

C++

Forelesning 10, vår 2014
Alfred Bratterud

I dag:

- * Oblig2: Utsatt frist - stå på
- * Prøve og Prosjektoppgave
- * Design patterns
 - * Fullfører tankerekken fra forrige ukes forelesning
 - * Noen flere patterns

Avsluttende prøve!

- * Prøve fredag 13.april (sist dag)
- * Rom og klokkeslett kommer i neste uke
- * Pensum:
 - * Alt som er gått igjennom på forelesning - både slides, eksempelkode
 - * Bakgrunnsstoff fra læreboken
 - * Alt som er gitt i oppgaver frem tom. 13.april
- * Mål: Kontrollere at dere kan det viktigste ved språket.
 - * Er ikke meningen å gi vanskelige "nøtter", men forståelse er viktig.
- * Tips: Gjør oppgaver heller enn å lese. Hver gang du er usikker på noe - slå det opp, og prøv det ut isolert til du vet du skjønner det.

Prosjektoppgave

- * Oppgavebeskrivelse:
 - * Lag et fungerende og brukbart C++ program som du liker tanken på å vise frem.
- * Generelle krav:
 - * 1 - 3 deltakere pr. gruppe.
 - * Med flere enn 1 deltaker vil hver deltaker kunne vurderes etter det commit-loggen sier den har gjort.
 - * Altså må dere dele opp i klare ansvarsområder - lag gjerne noen deler selvstendig, andre sammen.
 - * Terminalbasert (Linux/Studssh), eller GUI - men da kun via FLTK, Qt eller ncurses
 - * Koden skal minimum funke på linux, men bør være portabel mellom mac, windows og linux (med mindre unntak gis)
 - * Koden skal bygges med "make" (evt. kjøres via Qt) og dokumentasjonen må tydelig angi alle avhengigheter (biblioteker, filer etc. som må finnes og hvor)

Prosjektoppgave

- * Vurderingskriterier:
 - * Helhetsinntrykk av sluttproduktet - **God Kode Virker!**
 - * Kodens tydelighet og lesbarhet
 - * Hensiktsmessig Programdesign
(Objektorientering, flyt, algoritmer, skalerbarhet, effektivitet etc., avhengig av prosjektet)
 - * Patterns og gjenbrukbar kode gir pluss
 - * Robusthet / feilfrihet (feilhåndtering, ressurshåndtering)
 - * Dokumentasjon - med diagram av designet.
- * Koeffisienter:
 - * Skop / Omfang
 - * Vanskelighetsgrad
- * Evt. pluss for Kreativitet eller wow-faktor (Kan få A uten, men det hjelper)

Prosjektoppgave

- * link til nytt, privat Github-repo for gruppen og meg med oppgaveforslag i, må leveres på fronter innen fredag 06.april kl.23:59. (senere innlevering spiser av tiden deres)
- * Prosjektbeskrivelsen må ha med:
 - * Beskrivelse av hovedfunksjonaliteten
 - * Tydelige designmål (Utvidbart? skalarbart? Effektivt? Integrerbart? Modulært?)
 - * Avgrensning: Noen minimumsmål, og noen hårete mål
- * Prosjektoppgaven må godkjennes av meg - gjøres i github
- * Innleveringsfrist (hard): 04.mai 2014 kl.23:59, da klones alle repoer.
- * Dere skal presentere prosjektet deres for meg, etter fristen.

