
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

Listen-Paket

Element.java

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

Liste.java

```
public class Liste      {  
    private Element kopf;  
    ...                  }  
}
```

Listen-Paket

package listen

Element.java

```
package listen;  
  
class Element      {  
    Vergleichbar wert;  
    Element next;   ... }  
}
```

Liste.java

```
package listen;  
  
public class Liste      {  
    private Element kopf;  
    ...                  }  
}
```

Test1.java

```
class Test1      {  
    listen.Liste l; ...  
}
```

Test2.java

```
import listen.Liste;  
  
class Test2      {  
    Liste l; ... }  
}
```

Test3.java

```
import listen.*;  
  
class Test3      {  
    Liste l; ... }  
}
```

Pakete und Package listen

package listen

```
package listen;
```

```
class Element { ... }
```

```
package listen;
```

```
public class Liste { ... }
```

exportiert

importiert

Test.java

```
import listen.*;
```

```
class Test {  
    Liste l; ... }  
}
```

Package listen

Class Summary

Class	Description
Liste	Datentyp fuer lineare Listen von vergleichbaren Objekten

listen

Class Liste

java.lang.Object
listen.Liste

```
public class Liste  
extends java.lang.Object
```

Datentyp fuer lineare Listen von vergleichbaren Objekten

See Also:

Vergleichbar

Constructor Summary

Constructors

Constructor and Description

Liste()	erzeugt eine neue leere Liste
---------	-------------------------------

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
void	drucke () gibt den Inhalt der Liste (von vorne nach hinten) auf dem Bildschirm aus.
void	druckeRueckwaerts () gibt den Inhalt der invertierten Liste (d.h., von hinten nach vorne) auf dem Bildschirm aus.
void	fuegeVorneEin (Vergleichbar wert) fuegt ein Element vorne in die Liste ein.
void	loesche () loescht die komplette Liste.
void	loesche (Vergleichbar wert) loescht das erste Element mit dem angegebenen Wert aus der Liste.
java.lang.String	toString () erzeugt einen String, der die Elemente der Liste von vorne nach hinten aufzaehlt.
java.lang.String	toStringRueckwaerts () erzeugt einen String, der die Elemente der invertieren Liste (d.h., von hinten nach vorne) aufzaehlt.

Pakethierarchie

package wert

Aenderbar.java

Vergleichbar.java

Zahl.java

```
package wert;
```

```
public abstract class Zahl implements Vergleichbar {  
    protected abstract int runde ();                ...  
}
```

package wert.zahlen

Int.java

Bruch.java

```
package wert.zahlen;
```

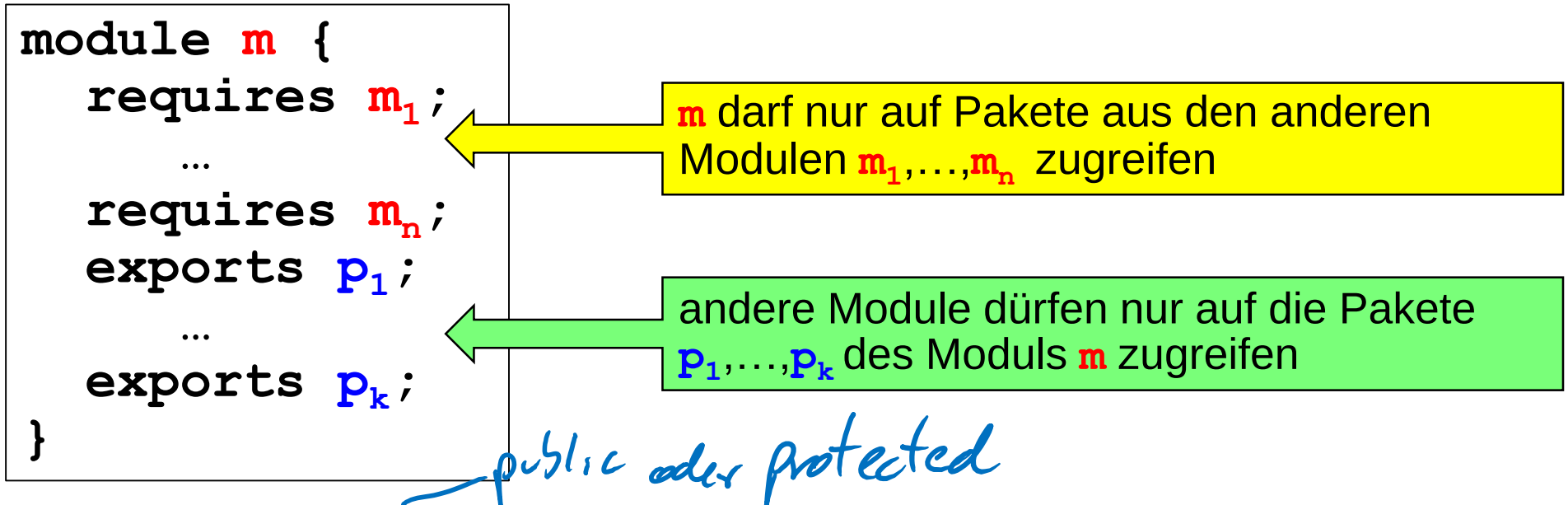
```
import wert.*;
```

```
import io.*;
```

```
public class Bruch extends Zahl implements Aenderbar {  
    public int runde () {...}                        ...  
}
```

