3. AuD Tafelübung T-C3

Simon Ruderich

10. November 2010

Wiederholung: Methoden

- Kapselung von Aufgaben
- unabhängig vom restlichen Programm (globale Variablen ausgenommen)

Beispiel

```
public static void main(String[] args) {
    int a = 5;
    int b = 10;
    int c = rechne(a, b); // = 15
}
public static int rechne(int x, int y) {
    return x + y;
}
```

Idee

- sequentielle Programme bei großen Problemen unübersichtlich
- Funktionen können auf alle Daten zugreifen und manipulieren (problematisch!)
- Idee: Kapselung von Aufgaben in „Gebilden“ die ein Problem lösen und mit anderen „Gebilden“ kommunizieren $\Rightarrow$ Objektorientierung

Übersicht

- Verbund: Vereinigung von mehreren Datentypen
- Klasse: Bauplan
- Instanz/Exemplar: Ausprägung einer Klasse
- Variablen in Klassen: Attribute
- Sichtbarkeit
- Vererbung: „Übertragung“ von Variablen/Methoden zwischen Klassen

Verbund

- „Datentyp“ der mehrere Datentypen vereint

Beispiel

```
public class Verbundname {
    // Attribute
    int x;
    int y;
    boolean z;
}
public static void main(String[] args) {
    Verbundname a = new Verbundname();
    a.x = 10;
    a.y = 15;
    a.z = false;
    System.out.println(a.x + "," + a.y + "," + a.z);
}
```

Klassen

- Verbindung von Verbund und Methoden
- Bauplan für Objekte
- Klassen sind (prinzipiell) unabhängig vom restlichen Programm
- dienen zur Kapselung von Aufgaben

Klassen

Beispiel

```
public class Komplex {
    // Attribute
    double r = 0;
    double i = 0;

    // Methoden
    public double betrag() {
        return Math.sqrt(i*i + r*r);
    }
    public void add(Komplex x) {
        this.r += x.r;
        this.i += x.i;
    }
}
```

Objekte

- auch Instanzen oder Exemplare genannt
- Ausprägung einer Klasse
- jedes Objekt besitzt eigene *unabhängige* Variablen

Beispiel

```
Komplex a = new Komplex();
a.r = 10;
a.i = 2;
Komplex b = new Komplex();
b.r = 20;
b.i = 0;
a.add(b);
// a.r = 30, a.i = 2
// b.r = 20, b.i = 0
```

Konstruktor

- wird beim Erstellen des Objekts aufgerufen
- erlaubt es dem Objekt beim Erstellen Parameter zu übergeben
- `Klassenname()` impliziter Konstruktor
- eigener Konstruktor überschreibt den impliziten Konstruktor

impliziter Konstruktor

```
Komplex a = new Komplex();
```

Konstruktor

Beispiel

```
public class Komplex {
    // ...
    Komplex(double r, double i) {
        this.r = r;
        this.i = i;
    }
    Komplex(Komplex k) {
        this.r = k.r;
        this.i = k.i;
    }
}
Komplex a = new Komplex(); // Compiler-Fehler
Komplex b = new Komplex(5, 2); // b.r = 5, b.i = 2
Komplex c = new Komplex(b);    // c.r = 5, c.i = 2
```

Referenzvariablen (= Verweisveränderliche)

- zeigen auf Speicherbereich bzw. Objekt
- ähnlich zu Pointern/Zeigern (wie z. B. in C), erlauben allerdings keine Zeigerarithmetik
- werden automatisch für Objekte verwendet (gilt auch für Arrays und Strings!)
- mit `null` initialisiert

Referenzvariablen

Variable einfachen Typs

```
int a = 5;
int b = a; // Kopie
b = 10; // a = 5, b = 10
// Variablen a und b unabhaengig
```

Referenzvariable

```
Klasse c = new Klasse(); // mit Attributen x und y
c.x = 10;
c.y = 15;
Klasse d = c; // nur Referenz wird kopiert
d.x = 20; // c.x = 20, d.y = 15
// Variablen c und d zeigen auf das selbe Objekt!
```

(Instanz-)Methoden

- können nur auf Objekten aufgerufen werden!
- haben Zugriff auf Variablen des Objekts

Beispiel

```
public class Test {
    int c = 10;
    int add(int a, int b) { return a + b + c; }

    public static void main(String[] args) {
        Test.add(0,0); // Compiler-Fehler,
                       // Test kein Objekt
        Test a = new Test();
        a.add(0,0); // funktioniert, a ist Objekt
    }
}
```

Statische Methoden

- werden mit **static** deklariert
- können auf Objekten und Klassen aufgerufen werden
- haben keinen Zugriff auf Variablen des Objekts

Beispiel

```
public class Test {
    int c = 10;
    static int add(int a, int b) { return a + b; }

    public static void main(String[] args) {
        Test.add(0,0); // funktioniert
        Test a = new Test();
        a.add(0,0); // funktioniert auch, add()
    }               // hat keinen Zugriff auf c!
}
```

Instanzvariablen

- gehören zu einem Objekt
- jedes Objekt hat seine eigenen Instanzvariablen
- können von statischen Methoden nicht verwendet werden

