

C++

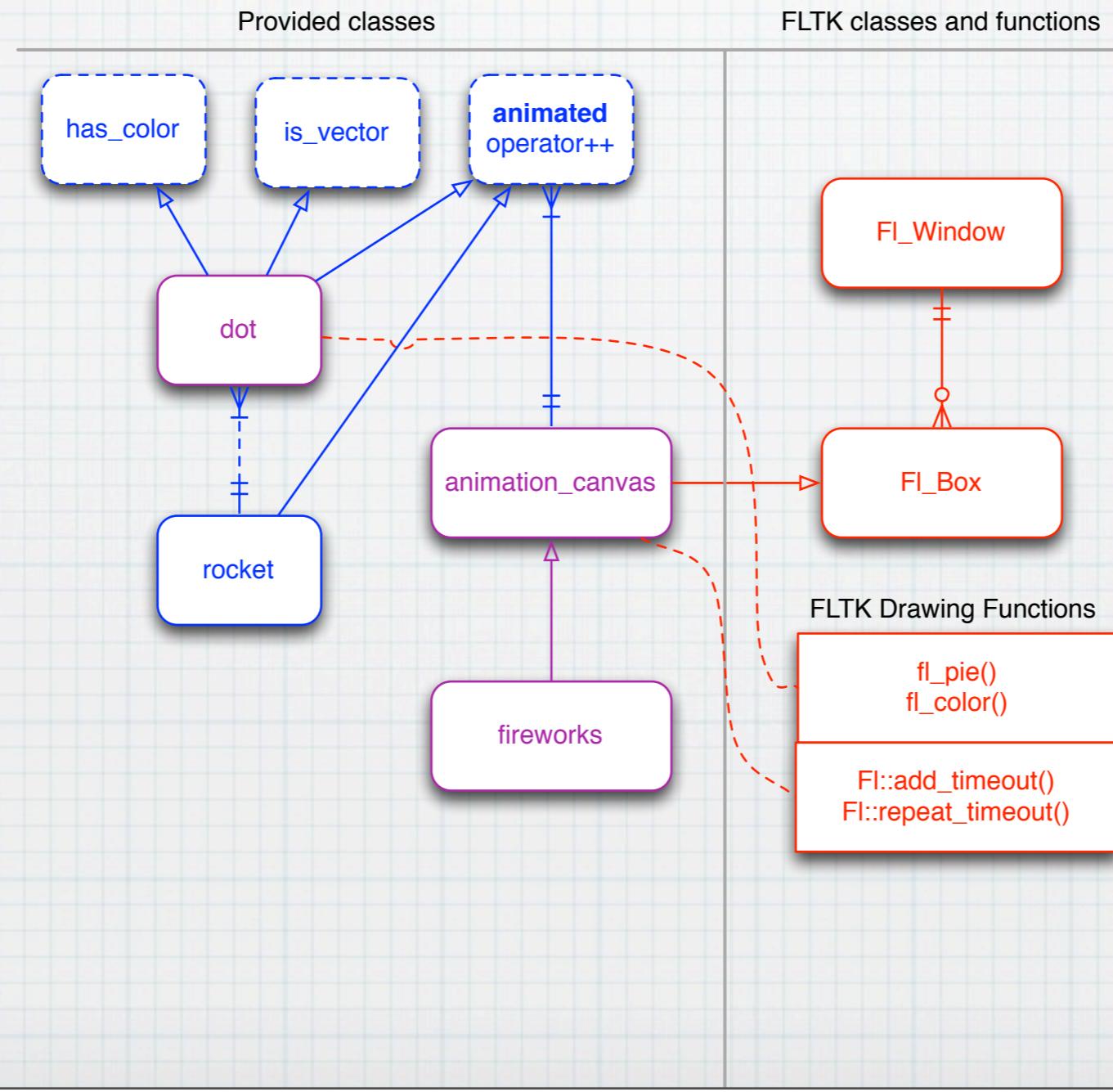
Forelesning 9, vår 2014
Alfred Bratterud

I dag:

