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**

Collection Framework

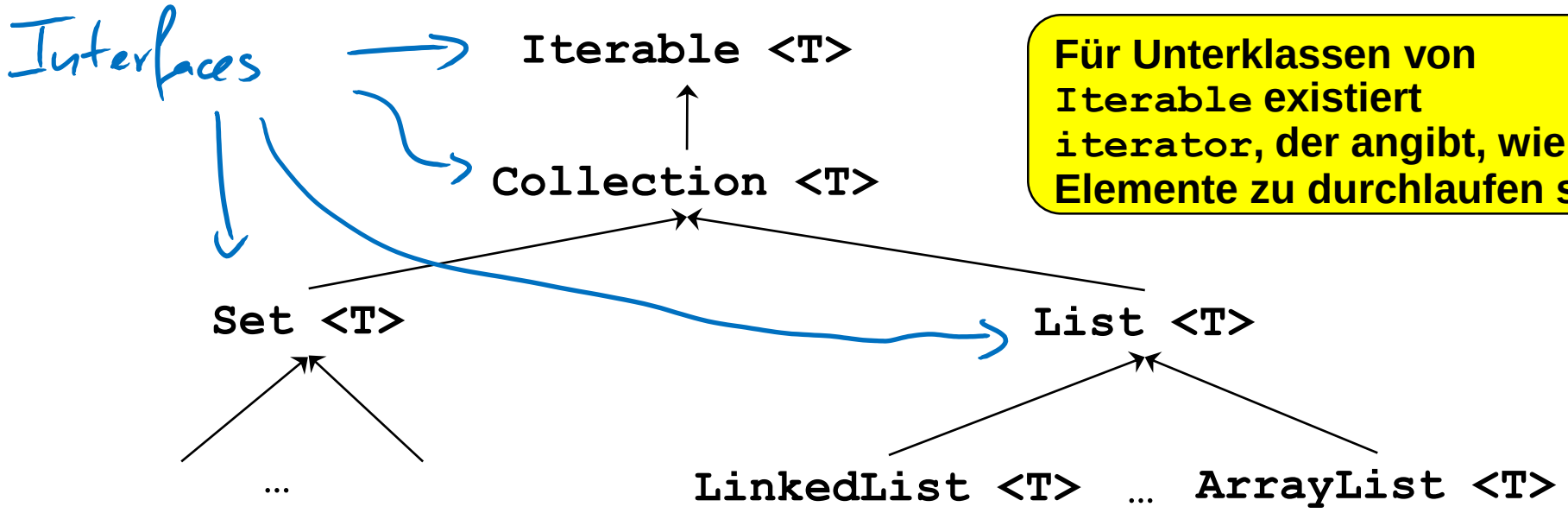
Typische Datenstrukturen im Paket `java.util` vordefiniert

