DM575

Forelæsning 12: Opførselsmønstre

Casper Bach
Institut for Matematik og Datalogi

SDU

Baseret på originalslides af Robin Kaarsgaard Sales

Opførselsmønstre

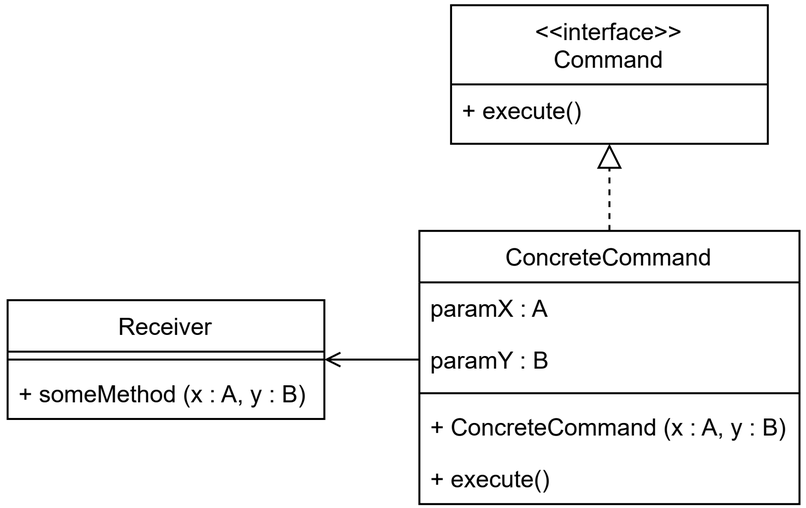
- Opførselsmønstre dækker over mønstre der kontrollerer eller faciliterer kommunikationen imellem programkomponenter.
- Disse kan tillade lavere kopling imellem klasser.
- **Husk på:** Designmønstre er ikke en tjekliste!

Katalog

- Chain of responsibility
- ★ Command
- Interpreter
- ★ Iterator
- Mediator
- Memento
- ★ Observer
- ★ State
- Strategy
- Template Method
- Visitor
- ★ **Bonus:** MVC

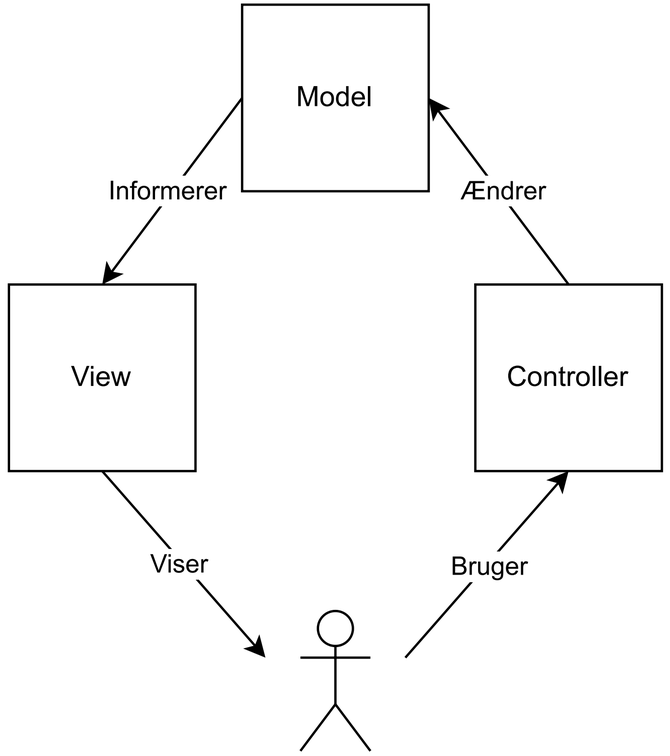
Command

- **Motivation:** At omdanne metodekald til programlogik til objekter der (indeholder al information om kaldet og) kan passeres rundt.
- **Deltagere:** Kommandoer er klasser der implementerer `Command`-grænsefladen og indeholder al information der skal bruges under udførsel (herunder formelle parametre til metoder, mv.). Konkrete kommandoer som `ConcreteCommand` kalder programlogik i passende modtagerklasse(r) `Receiver`.
- **Konsekvenser:** Adskiller forespørgsler (“slet første linje af teksten”, “gør baggrunden rød”, etc.) fra den kode der implementerer dem. Gør det nemmere at f.eks. sammensætte, forsinke og fortryde forespørgsler. Kræver noget konfigurationskode.