- * Oblig 2 - design med rocket
- * Design patterns
 - * Objektorienterte designmønstre
 - * Et avansert - og kult - emne
 - * Relevant for oblig2 (spørsmålene)
 - * Gir mye uttelling i prosjektoppgave

Oblig2 - med rocket

Fireworks class hierarchy

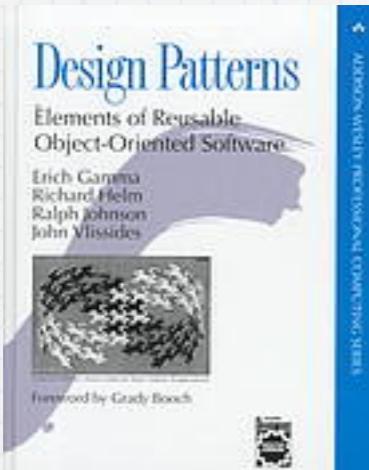


Design Patterns

- * Begrep fra arkitektur
- * 1977: Arkitekt Christopher Alexander
“A pattern Language: Towns, Buildings, Construction”
- * Et pattern: En konsist beskrevet løsning på et gjentagende problem
- * En generell, gjenbrukbar løsning
- * 1994: Design Patterns for OO-programmering

“Design Patterns”

Elements of Reusable Object-oriented Software



- * Klassiker, utgitt i 1994,
nå i minst 39'ende opplag (min er fra
mars 2011)
- * Skrevet av E. Gamma, R. Helm, R.
Johnson, J. Vlissides
“Gang of Four”
- * Beskriver 23 klassiske patterns
- * Følger retningslinjene fra Chris.
Alexander.
- * Støttelitteratur i dette kurset

Et enkelt eksempel “Singleton”

- * “Ensure a class only has one instance, and provide a global point of access to it”
- * En unik instans av et objekt - hvordan garanterer vi det?
- * Med privat konstruktør
- * ...Men oppnår vi ikke det samme med kun “static”-medlemmer i klassen?

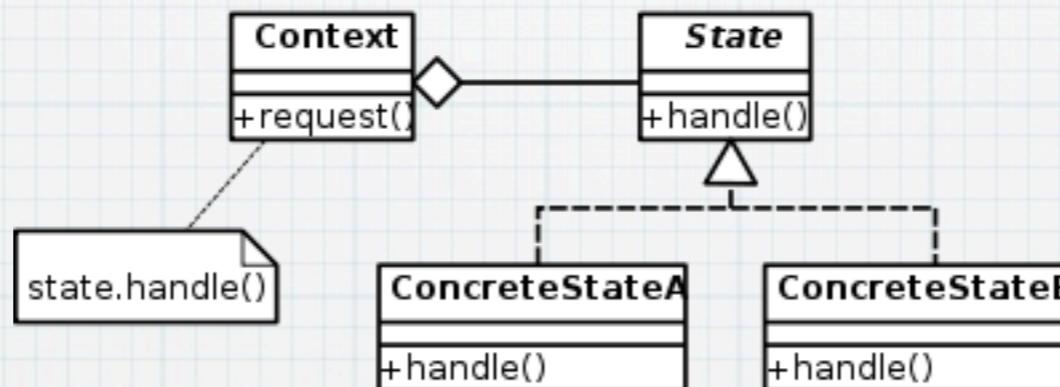
Demo!

