Objektorientierte Programmierung

Studiengang Medieninformatik

Hans-Werner Lang

Hochschule Flensburg

Vorlesung 8
03.05.2017

Was bisher geschah...

Hochschule Flensburg
University of Applied Sciences

Objektorientierte Programmierung

- Klassen und Objekte, Attribute, Methoden
- Klassenhierarchie, Vererbung, Methoden überschreiben
- Abstrakte Klasse, Interface, Klassenattribute und -methoden

Grafische Benutzungsoberfläche (GUI)

- GUI-Elemente: Frames, Panels, Labels, Buttons, Textfelder
- Reaktion auf Ereignisse mit Listenern

Datenstrukturen

- ArrayList, LinkedList
- Queue, Stack

Sonstiges

- Refactoring, Testen

Iterator

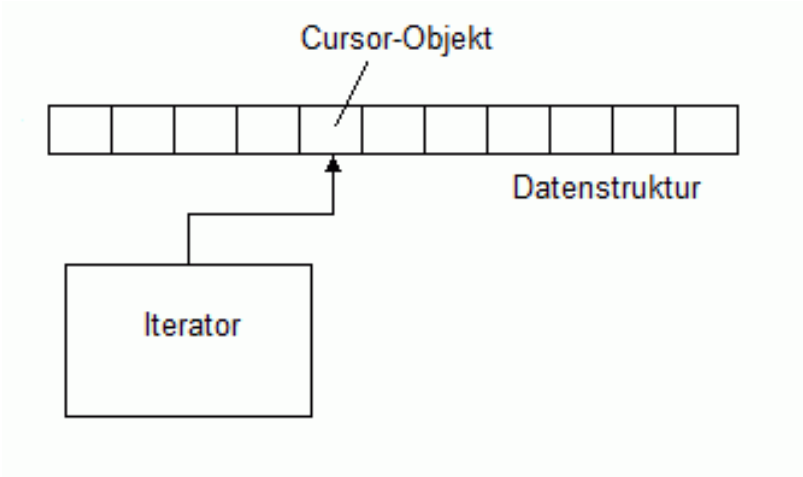
Hochschule Flensburg
University of Applied Sciences

Es kommt häufig vor, dass im Programm ein Array oder eine Liste durchlaufen wird, etwa um alle Elemente auszugeben oder etwas anderes damit zu machen.

Normalerweise schreiben wir hierzu eine For-Schleife; dabei müssen wir aber jedesmal neu überlegen, von wo bis wo die Laufvariable läuft, wie auf die Elemente zugegriffen wird usw.

Eine bessere Lösung ist ein Iterator. Ein Iterator stellt zwei Methoden zur Verfügung:

```
public boolean hasNext()  // true, wenn ein weiteres Objekt vorhanden ist  und
public Object next()      // liefert das Cursor-Objekt und schiebt den Cursor weiter
```



ArrayList-Iterator

Hochschule Flensburg
University of Applied Sciences

Eine *ArrayList* liefert ihren Iterator gleich mit. Wenn wir eine *ArrayList* mit Typ-Parameter verwenden, hat der Iterator der *ArrayList* denselben Typ-Parameter.

```
ArrayList<String> a=new ArrayList<String>();
a.add("ene");
a.add("mene");
a.add("muh");

Iterator<String> it=a.iterator();  // mitgelieferter Iterator von a
while (it.hasNext())
    System.out.print(it.next()+" ");
```

Interface *Iterable*

Hochschule Flensburg
University of Applied Sciences

Die Klasse *ArrayList* implementiert das Interface *Iterable*. Sie verpflichtet sich damit, eine Methode *iterator* zur Verfügung zu stellen, die einen Iterator zurückgibt.

Datenstrukturen, die das Interface *Iterable* implementieren, lassen sich mit einer speziellen For-Schleife durchlaufen, beispielsweise eine `ArrayList<String> a` in folgender Weise:

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

Es empfiehlt sich, den Doppelpunkt als "in" zu lesen: "Für jeden String s in Datenstruktur a ...".

