Der Java Event-Mechanismus

Wie erfährt Objekt a, dass Objekt b sich geändert hat?

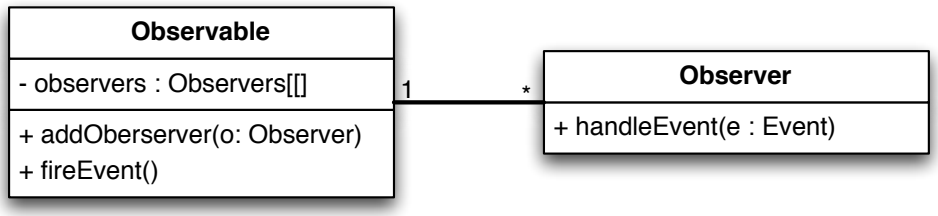
Möglichkeit I (Polling)

- Objekt a untersucht den Zustand von b.
  - a prüft auch, wenn sich b nicht geändert hat.
  - Wie merkt a, **dass** sich b geändert hat?
  - die Attribute von b müssen a alle bekannt sein.

Möglichkeit II (Events)

- Objekt b teilt Objekt a mit, dass sich etwas geändert hat.
  - ✓ keine unnötigen Polls
  - ✓ Geheimnisprinzip bleibt erhalten.
  - - Objekt b muss alle Objekte kennen, die benachrichtigt sein wollen.

Das Observer-Muster




```
class Observable {
   Observer[] observers;
   ...
```

```
} // Klassendeklaration
```

```
class Observable {
   Observer[] observers;
   ...

   public void addObserver(Observer obs) {
      observers[n+1] = obs; //so ähnlich
   }









} // Klassendeklaration
```



```
class Observable {
   Observer[] observers;
   ...

   public void addObserver(Observer obs) {
      observers[n+1] = obs; //so ähnlich
   }

   public void fireEvent() {
      // erzeuge ein neues Event-Objekt
      Event evt = new Event();

      // Benachrichtige alle Observer
      for(int i = 0; i<observers.length(); i++) {
         observers[i].handleEvent(evt);
      }
   } // fireEvent

} // Klassendeklaration
```

```
class Observer {

   public void handleEvent(Event e) {
      // Aha, b hat sich geändert.
      // Neue Werte stehen in e.
      // Ich kann darauf reagieren
      // ... und weiß sonst nichts von b
   }

} // Klassendeklaration
```


Bezeichnung

- Alle GUI-Elemente sind Observables
- ➡ In Swing kommt der Name Observable **nicht** vor!
- Oberserver heißen in Swing Listener

Event-Zoo

- Mausclick
- Fenstergröße verändert
- Button gedrückt
- Listenelement ausgewählt
- Maus bewegt
- Taste gedrückt
- Text markiert
- Maustaste gedrückt
- Maustaste losgelassen
- Textfeldeintrag geändert
- Fensterposition geändert
- Maus ist über dem Button
- Maus ist nicht mehr über dem Button
- .....

Event-Zoo

AWTEvent, BeanContextEvent, CaretEvent, ChangeEvent, ConnectionEvent, DragGestureEvent, DragSourceEvent, DropTargetEvent, FlavorEvent, HandshakeCompletedEvent, HyperlinkEvent, LineEvent, ListDataEvent, ListSelectionEvent, MenuEvent, NamingEvent, NamingExceptionEvent, NodeChangeEvent, Notification, PopupMenuEvent, PreferenceChangeEvent, PrintEvent, PropertyChangeEvent, RowSetEvent, RowSorterEvent, SSLSessionBindingEvent, StatementEvent, TableColumnModelEvent, TableModelEvent, TreeExpansionEvent, TreeModelEvent, TreeSelectionEvent, UndoableEditEvent, UnsolicitedNotificationEvent

```
java.lang.Object
  └ java.util.EventObject
      └ java.awt.AWTEvent
          └ java.awt.event.ComponentEvent
              └ java.awt.event.InputEvent
                  └ java.awt.event.MouseEvent
```

Method Summary	
void	**mouseClicked**(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	**mouseEntered**(MouseEvent e) Invoked when the mouse enters a component.
void	**mouseExited**(MouseEvent e) Invoked when the mouse exits a component.
void	**mousePressed**(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(MouseEvent e) Invoked when a mouse button has been released on a component.

Granularität von Events

- Benutzer macht einen Mausklick
- ➡ Events:
  - mouseClicked(MouseEvent e)
  - mousePressed(MouseEvent e)
  - mouseReleased(MouseEvent e)

Fragen?