Noen prosjektforslag

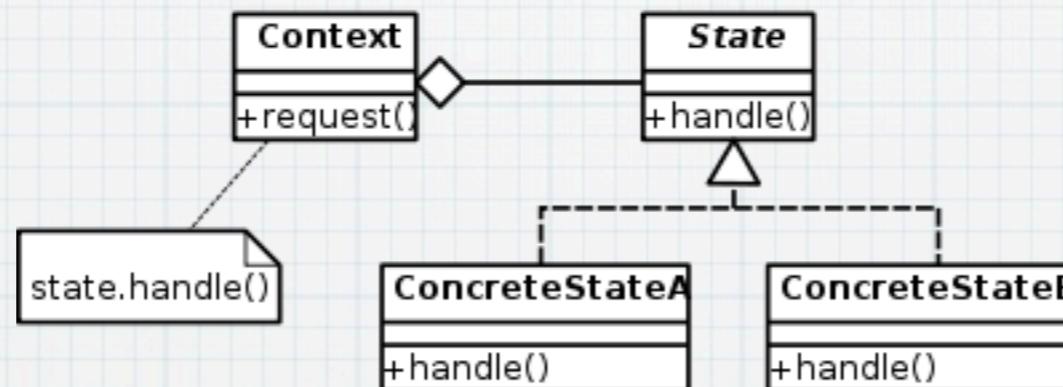
- * Et enkelt spill i FLTK/Qt
 - * PacMan? Tetris? Snake? Mario-klone? Rogue-like? Rollespill? Sjakk? Poker? AngryBirds? Asteroids?
 - * Utfordring: Koble til fysikkmotor, feks. Bullet
 - * Multiplayer over nett?
 - * 3D eller 2D i OpenGL?
 - * En animasjon (film) eller simulering (feks. populasjonsvekst med ressurser, formering, sykdommer etc.)
 - * En enkel synthesizer - ta lyd gjennom ulike filtre (frekvensendringer, fouriertransformasjoner etc.)
 - * Lage beats med samples?
 - * Webserver med logikk for dynamisk innhold og RESTful API
 - * Veldig nyttig - og C++ trengs for å spare verden for strøm
 - * En kryssplatform native-app? (Kun Qt som kan dette)
 - * Editor; markdown, C++, Python, Txt? Kryptert dagbok? Eller et tegneprogram?

Design patterns og idiomer

Design Patterns

- * Et pattern: En konsist beskrevet løsning på et gjentagende problem
- * En generell, gjenbrukbar løsning
- * 1994: Design Patterns for OO-programmering
- * Før det: "Patterns" i C
- * Enkleste eksempel: Singleton
 - * Garantér kun én instans, ved å gi klassen privat konstruktør.

Mer avansert: State



Mer avansert: State

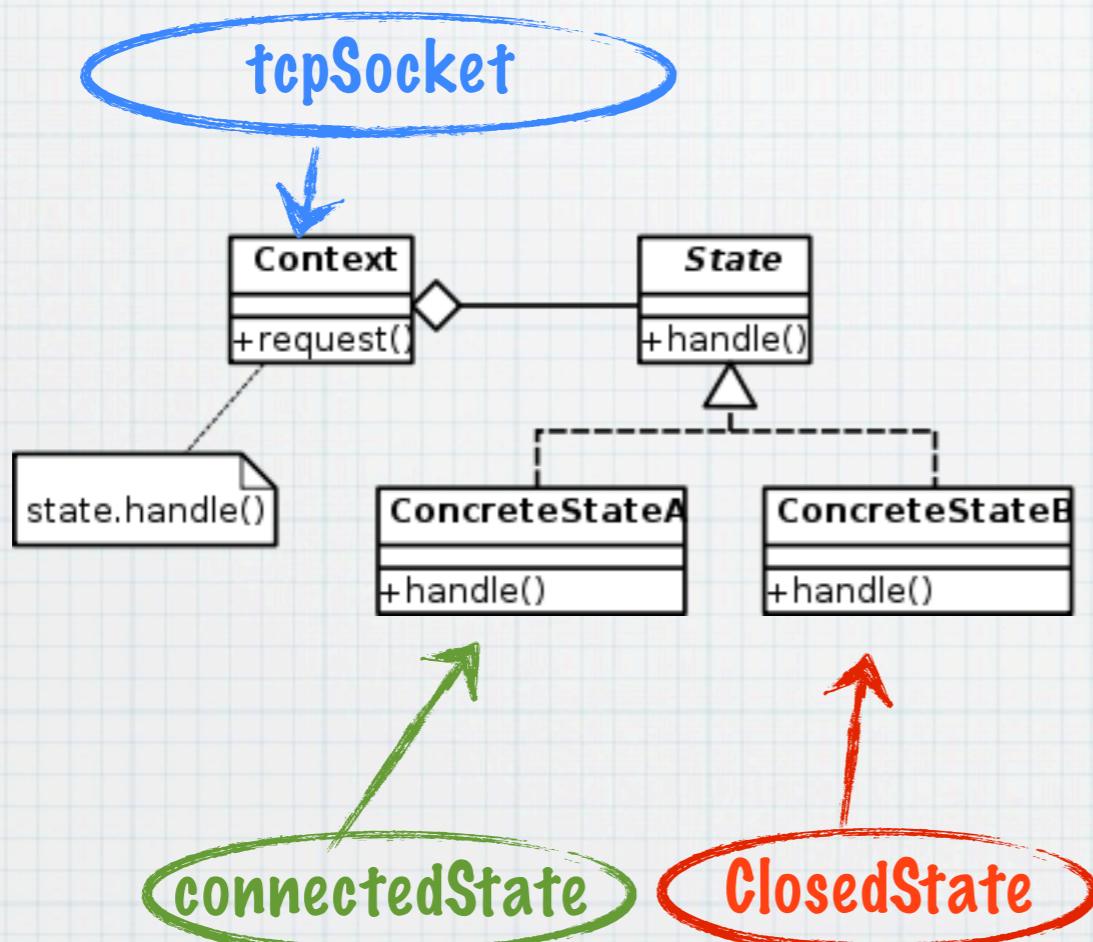
Erstatter:

