II.4. Erweiterungen von Klassen und fortgeschrittene Konzepte

- 1. Unterklassen und Vererbung
- 2. Abstrakte Klassen und Interfaces
- 3. Modularität und Pakete
- 4. Ausnahmen (Exceptions)
- 5. Generische Datentypen
- 6. Collections

Ähnliche Programmteile

```
public class Bruchelement {

  Bruch wert;
  Bruchelement next; ...    }
```

```
public class Wortelement {

  Wort wert;
  Wortelement next; ...    }
```

```
public class Bruchliste {

  Bruchelement kopf;

  Liste () { kopf = null; }

  void fuegeVorneEin (Bruch wert) {
    ... }

  Bruchelement suche (Bruch wert) {
    ...}
```

```
public class Wortliste {

  Wortelement kopf;

  Liste () { kopf = null; }

  void fuegeVorneEin (Wort wert) {
    ... }

  Wortelement suche (Wort wert) {
    ...}
```

Allgemeine Liste

```
public class Bruchelement {

  Bruch wert;
  Bruchelement next; ...    }
```

```
public class Element {

  Object wert;
  Element next; ...    }
```

```
public class Bruchliste {

  Bruchelement kopf;


 Liste () { kopf = null; }

 void fuegeVorneEin (Bruch wert) {
   ... }


 Bruchelement suche (Bruch wert) {
   ...}
```

```
public class Liste {

 Element kopf;


 Liste () { kopf = null; }

 void fuegeVorneEin (Object wert) {
   ... }


 Element suche (Object wert) {
   ...}
```

Verwendung der allgemeinen `Liste`

```
Bruch b1 = new Bruch (1,2),
      b2 = new Bruch (5,4);

Element e;


Liste xs = new Liste ();

xs.fuegeVorneEin (b1);
xs.fuegeVorneEin (b2);

e = xs.suche (b1);


xs.fuegeVorneEin ("hallo");

e = xs.suche ("hallo");
```

```
public class Element {

 Object wert;
 Element next; ...    }
```

```
public class Liste {

 Element kopf;


 Liste () { kopf = null; }

 void fuegeVorneEin (Object wert) {
   ... }


 Element suche (Object wert) {
```

Listen mit beliebigen Objekten durcheinander

Generische Liste

```
public class Element <T> {

 T wert;
 Element <T> next;  ...    }
```

```
public class Element {

 Object wert;
 Element next;  ...    }
```

```
public class Liste <T> {

 Element <T> kopf;


 Liste () { kopf = null; }

 void fuegeVorneEin (T wert) {
  ... }


 Element <T> suche (T wert) {
  ...}
```

```
public class Liste {

 Element kopf;


 Liste () { kopf = null; }

 void fuegeVorneEin (Object wert) {
  ... }


 Element suche (Object wert) {
  ...}
```

Generische Liste

```
public class Element <T> {

 T wert;
 Element <T> next;  ...    }
```

```
public class Liste <T> {

 Element <T> kopf;


 Liste () { kopf = null; }

 void fuegeVorneEin (T wert) {
  ... }


 Element <T> suche (T wert) {
  ...}
```

```
Bruch b1 = new Bruch (1,2),
      b2 = new Bruch (5,4);

Element <Bruch> e;


Liste <Bruch> xs =
          new Liste <>      ();

xs.fuegeVorneEin (b1);
xs.fuegeVorneEin (b2);

e = xs.suche (b1);
```

```
xs.fuegeVorneEin ("hallo");

e = xs.suche ("hallo");
```

Typfehler (compiliert nicht)

Generische Klasse

```
public class Element <T> {

 T wert;
 Element <T> next; ...    }
```

```
public class Liste <T> {

 Element <T> kopf;
```