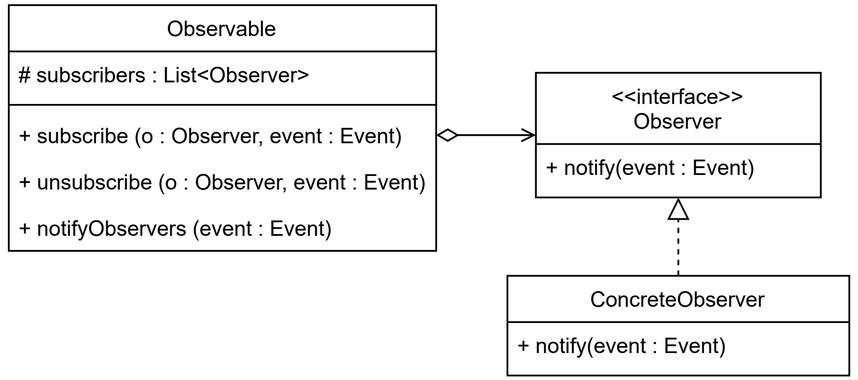
Bonusmønster: Model-View-Controller

- Ikke et af de originale designmønstre fra Gang of Four-bogen, men meget hyppigt brugt i forbindelse med grafiske brugergrænseflader.
- **Motivation:** At holde data, programlogik, og præsentation adskilt.
- **Deltagere:** `Model` holder data og tilbyder metoder til databehandling og -adgang; `View` trækker information fra `Model` og styrer visning; `Controller` tager inddata fra brugeren og styrer programlogik (herunder hvordan `Model` skal opdateres og hvad `View` skal vise).
- **Konsekvenser:** Holder programdele til databehandling, visning, og kontrol adskilt. Kræver noget setup og konfigurationskode, så er typisk ikke kompleksiteten værd til små programmer.



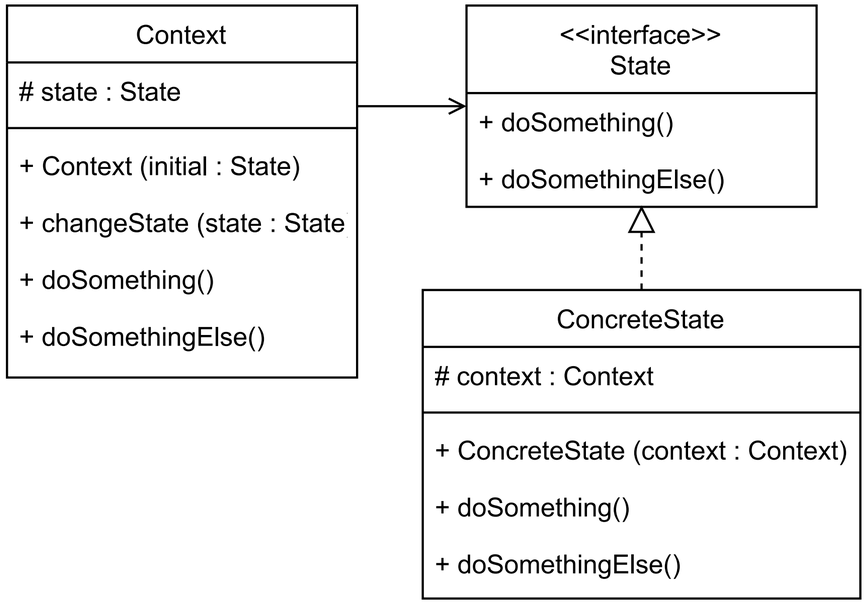
Observer

- **Motivation:** At lade programkomponenter reagere på handlinger andre steder i programmet.
- **Deltagere:** `Observable` vedligeholder en liste af observatører (der implementerer `Observer`) til en given handling. Når denne handling sker gives der besked til alle observatører ved kald til `notify()`.
- **Konsekvenser:** Kan gøre programmer mere responsive og mindre afhængige af hinanden ved at undgå periodevise checks.



State

- **Motivation:** At lade objekter skifte opførsel alt efter tilstand uden konstant at skulle checke tilstanden.
- **Deltagere:** `Context` holder en instans af `State`, som kan varieres ved kald til `changeState()`. Tilstandsafhængige metoder implementeres ved at kalde den tilsvarende metode på `State`-objektet.
- **Konsekvenser:** Tilstandsafhængige metoder kan implementeres uden eksplicit check af tilstand. Nye tilstande kan tilføjes med nye klasser der implementerer `State`.



Til VS Code!

- Lad os implementere eksempler på…
  - State
  - Observer