

Zentralübung zur Informatik I

Dr. Markus Schneider
Technische Universität München

Wintersemester 2000/2001
8. Januar 2001

Überblick

- Die Modifikatoren `public` und `private`
- Qualifizierte Namen (Punktnotation)
- Die Reihung (Arrays)
- Der Zusammenhang zwischen Rekursion und Iteration

Die Modifikatoren public und private

- **public** : Durch diesen Modifikator wird eine Klasse, ein Attribut oder eine Methode öffentlich, das bedeutet, eine Verwendung durch andere Klassen ist erlaubt.
- **private** : Durch diesen Modifikator wird ein Attribut oder eine Methode nichtöffentlich. Das heißt, ein Zugriff ist nur innerhalb der Klasse, in der das Attribut bzw. die Methode definiert wurde, möglich.
- **Anwendungsbeispiel:** Gesucht ist eine Klasse zur Beschreibung des Resultates der Abschlussklausur.
 - Anforderung:
 - Ein Attribut, das angibt, ob die Klausur bestanden wurde.
 - Eine weiteres Attribut, das angibt, ob abgeschrieben wurde.
 - entsprechende **get-** und **set-** Methoden
 - Wenn abgeschrieben wurde, ist die Klausur nicht bestanden.

```

class Klausurergebnis {
    private boolean hatBestanden;
    private boolean hatAbgeschriebe;
    Klausurergebnis () { hatBestanden = false; hatAbgeschriebe = false; }
    public boolean getHatBestanden () { return hatBestanden; }
    public boolean getHatAbgeschriebe() { return hatAbgeschriebe; }
    public void setHatBestanden (boolean x) { hatBestanden = x; return; }
    public void setHatAbgeschriebe (boolean x) { hatAbgeschriebe = x;
        return; }
}

```

- **Problem:**

- Die beiden **set** – Methoden sollten nicht unabhängig sein; bisher wäre es etwa möglich, dass ein Klausurteilnehmer abgeschriebe, aber dennoch bestanden hat.
- **Abhilfe:** Definition einer öffentlichen Operation **bestanden** und Kapselung der **set**–Methoden. Die Methode **bestanden** gewährleistet, dass nur bestanden hat, wer nicht abgeschriebe hat und die erforderliche Punktezahl hat.

```

class Klausurergebnis {
    private boolean hatBestanden;
    private boolean hatAbgeschrieben;
    Klausurergebnis () { hatBestanden = false; hatAbgeschrieben = false; }
    public boolean getHatBestanden () { return hatBestanden; }
    public boolean getHatAbgeschrieben() { return hatAbgeschrieben; }
    private void setHatBestanden (boolean x) { hatBestanden = x; return; }
    private void setHatAbgeschrieben (boolean x) { hatAbgeschrieben = x;
        return; }
    public void bestanden (boolean abges, boolean hatGenugPunkte) {
        if (abges) { setHatBestanden(false); setHatAbgeschrieben(true); }
        else { setHatAbgeschrieben(false);
            setHatBestanden(hatGenugPunkte); }
        return; }
}

```

- **Problem:**

Wie lässt sich die Klasse in eine ablauffähige Umgebung einbauen?

Qualifizierte Namen (*Punktnotation*)

- Durch einen qualifizierten Namen wird ein Attribut oder eine Methode eines **instanziierten** Objektes bezeichnet.
- **Struktur:**
 - objektName.attributName;
 - objektName.methodenName;

Beispiel:

- Die Klasse **Klausurergebnis** definiert nur die Eigenschaften, die wir von einem Objekt vom Typ **Klausurergebnis** erwarten.
- **Instanziierung:**
 - Klausurergebnis resultat = new Klausurergebnis();
 - **resultat** ist nun der Name des Objektes
 - boolean a = resultat.getHatAbgeschrieben(); // wurde abgeschlossen ?
 - resultat.bestanden(false, true); // setzt die privaten Attribute des Objektes
 - resultat.hatAbgeschrieben = true; // Fehlermeldung, da privates Attribut