void handle(){

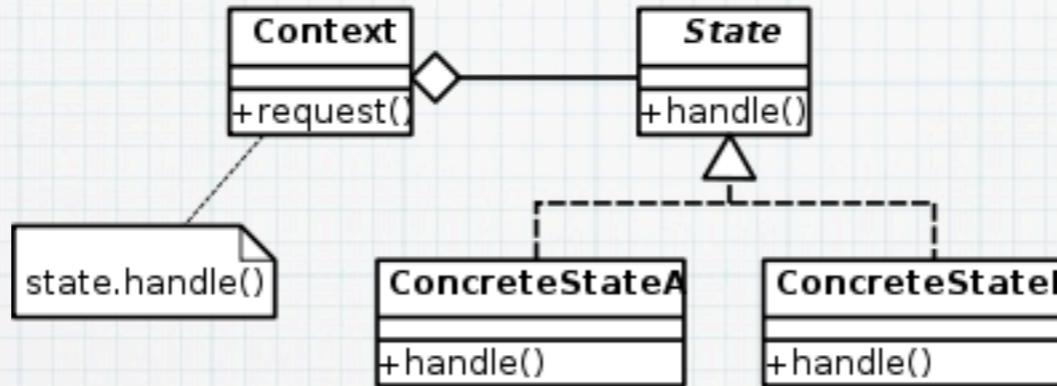
```
if (state==connected){  
    // do stuff ...  
}  
}
```

```
if(state==closed){  
    //Do other stuff...  
}  
}
```

