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;



  void fuegeVorneEin (Bruch wert){
    ... }
```

```
public class Wortliste {

  Wortelement kopf;



  void fuegeVorneEin (Wort wert){
    ... }
```

Allgemeine Liste

```
public class Bruchelement {

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

```
public class Element {

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

```
public class Bruchliste {

  Bruchelement kopf;



  void fuegeVorneEin (Bruch wert){
    ... }
```

```
public class Liste {

  Element kopf;



  void fuegeVorneEin (Object wert) {
    ... }
```

Verwendung der allgemeinen `Liste`

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



Liste xs = new Liste ();

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


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

Listen mit beliebigen Objekten durcheinander

```
public class Element {

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

```
public class Liste {

 Element kopf;


 void fuegeVorneEin (Object wert) {
   ... }
```

Vergleich in der Klasse `Liste`

```
public class Element {
 Object wert;
 Element next; ...    }
```

```
public class Liste {
 Element kopf;

public void fuegeSortiertEin (Object wert) {
 kopf = fuegeSortiertEin (wert, kopf);  }

private static Element fuegeSortiertEin (Object wert,
                                         Element e){
 if        (e == null)
           return new Element (wert);
 else if   (wert < e.wert)
           return new Element (wert, e);
 else {    e.next = fuegeSortiertEin (wert, e.next);
           return e;                              } }
```

Vergleich mit < nicht möglich auf `Object`

Abstrakte Klasse

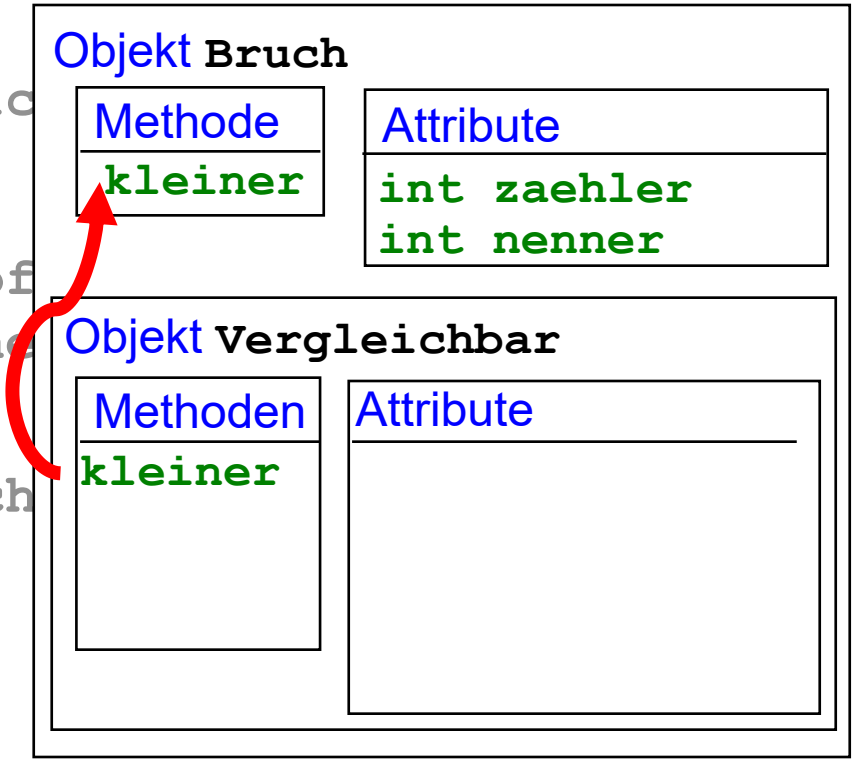
```
public abstract class Vergleichbar {

 public abstract boolean kleiner (Vergleichbar zuvergleichen);...}
```

```
public class Bruch extends Vergleichbar {

 private int zaehler, nenner;

 public boolean kleiner (Vergleichbar zuvergleichen) {


     if (zuvergleichen instanceof Bruch zb) {
       return (zaehler * zb.nenner < zb.zaehler * nenner);
     }
     else {IO.println("Kein Bruchvergleich");
           return false;
     }

...}
```

Abstrakte Klasse

```
public abstract class Vergleichbar {

 public abstract boolean kleiner (Vergleichbar zuvergleichen);...}
```

```
public class Bruch extends Vergleichbar {

 private int zaehler, nenner;

 public boolean kleiner (Vergleic



     if (zuvergleichen instanceof
       return (zaehler * zb.nenn
     }

     else {IO.println("Kein Bruch
           return false;
     }

...}
```



Liste mit abstrakter Klasse

```
public abstract class Vergleichbar {
 public abstract boolean kleiner (Vergleichbar zuvergleichen);...}
```

```
public class Bruch extends Vergleichbar { ... }
```

```
public class Wort extends Vergleichbar { ... }
```

```
public class Element                                    {
     Vergleichbar wert;  Element next; ...              }
```

```
public class Liste {
 Element kopf;
 void fuegeSortiertEin (Vergleichbar wert) {
  kopf = fuegeSortiertEin (wert, kopf);  }

 static Element fuegeSortiertEin (Vergleichbar wert, Element e){
  if       (e == null) return new Element (wert);
  else if  (wert.kleiner(e.wert)) return new Element (wert, e);
  else {e.next = fuegeSortiertEin (wert, e.next); return e;    } }
```