Ablauffähiges Programm

```
class Klausurumgebung {  
    public static void main(String[] args) {  
        Klausurergebnis resultat = new Klausurergebnis();  
        resultat.bestanden(false, true);  
        boolean a = resultat.getHatAbgeschriben();  
        System.out.println(a + " " ^ +  
            resultat.getHatBestanden() ); }  
}
```

- **Beachte:**
 - Die beiden Klassen **Klausurergebnis** und **Klausurumgebung** können in verschiedenen Dateien stehen (bei größeren Programmen ist dies sinnvoll).
 - Wenn sie in verschiedenen Dateien stehen, müssen beide Dateien durch **javac** kompiliert werden;
 - es entsteht dann eine Datei **Klausurergebnis.class** und eine Datei **Klausurumgebung.class**.
 - Nur die Datei **Klausurumgebung.class** kann ausgeführt werden

Reihungen

- Eine Reihung (oder Array) erlaubt es, verschiedene Daten desselben Typs zu einem Objekt zusammenzufassen.
- **Syntax:**
 - `int[] zahlenreihe; // definiert eine Reihung ganzer Zahlen bel. Länge`
 - `zahlenreihe = new int[10]; // legt die Länge auf 10 fest`
 - `Student[] tabelle = new Student[10]; // definiert eine leereTabelle`
`// für 10 Studenten`
 - `int[] zahlen = {2, 4, 23, 187}; // definiert und initialisiert ein Array`
`// ganzer Zahlen der Länge 4`
 - `zahlen[2] = 24; // weist dem Element mit Index 2 den Wert 24 zu`
- **Beachte:** Bei einer Reihung hat das erste Element den Index 0
- **Beispiel:** Es soll eine Tabelle von 11 Klausurergebnissen erstellt werden. Bei allen Klausurergebnissen wurde jeweils bestanden und nicht abgeschrieben. Die Tabelle soll am Bildschirm ausgegeben werden.

Beispielprogramm: Reihung

```
class Klausurtabellenumgebung {
    public static void main (String[] args) {
        int laenge = 11;
        Klausurergebnis[] tabelle = new Klausurergebnis[laenge];
        for (int i = 0; i<laenge; i = i++) {
            tabelle[i] = new Klausurergebnis();
            tabelle[i].bestanden(false, true);
        }
        System.out.println( "Abgeschrieben \t Bestanden" );
        for (int i = 0; i<laenge; i = i++)
            System.out.println(
                tabelle[i].getHatAbgeschrieben() + ^^^\t" +
                tabelle[i].getHatBestanden() );
        }
    }
```

Zusammenhang: Rekursion vs. Iteration

Beispielprogramm: Größter gemeinsamer Teiler

```
class Funktionsdienst {  
    Funktionsdienst () {}  
    public int ggT_rek (int a, int b) {  
        return a==b ? a  
            : a>b ? ggT_rek(a-b, b)  
            : ggT_rek(a, b-a); }  
}  
class Funktionsumgebung {  
    public static void main (String[] args) {  
        Funktionsdienst f = new Funktionsdienst();  
        int a = f.ggT_rek (356, 10345);  
        System.out.println(a); }  
}
```

Größter gemeinsamer Teiler (Iterativ)

- Besonders einfach ist die Umwandlung, wenn die rekursive Funktion repetitiv ist:

```
public int ggT_rek (int a, int b) {return a==b ? a
      : a>b ? ggT_rek(a-b, b)
      : ggT_rek(a, b-a); }
```

- Schleifenstruktur: **While** Schleife
- Schleifenbedingung: Solange **a==b** nicht erfüllt ist, erfolgt ein rekursiver Aufruf => **while (!(a==b)) { }**
- Variablen: Da die repetitive Funktion zwei Variablen verwendet, genügen auch in der iterativen Form zwei Variablen **a** und **b**.
- iterative Form

```
public int ggT_iter (int a, int b) {
    while (!(a ==b)) {
        if (a>b) a = a-b ; else b = b-a; }
    return a;
}
```