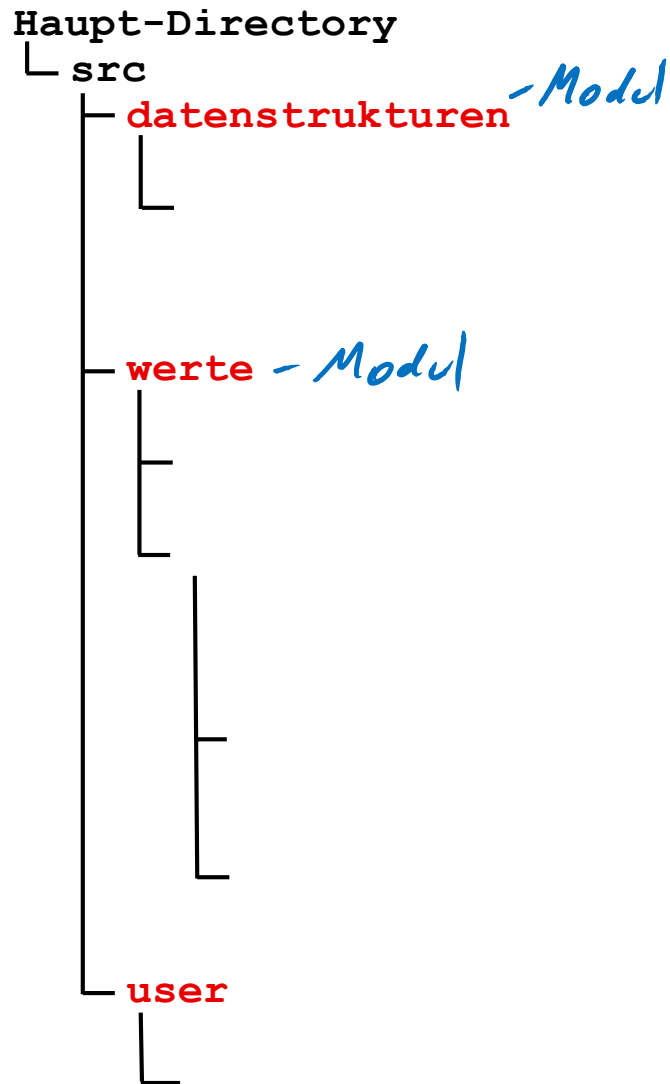
Module

- Modul **m** fasst Pakete zusammen
- Verzeichnis **m** enthält Deskriptor-Datei `module-info.java`



- Sichtbarkeiten wie bisher, aber nur innerhalb eines Moduls und zugreifbarer Komponenten anderer Module
- Jedes Modul darf auf `java.base` zugreifen. *← enthält Pakete java.lang, java.util, ...*
- "Unnamed" Modul darf auf alle anderen Module zugreifen