...
}

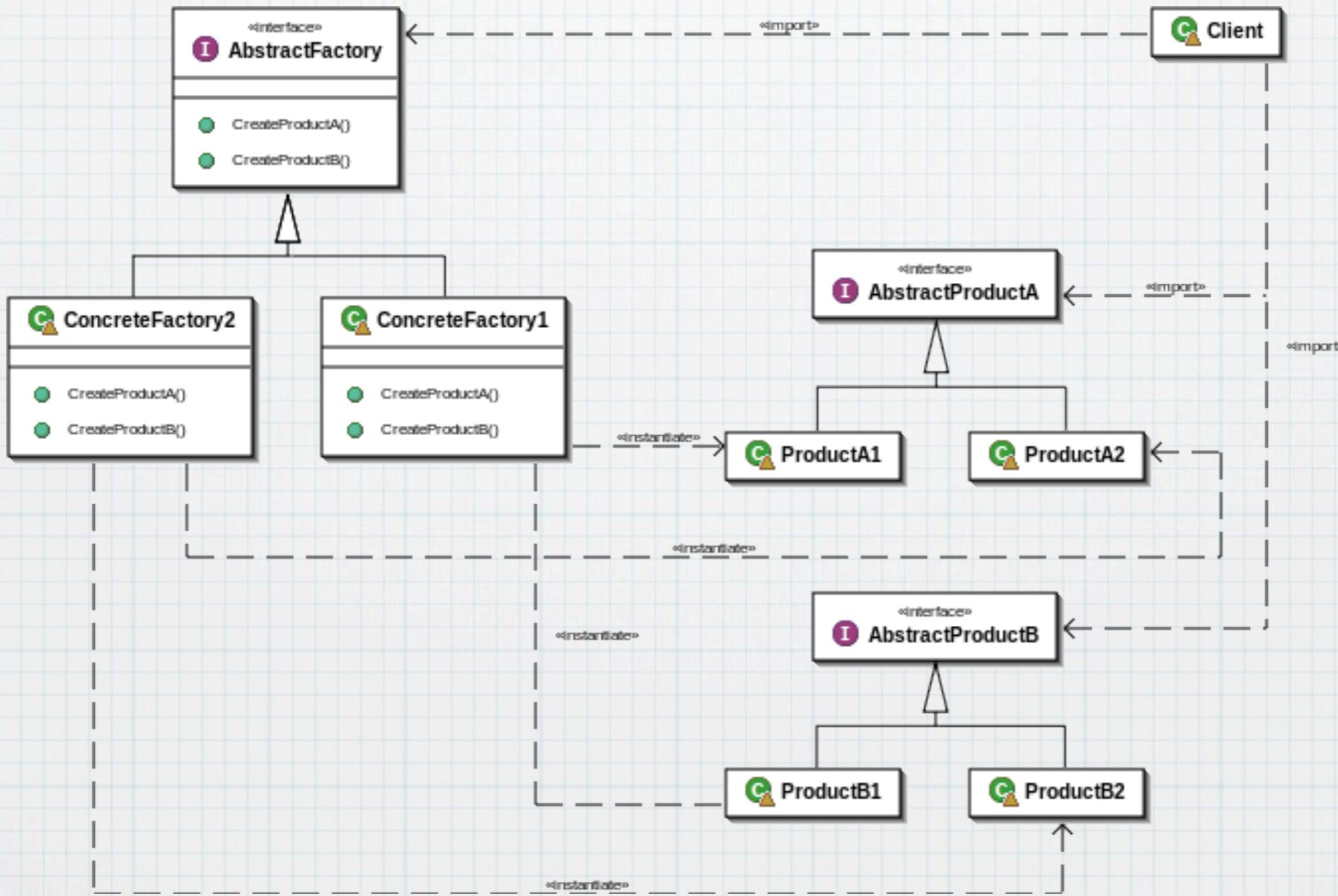


Mer avansert: State

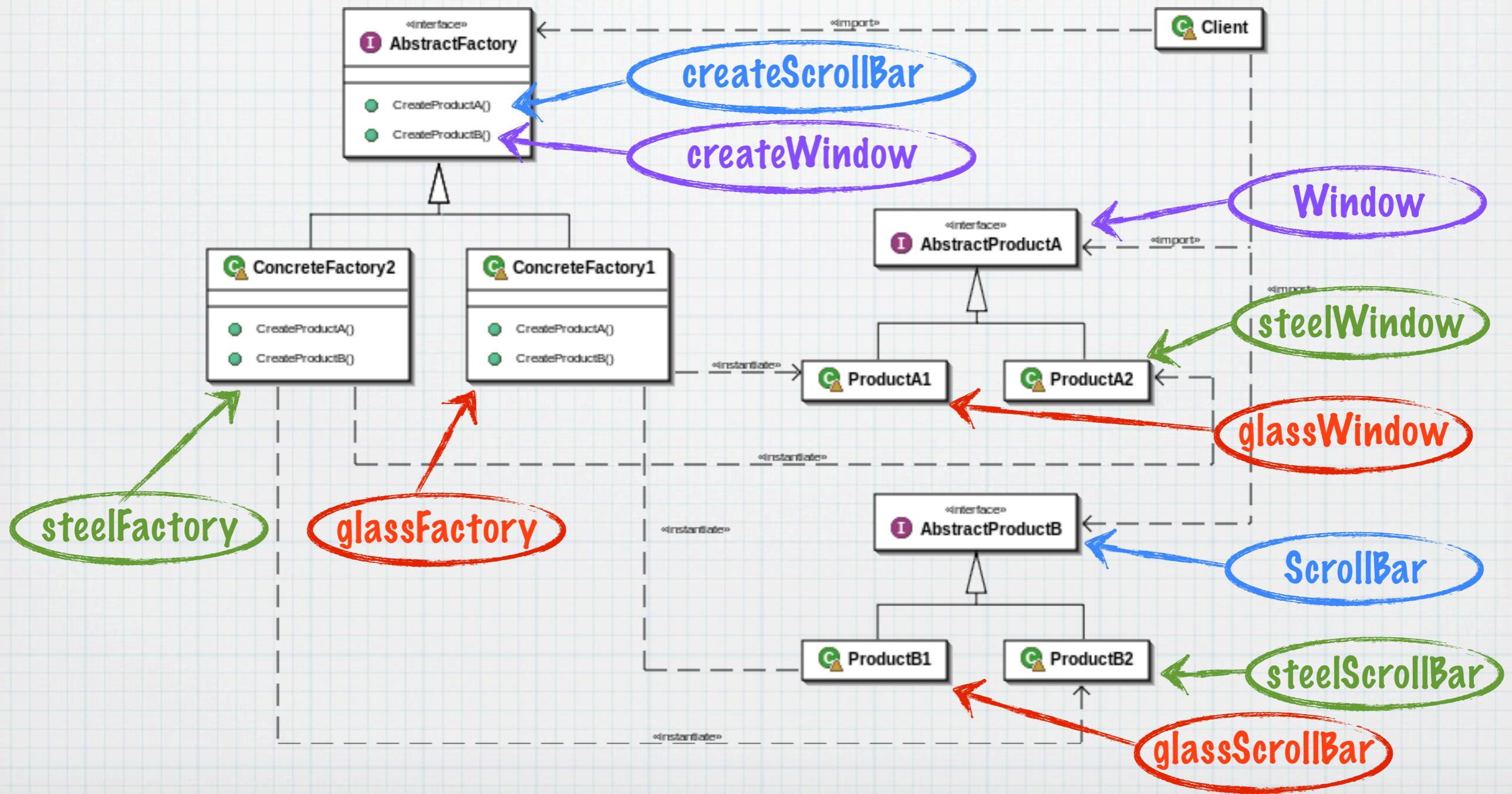


- * Bruk når:
 - En klasse "oppfører seg forskjellig" avhengig av tilstand