singleton.cpp

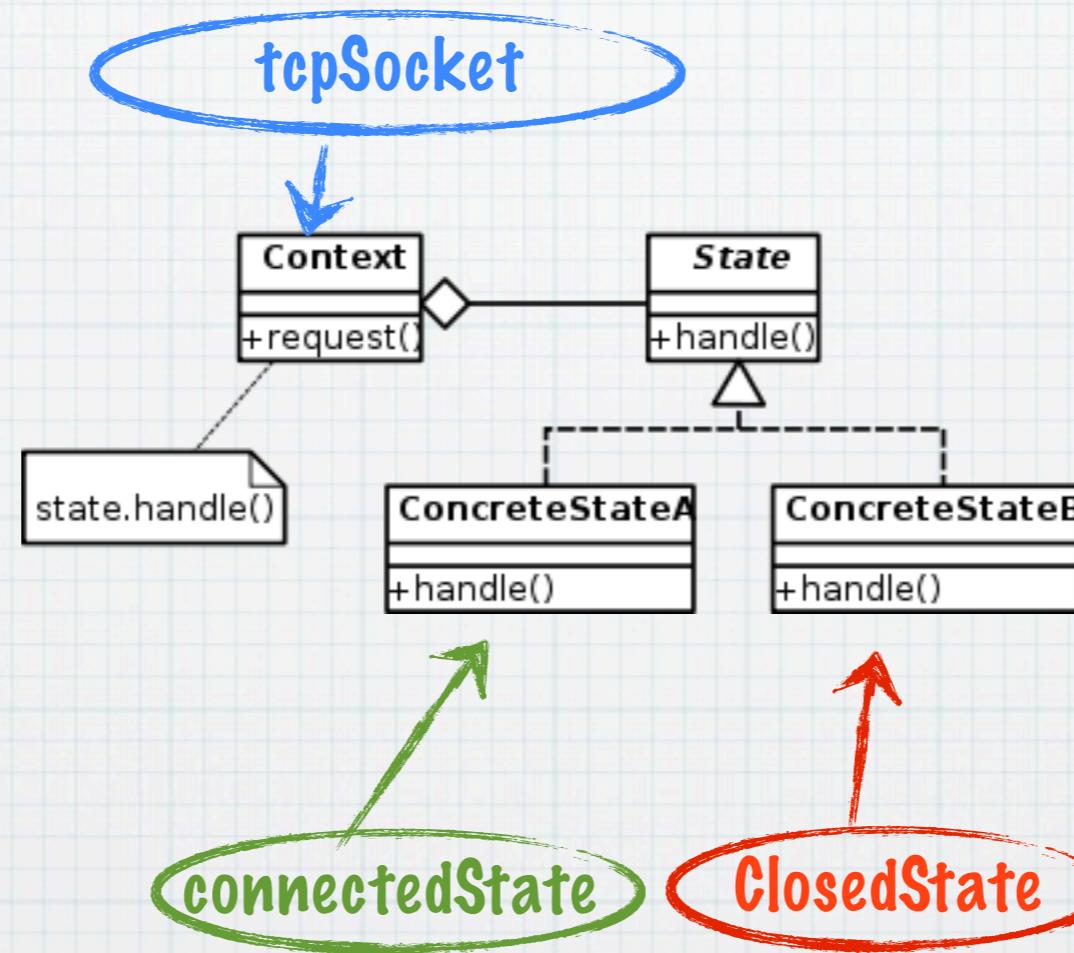
Fordeler med singleton kontra “vanlig” static?

- * Singleton har “lazy construction” - den opprettes først når den trengs.
 - * Den kan da også være avhengig av ting som skjer “runtime” - i motsetning til det som er static.
 - * Singleton kan (delvis) arves - feks. logfileXML, logfileCSV...
 - * Samme struktur kan garantere 2, 3, 4, eller n -antall instanser.
 - * Da heter det “multiton”
 - * Anvendelser?
 - * usbPort(n), screen(n), serverConnection(n)
 - * Hvorfor ikke bare ha dem i et array?
 - * Fordi da kan hvem som helst instansiere nye

