Vordiplomsklausur

Allgemeine Informatik I

Prof. J. Fürnkranz
Technische Universität Darmstadt — Wintersemester 2004/05
Termin: 8. 4. 2005

Name: **Vorname:** **Matrikelnummer:**

Fachrichtung:

Punkte: (1) .. (2) .. (3) .. (4) .. (5) .. (6) .. (7) .. **Summe:**

Aufgabe 1 (10 Punkte)

Ein Commodore C64-Rechner hatte seinerzeit einen 64 kilo-Byte großen Speicher, jede Speicherzelle war ein Byte groß.

1–a Wie viele verschiedene Speicherzellen hatte dieser Computer?

1–b Wie viele Bits benötigte man für die Adressierung, um eine dieser Speicherzellen auszulesen?

1–c Geben Sie die Binärdarstellung für die Speicher-Adresse (dezimal) 204 an.

1–d Geben Sie die Hexadezimaldarstellung für diese Adresse an.

Aufgabe 2 (12 Punkte)

Beantworten Sie die folgenden Fragen möglichst kurz und prägnant:

2–a Was versteht man unter Digitalisierung? Warum ist das wichtig für moderne Computer?

2–b Was ist die zentrale Idee, die die theoretischen Modelle der Turing-Maschinen und von Neumann-Rechnern verbindet, und die die Grundlage heutiger Computer-Systeme darstellt?

2–c Wofür benötigt man Programme wie Compiler oder Interpreter?

2–d Erklären Sie den Unterschied zwischen Compiler und Interpreter und deren Vor- und Nachteile.

2–e Skizzieren Sie, welche Strategie (kompilieren und/oder interpretieren) Java verfolgt, und benennen Sie die wesentlichen Komponenten.

Aufgabe 3 (8 Punkte)

In der Vorlesung haben Sie die logischen Operationen $\wedge$ (und), $\vee$ (oder), $\leftrightarrow$ (exklusives oder) und $\neg$ (Negation) kennen gelernt.

3–a Geben Sie für den logischen Ausdruck $(\neg a \wedge b) \vee (a \wedge \neg b)$ die Wahrheitstafel an. Geben Sie hierbei auch für jeden Teilausdruck den Wahrheitswert an.

a	b	$(\neg a \wedge b)$	$(a \wedge \neg b)$	$(\neg a \wedge b) \vee (a \wedge \neg b)$
0	0			
0	1			
1	0			
1	1			

3–b Geben Sie einen einfacheren logischen Ausdruck an, der die Ergebnis-Spalte der Wahrheitstabelle beschreibt.

Aufgabe 4 15 Punkte

Betrachten Sie das folgende KarelJ-Programm:

```
1   class A extends Robot
2   {
3     void move()
4     {
5       super.move();
6       super.move();
7     }
8
9     void bigStep() {
10      move();
11    }
12  }
13
14  class B extends A
15  {
16    void move()
17    {
18      super.move();
19      super.move();
20    }
21  }
22
23  task {
24    A aarel = new A(1, 1, 0, East);
25    B bbrel = new B(2, 1, 0, East);
26    A abrel = new B(3, 1, 0, East);
27
28    aarel.bigStep();
29    bbrel.bigStep();
30    abrel.bigStep();
31  }
```

Beantworten Sie die folgenden Fragen und begründen Sie Ihre Antworten kurz.

4–a An welcher Position *(Street, Avenue)* befinden sich die Roboter `aarel`, `bbrel` und `abrel` nach Ausführung des Programms?

4–b Geben Sie für jeden der drei Roboter den statischen und den dynamischen Typ an.

4–c Was würde sich ändern, wenn man in der Definition der Klasse `B` (zwischen Zeilen 20 und 21) ebenfalls eine `bigStep` Deklaration wie folgt einfügen würde:

```
void bigStep() {
  move();
}
```

4–d Was würde sich ändern, wenn man in der Definition der Klasse `B` (zwischen Zeilen 20 und 21) eine `bigStep` Deklaration wie folgt einfügen würde:

```
void bigStep() {
  super.move();
}
```

4–e Was würde passieren, wenn das Programm im `task`-Teil noch die Anweisungen

```
B barel = new A(4, 1, 0, East);   // Nach Zeile 26 einfuegen
barel.bigStep();                  // Nach Zeile 30 einfuegen
```

enthalten würde?

Aufgabe 5 (15 Punkte)

Implementieren Sie die folgende Methode

```
1       int[] append(int[] a, int[] b)
2       {
3         /* ... */
4       }
5
```

Die Methode soll für die beiden Arrays `a` und `b` einen neuen Array erzeugen, der zuerst die Elemente von `a` und dann die Elemente von `b` enthält. Dieser neue Array soll als Ergebnis der Methode zurückgegeben werden.

