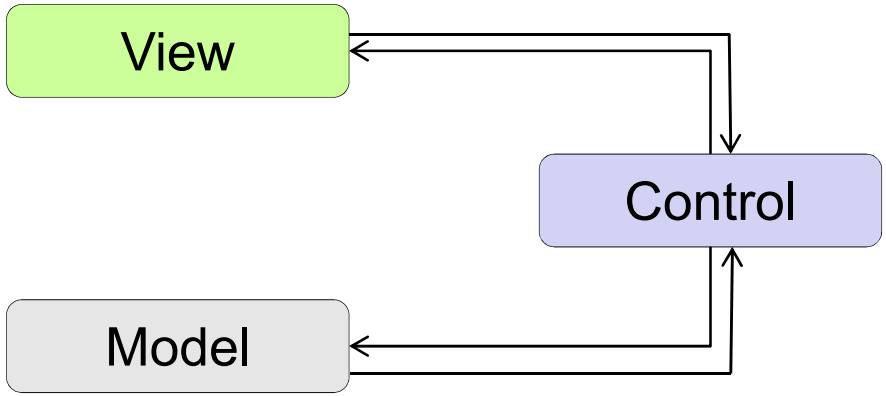
Grundlagen der Objektorientierung

Model-View-Control-Muster
(MVC Pattern)

Was bedeutet MVC?

Eine Software, die auf dem MVC-Pattern basiert, besteht aus

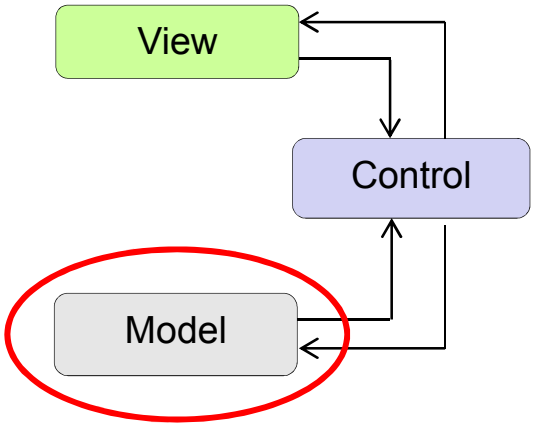
- einem Modell (Modellklassen, *Model*),
- einer oder mehreren Darstellungskomponenten (*View*) sowie
- einem Steuerungsmodul (*Control*)



Bildet den Grundstock für die komponentenbasierte Softwareentwicklung

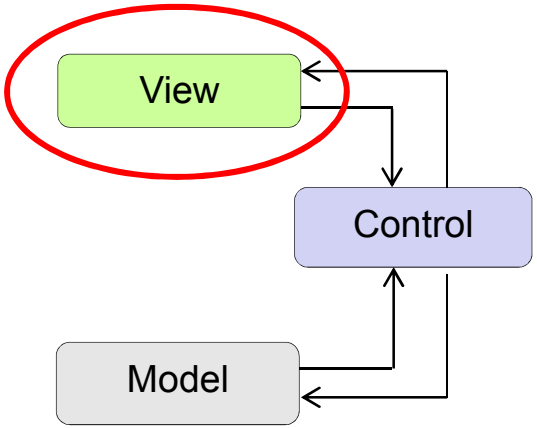
MVC: Modell

Modell(-klassen):



- Basisklassen, welche meist persistent (z.B. Datenbank) gespeichert werden
- Statisch, d.h. passiv (bei Modelklassen werden nur Attributwerte gesetzt, lediglich *PropertyChangeEvents* werden gesendet)
- Basisklassen werden ausschließlich von der Steuerung aktiv referenziert (angelegt, geändert, gelöscht)
- Basisklassen können via *EntityManager* verwaltet werden

MVC: View

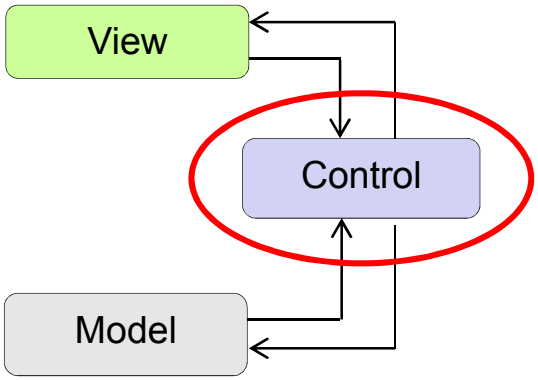


View:

- empfängt Signale (Events) von der Steuerung
- stellt die Eigenschaften von Modellklassen und anderen Klassen dar (GUI)
- gibt vom Benutzer ausgelöste Signale bzw. Events an die Steuerung weiter
- dynamisch, d.h. aktiv (auf Benutzereingaben wartend)
- Implementierung:
  - typische GUI-Klassen mit GUI-Elementen
  - mit Observer-Pattern (für Updates) realisiert (s. Entwurfsmuster)

MVC: Control

Steuerung (Control):



- empfängt Signale (Events) von der GUI und von den Modellklassen
- gibt die Signale bzw. Events an die GUI bzw. Modellklassen weiter
- dynamisch, d.h. aktiv (auf Events wartend)
- Implementierung:
  - besteht genau genommen überwiegend aus `if-then-else`-Konstrukten, welche die eingehenden Events individuell behandeln
  - realisiert Observer-Pattern (s. Entwurfsmuster)