- * Erstatter lange, grisete kondisjonaler
- * Implementerer gjerne hver tilstand som singleton

Mye brukt: Abstract Factory



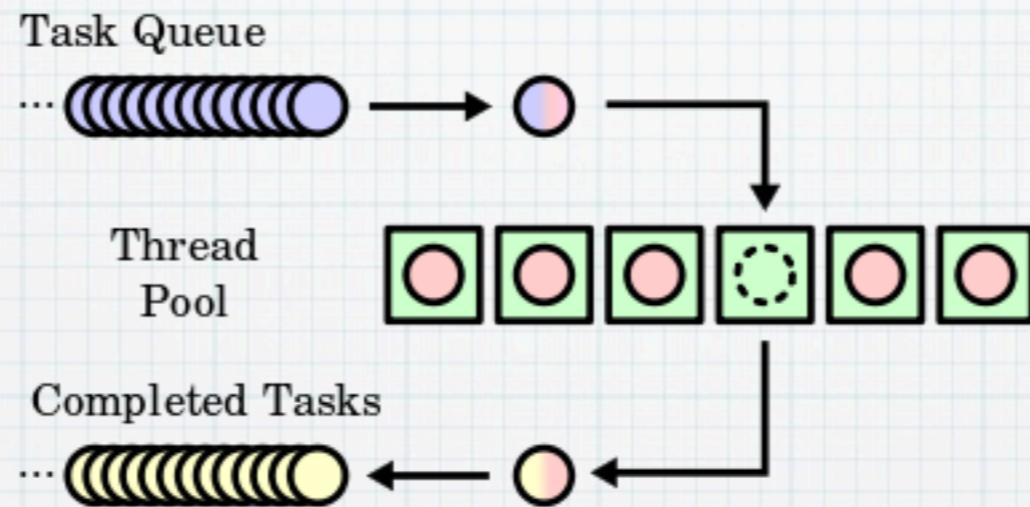
Mye brukt: Abstract Factory



Vår “animated_factory”

