Informatik B - Objektorientierte Programmierung in Java

Vorlesung 12:
GUI 2: AWT (2) - Ereignisverarbeitung

Inhalt

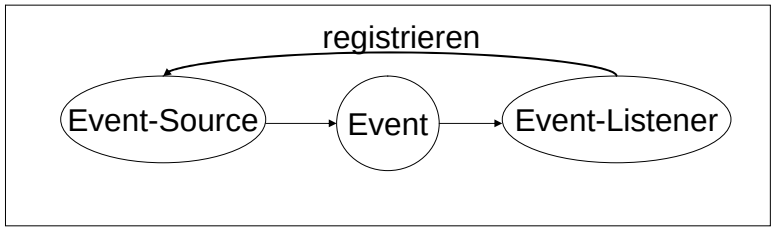
- Ereignisverarbeitung
- Anonyme Klassen als Ereignisempfänger
- AWT-Menüs
- Zwischenablage

Ereignisverarbeitung

- Überblick
  - Bisher haben wir nur Steuerelemente definiert und angezeigt.
  - Damit eine Benutzungsschnittstelle auch wie vorgesehen auf Eingaben reagiert, müssen Ereignisse (events) verarbeitet werden.
  - Ereignisse entstehen aufgrund von Aktionen des Benutzers und müssen von den jeweils betroffenen Elementen der Schnittstelle verarbeitet werden.
  - Zu den Ereignissen gehören u.a.:
    - Mausbewegungen
    - Tastendruck (Maus/Tastatur)
    - Betätigen von Rollbalken (Scrolling)
    - Entering

Ereignisverarbeitung

- Grundprinzip (seit JDK 1.1): Delegation Event Model
  - Ereignisse entstehen an einer Ereignisquelle (event source).
  - Die Quelle erzeugt ein Ereignis und übergibt es an alle registrierten Listener-Objekte (event listener) oder verwirft es.
  - Bei Ereignisquelle und Listener-Objekt kann es sich um dasselbe Objekt handeln, muss es aber nicht.



Ereignisverarbeitung

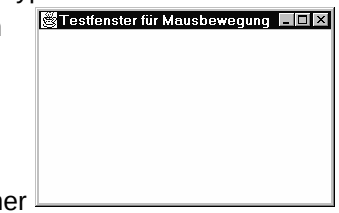
- Das Listener-Objekt wartet auf Ereignisse eines bestimmten Typs.
- Für jeden Ereignistyp (Tastendruck, Mausbewegung) existiert eine eigene Listener-Klasse.
- Ist für einen Ereignistyp kein Listener-Objekt registriert, findet keine Ereignisverarbeitung statt.
- Somit werden nur diejenigen Ereignisse verarbeitet, auf die ein Listener-Objekt „hört“ (lauscht).

Ereignisverarbeitung

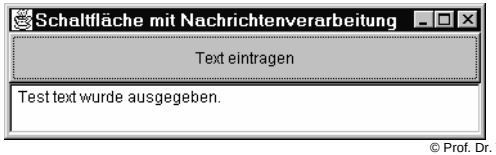
- Beispiel:
- `Button` verfügt über folgende Operationen zur Ereignisverarbeitung:
  - `void setActionCommand(String command)`
    - setzen des Befehlskürzels für die Versendung von Aktionsnachrichten
  - `String getActionCommand()`
  - `void addActionListener(ActionListener l)`
    - Zuordnen eines „Aktions-Listener-Objekts“
    - Dadurch werden Aktionsereignisse, die bei Mausklick auf die Schaltfläche entstehen, an das Listener-Objekt weitergegeben.
  - `void removeActionListener(ActionListener l)`

Ereignisverarbeitung

- In den Beispielen sind die Operationen der Listener-Schnittstellen häufig in den Containerobjekten (z.B. das Hauptfenster einer Anwendung) implementiert, die als Listener-Objekt ihren Komponenten zugeordnet werden (`addXXXListener`-Operationen). Dies ist aber nicht typisch.
  - Hauptfenster MausLauschen

