Kapitel 06

Klassen und Objekte

Fachgebiet Eingebettete Systeme und ihre Anwendungen
Prof. Dr. Andreas Koch

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhalt des 6. Kapitels
Klassen und Objekte

6.1 Was ist eine Klasse?

- Definition
- Die Klasse String

6.2 Klassendefinition

- äußerer Teil
- innerer Teil
  - Datenfelder
  - Konstruktor
  - Methoden

6.3 Variablen in Klassen

6.4 Zusammengesetzte Objekte

Was ist eine Klasse?

In Kapitel 5 wurde näher auf die eingebauten Datentypen eingegangen. Im Kapitel 6 wenden wir uns nun den Klasse, die ebenfalls einen Datentyp repräsentieren, zu.

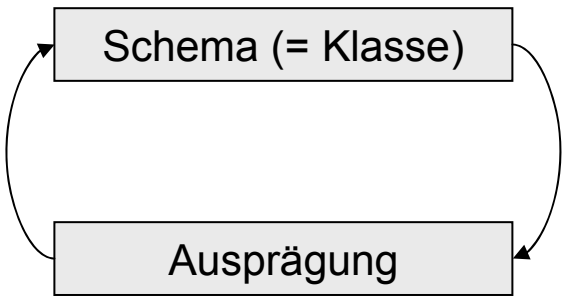
- Programmierkonzepte
  - prozedurale Programmierung
    - im Zentrum stehen Unterprogramme (Prozeduren)
  - objektorientierte Programmierung (Java)
    - im Zentrum stehen Datenstrukturen und somit Klassen und Objekte
- Definition Klasse aus Wikipedia
  - Klasse ist in der Objektorientierung ein abstrakter Oberbegriff für die Beschreibung der gemeinsamen Struktur und des gemeinsamen Verhaltens von Objekten (Klassifizierung).

Was ist eine Klasse?

- Prinzip der Koppelung
  - Es wird in der Regel nicht erlaubt, auf die Datenstrukturen selbst zuzugreifen, sondern nur mittels vordefinierter Methoden (Schnittstelle nach außen)
    - **`get`**-Methoden
    - **`set`**-Methoden (später mehr hierzu)
  - dadurch wird gewährleistet, dass nicht unvorhergesehene Dinge passieren können
    - man kann z. B. die Anzahl der Beeper oder die Position eines Roboters nicht direkt verändern, sondern nur mit den entsprechenden Methoden **`pickBeeper()`** oder **`move()`**

Was ist eine Klasse?

- Prinzip der Abstraktion
  - Klassen ermöglichen, reale Objekte abstrakt darzustellen. Es erfolgt eine schematische Definition der realen Objekte und der Zustände, die sie annehmen können.
  - Beispiel im Bezug auf die Datenfelder einer Klasse:




```
String Nachname, String Vorname;
```

„Thomas“, „David“

Beispiele von Klassen

- aus der KarelJ-Welt
  - **`Robot`** ist eine KarelJ-Klasse, die abstrakt Roboter definiert
- aus der Java-Welt
  - **`Applet`** ist eine Klasse für Internet-Applikationen
  - **`String`** ist eine Klasse für unveränderliche Zeichenketten.
    - Sie stellt verschiedene Methoden zur Inspektion und Konvertierung aus anderen Datentypen zur Verfügung.
    - **`String`** ist kein eingebauter Datentyp!
    - **`String`** ist eine Klasse und wird daher um Gegensatz zu **`int`** und **`double`**, welches eingebaute Datentypen sind, groß geschrieben.
    - Die Klasse für veränderliche Zeichenketten ist **`StringBuffer`**.

Die Klasse `String`

- Deklarieren
  - **`String s = „Dies ist ein String mit 35 Zeichen.“`**
    - Kurzform:
      `String x = new String("test")` entspricht **`String x = "test"`**
    - Ein **`String`** kann auch nicht druckbare Zeichen enthalten
    - spezielle Zeichencodes:

Zeichen	Bedeutung
\t	Horizontaler Tabulator
\n	Zeilenumbruch (Newline)
\f	Seitenumbruch (Formfeed)
\“	Doppeltes Anführungszeichen
\‘	Einfaches Anführungszeichen
\\	Backslash

Fortsetzung Klasse `String`

- Informationen zu **`String`**
  - **`int len = s.length();`**
    - Diese Methode gibt die Länge des **`String`**, bzw. **`StringBuffer`** zurück. Für den obigen **`String s`** liefert **`s.length()`** also den Wert 35.
  - **`char c = s.charAt(i);`**
    - Damit kann das Zeichen (Typ **`char`**) des **`String s`** an der **`i`**-ten Position bestimmt werden. Vorsicht: **`i`** läuft wie bei Arrays von 0 bis **`s.length-1`**.
  - **`String s2 = s1.substring(i,j);`**
    - **`i`** ist der Index bei dem der Substring beginnt, **`j`** ist der Index des Buchstabens, der als erstes nicht mehr Teil des Substrings ist.

