Klassen und höhere Datentypen

Objekte, Felder, Methoden

Küchlin/Weber: Einführung in die Informatik

Klassen

- **Klasse** (*class*) stellt einen (i.a. benutzerdefinierten) Verbund-Datentyp dar
- Objekte sind Instanzen (*instances*) der Klasse
- Klasse bedeutet zweierlei
  - Menge aller Objekte dieses Typs
  - Deklaration, wie Objekte dieses Typs aufgebaut sind

Klassen

- Jedes Objekt hat individuellen Datensatz (Zustand)
- Objektzustand gespeichert in Instanzvariablen (*instance variables*)
  - heißen in jedem Objekt gleich, haben aber i.a. verschiedene Werte.
- Instanzmethoden
  - werden auf einem Objekt aufgerufen
  - laufen auf diesem
  - haben Zugriff auf dessen Instanzvariablen

Klassen

- Klassenvariablen
  - nur ein Satz pro Klasse
  - speichern globalen Zustand der Klasse
- Klassenmethoden
  - werden wie normale Funktionen aufgerufen
  - unabhängig von Objekten

Klassendeklaration

- Eine Klassendeklaration spezifiziert
  - die **Instanzvariablen**
    (*instance variables, non-static fields, attributes*)
  - die **Instanzmethoden**
    (*instance methods*, *non-static methods*)
  - die **Klassenvariablen**
    (*static fields*)
  - die **Klassenmethoden**
    (*static methods*)
  - die **Konstruktoren**
    mit denen neue Objekte des Typs initialisiert werden

Klassendeklaration

- Klassendeklaration
  - beginnt mit dem Schlüsselwort class
  - gefolgt vom Namen der Klasse
- Konvention:
  Klassennamen werden immer großgeschrieben

Klassendeklaration: Beispiele (1)

```
class Messwert2D {
        int x;              // x-Koordinate
        int y;              // y-Koordinate
        double wert;        // Messwert
}
```

Reiner Verbundtyp (Container)
Keine Methoden

Klassendeklaration: Beispiele (2)

```
class Time {
  int sec=0;      // seconds, 0 <= sec < 60
  int min=0;      // minutes, 0 <= min < 60
  int hrs=0;      // hours, 0 <= hrs

  void tick() {
    sec++;
    if ( sec >= 60 ) {
          sec -= 60;
          min++;
          if ( min >= 60 ) {
                min -= 60;
                hrs++;
          }
    }
  }
}
```

Klassendeklaration: Beispiele (3)

- Konstruktor ist „Methode" mit dem Namen der Klasse
- Im Beispiel: M2DT()
- Wird automatisch bei Erzeugung eines neuen Objekts ausgeführt (Initialisierung)
- new T()
  erzeugt neues Objekt der Klasse T

```
class M2DwithTime {
  Messwert2D m;      // Messwert
  Time t;            // Zeitpunkt
  M2DwithTime() {    // Konstruktor
    m    = new Messwert2D();
    t    = new Time();
  }
}
```

Deklaration von Variablen mit Klassentyp

- K v; deklariert Variablen v vom Typ K
- Wert von new K() ist (Referenz auf) neues Objekt
- Darum häufig:

```
K v = new K();
```

Initialisierung und Konstruktoren

- new K()
  - reserviert nicht nur den benötigten Speicherplatz für das neue Objekt
  - sorgt auch für dessen Initialisierung
  - führt die Konstrukturen aus

Initialisierung und Konstruktoren

- **Konstruktoren**
  - haben Ähnlichkeit mit Methoden
  - haben aber keinen Ergebnistyp (auch nicht void)
  - tragen immer den Namen ihrer Klasse

Zugriff auf Instanz-Variablen und -Methoden

- Instanzvariable wird angesprochen durch:

  objekt.variablenname

- Instanzmethode wird aufgerufen durch:

  objekt.methodenname(arg1, ..., argn)

```
Time t = new Time();
t.sec   = 21;                  // Setze sec in t auf 21
t.min   = 35;                  // Setze min in t auf 35
t.hrs   = 2;                   // Setze hrs in t auf 2
t.tick();                      // Rufe tick() auf Objekt t auf
```

Klassenvariable und Klassenmethoden

- **Klassenvariable**
  - als static deklariert
  - nur ein einziges Mal in der Klasse vorhanden
  - egal wie viele Objekte dieser Klasse es gibt
  - Wird angesprochen durch

    Klassenname.variablenname

    oder

    variablenname (innerhalb der Klasse)

Klassenvariable und Klassenmethoden

- **Klassenmethoden**
  - als static deklariert
  - kann nicht auf einem Objekt aufgerufen werden
  - Wird angesprochen durch

    Klassenname.methodenname(. . .)

    oder

    methodenname(. . .)