Beispielsweise soll die Methode für die Arrays

a

1	3	5	7	9

b

2	4	6	8

den Array

1	3	5	7	9	2	4	6	8

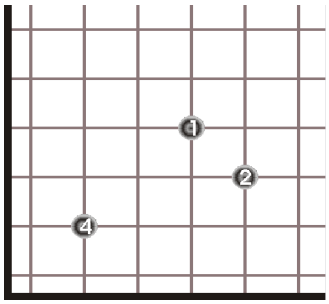
als Resultat liefern (aber sie muß natürlich auch für andere Arrays funktionieren).

Aufgabe 6 (25 Punkte)

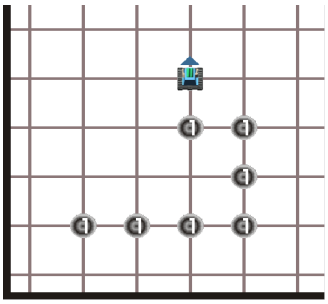
Programmieren Sie eine Klasse von Robotern namens `Zeichner`, die in der Lage ist, nach einem in der Roboter-Welt ausgelegten Plan eine Figur zu zeichnen. Die Kodierung des Pfades erfolgt auf folgende Weise:

- Die Anzahl der Beeper auf dem momentanen Feld gibt die Länge der Linie an, die gezogen werden soll (das momentane Feld wird mitgezählt).
- Nach Ziehen der Linie wird kontrolliert, ob auf dem linken Nachbarfeld ein Beeper liegt. Wenn ja, bewegt sich der Roboter ein Feld nach links, und setzt das Zeichnen in diese Richtung fort. Wenn nein, bewegt er sich nach rechts und versucht, die Zeichnung nach rechts fortzusetzen.
- Landet der Roboter auf einem Feld ohne Beeper, wird das Zeichnen beendet.

Zur Veranschaulichung sehen Sie im folgenden eine mögliche Startposition, einen Task, und die dazugehörige Endposition.




```
task {
  Zeichner karel =
    new Zeichner(2,2,0,East);
  karel.draw();
}
```



Im Detail sollen folgende Komponenten erstellt werden:

6–a eine Klassendefinition, die alle folgenden Methoden enthält.

6–b die Methode `move` soll so modifiziert werden, daß sich der Roboter solange vorwärts bewegt, bis er keinen Beeper mehr hält. Bei jedem Teil-Schritt soll der Roboter auf dem Zielfeld des Schrittes einen Beeper ablegen. Hält der Roboter keine Beeper, passiert nichts.

6–c eine Methode `pickAllBeepersButOne`, die alle auf einem Feld befindlichen Beeper aufhebt außer einem. Das heißt, wenn auf einem Feld n Beeper liegen, hebt der Zeichner $n-1$ Beeeper auf und läßt 1 Beeper liegen. Wenn sich kein Beeper auf dem Feld befindet, passiert nichts.

6–d eine Methode `beepersLeft`, die feststellt, ob sich links vom Roboter (mindestens) ein Beeper befindet, aber nichts am Zustand der Welt ändert.

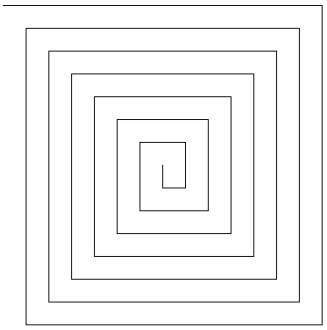
6–e eine Methode `draw`, die, unter Verwendung der vorhergehenden Methoden, die Zeichnung gemäß den Vorschriften ausführt.

Aufgabe 7 (15 Punkte)

Erweitern Sie das folgende Applet, sodaß es eine Spirale wie die abgebildete zeichnet. Die Anzahl der Umdrehungen soll jedoch von der Fensterbreite abhängen.

Gegeben sei folgendes Gerüst für das Applet.

```
1   class Spiral extends Applet
2   {
3     public void paint(Graphics g)
4     {
5       /* ... */
6     }
7
8     public void eineRunde(Graphics g, int x, int y, int length)
9     {
10      /* ... */
11    }
12  }
```



7–a Implementieren Sie die Methode `eineRunde`, sodaß sie eine Umdrehung der Spirale zeichnet. Die Runde soll beim Punkt (`x,y`) beginnen und beim Punkt (`x+25,y+25`) enden. Die obere und die rechte Seite haben jeweils eine Länge von `length`, die untere und die linke Seite haben jeweils eine Länge von `length-25`.

7–b Implementieren Sie die Methode `paint`, sodaß sie unter wiederholter Verwendung von `eineRunde` eine Spirale zeichnet. Die Spirale soll beim Punkt (0,0) beginnen, und über die gesamte Fensterbreite gehen, die Sie durch folgende Anweisung ermitteln können:

```
int l = g.getClipBounds().width;
```

Der Abstand zwischen den Linien der Spirale soll 25 Punkte betragen.