Mer avansert: State



Mer avansert: State



Mer avansert: State

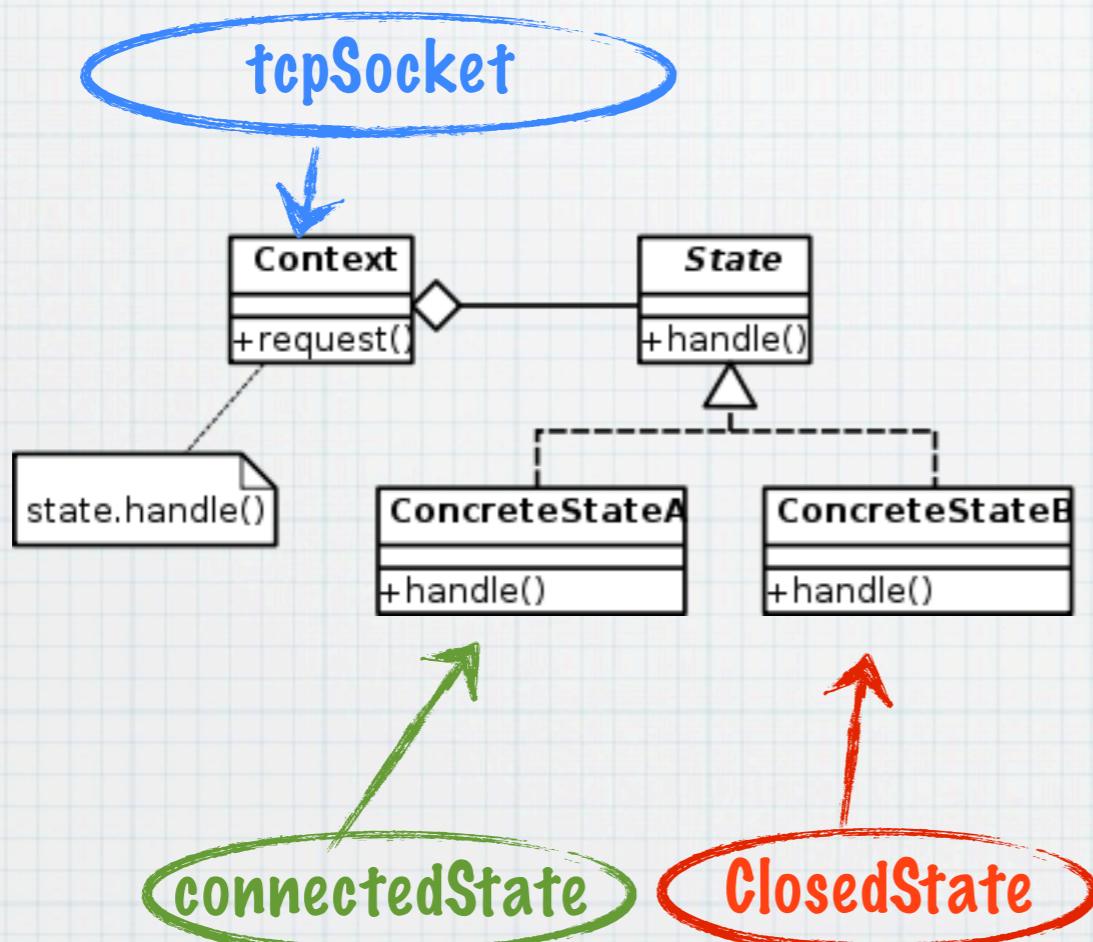
Erstatter:

void handle(){

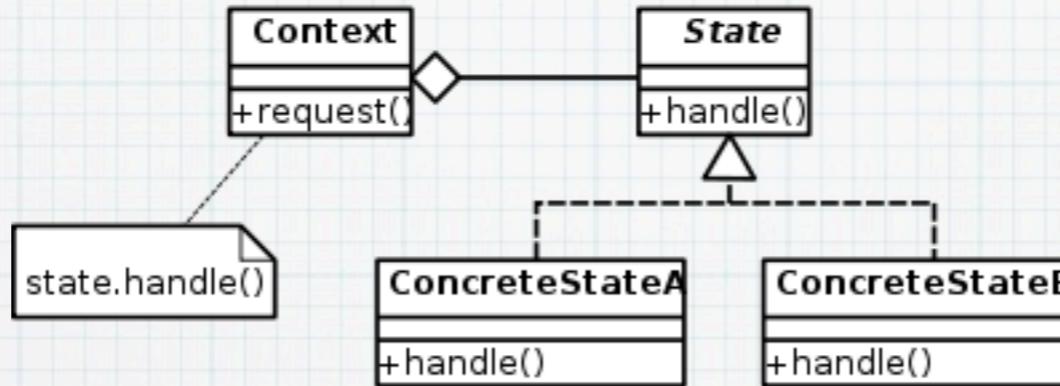
```
if (state==connected){  
    // do stuff ...  
}  
}
```