Mehrere Anforderungen an Klassen

```
public abstract class Vergleichbar {

 public abstract boolean kleiner (Vergleichbar zuvergleichen);...}
```

```
public abstract class Aenderbar {

 public abstract void aenderung ();  ...}
```

```
public class Bruch extends Vergleichbar, Aenderbar {

 public boolean kleiner (Vergleichbar zuvergleichen){ ... }

 public void aenderung () { ... }                    ...}
```

```
public class Wort extends Vergleichbar {

  public boolean kleiner (Vergleichbar zuvergleichen) {...}
```

Geht nicht, Java hat nur Einfachvererbung

Mehrere Anforderungen an Klassen

```
public interface Vergleichbar                          {
 boolean kleiner (Vergleichbar zuvergleichen);         }
```

```
public interface Aenderbar                             {
 void aenderung ();                                    }
```

```
public class Bruch implements Vergleichbar, Aenderbar {
 public boolean kleiner (Vergleichbar zuvergleichen){ ... }
 public void aenderung () { ... }                          ...}
```

```
public class Wort implements Vergleichbar {
  public boolean kleiner (Vergleichbar zuvergleichen) {...}
```

Interfaces und abstrakte Klassen

```
public interface Vergleichbar                                    {
     boolean kleiner (Vergleichbar zuvergleichen);               }
```

```
public interface Aenderbar                                       {
     void aenderung ();                                          }
```

```
public abstract class Zahl implements Vergleichbar               {
     protected abstract int runde ();
     public String rundungsinfo ()            {
          return "in etwa " + runde ();       }                  }
```

```
public class Bruch extends Zahl implements Aenderbar             {
     public boolean kleiner (Vergleichbar zuvergleichen) { ... }
     protected int runde () { ... }
     public void aenderung () { ... }                      ...   }
```

```
public class Int extends Zahl                                    {
     public boolean kleiner (Vergleichbar zuvergleichen) { ... }
     public int runde () { ... }                           ...   }
```

```
public class Wort implements Vergleichbar                        {
     public boolean kleiner (Vergleichbar zuvergleichen) {...}... }
```

Sealed Interfaces und abstrakte Klassen

```
public sealed interface Vergleichbar permits Zahl, Wort      {
     boolean kleiner (Vergleichbar zuvergleichen);                }
```

```
public interface Aenderbar                                        {
     void aenderung ();                                           }
```

```
public abstract sealed class Zahl implements Vergleichbar
                                        permits Bruch, Int {
     protected abstract int runde ();
     public String rundungsinfo () { ... }                        }
```

```
public final class Bruch extends Zahl implements Aenderbar  {
     public boolean kleiner (Vergleichbar z
     protected int runde () { ... }
     public void aenderung () { ... }
```

```
public final class Int extends Zahl
     public boolean kleiner (Vergleichbar z
     public int runde () { ... }
```

```
public final class Wort implements Verg
     public boolean kleiner (Vergleichbar z
```

switch mit **sealed** Interface oder **sealed** abstrakter Klasse

wegen **sealed** abstrakter Klasse vollständig ohne **default**-Fall

```
Zahl z;

String s = switch (z) {
               case Bruch b -> "Bruch";
               case Int i   -> "Int";
}
```

Liste mit Interfaces

```
public interface Vergleichbar                                    {
     boolean kleiner (Vergleichbar zuvergleichen);               }
```

```
public abstract class Zahl implements Vergleichbar       {...}
public class Bruch extends Zahl implements Aenderbar     {...}
public class Int extends Zahl                             {...}
public class Wort implements Vergleichbar                 {...}
```

```
public class Element                                 {
     Vergleichbar wert;  Element next; ...           }
```

```
public class Liste {

 Element kopf;

 void fuegeSortiertEin (Vergleichbar wert) {
  kopf = fuegeSortiertEin (wert, kopf);   }

 static Element fuegeSortiertEin (Vergleichbar wert, Element e){
  if        (e == null) return new Element (wert);
  else if   (wert.kleiner(e.wert)) return new Element (wert, e);
  else {e.next = fuegeSortiertEin (wert, e.next); return e;     } }
```

Datenzugriff mit Interfaces

```
public interface I                                          {
      int x = 4, y = 6;
      void b (int i);
      void q (int n);                                       }

public interface J                                          {
      int x = 3;
      void b (double d);
      void q (int n);                                       }

public class C implements I_and_J                           {
      public void b (int i) { ... }
      public void b (double d) { ... }
      public void q (int n) { ... }                         }
```

```
public interface I_and_J extends I, J  {
                                    ... }
```

```
C z = new C ();
I i = z;
J j = i;
i.b (5); j.b (5);
i.q (5); j.q (5);
IO.println(I.x + "," + J.x + "," + C.x + "," + C.y);
```

nicht erlaubt, stattdessen `J j = (C) i;`

nicht erlaubt, da nicht eindeutig

Interfaces mit Default-Methoden

```
public interface I  extends H                                          {
     default int u () { return 0; }


                                                                       }
```

```
                          ;
```

```
public class C implements I
                          return    super.u() + 1; }
```

nicht erlaubt, **v** muss in **C** implementiert werden

```
C z = new C ();

IO.println(z.u ());

IO.println( I.w ());
```

1

4

nicht erlaubt, da Mehrfachvererbung nicht eindeutig