- * Vår factory har bare “ett produkt”; en “animated”.
- * I prinsippet en klasse med en “virtual constructor”: “create_animated”
- * Factories blir enda mye nyttigere med flere “produkter” som skal høre sammen.
Eksempel:
 - * Klassikeren: GUI med ulik “look and feel”
 - * Alle “looks’n feels” har knapper, vinduer, scrollbars etc.
 - * Men de har ulikt utseende og oppførsel basert på “look’n feel”
 - * En animasjon med ulike tropper; tyske, franske amerikanske.
 - * Alle nasjoner har “cavalery”, “infantry” og “artillery”.
 - * Men de har ulikt utseende og ulike egenskaper basert på nasjonalitet
 - * Da kunne vi hatt “animated_troops_factory”, med “virtuelle konstruktører” for hver troppetype

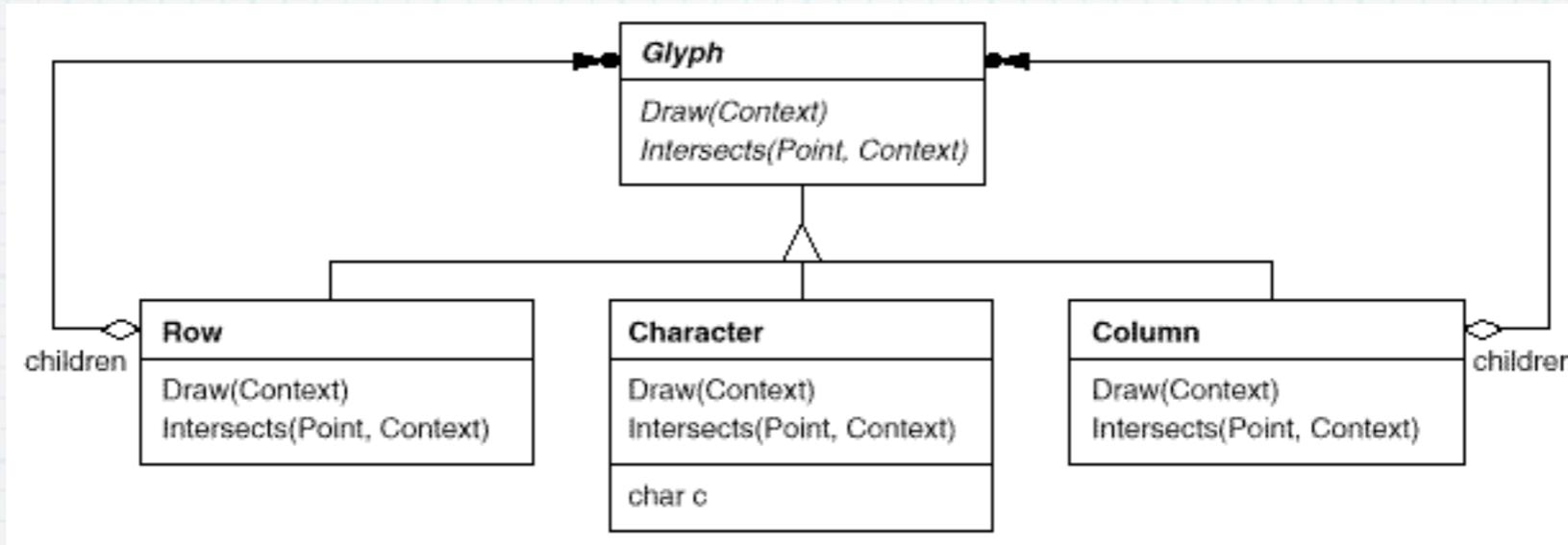
Patterns: ikke bare “convenience”



- * “Thread-pool” er ett eksempel på pattern som effektiviserer kode
- * Tråde er “dyre” å opprette, så vi tar vare på dem.