- Eine Klasse, viele Typen:
  `Liste <Bruch>, Liste <Wort>, ...`
- Generische Typen nur vom Compiler überprüft, nicht zur Laufzeit

nicht möglich (statische Methode existiert nur einmal pro Klasse)

```
static     Element <T> suche (T wert, Element <T> kopf) {
 if   (kopf == null)                    return null;
 else if (wert.gleich(kopf.wert))  return kopf;
 else                              return suche (wert, kopf.next);
}
```

Generische Klasse

```
public class Element <T> {

 T wert;
 Element <T> next; ...    }
```

```
public class Liste <T> {

 Element <T> kopf;
```

- Eine Klasse, viele Typen:
  `Liste <Bruch>, Liste <Wort>, ...`
- Generische Typen nur vom Compiler überprüft, nicht zur Laufzeit

erlaubt (generische Methode)

```
static <T> Element <T> suche (T wert, Element <T> kopf) {
 if   (kopf == null)                    return null;
 else if (wert.gleich(kopf.wert))  return kopf;
 else                                  return suche (wert, kopf.next);
}
```

Generische Klasse

```
public class Element <T> {

 T wert;
 Element <T> next; ...    }
```

```
public class Liste <T> {

 Element <T> kopf;
```

- Eine Klasse, viele Typen:
  `Liste <Bruch>, Liste <Wort>, ...`
- Generische Typen nur vom Compiler überprüft, nicht zur Laufzeit

erlaubt (generische Methode)

```
static <S> Element <S> suche (S wert, Element <S> kopf) {

 if   (kopf == null)                    return null;

 else if (wert.gleich(kopf.wert))   return kopf;

 else                                   return suche (wert, kopf.next);
}
```

Typparameter der Methode

Rückgabetyp

Generische Klasse

```
public class Element <T> {

 T wert;
 Element <T> next; ...    }
```

```
public class Liste <T> {

 Element <T> kopf;
```

- **Eine Klasse, viele Typen:**
  `Liste <Bruch>, Liste <Wort>, ...`
- **Generische Typen nur vom Compiler überprüft, nicht zur Laufzeit**

nicht möglich

Methode **`gleich`** nur in Klassen, die Interface **`Vergleichbar`** implementieren

```
static <S> Element <S> suche (S wert, Element <S> kopf) {

 if   (kopf == null)                    return null;

 else if (wert.gleich(kopf.wert))   return kopf;

 else                                return suche (wert, kopf.next);

}
```

Typebounds

```
public class Element <T extends Vergleichbar> {

 T wert;
 Element <T> next; ...    }
```

```
public class Liste <T extends Vergleichbar> {

 Element <T> kopf;
```

```
static <S> Element <S> suche (S wert, Element <S> kopf) {
 if   (kopf == null)                    return null;
 else if (wert.gleich(kopf.wert))  return kopf;
 else                                  return suche (wert, kopf.next);
}
```

Typebounds

```
public class Element <T extends Vergleichbar> {

 T wert;
 Element <T> next; ...    }
```

```
public class Liste <T extends Vergleichbar> {

 Element <T> kopf;
```

```
static <S extends Vergleichbar> Element <S> suche (…) {
 if   (kopf == null)                    return null;
 else if (wert.gleich(kopf.wert))   return kopf;
 else                                   return suche (wert, kopf.next);
}
```

Ober- und Unterklassen

```
class A implements Vergleichbar { ... }

class B extends A { ... }
class C extends A { ... }
```

B ist Unterklasse von **A**

C ist Unterklasse von **A**

```
A [] aArray = new A [5];

B [] bArray = new B [17];

aArray = bArray;

aArray[0] = new C ();
```

B[] ist Unterklasse von **A[]**

C[] ist Unterklasse von **A[]**

compiliert, aber Typfehler zur Laufzeit

```
Liste <A> aList = new Liste <> ();

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

aList = bList;

aList.fuegeVorneEin(new C ());
```

Liste <B> ist keine Unterklasse von **Liste <A>**

Liste <C> ist keine Unterklasse von **Liste <A>**