    (innerhalb der Klasse)

Klassenvariable und Klassenmethoden

- Klassenvariablen typischerweise für globale Buchhaltungsaufgaben
- Klassenmethoden typischerweise dann, wenn kein Objekt bei der Operation besonders ausgezeichnet

Pakete und Namensräume

- Namensräume sind vor allem bei großen Softwaresystemen wichtig
  - In denen mehrere Entwickler zusammenarbeiten
  - „Fremder“ (oder „alter“) Code integriert wird
- Ohne Namensräume Gefahr groß, dass in einem anderen Programmteil der gleiche Name (für eine Klasse) schon gebraucht wurde

Pakete und Namensräume

- Eine Deklaration import PName.* macht die im Paket PName deklarierten Klassen auch ohne Verwendung des vollständig qualifizierten Namens bekannt
  - Diese können also in Kurzform aufgerufen werden, es muss also nicht der vollständig qualifizierte Namen verwendet werden
  - „Intern" wird aber immer der vollständig qualifizierte Namen verwendet
  - Die Klassen im Paket java.lang werden immer automatisch importiert

Kapselung und Zugriffskontrolle

- Jeder Name eines Mitglieds (Attribut oder Methode) einer Klasse hat einen **Sichtbarkeitsbereich** (*scope*)
  - Ein Mitgliedsname kann nur dort verwendet werden, wo er sichtbar ist
- Zugriffe auf das Mitglied unterliegen damit einer **Zugriffskontrolle** (*access control*)
- In Java gibt es die 4 Sichtbarkeitsbereiche: **global**, **geschützt**, **Paket** und **Klasse**

Kapselung und Zugriffskontrolle

- *Access modifiers*:
  - public: öffentlich, global
  - protected: geschützt
  - private: eigene Klasse
  - keiner: eigenes Paket
- Protected bedeutet:
  Zugriff von abgeleiteten Klassen aus

Kontrakt und Aufrufschnittstelle

- **Bsp.:**

```
public class Date {          // the class name is public
  private int day;           // private field
  private int month;         // private field
  private int year;          // private field
  public void m1() {         // public method
   // ...
  }
  void m2() {                // package method
    // ...
  }
  private boolean m3() {     // private method
   // ...
  }
}
```

Kapselung und Zugriffskontrolle

- **Problem:**
  in Java Zugriffsschutz auf Klassenebene und nicht auf Objektebene
  - Eine Methode der Klasse A, die auf einem Objekt a1 abläuft, kann auf alle Attribute eines anderen Objekts a2 vom Typ A mit genau denselben Rechten zugreifen, als seien es ihre eigenen

Kontrakt und Aufrufschnittstelle

- Die zugänglichen Daten und Methoden einer Klasse bilden die **(Aufruf-)Schnittstelle** (*call interface*) der Klasse nach außen
- Interne Daten und Methoden bleiben verborgen und können geändert werden, ohne dass extern davon etwas zu merken ist
- **-> Geheimnisprinzip**
  (*principle of information hiding*)

Kontrakt und Aufrufschnittstelle

- Syntax und Semantik der Aufrufschnittstelle bilden den **Kontrakt** (*contract*) zwischen den Programmierern der Klasse und ihren Nutzern
- Wenn und solange der Kontrakt beachtet wird, funktioniert die Zusammenarbeit zwischen den Nutzern der Klasse und dem Code der Klasse selbst
- Die Art und Weise, wie die Klasse ihre Seite des Kontrakts erfüllt, ist unerheblich und kann wechselnden Erfordernissen angepasst werden
- In bestimmten Fällen kann es nötig werden, weitere Zusicherungen in den Kontrakt aufzunehmen, z. B. Leistungsdaten
- **Kontrakt muss dokumentiert werden**