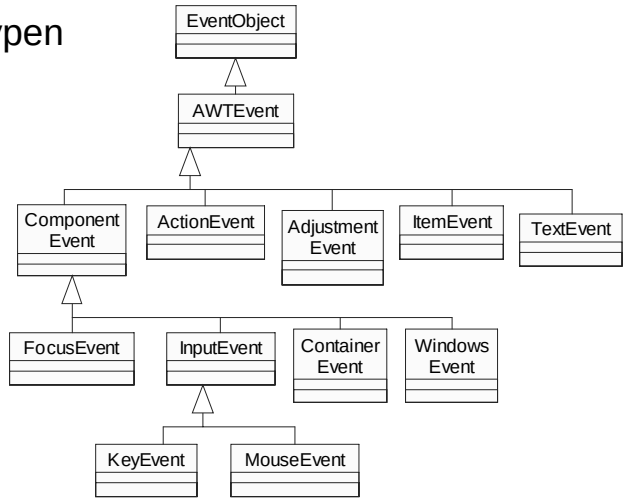


  - Komponente ButtonmitListener



Ereignisverarbeitung

- Ereignistypen



Ereignisverarbeitung

- Ereignistypen

Ereignisklasse	Bedeutung
EventObject	Basisklasse aller Ereignisse
--- AWTEvent	Ereignis in der Benutzerschnittstelle (abstrakte Basisklasse)
--- ActionEvent	Aktionsauslösung in der Benutzerschnittstelle
--- AdjustmentEvent	Anpassung von Rollleisten/Rollflächen (scroll bars)
--- ItemEvent	Änderung eines Auswahl- oder Listenelements
--- TextEvent	Textänderung in TextField, TextArea
--- ComponentEvent	Änderung von Komponenten
--- FocusEvent	Fokuswechsel von Komponenten in einem Container
--- ContainerEvent	Hinzufügen/Löschen von Komponenten in einem Container
--- WindowEvent	Öffnen/Schließen, Aktivieren, Minimieren etc. von Fenstern
--- InputEvent	Eingabeereignisse (abstrakte Oberklasse)
--- MouseEvent	Mauseingabeereignisse (bewegen, klicken etc.)
--- KeyEvent	Tastatureingabeereignisse (Taste drücken, loslassen)

Ereignisverarbeitung

- Ereignistypen
  - Ereignisse werden in zwei grobe Kategorien eingeteilt:
    - „primitive“ Ereignisse
      - Ereignisse unterhalb von `ComponentEvent`
      - Abfangen von Benutzereingaben (`InputEvent`: Maus, Tastatur)
      - Fenstermanipulation (Minimieren von Fenstern, Fokuswechsel)
    - „semantische“ Ereignisse
      - Benutzer hat eine bestimmte Taste gedrückt
      - Statuswechsel beim Anklicken eines Kontrollkästchens (Checkbox)
      - Jeder Typ von Steuerelement kann dabei unterschiedliche Typen semantischer Ereignisse auslösen:
        - ein Button z.B. `ActionEvent`
        - eine Scrollbar z.B. `AdjustmentEvent`
        - eine Liste z.B. `ActionEvent` und `ItemEvent`

Ereignisverarbeitung

- Ereignistypen
  - Zuordnung von AWT-Komponenten zu den von ihnen ausgelösten Ereignistypen:

	ActionEvent	AdjustmentEvent	ComponentEvent	ContainerEvent	FocusEvent	ItemEvent	KeyEvent	MouseEvent	TextEvent	WindowEvent
Button	x		x		x		x	x		
Canvas			x		x		x	x		
Checkbox			x		x	x	x	x		
CheckboxMenuItem	x									
Choice			x		x	x	x	x		
Component			x		x		x	x		
Container			x	x	x		x	x		

