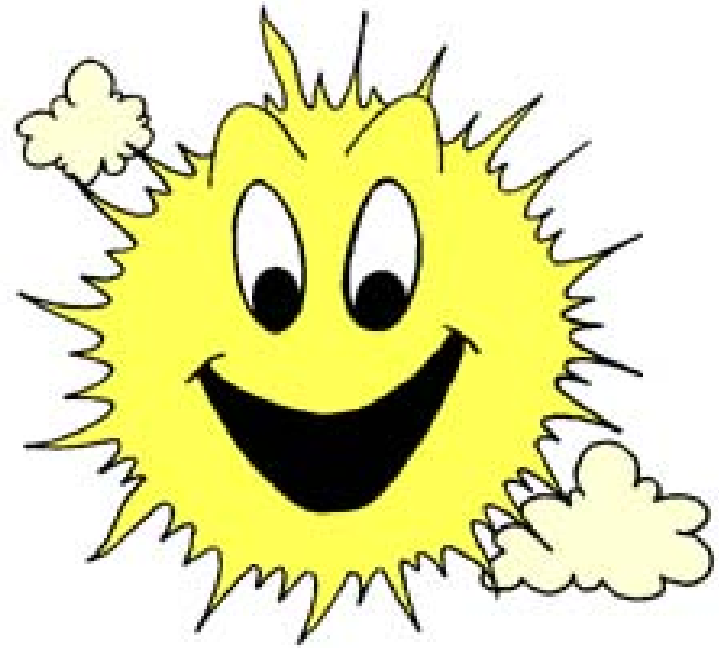
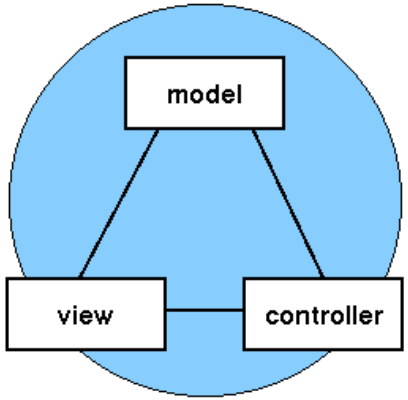
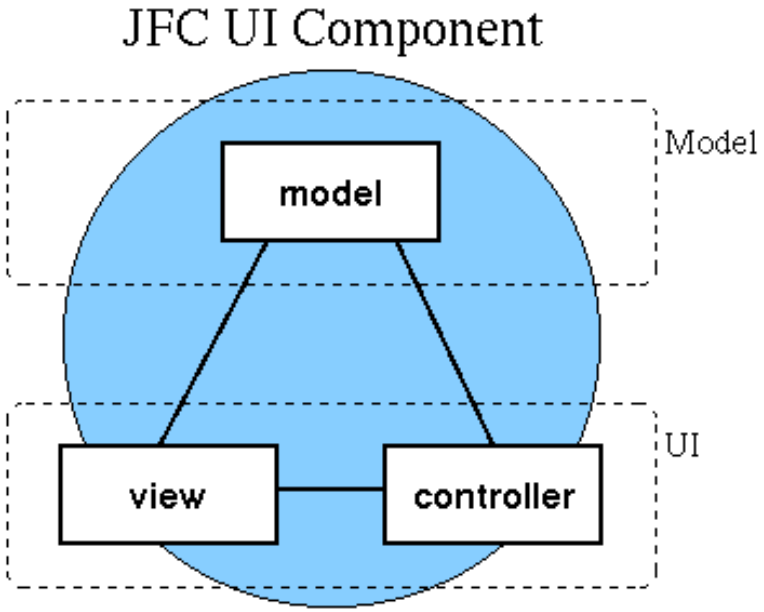
Wiederholung 7. Vorlesung



6.2 MVC Java Foundation Classes (JFC)

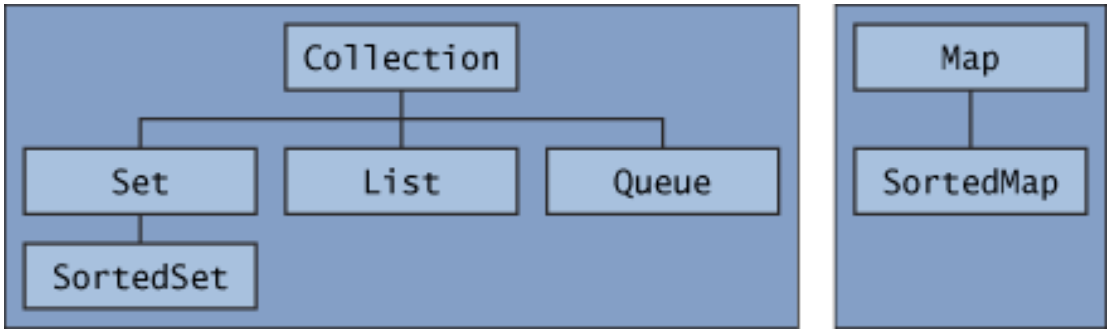


Bei kleineren Anwendungen wird, wie in den JFC häufig die Benutzungsschnittstelle ((User Interface (**UI**)) gewissermaßen als **Einheit** von **View** und **Controller** gesehen.



6.5 Collections in Java

- **Collections** können allgemein klassifiziert werden in die folgenden Interfaces (siehe auch Literatur)



- **Collections** stellt Gruppe von Objekten dar
- **Set** ist eine Collection, die *keine doppelten* Elemente hat
- **List** ist eine *geordnete* Collection, die doppelte Elemente enthalten kann
- **Queue** stellt ein FIFO dar
- **Map** stellt Paare aus **{ Schlüssel , Objekt }** dar. Schlüssel müssen eindeutig sein.

6.6 Speichern von Objekten in Java

- Neben Datenbanken stellt die **Persistenz** (*etwas dauerhaft machen*) eine elegante Möglichkeit zur Speicherung (**Serialisierung**) von Objekten dar.
- In Java sind alle Objekte, welche das ***Serializable*** Interface implementieren, *serialisierbar*.

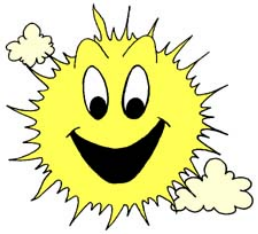
```
FileOutputStream fos = new FileOutputStream(fName);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(object);
```

- Aus gespeicherten (serialisierten) Daten können durch **Deserialisierung** wieder Objekte erzeugt werden.

```
FileInputStream fis = new FileInputStream(fName);
ObjectInputStream ois = new ObjectInputStream(fis);
MyObject object = (MyObject) ois.readObject();
```

- Siehe Dokumentation und Beispiele in der Literatur

Ende der Wiederholung



6.7 Tabellen in Java

- Tabellen werden von *JTable* unterstützt.
- ***JTable*** unterstützt die Verwendung von Modellen, die das *TableModel* (Interface) implementieren.
- *TableModel* enthält 9 Methoden die implementiert sein müssen. → viel Arbeit.
- Aber: es gibt ein ***AbstractTableModel***, das dem Programmierer viel Arbeit abnimmt, nur wenige anwendungsspezifische Methoden *müssen* implementiert werden:


```
public int getRowCount();
public int getColumnCount();
public String getColumnName(int column);
public Object getValueAt(int rowIndex, int columnIndex);
```

- Weitere Methoden nur nach Bedarf, beispielsweise zum Verändern der Tabelle.