Fortsetzung Klasse `String`

- Vergleich von Zeichenketten:
  - **`if (s1.equals(s2)) return s1;`**
    - bedeutet: Falls die **`Strings s1`** und **`s2`** gleich sind, wird die Bedingung der if-Schleife **`true`** und **`s1`** zurückgegeben.
    - Bem.: Der Code **`if (s1==s2) return s1;`** würde bedeuten, dass überprüft wird, ob beide Stringvariablen auf den identischen String weisen.
  - **`s1.equalsIgnoreCase(s2):`**
    - Hier wird bei der Überprüfung nicht zwischen Groß- und Kleinschreibung unterschieden.

6.2 Die Klassendefinition

Klassen lassen sich in einen äußeren und inneren Teil unterteilen.

- der äußere Teil:
  - ist in der Regel für die Benennung der Klasse zuständig
  - legt fest von welcher Klasse geerbt wird (optional).
    - auf Vererbung wird zu einem späteren Zeitpunkt noch einmal eingegangen

```
public class Ticketautomat extends Automat {
        //Innenteil der Klasse ausgelassen
}
```

Fortsetzung Klassendefinition

- der innere Teil:
  - Definition der Datenfelder, Konstruktoren und Methoden, die den Instanzen oder Objekten ihre Struktur und ihr Verhalten geben.
  - Die Reihenfolge dieser Hauptbestandteile sind in Java zwar nicht festgelegt, es hat sich jedoch eine Art Standard herausgebildet.

```
public class Klassenname {

        Datenfelder
        Konstruktoren
        Methoden
}
```

Fortsetzung Klassendefinition innerer Teil

```
public class Klassenname {
   >>  Datenfelder
       Konstruktoren
       Methoden
}
```

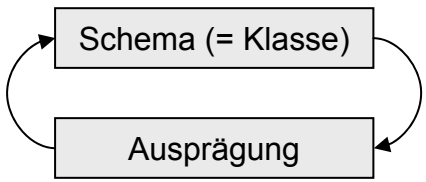
- Datenfelder
  - Datenfelder werden auch als Instanzvariablen oder Klassenattribute bezeichnet.
  - Sie können als kleine Bereiche innerhalb eines Objektes verstanden werden, in denen Werte gespeichert werden.
    - Beispiel KarelJ: Anzahl der Beeper, Position des Roboters
    - Beispiel Ticketautomat:

```
public class Ticketautomat {

        private int preis;
        private int bisherGezahlt;
        private int gesamtSumme;

        Konstruktor & Methoden ausgelassen
}
```

Fortsetzung Klassendefinition innerer Teil

```
public class Klassenname {
   Datenfelder
   Konstruktoren
   Methoden
}
```

- Konstruktor
  - Konstruktoren ermöglichen, dass ein Objekt nach seiner Erzeugung in einen gültigen Zustand versetzt wird.
  - Dieser Vorgang wird auch als Initialisierung bezeichnet.
  - Objekte bzw. Instanzen sind konkrete Ausprägungen einer abstrakten Klasse

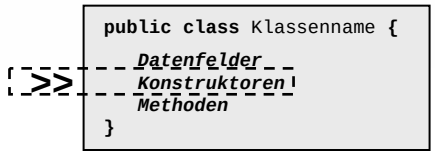


`String` Nachname, **`String`** Vorname

„Thomas“, „ David“

    - z.B. **`karel`** ist ein **`Robot`**
  - Definition von Klassen und Objekten
    - Klassen werden mittels **`class`** definiert und zur Compile-Zeit übersetzt
    - Objekte werden mittels **`new`** definiert und zur Laufzeit angelegt

Fortsetzung Klassendefinition innerer Teil



- Konstruktor
  - Ein Konstruktor einer Klasse ist eine bestimmte Art von Methode.
  - Syntaktische Eigenschaften
    - Der Name des Konstruktors muss dem der Klasse entsprechen.
    - Der Konstruktor darf keine Angaben über einen Rückgabetyp haben. Beachten Sie, **`void`** ist eine solche Angabe und hat somit vor einem Konstruktor nichts zu suchen.
    - Eine Methode, die den selben Namen wie die Klasse hat, ist automatisch ein Konstruktor.
  - Eine Klasse darf mehrere gleichnamige Konstruktoren besitzen.
    - genauso auch mehrere gleichnamige Methoden, dazu später mehr.