Beispiel

```
public class Komplex {
    double r = 0;
    double i = 0;
}
```

Klassenvariablen

- werden mit **static** deklariert
- gehören zu einer Klasse
- werden von allen Objekten der Klasse geteilt
- können von allen Methoden verwendet werden

Klassenvariablen

Beispiel

```
public class Komplex {
    static int count = 0;
    Komplex() {
        count++;
    }
    public static void main(String[] args) {
        Komplex a = new Komplex();
        Komplex b = new Komplex();
        Komplex c = new Komplex();
        System.out.println(Komplex.count); // = 3
    }
}
```

Konstanten

- werden mit **final** deklariert
- sind unveränderlich
- werden meist in Großbuchstaben geschrieben
- können Zahlen oder Objekte sein

Beispiel

```
public class Konstanten {
    static final double PI = 3.1415;
    static final String ANSWER = "42";
    static final Komplex Z = new Komplex(-5, 2);

    public static void main(String[] args) {
        ANSWER = "47"; // Compiler-Fehler!
    }
}
```

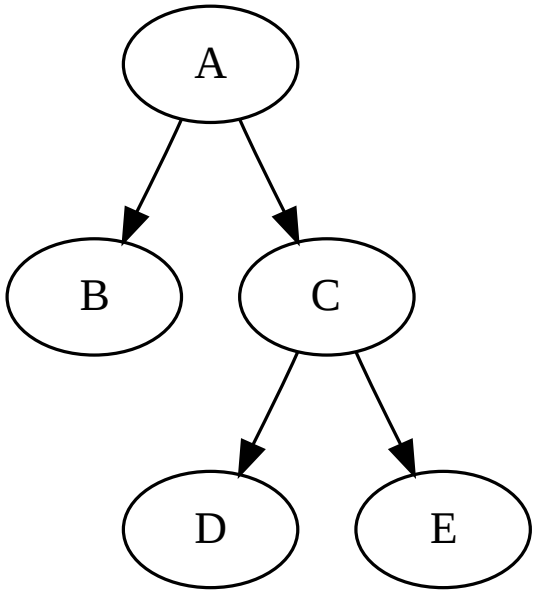
Sichtbarkeitsmodifikatoren

default	(keine Modifikator) Zugriff innerhalb des Pakets
public	öffentlich, Zugriff von überall
private	privat, nur die eigene Klasse darf darauf zugreifen
(protected)	

Vererbung

- Unterklasse übernimmt Variablen und Methoden der Oberklasse
- in Java mit dem Keyword **extends**
- jede Klasse kann (in Java) nur eine Oberklasse haben
- `Object` ist die Oberklasse aller Klassen

Vererbung




```
class A { }
class B extends A { }
class C extends A { }
class D extends C { }
class E extends C { }
```

Vererbung

Beispiel

```
class A {
    int a = 10;
    int add(int b) {
        return this.a + b;
    }
}
class B extends A {
    int sub(int b) {
        return this.a - b;
    }
}
A a = new A();
B b = new B();
System.out.println(a.add(5));
System.out.println(b.add(5) + "," + b.sub(5));
```

Polymorphe Variablen

- eine Variable kann im Laufe des Programms auf unterschiedliche Objekte/Klassen zeigen
- jede Variable hat einen

  statischen Typ wird zur Kompilierung bestimmt, legt fest welche Methoden verwendet werden können

  dynamischen Typ wird zur Laufzeit bestimmt, legt fest welche Methoden *aufgerufen* werden

- Variablen werden immer statisch aufgelöst
- Methoden werden immer dynamisch aufgelöst (Ausnahme, statische Methoden)

Polymorphe Variablen

Beispiel

```
// statischer Typ A, dynamischer Typ A
A a = new A();
// statischer Typ B, dynamischer Typ B
B b = new B();
// statischer Typ A, dynamischer Typ B
A c = new B();
```

Polymorphe Variablen

Beispiel Typsicherheit

```
class A {
    void test() { /* ... */ }
}
class B extends A {
    void test2() { /* ... */ }
}

A a = new B();
a.test();  // funktioniert
a.test2(); // Compiler-Fehler, statischer Typ A!
```

Polymorphe Variablen

Beispiel

```
class A {
    int x = 5;
    int y() { return 3; }
}
class B extends A {
    int x = 10;
    int y() { return 6; }
}

A a = new A(); B b = new B();
System.out.println(a.x + "," + b.x);       // 5,10
System.out.println(a.y() + "," + b.y()); // 3,6
A c = new B();
System.out.println(c.x + "," + c.y());     // 5,6
```

Polymorphe Variablen

Beispiel: static

```
class A {
    static int x = 5;
    static int y() { return 3; }
}
class B extends A {
    static int x = 10;
    static int y() { return 6; }
}

A a = new A(); B b = new B();
System.out.println(a.x + "," + b.x);       // 5,10
System.out.println(a.y() + "," + b.y());   // 3,6
A c = new B();
System.out.println(c.x + "," + c.y());     // 5,3
```

Fragen

Fragen?