Hinweis: *Das Zeichnen einer weiteren Umdrehung macht nur solange Sinn, solange die kürzeren Seiten eine Länge* ≥ 0 *haben.*

Lösung 1

1–a 64kByte = 64×1024 Bytes = 65,536 Bytes $\Rightarrow$ 65,536 Adressen

1–b Man benötigt 16 Bits ($2^{16} = 65,536$).

1–c 11001100_2

1–d CC_{16} (da $1100_2 = C_{16}$)

Lösung 2

2–a Moderne Rechner sind Digitalrechner, d.h., jegliche Information (Zeichen, Zahlen, Texte, Bilder, Musik, Videos, etc.) muß in digitale Form gebracht werden, d.h. durch eine Folge von 0 und 1 dargestellt werden.

2–b Die Grundidee ist, daß Programme wie Daten gespeichert werden können. Das ermöglicht die Konstruktion einer "universellen" Rechenmaschine, auf der beliebige Berechnungen bzw. Informationsverarbeitungsschritte durchgeführt werden können.

2–c Compiler und Interpreter übersetzen Programme von einer Programmier-Sprache in eine andere, zumeist von einer höheren Programmiersprache in Maschinensprache.

2–d Compiler übersetzen das Programm einmalig, Interpreter übersetzen das Programm vor bzw. während jeder Ausführung neu. Compiler produzieren schnellere Programme, Interpreter erlauben höhere Interaktivität.

2–e Java verfolgt einen Mittelweg: Der Java Quell-code wird in den Java Byte Code kompiliert, dieser wird auf der Java Virtual Machine interpretiert.

Lösung 3

	a	b	$(\neg a \wedge b)$	$(a \wedge \neg b)$	$(\neg a \wedge b) \vee (a \wedge \neg b)$
	0	0	0	0	0
3–a	0	1	1	0	1
	1	0	0	1	1
	1	1	0	0	0

3–b Der vereinfachte Ausdruck lautet: $a \leftrightarrow b$.

Lösung 4

4–a `aarel`: (1,3)

`bbrel`: (2,5)

`abrel`: (2,5)

4–b statischer Typ von `aarel` : `A`

dynamischer Typ von `aarel` : `A`

statischer Typ von `bbrel` : `B`

dynamischer Typ von `bbrel` : `B`

statischer Typ von `abrel` : `A`

dynamischer Typ von `abrel` : `B`

4–c Es würde sich nichts ändern, da diese Deklaration der Vererbung entspricht.

4–d `bbrel` und `abrel` würden nun nur mehr zwei Schritte machen.

4–e Das Programm kompiliert nicht mehr, da man einer Unterklasse keine Objekte einer Überklasse zuordnen kann, weil diese möglicherweise nicht alle Fähigkeiten der Überklasse haben.

Lösung 5

```
1       int[] append(int[] a, int[] b) {
2
3         int[] c = new int[a.length + b.length];
4
5         for (int i = 0; i < a.length; i++) {
6           c[i] = a[i];
7         }
8
9         for (int i = 0; i < b.length; i++) {
10          c[i+a.length] = b[i];
11        }
12
13        return c;
14      }
15
```

Lösung 6

```
1   class Zeichner extends Robot
2   {
3      void move()
4      {
5         while (anyBeepersInBeeperBag()) {
6            super.move();
7            putBeeper();
8         }
9      }
10
11     void pickAllBeepersButOne()
12     {
13        if (nextToABeeper()) {
14          while (nextToABeeper())
15             pickBeeper();
16          putBeeper();
17        }
18     }
19
20     boolean beepersLeft()
21     {
22        turnLeft();
23        super.move();
24
25        boolean value = nextToABeeper();
26
27        loop (2)
28           turnLeft();
29
30        super.move();
31        turnLeft();
32
33        return value;
34     }
35
36     void draw()
37     {
38        while (nextToABeeper()) {
39
40           pickAllBeepersButOne();
41           move();
42
43           if (beepersLeft()) {
44             turnLeft();
45           }
46           else {
47             loop(3)
48               turnLeft();
49           }
50
51           super.move();
52        }
53     }
54  }
55
```

Lösung 7

```
1     public void paint(Graphics g)
2     {
3        int l = g.getClipBounds().width;
4        int a = 0;
5        int b = 0;
6
7        while (l >= 25) {
8           eineRunde(g, a, b, l);
9           a += 25;
10          b += 25;
11          l -= 50;
12       }
13    }
14
15    public void eineRunde (Graphics g, int x, int y, int length)
16    {
17       g.drawLine( x,        y,        x+length, y        );
18       g.drawLine( x+length, y,        x+length, y+length );
19       g.drawLine( x+length, y+length, x+25      y+length );
20       g.drawLine( x+25,     y+length, x+25,     y+25     );
21    }
```