Andere Programmiersprachen kennen derartige For-Each-Schleifen ebenfalls:

Python:

```
for s in a:
    print s
```

PHP:

```
foreach ($a as $s)
    echo $s;
```

Einen eigenen Iterator schreiben


```
public class StringArrayIterator implements Iterator<String>
{
     // Attribute
     private String[] a;
     private int n, i=0;

     // Konstruktor
     public StringArrayIterator(String[] a)
     {
          this.a=a;
          n=a.length;
     }

     @Override
     public boolean hasNext()
     {
          return i<n;
     }

     @Override
     public String next()
     {
          return a[i++];
     }

     @Override
     public void remove()
     {
          // not implemented
     }
}
```

Anwendung des Iterators

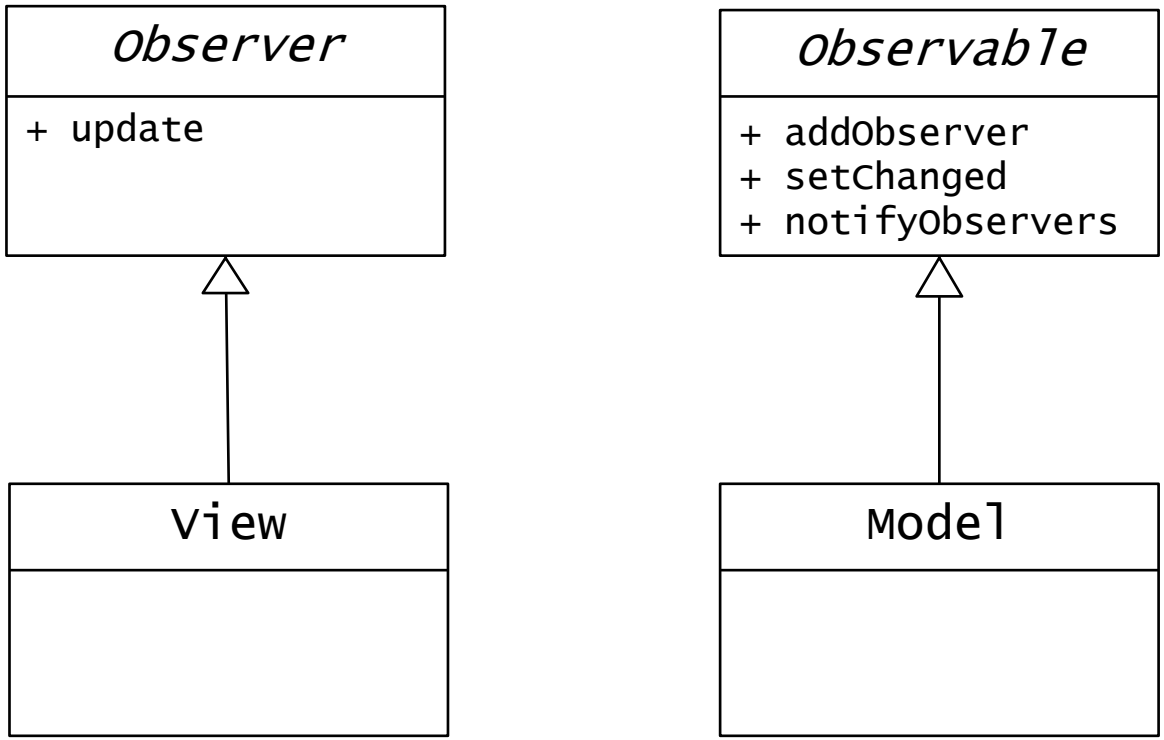

```
String[] s=new String[3];
s[0]="ene";
s[1]="mene";
s[2]="muh";

Iterator<String> it=new ArrayIterator<String>(s);
while (it.hasNext())
    System.out.print(it.next()+" ");
```

Eigener Iterator mit Typ-Parametern


```
public class ArrayIterator<Type> implements Iterator<Type>
{
     // Attribute
     private Type[] a;
     private int n, i=0;

     // Konstruktor
     public ArrayIterator(Type[] a)
     {
          this.a=a;
          n=a.length;
     }

     @Override
     public boolean hasNext()
     {
          return i<n;
     }

     @Override
     public Type next()
     {
          return a[i++];
     }

     @Override
     public void remove()
     {
          // not implemented
     }
}
```

Observer-Softwaremuster

Hochschule Flensburg
University of Applied Sciences



Einfachstes Beispiel für das Observer-Softwaremuster

Hochschule Flensburg
University of Applied Sciences

```
public class Mvc
{
    public static void main(String[] args)
    {
        Model model=new Model();
        View view=new View();
        // das Objekt view in die Liste der Beobachter des Objekts model eintragen:
        model.addObserver(view);
        model.change();
        model.change();
    }
}
public class Model extends Observable
{
    public void change()
    {
        this.setChanged();
        this.notifyObservers("Modell hat sich geändert");
    }
}
public class View implements Observer
{
    @Override
    public void update(Observable arg0, Object arg1)
    {
        System.out.println(arg1);
    }
}
```

Zusammenfassung

- Iterator-Softwaremuster
- Observer-Softwaremuster