Ereignisverarbeitung

- Fortsetzung

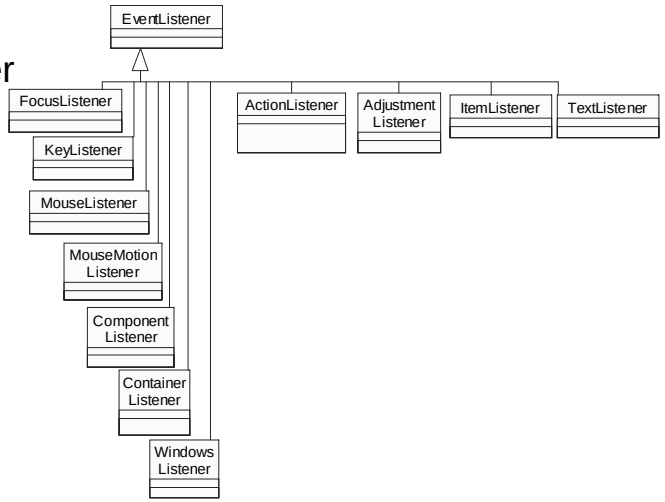
	ActionEvent	AdjustmentEvent	ComponentEvent	ContainerEvent	FocusEvent	ItemEvent	KeyEvent	MouseEvent	TextEvent	WindowEvent
Dialog			x	x	x		x	x		x
Frame			x	x	x		x	x		x
Label			x		x		x	x		
List	x		x		x	x	x	x		
MenuItem	x									
Panel			x	x	x		x	x		
Scrollbar		x	x		x		x	x		
ScrollPane			x	x	x		x	x		
TextArea			x		x		x	x	x	
TextComponent			x		x		x	x	x	
TextField			x		x		x	x	x	
Window			x	x	x		x	x		x

Ereignisverarbeitung

- Ereignis-Empfänger
  - Auf ein Ereignis wird nur reagiert, wenn ein zu ihm passendes Listener-Objekt existiert.
  - Die `EventListener` sind Schnittstellen (Interfaces), die analog zu den entsprechenden Ereignistypen (z.B. `MouseEvent`) Operationen für das Verarbeiten dieser Nachrichten zusammenfassen.
  - Damit ein Listener-Objekt auf Ereignisse reagieren kann, muss sie eine der `EventListener`-Schnittstellen implementieren, d.h. für alle Operationen eines `EventListener` eine Implementierung angeben.

Ereignisverarbeitung

- Ereignis-empfänger



Ereignisverarbeitung

Listener-Schnittstelle	Nachrichtenbearbeitungsoperationen, die bei Verwenden der Schnittstelle implementiert werden müssen
ActionListener	`void actionPerformed(ActionEvent e)`
AdjustmentListener	`void adjustmentValueChanged(AdjustmentEvent e)`

ComponentListener	`void componentHidden(ComponentEvent e)`
	`void componentMoved(ComponentEvent e)`
	`void componentResized(ComponentEvent e)`
	`void componentShown(ComponentEvent e)`

ContainerListener	`void componentAdded(ContainerEvent e)` `void componentRemoved(ContainerEvent e)`
FocusListener	`void focusGained(FocusEvent e)` `void focusLost(FocusEvent e)`
ItemListener	`void itemStateChanged(ItemEvent e)`

KeyListener	`void keyPressed(KeyEvent e)`
	`void keyReleased(KeyEvent e)`
	`void keyTyped(KeyEvent e)`

Ereignisverarbeitung

Listener-Schnittstelle	Nachrichtenbearbeitungsoperationen, die bei Verwenden der Schnittstelle implementiert werden müssen

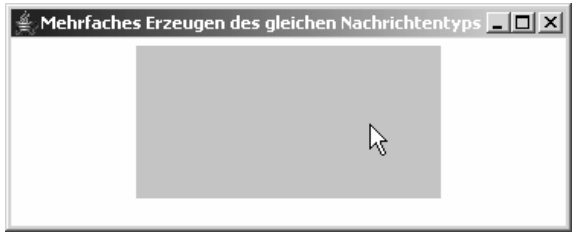
MouseListener	`void mouseClicked(MouseEvent e)`
	`void mouseEntered(MouseEvent e)`
	`void mouseExited(MouseEvent e)`
	`void mousePressed(MouseEvent e)`
	`void mouseReleased(MouseEvent e)`