6.7 Tabellen in Java (2) (Beispiel: „tabelle“)

- Tabellen sind meist länger, daher ist ein Scroll-Balken von Vorteil.
- Ein Model vorausgesetzt, welches ein ***AbstractTableModel***, implementiert, ist folgender Code erforderlich:

```
MyTableModel tm = new MyTableModel();  // TableModel
JScrollPane scrollPane = new JScrollPane(); // Pane
JTable table = new JTable();              // Tabelle

  ... // Initialisierung
  // Scrollbalken anzeigen, falls benötigt
  scrollPane.setVerticalScrollBarPolicy(
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED );
  // Tabelle in Scroll-Pane einfügen
  scrollPane.getViewport().add( table );
  // Model für Tabelle setzen
  table.setModel( tm );
```

6.7 Tabellen in Java (3) (Beispiel: „tabelle“)

- Art der Auswahl einer Tabellenzeile siehe auch JList.
- Art der Bearbeitung der Ereignisse, bei der Auswahl in der Tabelle durch den Benutzer auf unterschiedliche Weise möglich. Hier die Verwendung durch ***ListSelectionListener***.

```
table.setSelectionMode(
   ListSelectionModel.SINGLE_SELECTION);
ListSelectionModel lsm = table.getSelectionModel();
// TableModelListener registrieren
lsm.addListSelectionListener( this );

... // Implementierung ListSelectionModel
public void valueChanged( ListSelectionEvent e ) {
   int row = table.getSelectedRow();
   ...
}
```

6.8 Tree-Views in Java (Beispiel: „tree“)

- Eine sehr vielseitige und mächtige Darstellung von Hierarchien in grafischen Benutzungsoberflächen sind Trees (Bäume)
- Baumartige Ansichten können in Java mit **JTree** realisiert werden
- Einbindung eines JTree erfolgt ähnlich wie JTable
- Die Daten eines Baums können unterschiedlich verwaltet werden:
  - Als **TreeNode**, beispielsweise mit **DefaultMutableTreeNode**
  - Als Model, beispielsweise mit **TreeModel**

- Im Beispiel „tree“ betrachten wir zuerst **DefaultMutableTreeNode**

6.8.1 JTree mit TreeNode

```
// Root-Knoten für Baum
DefaultMutableTreeNode root = new
    DefaultMutableTreeNode("Tierheim");
// Datenbaum erzeugen
DefaultMutableTreeNode hund = new
    DefaultMutableTreeNode("Hund");
DefaultMutableTreeNode katz = new
    DefaultMutableTreeNode("Katze");
DefaultMutableTreeNode sonst = new
    DefaultMutableTreeNode("Sonstige Haustiere");

protected void init() {
  root.add(hund); // Hund als Verzeichnis
  root.add(katz); // Katze als verzeichnis
  root.add(sonst); // ...
  // dem Verzeichnis Hund einen Hund hinzufügen
  Object obj = new EinTier( id, “DSH“, “Bello“);
  hund.add( new DefaultMutableTreeNode( obj ) );
  ...
```

6.8.1 JTree mit TreeNode (2)

```
    ...
    tree = new JTree( root ); // ! beachte Konstruktor

    // ... JFrame implements TreeSelectionListener
    tree.addTreeSelectionListener( this ); // Listener
}

// Implementierung des TreeSelectionListeners
public void valueChanged( TreeSelectionEvent event) {
    System.out.println( event.getPath() );
    Object obj = tree.getLastSelectedPathComponent();
    if ( obj != null )
        ...  // hier etwas tun
}
```

Alle weiteren Details im Zusammenhang im Beispiel “tree“

6.8.2 JTree mit TreeModel

- Will man konsequent das MVC-Muster verfolgen, ist ein für JTree geeignetes **Daten-Modell** zu realisieren.
- **TreeModel** ist ein Interface. Es wird beispielsweise durch **DefaultTreeModel** realisiert.
- Dieses Modell model wird der Darstellungskomponente
  `JTree tree = new JTree();`
  mit
  `tree.setModel( model );`
  bekannt gemacht.
- Für ein TreeModel sind eine ganze Reihe von Methoden zu implementieren (siehe Dokumentation).
- Beispiel: Darstellung der Hierarchie einer Dateistruktur (FileTreeDemo.java, *David Flanagan, JFC in A Nutshell*):