Fortsetzung Klassendefinition innerer Teil

```
public class Klassenname {
   Datenfelder
>> Konstruktoren
   Methoden
}
```

- Konstruktorverwendung
  - Ein Konstruktor einer Klasse wird normalerweise in einer einzigen spezifischen Situation aufgerufen: bei der Erzeugung eines Objekts dieser Klasse mit **new**.
  - Der Klassenname hinter **new** ist also genauer gesagt der Name des Konstruktors bei seinem Aufruf:

```
String str = new String ( "Hello" );
```

  - Damit wird auch klar, was die Klammern hinter dem Klassennamen in einem **new**–Ausdruck sollen:
    - Das ist einfach die Parameterliste für den Aufruf des Konstruktors.
    - Ein leeres Klammerpaar bedeutet dann, dass ein Konstruktor mit leerer Parameterliste aufgerufen wird.

Fortsetzung Klassendefinition innerer Teil

```
public class Klassenname {
   Datenfelder
>> Konstruktoren
   Methoden
}
```

- Konstruktorverwendung am Beispiel „Ticketautomat“

```
Ticketautomat gondelbahn = new Ticketautomat ( 500 );
```

```
public class Ticketautomat {
        Datenfelder ausgelassen

        /**
        * Erzeugt gültigen Automat, Preis(cent)
        */
        public Ticketautomat (int ticketpreis){
                preis = ticketpreis;
                bisherGezahlt = 0;
                gesamtSumme =0;
        }
        Methoden ausgelassen
}
```

Fortsetzung Klassendefinition innerer Teil

- Methoden
  - Methoden erlauben es, den Zustand eines Objektes zu ändern, bzw. ein vom Zustand abhängiges Verhalten an den Tag zu legen.
    - Beipiel KarelJ: **`move(), putBeeper(),`** etc.
  - Bestehen aus einem Kopf und einem Rumpf.
    - vergleichbar mit dem äußeren und inneren Teil einer Klasse
  - Methodenkopf:
    - Kommentar was die Methode tut
    - Signatur der Methode (siehe nächste Folie)
  - Methodenrumpf:
    - Enthält die Deklarationen und die Anweisungen. Diese legen fest, was im Falle eines Methodenaufrufs passiert.

Fortsetzung Klassendefinition innerer Teil

- Methodensignatur
  - Die Signatur einer Methode setzt sich zusammen aus
    - dem Rückgabe- / Ergebnistyp der Methode (bzw. **`void`**),
    - dem vollständigen Namen,
    - der Anzahl der formalen Parameter (potentiell auch ohne Parameter) und
    - den Typen der formalen Parameter und ihrer Reihenfolge in der Parameterliste.
  - Unterschiede zwischen der Signatur von Methoden und der Signatur von Konstruktoren
    - Methoden haben im Gegensatz zu Konstruktoren einen Rückgabetyp.

Fortsetzung Klassendefinition innerer Teil

- Methoden
  - Sondierende Methoden (get-Methoden)
    - liefern Informationen über ein Objekt

```
public class Ticketautomat {

        Datenfelder & Konstruktor ausgelassen

        /**
        * Liefert den Preis in Cent
        */
        public int gibPreis(){
                return preis;
        }

}
```

Fortsetzung Klassendefinition innerer Teil

- Methoden
  - Verändernde Methoden (set-Methoden)
    - ändern den Zustand eines Objekts

```
public class Ticketautomat {

        Datenfelder & Konstruktor ausgelassen

        /**
        * Nimmt den übergebenen Betrag als
        * Anzahlung
        */
        public void geldEinwerfen(int betrag){
                bisherGezahlt += betrag;
        }
}
```

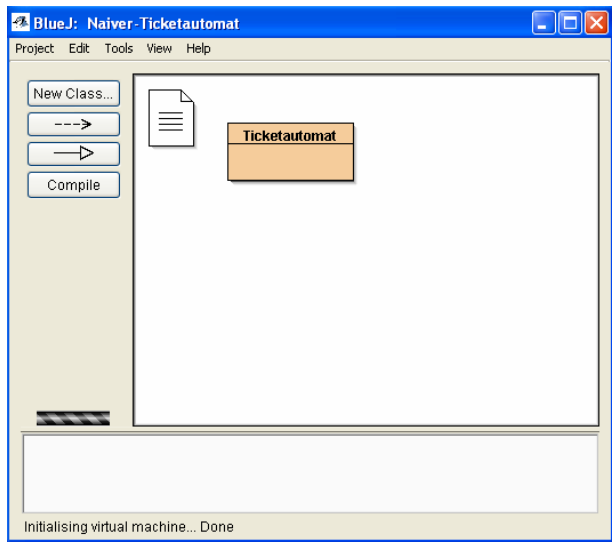
Fortsetzung Klassendefinition innerer Teil