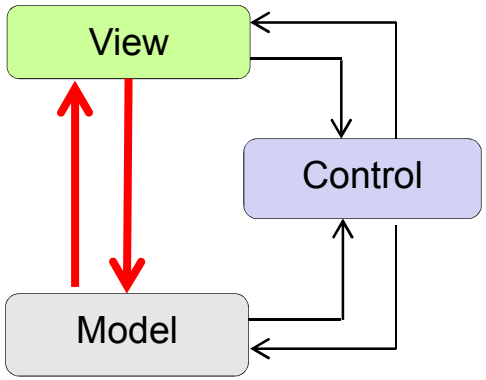
Problematik: Trennung von M, V und C

MVC wird in der Praxis auf mehrere Weisen verwendet:

- In der VIEW werden MODEL-Elemente direkt referenziert
  ==> unübersichtlich bei vielen GUI-Komponenten

- In der VIEW wird CONTROL mitrealisiert
  ==> ebenfalls unübersichtlich

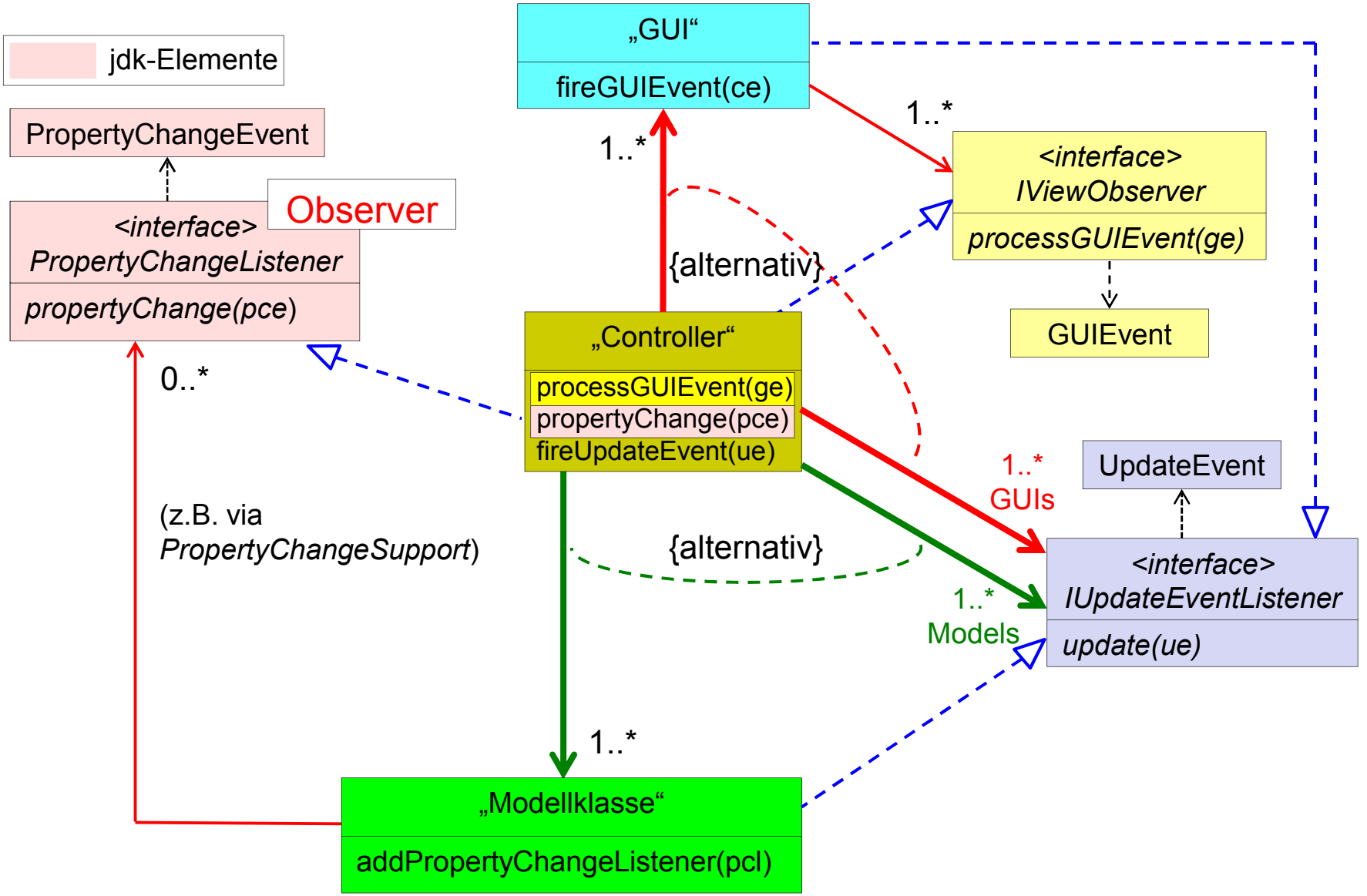
  Bsp: direktes Verarbeiten von Button-Events in (anonymen) *ActionListenern*



=> Bei hierarchischen GUIs aus Performancegründen oft sinnvoll

=> Saubere Trennung erfolgt mit MVP-Pattern (Model-View-Presenter)

MVC: Kommunikation



Besonderheiten

Besteht eine GUI-Komponente aus weiteren GUI-Komponenten:

- so kann die „oberste“ Komponente zusätzlich als *IViewObserver* fungieren und die *GUIEvents* der Unterkomponenten an die Haupt-GUI weiterleiten
- Dazu werden die Unterkomponenten zusätzlich zu „Sendern“ mit Observer-Management (*addObserver() & removeObserver()*)
- Analog werden die *UpdateEvents* an die Unterkomponenten weitergeleitet:
  - direkt (nur aus Performancegründen direkt an die jeweilige Instanz) oder
  - indirekt (als *IUpdateEventListener*)