MouseMotionListener	`void mouseDragged(MouseEvent e)` `void mouseMoved(MouseEvent e)`
TextListener	`void textValueChanged(TextEvent e)`

WindowListener	`void windowActivated(WindowEvent e)`
	`void windowClosed(WindowEvent e)`
	`void windowClosing(WindowEvent e)`
	`void windowDeactivated(WindowEvent e)`
	`void windowDeiconified(WindowEvent e)`
	`void windowIconified(WindowEvent e)`
	`void windowOpened(WindowEvent e)`

Ereignisverarbeitung

- Beispiele:
  - Die Klasse `MausLauschen` (siehe vorne)
  - Die Klasse `ButtonMitListener` (siehe vorne)
  - Die Klasse `Multiple`:



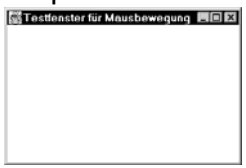
Bewegen Sie die Maus vom Panel (weißer Rand) in die graue Zeichenfläche und schauen Sie sich die Ausgabe an.

Ereignisverarbeitung

- Adapterklassen für Listener-Schnittstellen
  - Bei der Deklaration einer Klasse, die eine Listener-Schnittstelle implementieren soll, müssen in dieser Klasse immer alle Operationen der Listener-Schnittstelle implementiert werden, was oft gar nicht wünschenswert ist.
  - Will man nur eine Operation benutzen, so müssten alle anderen Operationen einer Schnittstelle als Operation ohne Rumpf implementiert werden, was sehr umständlich ist.

Ereignisverarbeitung

- Im Paket `java.awt.event` existiert daher parallel zu jeder Listener-Schnittstelle, die mehr als eine Operation enthält, eine entsprechende Adapterklasse, die eine „leere“ Implementierung der Schnittstelle darstellt.
- Statt eine Listener-Schnittstelle zu implementieren, kann man eine neue Klasse auch vom entsprechenden Adapter ableiten und muss dann nicht mehr alle Operationen der Schnittstelle implementieren.
- Beispiel:
  - MausLauschenMitAdapter



Ereignisverarbeitung

- Bei den bisher vorgestellten Beispielen diente die GUI-Komponente selbst (siehe MausLauschen) bzw. der übergeordnete Container (siehe ButtonMitListener) als Ereignisempfänger.
- Dabei treten folgende Probleme auf:
  - Implementierung einer Listener-Schnittstelle
    - Alle Operationen des Interface müssen implementiert werden.
  - Ableitung einer Adapterklasse
    - Der Ereignisempfänger kann aus keiner anderen Klasse abgeleitet werden.
- Alternativ dazu kann ein Ereignisempfänger auch als *anonyme Klasse* realisiert werden.

Anonyme Klassen

- Eigenschaften
  - Die Klasse erhält keinen Namen.
  - Anonyme Klassen werden innerhalb eines Ausdrucks definiert und instantiiert.
  - Nach `new` *`Klassenname`*`()` folgt in geschweiften Klammern die Definition der Klasse.
  - Wird für *`Klassenname`* der Name eines Interfaces angegeben, implementiert die anonyme Klasse dieses Interface und erbt von der Klasse **`Object`**.
  - Handelt es sich bei *`Klassenname`* um den Namen einer Klasse, so wird die anonyme Klasse daraus abgeleitet.