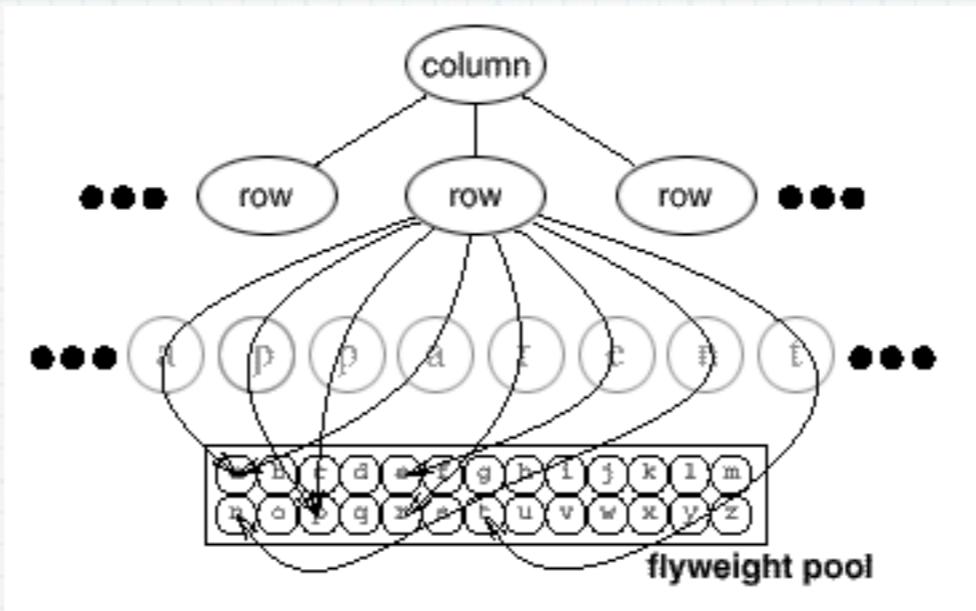
http://en.wikipedia.org/wiki/Thread_pool_pattern

Flyweight



- * “Use sharing to support large numbers of fine-grained objects efficiently” (“Design Patterns” s.195)
- * Kun egenskaper som gjelder “mange” objekter skal ligge inni klassen (intrinsic)
- * Alle “kontekstavhengige” egenskaper skal sendes med fra utsiden (extrinsic)
- * Eks: I stedet for ett objekt pr. tegn i et dokument, lag ett objekt pr. tegn punktum.
 - * Når du skal “tegne” det, send med koordinatene (Context)

Flyweight



- * Her har "row" pekere til de samme "bokstavene" mange ganger
- * Vi har kun 26 objekter, men vi kan ha millioner av bokstaver
 - * For alle får vi alle fordelene av objektorientering
- * Eks: Vi kan merke én gal bokstav i et ord, ved å kalle `letter(row,col)->mark_error()`
- * Brukes gjerne sammen med factory/multiton:
`letter_factory.get('a')` gir en "letter*", oppretter hvis den ikke finnes fra før.

Maaange patterns

AbstractFactory

Builder

Flyweight

Proxy

FactoryMethod

Prototype

Adapter

Bridge

Iterator

Facade

Command

Composite

Memento

Thread Pool

State

Decorator

Strategy

AbstractFactory

Chain of Responsibility

“Idiomer” vs. Patterns

- * Et “Design Pattern” refererer gjerne til en objektorientert løsning med flere klasser - og ofte polymorfi
- * Et “idiom” er noe litt “enklere” eller mer generelt; en vanlig skrivemåte (hentet fra idiomer i naturlig språk)
- * “Patterns” og “idiomer” brukes litt om hverandre
- * Det finnes da mange rent “prosedurale” patterns, feks. i C:
 - * Semaforer, mutexer, “barriers” etc.
 - * “monitorer” og “condition variables”
 - * Map / Reduce fra funksjonell programmering og HPC
- * Nesten alle blir bedre med objektorientering

Bjarne Stroustrup's RAII

- * Husker dere "Exception leak?"
 - * Anta at vi allokerer minne, eller en hvilken som helst ressurs; en fil, et socket, en mutex
 - * Før vi rekker å slippe den (release) kastes en exception. Oops! Lekkasje!
- * Løsning: "Resource Acquisition Is Initialization"
 - * "Å anskaffe en ressurs er å initialisere".
 - * ...ikke av seg selv, men vi kan bruke et "pattern", eller et "idiom")
 - * Nøkkel: I C++ destrueres alle variabler når de går ut av skop
 - * Løsning: For hver ressurs du skal anskaffe, lag en klasse som anskaffer den i konstruktøren.
 - * Og la destruktoren "avskaffe" / deallokere / lukke.

Demo!

(exception_leak.cpp)
exception_leak2.cpp
raii.cpp