp)`, die ein Array mit den Elementen, die in beiden Arrays vorkommen, zurückgibt. Das zurückgegebene Array darf dabei nur so groß sein wie Elemente in den beiden Arrays übereinstimmen. Sie können davon ausgehen, dass in den Arrays `ziehung` und `tipp` keine Elemente mehrfach vorkommen.

Zum Erzeugen von Zufallszahlen können Sie die folgende Methode verwenden, die Zufallszahlen zwischen `low` (inklusive) und `high` (exklusive) erzeugt.

```
public static int myRandom(int low, int high) {
   return (int) (Math.random() * (high - low) + low);
}
```

```
public class Lotto {









}
```

Aufgabe 4 [10 Punkte] **Mergesort**

Vervollständigen Sie die folgende rekursive Implementierung des aus der Vorlesung bekannten Sortierverfahrens *Mergesort*. Der Bereich im Array `numbers` zwischen `left` und `right` (jeweils einschließlich) soll dabei von der Methode `sort` aufsteigend sortiert werden, indem der Bereich in zwei Teilbereiche aufgeteilt, diese separat sortiert und am Ende die sortierten Bereiche *gemerget* werden. Für den letzten Schritt (*merge*) wird ein Hilfs-Array verwendet.

Gehen Sie davon aus, dass die Methode `sort` beim Aufruf gültige Werte für `left` und `right` übergeben bekommt, d.h. beide Werte bezeichnen Indexe aus dem Indexbereich von `numbers`.

```
public static void sort(int[] numbers, int left, int right) {
   final int rightmost = right;
   final int leftmost = left;
   if (left >= right) [          ];

   // sort
   int middle = left + (right - left) / 2;
   sort(numbers, left, middle);
   sort([                                ]);

   // merge
   int[] sorted = new int[1 + right - left];
   right = [                  ];
   for (int i = 0; i < sorted.length; i++) {
      if (left > middle || (right <= rightmost &&
          [                                    ])) {
         sorted[i] = numbers[right];
         right++;
      } else {
          [                                  ];
         left++;
      }
   }

   // store
   for (int i = 0; i < sorted.length; i++) {
      numbers[leftmost + i] = sorted[i];
   }
}
```

Aufgabe 5 [10 Punkte] **Objekte**

Schreiben Sie eine Klasse `Mensch`, die die privaten Attribute `name, vorname` vom Typ `String` hat. Die Klasse soll einen Konstruktor `Mensch(String name, String vorname)` bereitstellen. Statten Sie die Klasse mit den entsprechenden get- und set-Methoden für ihre Attribute aus (`void setVorname(String vorname)`, `String getName()` usw.). Fügen Sie eine Methode `toString()` hinzu, die den vollen Namen des Menschen als String zurückgibt.

Leiten Sie von der Klasse `Mensch` eine Klasse `Arbeitnehmer` ab, die ein zusätzliches privates Attribut `gehalt` vom Typ `int` besitzt, welches zusätzlich zu Name und Vorname im Konstruktor übergeben wird. Die `toString`-Methode soll zusätzlich zu Name und Vorname das Gehalt des Arbeitnehmers ausgeben. Schreiben Sie weiterhin eine Methode `boolean compareGehalt(Arbeitnehmer a)`, die das Gehalt des übergebenen Arbeitnehmer-Objekts mit dem der aktuellen Instanz vergleicht und `true` zurückgibt, falls diese übereinstimmen und `false` sonst. **Nutzen Sie Vererbung möglichst gut aus, um zu vermeiden, den gleichen Code mehrfach zu schreiben.**

Aufgabe 6 [7 Punkte] **Polymorphie**

Betrachten Sie die angegebene Klassenhierarchie. Geben Sie zu jedem Ausgabebefehl in der `main`-Methode der Klasse `Polymorphie` an, ob er kompiliert, ob er ohne Fehler ausgeführt wird und welche Ausgabe er ggf. liefert.

```
1  class A {
2      int someMethod() { return 1; }
3      int someMethod(A a) { return 2; }
4      int someMethod(B b) { return 3; }
5      int someMethod(C c) { return 4; }
6      static A anotherMethod(Object obj) { return (A) obj; }
7  }
8
9  class B extends A {
10     int someMethod() { return 6; }
11     int someMethod(A a) { return 7; }
12     int someMethod(B b) { return 8; }
13     int someMethod(C c) { return 9; }
14     static A anotherMethod(Object obj) { return (B) obj; }
15 }
16
17 class C extends A {
18     int someMethod() { return 11; }
19     int someMethod(A a) { return 12; }
20     int someMethod(B b) { return 13; }
21     int someMethod(C c) { return 14; }
22     static C anotherMethod(Object obj) { return (C) obj; }
23 }
24
25 public class Polymorphie {
26
27     public static void main(String[] args){
28
29         A a = new A(); B b = new B(); C c = new C();
30
31         System.out.println(a.someMethod());
32         System.out.println(B.anotherMethod((C) b).someMethod());
33         System.out.println(A.anotherMethod(b).someMethod(b));
34         System.out.println(B.anotherMethod(b).someMethod(c));
35         System.out.println(a.someMethod((A) b));
36         System.out.println(B.anotherMethod(c).someMethod());
37         System.out.println(C.anotherMethod((A) c).someMethod(b));
38     }
39 }
```

Zeile

31:

32:

33:

34:

35:

36:

