Erweiterungen in Java 5.0

- **Übergang von SDK 1.4 -> 1.5 gewaltig, darum: Java 5.0**
- **wird auch Tiger-Release genannt**
- **hier einige Neuerungen:**
  - **Generische Typen**
  - **Aufzählungstypen**
  - **variable Anzahl von Parametern**
  - **for each-Schleife**
  - **Auto Boxing**
- **jetzt: sogar schon > Java 7.*, für unsere Belange uninteressant**

Generische Typen

- **in Üb-Aufg. 41:**
  - Datenstruktur **Liste** allgemein für **Object** definiert

  -> kann alle Objekte als Einträge enthalten

  -> nicht parametrisierbar
  - schön wäre: verwendbar nur für **Auto** und Unterklassen
- **in C++ Standard Template Library**
  - kann Klassen parametrisieren
- **in Java 5.0 Einführung von generischen Typen für**
  - Library-Klassen: **Vector<Auto>**
  - selbst definierte Klassen:

Generische Liste

- **in Klassenkopf: Einfügen eines Parameters für gen. Typ:**

```
public class Liste<DataFormal> {
```

- **in restlichem Programm, überall Ersetzen**
  - von **Object**
  - durch **DataFormal**

```
DataFormal obj = null;

public void vorhängen(DataFormal obj)

public DataFormal head()    …
```

- **als Listeneinträge nur noch Objekte**
  - der Klasse **DataAktuell**
  - und abgeleiteten Klassen

Instanzierung & Verwendung

- **bei Instanzierung Angabe des Eintragstyps**

```
Liste<Auto> listAuto = new Liste<Auto>();
```

-> nur noch Auto in Liste vorhanden:

```
listAuto.vorhängen(new Auto())                    //o.k.

listAuto.vorhängen(new Kreis())                   //geht nicht!
```

- **Aufruf der Methoden**

```
System.out.println(listAuto.head())
```

ruft **`toString()`** von **`Auto`** auf

- **hier: kein Cast erforderlich, manchmal: doch**
- **aktueller Parameter kann beliebige Klasse sein**

Einschränken des generischen Typs

- **Einschränkung zugelassener Klassen für akt. Par.**
  - rechts von formalem Parameter **`extends MaxKlassen`**

```
class Liste<DataFormal extends Kl & I1 & I2> {
```

  - nur Klassen als aktuelle Parameter zugelassen, die
    - Klasse **`Kl`** erweitern und
    - Interfaces **`I1`** und **`I2`** implementieren

```
class Liste<DataForm extends Auto>
```

  - dann:

```
Liste<Lkw> ...                  // o.k.
Liste<Kreis> ...                // geht nicht
```

Vererbung in generischen Klassen

- **generische Klassen nicht typkompatibel**

```
class A extends B ...

Liste<A> lista = new Liste<A>();

Liste<B> listb = new Liste<B>();

listb = lista          //geht nicht!!!
```

- **Einträge der generischen Klassen aber typkompatibel**

```
listb.vorhängen(new A());
```

Aufzählungstypen

- **Ein Aufzählungstyp ist eine Menge von Konstanten**
- **Definition**

```
enum GrundFarbe {blau, rot, gelb}
```

- **Zugriff auf eine Konstante**

```
GrundFarbe.blau
```

- **Zuweisung an Variable**

```
GrundFarbe variable = GrundFarbe.blau;
```

- **Einsatz in switch-Anweisung**

```
switch (variable) {
  case blau: ...        //ohne GrundFarbe
  case rot:      ...
    ...
```

variable Anzahl von Parametern

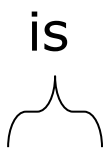
- **Methoden mit variabler Anzahl von Par. eines Typs**
- **Parameterliste intern als Array dargestellt**

-> Zugriff auf Parameterliste wie auf Array

- **höchstens eine Liste von Parametern pro Methode**
- **Liste von Parametern als letzter Parameter**
- **Beispiel:**
  - Summiere die **ersten** Elemente einer **int**-Liste.
  - Falls **ersten** > Listenlänge: summiere ganze Liste

Summation

```
static int sum(int ersten, int ...is) {
  int out = 0;
  int min = is.length;
  if (ersten < min) {
    min = ersten;
  }
  for(int i=0; i<min; i++) {
    out += is[i];
  }
  return out;
}
```



Aufruf: sum(5,1,2,3)

For Each Schleife

- **einfachere Form um über alle Elemente eines Feldes oder einer Collection (Vector, …) zu laufen**

```
 static int sum1(int ...is) {
    int out = 0;
    for(int element:is) {
      out += element;
// anstelle von:
// for (int i=0;i<is.length;i++) {out += is[i];}
    }
    return out;
  }
```

Auto Boxing

- **in Collection Klassen**
  - nur echte Objekte
  - keine primitiven Typen
- **dazu: zu jedem primitivem Typ Wrapper Klasse**

  int -> Integer
- **bisher: Umwandlung in Wrapper-Objekt umständlich**

  1 -> new Integer(1)
- **jetzt: in beiden Richtungen Umwandlung automatisch**

```
Stack<Integer> stack = new Stack<Integer>();

stack.push(1);                  //alt: stack.push(new Integer(1))

int wert = stack.pop();           //alt: stack.pop().intValue()
```