`Liste <T>`

- `Liste ()`
- `String toString ()`
- `void fuegeVorneEin (T wert)`

`LinkedList <T>`

- `LinkedList ()`
- `String toString ()`
- `void addFirst (T wert)`

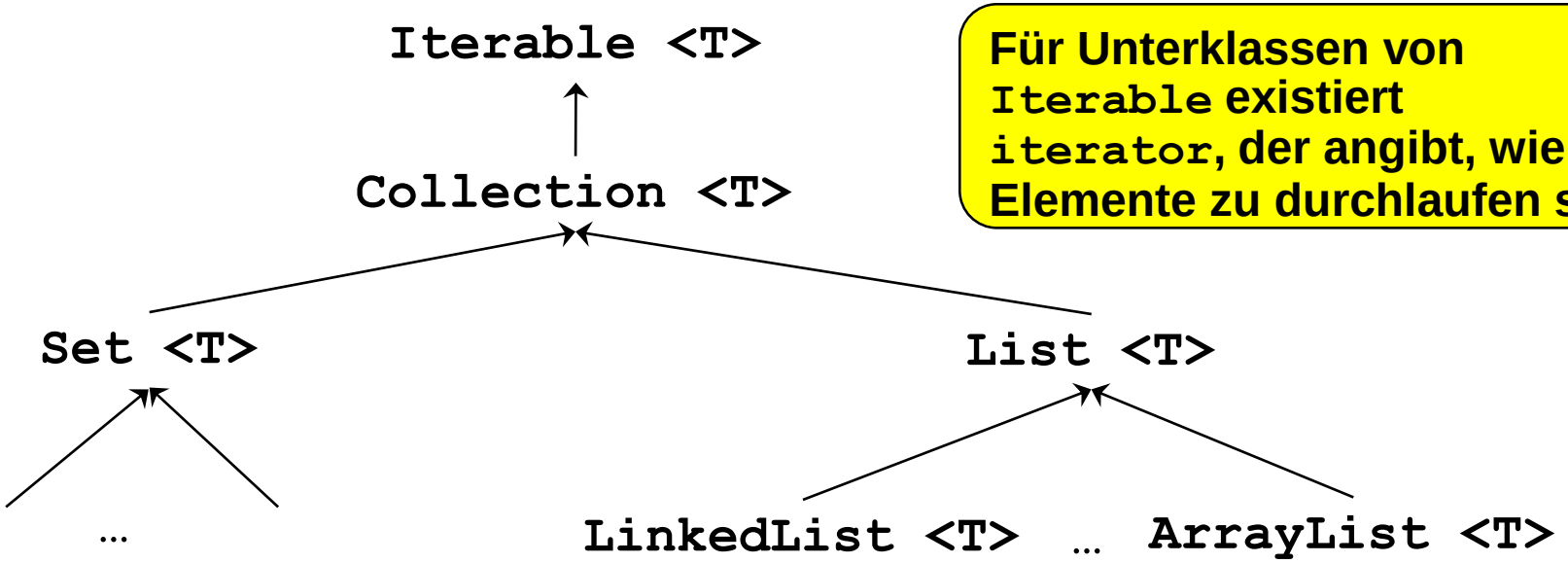


Für Unterklassen von `Iterable` existiert `iterator`, der angibt, wie Elemente zu durchlaufen sind

Collection Framework

```
interface Iterable <T> {

 Iterator <T> iterator();

  ...

}
```

```
interface Iterator <T> {
 boolean hasNext();
 T       next();
 void    remove();

  ...
}
```



Für Unterklassen von `Iterable` existiert `iterator`, der angibt, wie Elemente zu durchlaufen sind

Collection Framework


```
interface Iterator <T> {
 boolean hasNext();
 T       next();
 void    remove();
}
```

```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");
```

Collection Framework


```
interface Iterator <T> {
 boolean hasNext();
 T       next();
 void    remove();
}
```

```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();
```

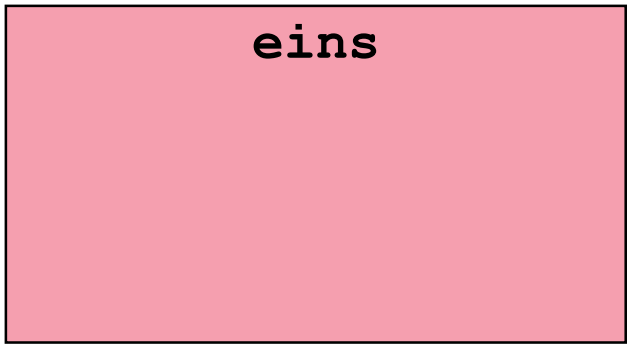
Collection Framework


```
interface Iterator <T> {
 boolean hasNext();
 T       next();
 void    remove();
}
```

```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

while (it.hasNext()) {
     String s = it.next();
     System.out.println(s);
}
```

setzt Iterator weiter und liefert "überlaufenes" Element als Ergebnis

Collection Framework




```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

while (it.hasNext()) {
     String s = it.next();
     System.out.println(s);
}
```

setzt Iterator weiter und liefert "überlaufenes" Element als Ergebnis

Collection Framework


```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

while (it.hasNext()) {
     String s = it.next();
     System.out.println(s);
}
```

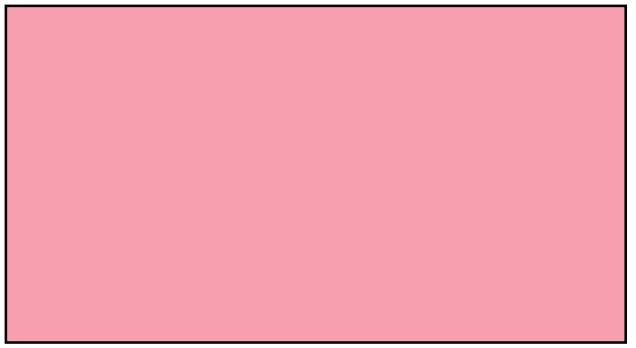
setzt Iterator weiter und liefert "überlaufenes" Element als Ergebnis

Collection Framework


```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

while (it.hasNext()) {
     String s = it.next();
     System.out.println(s);
}
```

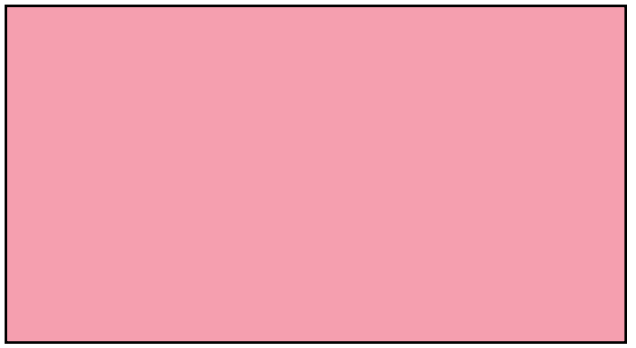
```
for (String s : sl) {
     System.out.println(s);
}
```

Collection Framework



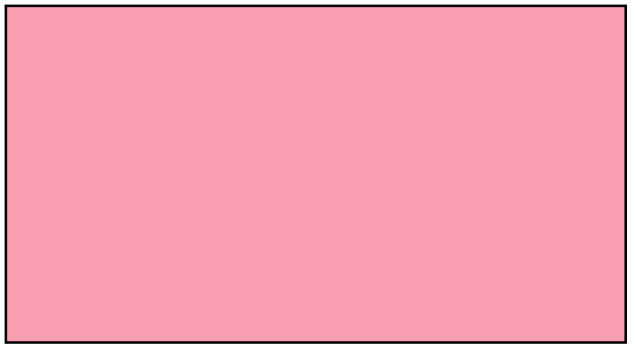
```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();
```

Collection Framework



```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

it.next();
```

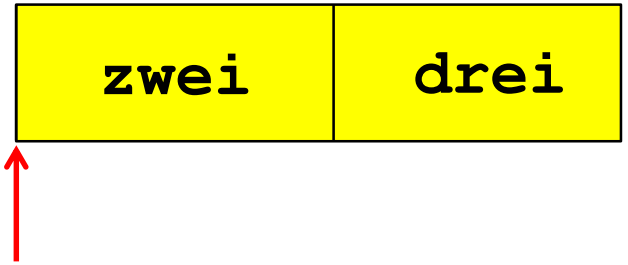
Collection Framework

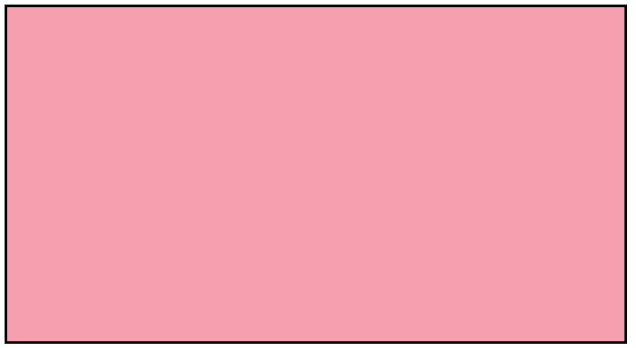


```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

it.next();

it.remove();
```