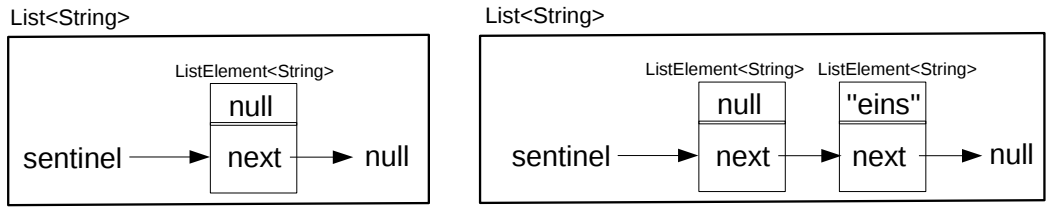
37:

Aufgabe 7 [8 Punkte] **Collections**

Es soll eine generische einfach-verkettete Liste implementiert werden. Vervollständigen Sie dazu den unten stehenden Programmcode. Zu implementieren sind die Methoden `public void add(T info)` und `public void delete(T info)` in der Klasse `List<T>` und die Methoden `public boolean hasNext()` und `public T next()` in der Klasse `ListIterator<T>`. Dabei gilt

- Die Methode `add(T info)` fügt am Ende der Liste ein Listenelement mit dem Wert `info` ein.
- Die Methode `delete(T info)` löscht das erste Listenelement, das den Wert `info` hat, falls ein solches exisitiert. Beachten Sie, dass, wenn es mehrere Listenelemente mit dem Wert `info` gibt, lediglich das erste gelöscht wird.
- Die Methode `hasNext()` gibt `true` zurück, falls es ein weiteres noch nicht vom Iterator betrachtetes Element in der Liste gibt, andernfalls `false`.
- Die Methode `next()` gibt den Wert des nächsten Listenelements in der Iteration wieder.

Der Anfang der Liste ist durch ein Listenelement markiert, dessen Inhalt leer (`null`) ist. D.h. auch eine leere Liste enthält ein Listenelement mit dem Wert `null`. Die folgende Abbildung zeigt eine leere String-Liste und eine String-Liste mit einem Element, wobei dieses Listenelement den Wert `"eins"` hat.




```
import java.util.Iterator;

public class List<T> implements Iterable<T>{

    private ListElement<T> sentinel;


    public List(){
        sentinel = new ListElement<T>(null);
    }







    public void add(T info){









    }

    public void delete(T info){
```

```
    }

    public Iterator<T> iterator() {
        return new ListIterator(this);
    }

    private class ListIterator implements Iterator<T>{
        private ListElement<T> current;

        public ListIterator(List<T> list) {
            this.current = (ListElement<T>) list.sentinel.getNext();
        }

        @Override
        public boolean hasNext() {









        }

        @Override
        public T next() {









        }
    }
}

public class ListElement<T>{
        private T info;
        private ListElement<T> next;

        public ListElement(T info){
            this.info = info;
            this.next = null;
        }

        public T getInfo(){
            return info;
        }
        public void setInfo(T info){
            this.info = info;
        }
```

```
        public ListElement<T> getNext(){
            return next;
        }

        public void setNext(ListElement<T> next){
            this.next = next;
        }

        @Override
        public String toString() {
            return info.toString();
        }
}
```

Aufgabe 8 [10 Punkte] **Callstack**

Betrachten Sie den folgenden Programmcode. Zeichnen Sie neben dem angegebenen Callstack die Namenstabellen, die sich nach den Aufrufen `new Name("Max", "Muster")` und `new Name("Alex", "Schulz")` ergeben. Zeichnen Sie die Namenstabellen zu den beiden im Callstack angegebenen Methoden und ergänzen Sie die notwendigen Referenzen. Welche Änderungen müssten Sie in Ihrer Abbildung vornehmen, um den Callstack einen Schritt später, also nach Abarbeitung des Befehls in Zeile 27, darzustellen? Welche Objekte wird der Garbage Collector auf dem Heap nach Abarbeitung des Befehls in Zeile 40 zur Bereinigung identifizieren? Welche Ausgabe liefert der Befehl `System.out.println(n1 + ", " + n2)` in Zeile 41?

neuerName(Name)	26	NT

main()	40	NT

```
1  public class Jasmin {
2
3
4      static class Vorname {
5          String vorname;
6
7          public Vorname(String vorname){
8              this.vorname = vorname;
9          }
10
11         @Override
12         public String toString(){
13             return vorname;
14         }
15
16     }
17
18     static class Name extends Vorname{
19         String nachname;
20
21         public Name(String vorname, String nachname){
22             super(vorname);
23             this.nachname = nachname;
24         }
25
26         void neuerName(Name n){
27             nachname = n.nachname;
28             n = new Name("Chris", "Mueller");
29         }
30
31         public String toString(){
32             return super.toString() + " " + nachname;
33         }
34
35     }
36
37     public static void main(String[] args){
38         Name n1 = new Name("Max", "Muster");
39         Name n2 = new Name("Alex", "Schulz");
40         n2.neuerName(n1);
41         System.out.println(n1 + ", " + n2);
42     }
43
44 }
```

Aufgabe 9 [9 Punkte] **Threads**

Geben Sie neun weitere mögliche Werteverläufe der statischen Variablen `number` bei Ausführung des folgenden Programms an.

```
public class Conflict extends Thread {

    static int number = 1;

    boolean add;

    public Conflict(boolean add) {
        this.add = add;
        this.start();
    }

    public void run() {
        // wait for 1-3 seconds
        try {sleep((long) (Math.random() * 2000 + 1000));
        } catch (InterruptedException e) {}

        int old = number;

        // wait for 1-3 seconds
        try {sleep((long) (Math.random() * 2000 + 1000));
        } catch (InterruptedException e) {}

        if (this.add)
            number = old + 1;
        else
            number = old * 3;
    }

    public static void main(String[] args) {
        for (int i = 1; i <= 3; i++) {
            Thread thread = new Conflict(i%2==0);
        }
    }
}
```

Mögliche Werteverläufe: