Ludwig-Maximilians-Universität München SS 2006
Institut für Informatik Übungsblatt 5
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

Informatik II Musterlösung

Zu jeder Aufgabe ist eine Datei abzugeben, deren Name rechts in der Aufgabenüberschrift steht. Stellen Sie die Dateien in ein extra Verzeichnis (mit beliebigem Namen) und packen Sie dieses zu einem ZIP-Archiv. Geben Sie dieses, wie üblich, per UniWorx ab.

Aufgabe 5-1 Beschäftigte bei Behörden (`Clerk.java`, 12 Punkte)

Wir modellieren einen Beschäftigten bei einer Behörde aus der Sicht des Personalreferats; naturgemäß ist unsere Modellierung *stark* vereinfacht.

Für das Personalreferat ist ein Beschäftigter gegeben durch

- seinen Nach- und Vornamen (jeweils vom Typ `String`)
- das Jahr seiner Geburt (von Typ `int`)
- seine Besoldungskategorie (vom Typ `int`)
- einen booleschen Wert, der anzeigt, ob der Beschäftigte verheiratet ist.

Ziel der Aufgabe ist es, einen Beschäftigten zu modellieren und aus den gegebenen Daten die monatliche Besoldung zu ermitteln, die sich wie folgt berechnet:

Der Grundsold von €2000 wird um $x \cdot 17/9$ Prozent erhöht, wobei x die Besoldungskategorie ist. Beginnend mit dem Jahr, in dem das 30. Lebensjahr vollendet wird, wird der so errechnete Betrag alle 5 Jahre zusätzlich um 1 Prozent erhöht. Der Zuschlag für Verheiratete beträgt 12.3 Prozent des Grundsolds.

a) Definieren Sie eine Klasse `Clerk`, deren Attribute einen Angestellten (wie oben modelliert) beschreiben.

b) Erweitern Sie Ihre Klasse um einen Konstruktor `Clerk(String firstName, String lastName, int yearOfBirth, int salaryClass, boolean married)`, der die Instanzvariablen eines Beschäftigten mit den übergebenen Werten vorbesetzt.

c) Erweitern Sie Ihre Klasse um eine Methode `void promote(int newSalaryClass)`, die einen Beschäftigten in eine neue Gehaltsstufe einordnet.

d) Erweitern Sie Ihre Klasse um Methoden `void marry()` und `void divorce()`, die den Familienstand verheiratet bzw. nicht verheiratet verändern.

e) Erweitern Sie Ihre Klasse um eine Methode `double salary(int thisYear)`, die den Sold bestimmt. In der Variablen `thisYear` wird das Jahr übergeben, für das der Sold berechnet werden soll.

f) Erweitern Sie Ihre Klasse um eine Methode `boolean envies(Clerk c)`, die genau dann `true` zurückliefert, wenn der Beschäftigte, bei dem die Methode aufgerufen wird, den Beschäftigten `c` beneidet. Dies ist der Fall, wenn

   - der Beschäftigte bereits älter ist als c
   - aber im Jahr 2006 einen geringeren Sold bezieht.

g) Erweitern Sie Ihre Klasse um eine Prozedur `main`, die zwei Beschäftigte erzeugt, und die folgenden Aktionen durchführt:

   - Ausgabe des Solds beider Beschäftigten auf dem Bildschirm.
   - Der erste Beschäftigte heiratet; danach erneute Ausgabe des Sold beider Beschäftigten.
   - Der zweite Beschäftigte wird in die Gehaltsklasse 7 übernommen.

- Es soll auf dem Bildschirm ausgegeben werden, ob der erste Angestellte den zweiten beneidet.
- Es soll auf den Bildschirm ausgegeben werden, ob der zweite Angestellte sich selbst beneidet.

h) Schreiben Sie eine Prozedur `boolean allHappy(Clerk[] cs)`, die für eine Reihung von `Clerk`-Objekten bestimmt, ob alle Beschäftigten glücklich sind, d.h. ob bei einem paarweisen Vergleich `envies` stets `false` liefert. Fügen Sie einen Test dieser Prozedur in die `main`-Prozedur ein!

Lösung:

```
public class Clerk { /* Teilaufgabe (a): Instanzenvariablen mit Defaultwerten */
    private String firstName, lastName;
    private int yearOfBirth;
    private int salaryClass;
    private boolean married;

    /*
     * wegen der vermeidung von magic numbers hier der Grundsold und die
     * weiteren Konstanten
     */
    private static final double basicSalary = 2000.0;
    private static final double classMultiplier = 17.0 / 9.0;
    private static final double marriageBonus = 12.3;
    private static final int year = 2006;

    public String toString() {
        return firstName + " " + lastName + "(born " + yearOfBirth + ", " + (married?"married":"ba
    }

    /**
     * Teilaufgabe (b): Initialisieren der Instanzenvariablen per Konstruktor
     */
    public Clerk(String firstName, String lastName, int yearOfBirth,
            int salaryClass, boolean married) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.yearOfBirth = yearOfBirth;
        this.salaryClass = salaryClass;
        this.married = married;
    }

    /** Teilaufgabe (c): Befoerdern eines Angestellten */
    public void promote(int newSalaryClass) {
        salaryClass = newSalaryClass;
    }

    /** Teilaufgabe (d): Heirat und Scheidung */
    public void marry() {
        married = true;
    }

    public void divorce() {
        married = false;
    }

    /** Teilaufgabe (e): Berechnung des Solds */
    public double salary(int thisYear) {
        /* berechnet den Sold */
```

```
        double thesalary;
        /* Grundsold + x * 17/9 Prozent mit x = salaryClass: */
        thesalary = basicSalary
        * ((100.0 + classMultiplier) / 100.0 * salaryClass);
        // System.out.println("s1: " + thesalary);
        /* Altersbonus */
        int age = thisYear - yearOfBirth;
        int nbonus = 0;
        if (age >= 30)
            nbonus = (age - 25) / 5;
        thesalary = ((100.0 + nbonus) / 100.0) * thesalary;
        /* Verheiratetenzuschlag */
        if (married)
            thesalary += ((100.0 + marriageBonus) / 100.0) * basicSalary;
        return thesalary;
    }

    /** Teilaufgabe (f): wer beneidet wen? */
    public boolean envies(Clerk c) {
        return (yearOfBirth < c.yearOfBirth) && (salary(year) < c.salary(year));
    }

    /** Teilaufgabe (g): main (enthaelt etwas mehr als verlangt) */
    public static void main(String[] args) {
        Clerk c1 = new Clerk("Jack", "Miller", 1970, 1, false);
        Clerk c2 = new Clerk("John", "Smith", 1980, 1, false);
        System.out.println(c1 + " earns " + c1.salary(year));
        System.out.println(c2 + " earns " + c2.salary(year));
        System.out.println(c1 + " envies " + c2 + ": " + c1.envies(c2));
        c2.promote(7);
        System.out.println(c2 + " earns " + c2.salary(year));
        System.out.println(c1 + " envies " + c2 + ": " + c1.envies(c2));
        System.out.println(c2 + " envies himself: " + c2.envies(c2));
        Clerk c3 = new Clerk("Clark", "Kent", 1962, 3, false);
        Clerk c4 = new Clerk("Jerry", "Maguire", 1974, 1, true);
        Clerk[] clerks = new Clerk[]{c1, c2, c3, c4};
        System.out.println(java.util.Arrays.toString(clerks) + "are happy: " + allHappy(clerks));
        // wir machen sie gluecklich
        c4.divorce();
        c2.promote(1);
        System.out.println(java.util.Arrays.toString(clerks) + "are happy: " + allHappy(clerks));
    }

    /** Teilaufgabe(h): Ist eine Reihung von Clerk-Objekten paarweise zufrieden? */
    public static boolean allHappy(Clerk[] cs) {
        for (Clerk c1 : cs) {
            for (Clerk c2 : cs) {
                if (c1.envies(c2)) {    // beachte die Nichtreflexivitaet!
                    return false;
                }
            }
        }
        return true;
    }
}
```

Aufgabe 5-2 **Keller als Klasse implementieren** (`ArrayStack.java`, 10 Punkte)