löscht zuletzt "überlaufenes" Element

Collection Framework

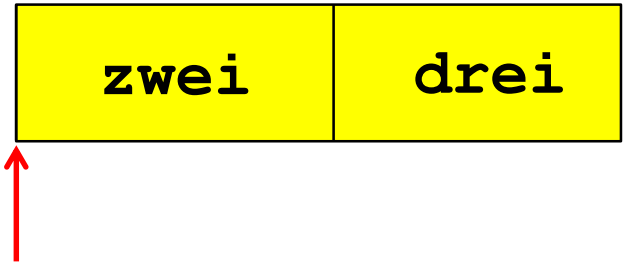




```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

it.next();

it.remove();
```

löscht zuletzt "überlaufenes" Element

Collection Framework




```
LinkedList <String> sl = new LinkedList <> ();

sl.addFirst("drei");  sl.addFirst("zwei");  sl.addFirst("eins");

Iterator <String> it = sl.iterator();

it.next();
it.remove();

for (String s : sl) System.out.println(s);
```

löscht zuletzt "überlaufenes" Element

Collection Framework

```
LinkedList <Integer> il = new LinkedList <> ();


il.addFirst(3); il.addFirst(2); il.addFirst(1);
```

Autoboxing (konvertiert **`int`** in **`Integer`**)

Collection Framework

```
LinkedList <Integer> il = new LinkedList <> ();

il.addFirst(3); il.addFirst(2); il.addFirst(1);

Iterator <Integer> it = il.iterator();

while (it.hasNext()) {
     Integer i = it.next();
     System.out.println(i);
}
```

Autoboxing (konvertiert **int** in **Integer**)

Collection Framework

```
LinkedList <Integer> il = new LinkedList <> ();


il.addFirst(3); il.addFirst(2); il.addFirst(1);


Iterator <Integer> it = il.iterator();


while (it.hasNext()) {
     int      i = it.next();
     System.out.println(i);
}
```

Autoboxing (konvertiert **int** in **Integer**)

Unboxing (konvertiert **Integer** in **int**)

Collection Framework

```
LinkedList <Integer> il = new LinkedList <> ();

il.addFirst(3); il.addFirst(2); il.addFirst(1);

Iterator <Integer> it = il.iterator();

while (it.hasNext()) {
     int     i = it.next();
     System.out.println(i);
}

for (Integer i : il) System.out.println(i);
```

Autoboxing (konvertiert **int** in **Integer**)

Unboxing (konvertiert **Integer** in **int**)

Collection Framework

```
LinkedList <Integer> il = new LinkedList <> ();

il.addFirst(3); il.addFirst(2); il.addFirst(1);

Iterator <Integer> it = il.iterator();

while (it.hasNext()) {
     int     i = it.next();
     System.out.println(i);
}

for (Integer i : il) System.out.println(i);

for (int i : il)   System.out.println(i);
```

Autoboxing (konvertiert **int** in **Integer**)

Unboxing (konvertiert **Integer** in **int**)

Collection Framework

```
LinkedList <Integer> il = new LinkedList <> ();

il.addFirst(3); il.addFirst(2); il.addFirst(1);

Iterator <Integer> it = il.iterator();

while (it.hasNext()) {
     int     i = it.next();
     System.out.println(i);
}

for (Integer i : il) System.out.println(i);

for (int i : il) { System.out.println(i); il.addFirst(0); }
```

Autoboxing (konvertiert **int** in **Integer**)

Unboxing (konvertiert **Integer** in **int**)

compiliert, aber Fehler zur Laufzeit