- Methoden mit lokalen Variablen
  - Dabei handelt es sich um Variablen die innerhalb einer Methode deklariert und benutzt werden.
  - Es kann nur innerhalb der Methode auf sie zugegriffen werden. Ihre Lebensdauer entspricht der der Methode.

```
public int wechselgeldAuszahlen(){
        int wechselgeld;
        wechselgeld = bisherGezahlt;
        bisherGezahlt = 0;
        return wechselgeld;
}
```

Beispiel Verbesserter Ticketautomat Live-Beispiel

- BlueJ Ansicht
  - Kompilieren
  - Instanz erzeugen
- Klassendefinition
  - Datenfelder
  - Konstruktor
    - erwartete Parameter
  - Methoden
    - Übersicht – Welche Methoden gibt es?
    - sondierende (get)
    - verändernde (set)
- Kommentierung
  - Generell
  - JavaDoc – Variablen
  - Implementierungs- versus Interface-Ansicht



6.3 Variablen in Klassen

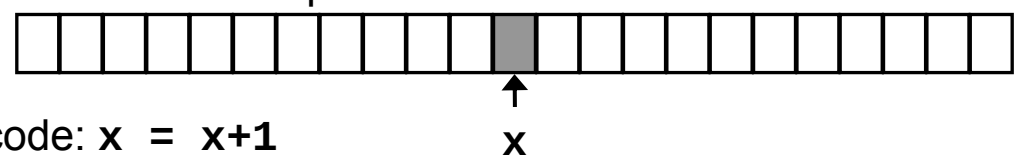
In Kapitel 6.2 haben wir drei Arten von Variablen kennen gelernt.

- Datenfelder
  - werden außerhalb von Methoden und Konstruktoren definiert und
  - sind für die ganze Klasse sichtbar,
- Formale Parameter
  - werden im Methoden- oder Konstruktorkopf definiert,
  - erhalten ihre Werte von außen und
  - sind nur innerhalb der Methode oder des Konstruktors sichtbar.
- Lokale Variablen
  - werden in Methodenrümpfen definiert und
  - sind nur innerhalb des Blocks sichtbar.

Variablen Fortsetzung

Allgemein:

- Der Begriff *Variable* bedeutet in Mathematik und Programmierung etwas fundamental Verschiedenes!
  - *Mathematik*: Eine Variable ist ein Platzhalter für die Elemente einer Menge.
    - Beispiel: Für alle reellen Zahlen x (kurz: für $x \in \mathbb{R}$) für die gilt $x^2 \geq 4$.
  - *Programmierung*: Eine Variable ist ein zusätzlicher, symbolischer Name für eine feste Speicheradresse.
    - Beispiel: schematische Speicheransicht



    - Beispielcode: **x = x+1**
    - Liest den Inhalt der Speicherzelle auf die **x** verweist in das Rechenwerk, addiert 1 dazu und schreibt das Ergebnis wieder in dieselbe Speicherzelle zurück.

6.4 Zusammengesetzte Objekte

- Objekte von eingebauten Typen sind aus Sicht eines Java–Programmierers *atomar*.
  - *Das heißt*: Eine etwaige weitere interne Struktur und Zerlegbarkeit eines solchen Objekts auf Maschinenebene wird im Java–Quelltext nicht sichtbar.
  - *Ausnahme*: Es gibt Operatoren in Java zur logischen Verknüpfung von Zahlen „Bit–für–Bit“.

    → Betrachten wir in dieser Veranstaltung nicht weiter (für Interessierte: Stichwort *Bitlogik*).
- Objekte von Klassen sind hingegen im allgemeinen aus mehreren Variablen von eingebauten Typen und/oder Klassen zusammengesetzt.