Anonyme Klassen

- Anonyme Klassen als Ereignisempfänger
  - Bei der Realisierung eines Ereignisempfängers implementiert die anonyme Klasse entweder eine Listener-Schnittstelle oder wird aus einer Adapterklasse abgeleitet.
  - Beispiele:
    - ButtonMitListenerAnonym
    - MausLauschenAnonym

AWT-Menüs

- Überblick
  - Ein wichtiger Teil von Benutzungsoberflächen sind Menüs, über die die Funktionen eines Programms aufzurufen sind.
  - Es gibt zwei verschiedene Typen von Menüs:
    - Menüleiste, die am oberen Fensterrand verankert ist und über einfache oder geschachtelte Listen von Menüeinträgen verfügt
    - Popup- oder Kontextmenü, welches per Mausklick aktiviert werden kann und sich meistens an der Position des Mauszeigers öffnet
  - Die Befehle der Menüs können meistens auch über sog. Tastatur-Shortcuts (z.B Ctrl+C) angesprochen werden.

AWT-Menüs

- Im AWT gibt es eine Reihe von Klassen, die die Menüfunktionalität realisieren:
  - Die Klassen `MenuBar` und `PopupMenu` definieren die zwei verschiedenen Menütypen. Sie werden einem Fenster (Frame) bzw. Container durch die Operationen
    - `void setMenuBar(MenuBar dieMenueLeiste)` bzw.
    - `Component add(Component dasPopupMenu)`

    zugefügt.
  - Dabei enthält eine Menüleiste ein oder mehrere Objekte vom Typ `Menu`, jedes Menü wiederum ein oder mehrere Menüeinträge (`MenuItem`).
  - Menüs können geschachtelt werden.

AWT-Menüs

- Damit auf das Anklicken eines Menüs auch die gewünschte Aktion ausgelöst wird, muss man
  - den Menüeinträgen Aktionskommandos zuordnen, mit Hilfe der Operation
    `void setActionCommand(String derBefehlsname)`
  - und ein Listener-Objekt für „semantische Aktionen“ registrieren, durch die Operation
    `void addActionListener(ActionListener einAktionsLauscher)`
    Die Menüaufrufe werden wie alle semantischen Ereignisse durch die Implementierung der `actionPerformed`-Operation im Listener-Objekt abgearbeitet.

AWT-Menüs

- Bei der Erzeugung eines Menüeintrages kann ihm ein passendes Tastaturkürzel zugewiesen werden.
- Dazu erzeugt man ein (anonymes) Objekt der Klasse `MenuShortCut` und übergibt es dem Konstruktor des Menüeintrages:
  - `MenuItem einMenueEintrag = new MenuItem(“Text“, new MenuShortcut(KeyEvent.VK_T));`
  - Die Anwahl von Menüs über die Tastatur ist betriebssystemabhängig – meist muss man zugleich mit dem angegebenen Shortcut (hier: KeyEvent.VK_T, d.h. der virtual key „T“) eine Sondertaste drücken, z.B. die Control- bzw. Steuertaste.

Zwischenablage

- Überblick
  - Für die Steuerung der Zwischenablage ist in Java ein eigenes Unterpaket (`java.awt.datatransfer`) des AWT vorgesehen.
  - Neben der Klasse `Clipboard` mit Operationen für die Ansteuerung der Zwischenablage
    - `void setContents (Transferable derInhalt, ClipboardOwner derClipboardBesitzer)`
      um etwas in die Zwischenablage einzufügen bzw.
    - `Transferable getContents(Object dasNachfragendeObjekt)`
      um ihren Inhalt zu lesen,

    ist dort die Klasse `DataFlavor` definiert.

Zwischenablage

- `DataFlavor` dient der Beschreibung von Datenformaten, die benutzt werden können, um in die Zwischenablage zu schreiben oder Daten aus der Zwischenablage zu holen.
- Für den Zugriff auf String-Objekte und Dateilisten sind die beiden Datenformate `stringFlavor` und `javaFileListFlavor` als Eigenschaften der Klasse `DataFlavor` vordefiniert.

Zwischenablage

- Beispiel `MenueClipBoard`:
  - Ein umfangreiches Beispiel zur Verwendung von Menüs und der Zwischenablage befindet sich in `MenueClipBoard` und hat folgendes Aussehen: