Informatik B

Vorlesung 18

GUI

Rückblick

- OOP
- Streams
- Collection
- Threads
- Reflection
- Netzwerk

AWT

- Das *Abstract Window Toolkit* (*AWT*) stellt ein API zur Erzeugung und Darstellung eines plattformunabhängigen GUI (*Graphical User Interface*) für Java-Programme dar
- AWT stellt das so genannte *Heavyweight*-Framework zur Darstellung von Steuerelementen dar
- Das bedeutet, dass AWT die nativen GUI-Komponenten des jeweiligen Betriebssystems zur Darstellung verwendet
- Diese nativen GUI-Komponenten werden *Peer* (englisch für "Partner") genannt
- *Heavyweight*, heißen diese Komponenten, weil teilweise umfangreiche Betriebssystem-Ressourcen mit ihnen verbunden sein können

AWT

- Die AWT-Klassen liegen im Package **`java.awt`**
- Dazu gehören unter anderem:
  - Fenstertypen
  - Steuerelemente
  - Klassen für das Layout
  - Klassen zur Menüsteuerung
  - Grafikkontext

AWT-Komponenten

- Alle visuell darstellbaren Klassen sind aus Basisklassen abgeleitet, die die benötigte Grundfunktionalität zur Verfügung stellen
- Basisklasse des AWT ist die Klasse **`Component`**
- Jedes GUI-Element ist von dieser Klasse abgeleitet
- Ausnahme bilden die Klassen zur Erstellung von Menüs, diese sind von **`MenuComponent`** abgeleitet

AWT-Komponenten

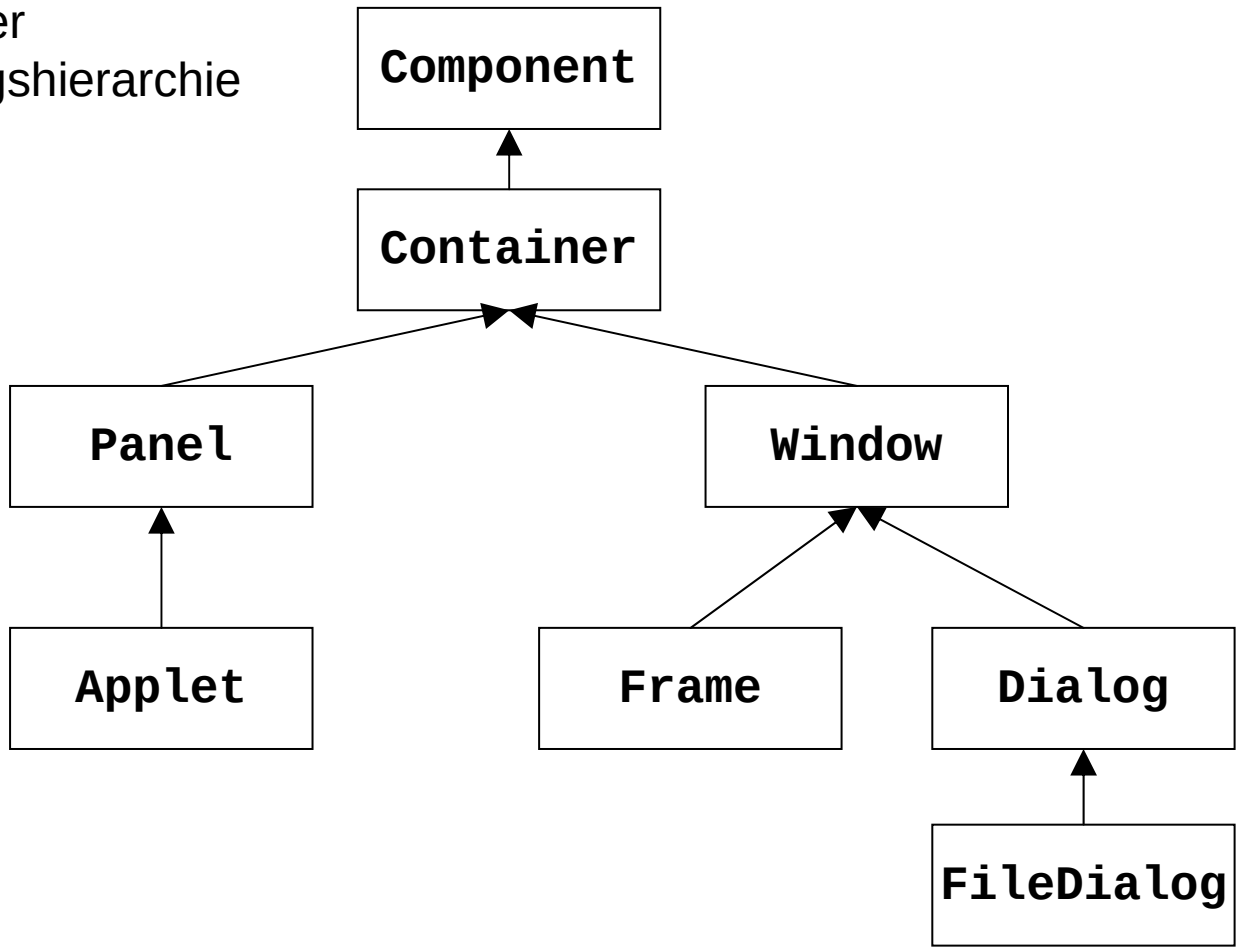
- **`Container`** ist eine Unterklasse von **`Component`**
- **`Container`** ist die Basisklasse für alle Komponenten, die wiederum weitere Komponenten beinhalten können
- Durch das Hinzufügen von Komponenten zu einem **`Container`** (**`add()`**) entsteht eine Komponentenhierarchie
- Jede Komponente bekommt den unmittelbar oberen **`Container`** als „parent“ zugeordnet

AWT-Komponenten

- Unterhalb von **`Container`** und **`Component`** liegen die konkreten Gestaltungselemente des AWT, mit denen eine Benutzeroberfläche aufgebaut werden kann
- Dazu gehören Fensterklassen (**`Window`**, **`Frame`**, **`Dialog`**, **`FileDialog`**,…) und Steuerelemente (**`Button`**, **`Slider`**, …)
- Die Fensterklassen sind **`Container`**, die wiederum weitere **`Component`**-Objekte (**`Container`** und Steuerelemente) enthalten können

AWT-Komponenten

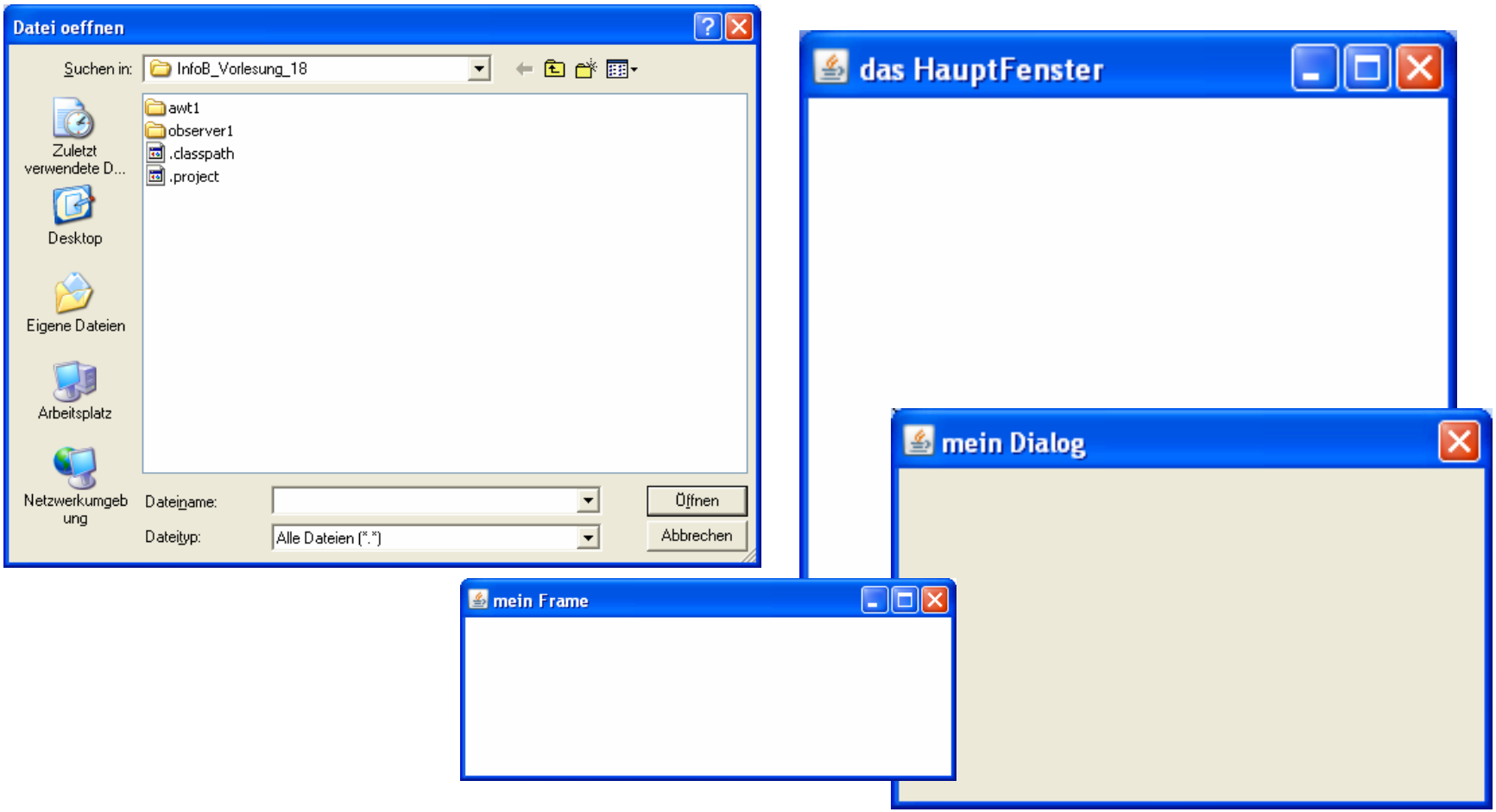
Auszug der
Vererbungshierarchie



AWT-Komponenten

- Für die Strukturierte Anordnung von Elementen dienen u.A. folgende Unterklassen von **`Container`**:
  - **`Panel`**: Darstellungsfläche, Containerklasse für Steuerelemente, kann in einem anderem **`Container`** enthalten sein
  - **`ScrollPane`**: Containerklasse die ein Steuerelement enthält und ggf. mit Rollbalken versieht
  - **`Dialog`**: (Modal-) Dialog, z.B. *OK/Cancel*-Anfrage
  - **`FileDialog`**: Datei öffnen/speichern-Dialog
  - **`Frame`**: Fenster mit Rahmen, Titelleiste und Menü
  - **`Window`**: Einfaches Fenster ohne Rahmen, Titelleiste oder Menü

Fenster

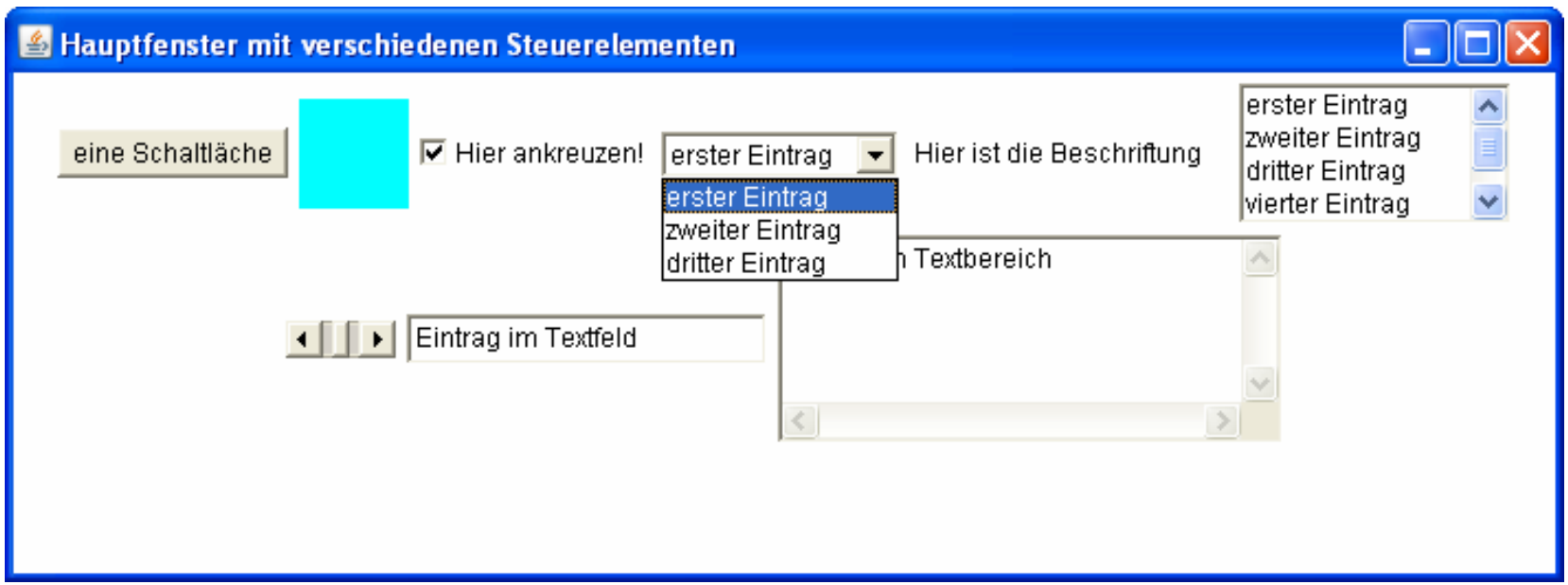


Beispiel: **`awt1`**

AWT-Steuerelemente

- In AWT gibt es verschiedene Steuerelemente:
  - **`Button`**: Eine beschriftete Schaltfläche
  - **`Label`**: Eine (änderbare) Textzeile
  - **`TextField`**: Eingabe/Anzeige einer Textzeile
  - **`TextArea`**: Eingabe/Anzeige mehrerer Textzeilen
  - **`Canvas`**: Zeichenfläche
  - **`Choice`**: Liste mit Einträgen, wobei einer ausgewählt werden kann
  - **`List`**: Auswahl eines oder mehrerer Einträge aus einer Liste, ggf. mit Rollbalken versehen

AWT-Steuerelemente



- Beispiel: **`awt2`**

AWT

- Es gibt sehr viele verschiedene Klassen, um eine GUI zu gestalten
- Dazu gehören die **`Container`** und die Steuerelemente
- Es soll im Folgenden nicht auf jede Klasse einzeln eingegangen werden
- Weitere Informationen finden sich unter: http://java.sun.com/javase/6/docs/api/java/awt/package-summary.html
- Im Folgenden sollen einige Konzepte näher erläutert werden

Hinzufügen von Komponenten

- Zu einem **`Container`** können Komponenten oder weitere Container hinzugefügt werden
- Sind die gewünschten Komponenten hinzugefügt, muss dem obersten **`Container`** das Signal gegeben werden, alles zusammenzubauen
- Dazu wird die Methode **`pack()`** aufgerufen, die die Größe der enthaltenen Komponenten bestimmt und die Größe des Fensters ermittelt
- Um das Fenster anzuzeigen muss dann noch die Methode **`setVisible(boolean b)`** aufgerufen werden, wobei **`b`** den Wert **`true`** haben muss, damit das Fenster sichtbar wird

Anordnung von Komponenten

- Um eine Oberfläche zu gestalten, müssen die einzelnen Komponenten passend angeordnet werden können
- Dazu kommt, dass ein Fenster ggf. seine Größe verändern kann und sich das Layout dabei anpassen muss
- Für diese Aufgabe gibt es verschiedene **`LayoutManager`**
- Je nach **`LayoutManager`** werden die einzelnen Komponenten angeordnet
- Ein Container kann genau einen **`LayoutManager`** besitzen

Anordnung von Komponenten

- Es gibt unter anderem folgende **`LayoutManager`**:
  - **`FlowLayout`**: Ordnet Komponenten von links nach rechts an
  - **`BoxLayout`**: Ordnet Komponenten horizontal oder vertikal an
  - **`GridLayout`**: Setzt Komponenten in ein Raster, wobei jedes Element die gleichen Ausmaße besitzt
  - **`BorderLayout`**: Setzt Komponenten in vier Himmelsrichtungen oder in der Mitte
  - **`GridBagLayout`**: Sehr flexibler Manager als Erweiterung von **`GridLayout`**
  - **`CardLayout`**: Verwaltet Komponenten wie auf einem Stapel, von dem nur eine Komponente sichtbar ist
- Um einem **`Container`** eine Ausrichtungstrategie zu setzen, wird die Methode **`setLayout()`** verwendet

Anordnung von Komponenten

- Oft reicht ein **`LayoutManager`** nicht aus
- Dann kann man in einem Container weitere Container einbauen, die einen eigenen **`LayoutManager`** besitzen
- So kann das Layout einer GUI fein granular aufgebaut werden
- Nachteil der geschachtelten Container ist die mangelnde Übersicht im Quellcode
- Daher ist es bei der GUI-Programmierung besonders wichtig auf sinnvolle Variablenbezeichner zu achten

`FlowLayout`

- Der **`FlowLayout`**-Manager setzt die Elemente von links nach rechts in eine Zeile
- Die Komponenten behalten ihre Größe, das heißt, der Layoutmanager gibt keine neue Größe vor
- Passen nicht alle Elemente in eine Zeile, so werden sie untereinander angeordnet
- Ein zusätzlicher Parameter bestimmt, wie die Elemente im **`Container`** positioniert werden: zentriert, rechts- oder linksbündig, ohne Einstellung ist die Anzeige zentriert
- Beispiel: **`flowlayout1`**

`BorderLayout`

- Ein **`BorderLayout`** unterteilt seine Zeichenfläche in fünf Bereiche:
  Norden, Osten, Süden, Westen und Mitte (Center)
- Die Elemente im Norden und Süden erstrecken sich immer über die gesamte Breite des **`Containers`**
- Die Höhe des Nordens und Südens ergibt sich aus der Wunschhöhe der enthaltenen Komponenten (Childs), und die Breite wird angepasst
- Die Elemente rechts und links bekommen ihre gewünschte Breite, werden aber in der Höhe gestreckt
- Das Element in der Mitte wird in Höhe und Breite angepasst

BorderLayout

- Für jeden dieser Bereiche (Richtungen) sieht die Klasse **`BorderLayout`** eine Konstante vor:
  - **`BorderLayout.CENTER`**
  - **`BorderLayout.NORTH`**
  - **`BorderLayout.EAST`**
  - **`BorderLayout.SOUTH`**
  - **`BorderLayout.WEST`**
- Dem **`Container`** wird mit der Methode **`add(Komponente, Ausrichtung)`** eine Komponente hinzugefügt
- Wird die Methode **`add()`** mit nur einem Argument aufgerufen, so wird die Komponente automatisch in die Mitte (Center) gesetzt
- Beispiel: **`borderlayout1`**

`GridLayout`

- Das **`GridLayout`** ordnet seine Komponenten in Zellen an
- Jeder Komponente in der Zelle wird dieselbe Größe zugeordnet, also bei drei Elementen in der Breite ein Drittel des Containers
- Wird der Container vergrößert, so werden die Elemente gleichmäßig vergrößert und bekommen so viel Platz wie möglich
- Beim Konstruktoraufruf kann die Spalten- und Zeilenzahl angegeben werden
- Der **`LayoutManager`** verteilt die Komponenten dennoch recht eigenmächtig
- Will man unbedingt drei Spalten, so sollte man dafür den Wert drei angeben, aber die Zeilenzahl auf null setzen
- Beispiel: **`gridlayout1`**

Null-Layout

- Das Argument **`null`** bei **`setLayout()`** setzt keinen **`LayoutManager`**, und die Komponenten müssen absolut positioniert werden
- Zum Setzen der Position und Ausmaße bietet jede **`Component`** die Methode **`setBounds(int x, int y, int breite, int höhe)`**
- Das Setzen vom Null-Layout sollte nicht die Regel sein, da Änderungen an der Zeichensatzgröße hässliche Effekte nach sich ziehen
- Da die Methode **`pack()`** die Größe der einzelnen Komponenten erfragt und die **`Container`** entsprechend skaliert, sollte die Methode aufgrund unerwünschter Seiteneffekte beim Null-Layout nicht aufgerufen werden
- Beispiel: **`nulllayout1`**

`CardLayout`

- Es können auf einer Oberfläche mehrere Ebenen angeordnet werden
- Dabei ist lediglich die erste Ebene sichtbar
- Über Methodenaufrufe kann die sichtbare Ebene gewechselt werden
- Dazu wird ein neues **`CardLayout`**-Objekt erzeugt: **`CardLayout cards = new CardLayout();`**
- Die Referenz sollte man sich merken, da darüber festgelegt wird, welche Ebene oben liegen soll (alternativ kann man den **`LayoutManager`** mit der Methode **`getLayout()`** erfragen)

`CardLayout`

- Je Ebene wird ein **`Container`** (mit beliebigem Layout) erzeugt
- Jede Ebene kann individuell gestaltet werden
- Die einzelnen **`Container`** der Ebenen werden dann dem **`Container`** mit dem **`CardLayout`** hinzugefügt
- Dies wird mit der Methode **`add`** erledigt, wobei als zweiter Parameter ein **`String`** angegeben werden muss, der den Card-Container identifizieren soll
- Beispiel: **`cardlayout1`**

Zusammenfassung

- AWT
- Container
- Component
- Layout

Ausblick

- Vom Klick zum Ereignis:
  AWT-Eventhandling