6.4 Zusammengesetzte Objekte

- Beispiel:

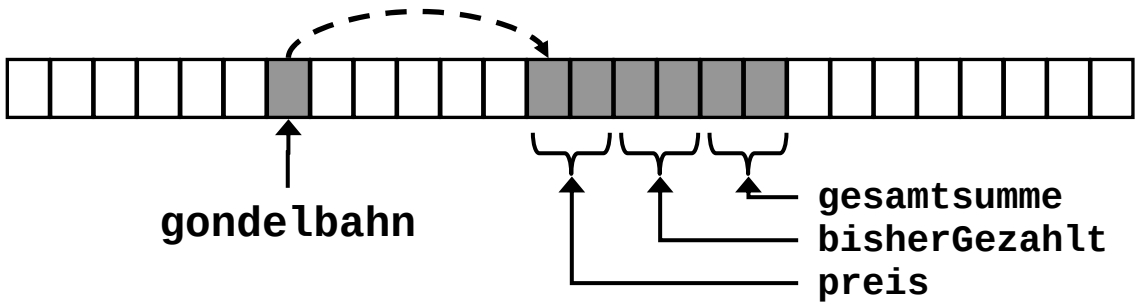
```
public class Ticketautomat {

        public int preis;
        private int bisherGezahlt;
        private int gesamtsumme;

        Konstruktor &
        Methoden ausgelassen
}
```

```
Ticketautomat gondelbahn = new Ticketautomat ();
```

- Erläuterung:
  - Mit obigem Code ist eine Klasse namens **`Ticketautomat`** definiert. Es wurde zudem eine Variable **`gondelbahn`** vom Typ **`Ticketautomat`** deklariert und initialisiert.
  - Die genauen Details der Syntax (insbesondere der Sinn von **`public`**) werden erst später in der Vorlesung behandelt.

6.4 Zusammengesetzte Objekte

- *Erinnerung*: Variablen von Klassen sind nur Verweise auf die Speicheradresse.



- Hinweis: Auf das Objekt **`preis`** vom eingebauten Typ **`int`** kann hier direkt zugegriffen werden (**`public`**).
  - **`gondelbahn.preis`**

6.4 Zusammengesetzte Objekte

- Beispiel 2:

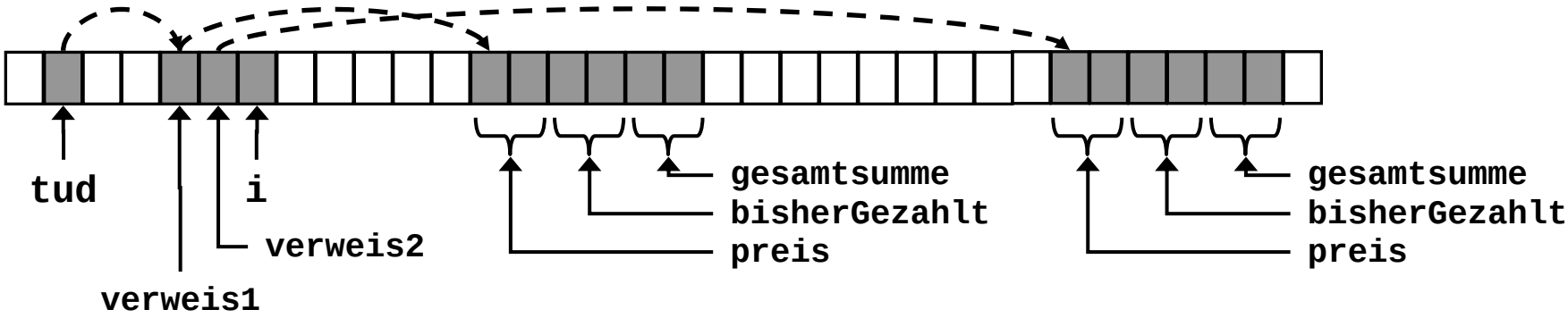
```
public class Verwaltung {

        public Ticketautomat verweis1;
        public Ticketautomat verweis2;
        public int i;

        Konstruktor & Methoden ausgelassen
}
```

```
public class Ticketautomat {

        public int preis;
        private int bisherGezahlt;
        private int gesamtsumme;

        Konstruktor & Methoden ausgelassen
}
```

```
Verwaltung tud = new Verwaltung();
tud.verweis1 = new Ticketautomat();
tud.verweis2 = new Ticketautomat();
```

6.4 Zusammengesetzte Objekte

- *Speichersicht*



- Zum Beispiel greift **`tud.verweis2.gesamtsummme`** in dieser Schemazeichnung auf die graue Speicherzelle ganz rechts zu.

Kontrollfragen zu diesem Kapitel

1. Welches sind die Hauptbestandteile einer Klasse?
2. Was unterscheidet eine Methodensignatur von einer Konstruktorsignatur?
3. Was ist der Sinn von fest vordefinierten Schnittstellen?
4. Was ist unter einer Variable im Sinne der Programmierung zu verstehen?