Modul-Beispiel



- Verwende Unterverzeichnis src für Java-Quellcode (... • java)
- Verwende Unterverzeichnis class für compilierten Code (... • class)

Modul-Beispiel

Haupt-Directory

└ src

└ datenstrukturen

└ module-info.java

└ listen

└ Element.java

└ Liste.java

└ werte

└ module-info.java

└ io

└ SimpleIO.java

└ wert

└ Aenderbar.java

└ Vergleichbar.java

└ Zahl.java

└ zahlen

└ Bruch.java

└ Int.java

└ zeichen

└ Wort.java

└ user

└ module-info.java

└ test

└ Test_Programm.java

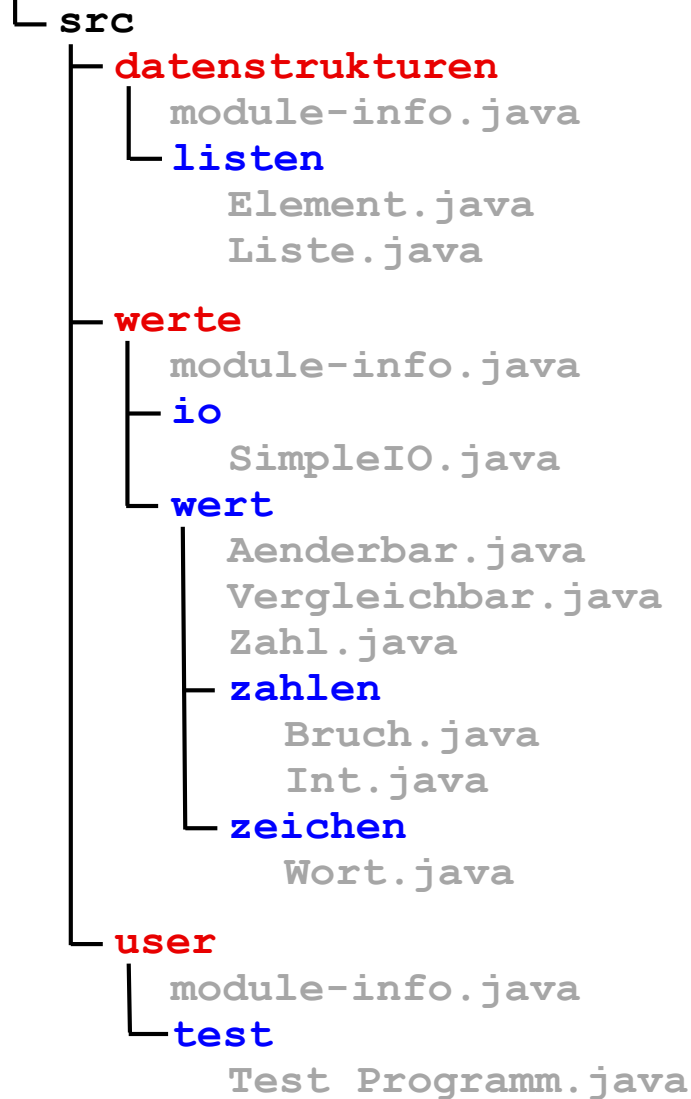
```
module datenstrukturen {  
    requires werte;  
    exports listen;  
}
```

```
module werte {  
    requires java.desktop;  
    exports wert;  
    exports wert.zahlen;  
    exports wert.zeichen;  
}
```

```
module user {  
    requires datenstrukturen;  
    requires werte;  
}
```

Modul-Beispiel

Haupt-Directory



Compilieren:

destination directory
`javac -d class` *Verzeichnis f. class-Dateien*
`--module-source-path src`
`src/user/test/Test_Programm.java`
Diese Klasse soll compiliert werden.

Ausführen:

Pfad, um die class-Dateien zu finden
`java -p class`
`-m user/test.Test_Programm`
Klasse, deren main-Methode ausgeführt werden soll
Modul
Paket
Klasse