Implementieren Sie einen Keller (engl. *Stack*) als eine Klasse. Erweitern Sie hierzu das unten angegebene Gerüst.

```
public class ArrayStackSkeleton {
    /**
     * Hier werden die Stack-Einträge gespeichert.
     */
    private int[] data;

    /**
     * Das nächste zu schreibende Element (= Anzahl der Elemente auf dem Stack)
     */
    private int topOfStack;

    /**
     * Legt einen Stack mit der Anfangskapazität <code>initialCapacity</code>
     * an. Die Kapazität wirkt sich nicht auf das beobachtbare Verhalten aus
     * (sprich: sie wird bei Bedarf erweitert).
     */
    public ArrayStackSkeleton(int initialCapacity) {
        // TODO
    }

    /**
     * Lege einen leeren Stack an.
     */
    public ArrayStackSkeleton() {
        // TODO
    }

    /**
     * Lege einen neuen Stack an, belege ihn mit den Elementen
     * <code>elements</code>.
     */
    public ArrayStackSkeleton(int[] elements) {
        // TODO
    }

    /**
     * Copy constructor: Erzeuge eine neue Instanz als echte Kopie von
     * <code>other</code>.
     */
    public ArrayStackSkeleton(ArrayStackSkeleton other) {
        // TODO
    }

    /**
     * Stelle einen neuen Wert auf den Stack, erweitere gegebenenfalls die
     * Kapazität.
     */
    public void push(int value) {
        // TODO
    }

    /**
```

```
     * Entferne den obersten Wert vom Stack und gib ihn zurück.
     */
    public int pop() {
        // TODO
    }

    /**
     * Lies den obersten Wert vom Stack aus.
     */
    public int peek() {
        // TODO
    }

    /**
     * Gib den momentanen Stackzustand als String zurück.
     */
    public String toString() {
        // TODO
    }

    public static void main(String[] args) {
        ArrayStackSkeleton stack = new ArrayStackSkeleton();
        System.out.println(stack);
        stack.push(5);
        System.out.println(stack);
        stack.push(3);
        System.out.println(stack);
        System.out.println("-->" + stack.pop());
        System.out.println(stack);
        System.out.println("-->" + stack.peek());
        System.out.println(stack);
    }
}
```

Hinweise:

- Methode `toString()`: Wenn man diese parameterlose Methode in eine Klasse schreibt, kann man deren Instanzen direkt per `System.out.println()` ausgeben. Mit dem `+`-Operator kann man in Java nicht nur Zahlen addieren, sondern auch Strings zusammenhängen. Kommt ein Objekt, das kein String ist, in so einer „Addition“ vor, dann setzt Java `toString()` ein, um dieses Objekt zu konvertieren. Die untenstehende Klasse `IntArrayContainer` enthält eine `toString`-Implementierung. Ihre Implementierung soll zu folgender Ausgabe führen:

  ```
  ArrayStack[]
  ArrayStack[5]
  ArrayStack[5, 3]
  -->3
  ArrayStack[5]
  -->5
  ArrayStack[5]
  ```

- Weiterhin kann ihnen die Klasse `StringBuilder` und die Methode `System.arraycopy()` bei der Implementierung helfen. Schlagen Sie beides im JavaDoc nach.

- Copy-Konstruktoren: sind ein einfaches Mittel, um Objekte zu kopieren. Prinzip: Einer der Konstruktoren einer Klasse bekommt als Argument eine andere Instanz der selben Klasse. Nun wird der interne Zustand des neuen Objekts komplett von dem existierenden übernommen und fertig ist die Kopie. `ArrayStack` hat einen Copy-Konstruktor.

- Assertions: Manche Methoden können, abhängig vom momentanen Zustand des Stacks, manchmal nicht aufgerufen werden. Überprüfen Sie diese Fälle über Assertions.
- Pseudo-Methode `this()`: Damit kann man von einem Konstruktor einen anderen aufrufen, allerdings nur in der ersten Zeile, also bevor andere Anweisungen aufgeführt werden. `IntArrayContainer` ist auch hierfür ein Beispiel.

Beispiel für ein paar der erwähnten Konstrukte:

```
public class IntArrayContainer {

    private int[] data;

    public IntArrayContainer(int[] myData) {
        this.data = myData;
    }
    public IntArrayContainer() {
        this(new int[0]);
    }

    public String toString() {
        // Bei leeren Arrays ist das Ergebnis nicht optimal!
        StringBuilder sb = new StringBuilder();
        sb.append("| ");
        for(int i : this.data) {
            sb.append(i);
            sb.append(" | ");
        }
        String myString = sb.toString();
        return myString;
    }

    public static void main(String[] args) {
        System.out.println(new IntArrayContainer(new int[] { 1, 2, 3}));
        System.out.println(new IntArrayContainer(new int[] { 77 }));
        System.out.println(new IntArrayContainer(new int[0]));
    }
}
```

Die `main`-Methode erzeugt folgende Ausgabe:

```
| 1 | 2 | 3 |
| 77 |
|
```

Lösung:

```
public class ArrayStack {
    /**
     * Hier werden die Stack-Einträge gespeichert.
     */
    private int[] data;

    /**
     * Das nächste zu schreibende Element (= Anzahl der Elemente auf dem Stack)
     */
    private int topOfStack;
```

```
/**
 * Lege einen leeren Stack an.
 */
public ArrayStack() {
    this(5);
}

/**
 * Legt einen Stack mit der Anfangskapazität <code>initialCapacity</code>
 * an. Die Kapazität wirkt sich nicht auf das beobachtbare Verhalten aus
 * (sprich: sie wird bei bedarf erweitert).
 */
public ArrayStack(int initialCapacity) {
    assert initialCapacity >= 0;
    this.data = new int[initialCapacity];
    topOfStack = 0;
}

/**
 * Lege einen neuen Stack an, belege ihn mit den Elementen
 * <code>elements</code>.
 */
public ArrayStack(int[] elements) {
    this(elements.length);
    // Nicht schnell, aber dafür einfach:
    for (int elem : elements) {
        push(elem);
    }
}

/**
 * Copy constructor: Erzeuge eine neue Instanz als echte Kopie von
 * <code>other</code>.
 */
public ArrayStack(ArrayStack other) {
    this.data = new int[other.data.length];
    System.arraycopy(other.data, 0, this.data, 0, other.data.length);
    this.topOfStack = other.topOfStack;
}

/**
 * Stelle einen neuen Wert auf den Stack, erweitere gegebenenfalls die
 * Kapazität.
 */
public void push(int value) {
    if (this.topOfStack >= data.length) {
        extendCapacity();
    }
    this.data[this.topOfStack] = value;
    this.topOfStack++;
}

/**
 * Hilfsfunktion: Verdopple Kapazität.
 */
```

```
private void extendCapacity() {
    int newLength = this.data.length * 2;
    if (newLength == 0) {
        // Zur Sicherheit: Möglicherweise war die Anfangskapazität = 0
        newLength = 2;
    }
    int[] newData = new int[newLength];
    System.arraycopy(this.data, 0, newData, 0, this.data.length);
    this.data = newData;
}

/**
 * Entferne den obersten Wert vom Stack und gib ihn zurück.
 */
public int pop() {
    assert this.topOfStack > 0;
    this.topOfStack--;
    return this.data[topOfStack];
}

/**
 * Lies den obersten Wert vom Stack aus.
 */
public int peek() {
    assert this.topOfStack > 0;
    return this.data[topOfStack - 1];
}

/**
 * Gib den momentanen Stackzustand als String zurück.
 */
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("ArrayStack[");
    for (int i = 0; i < this.topOfStack; i++) {
        if (i > 0) {
            sb.append(", ");
        }
        sb.append(this.data[i]);
    }
    sb.append("]");
    return sb.toString();
}

public static void main(String[] args) {
    ArrayStack stack = new ArrayStack();
    System.out.println(stack);
    stack.push(5);
    System.out.println(stack);
    stack.push(3);
    System.out.println(stack);
    System.out.println("-->" + stack.pop());
    System.out.println(stack);
    System.out.println("-->" + stack.peek());
    System.out.println(stack);
}
```

```
}
```

Abgabe: Per UniWorx, bis spätestens Montag, den 5.6.2006 um 9:00 Uhr.