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

...
}

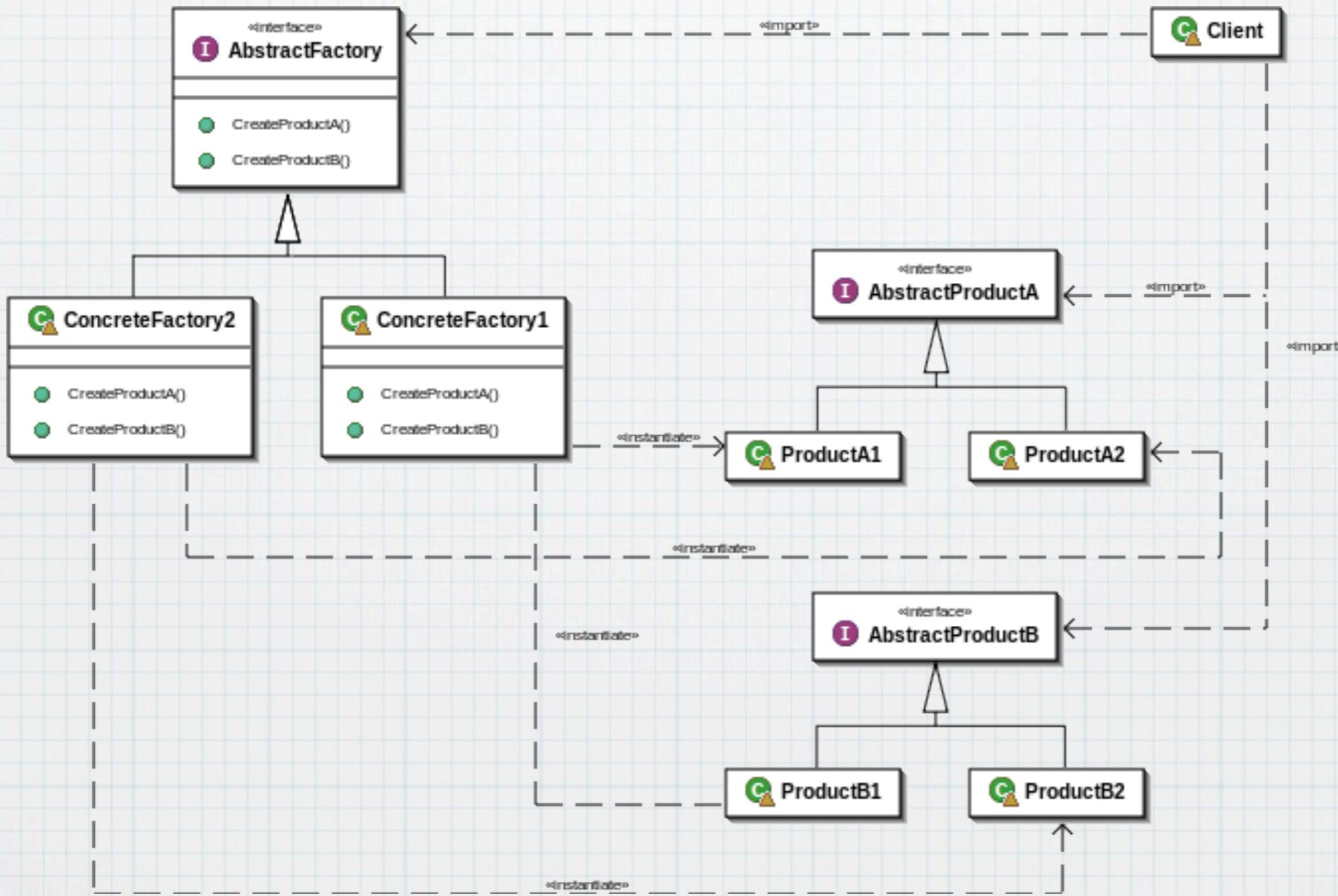


Mer avansert: State

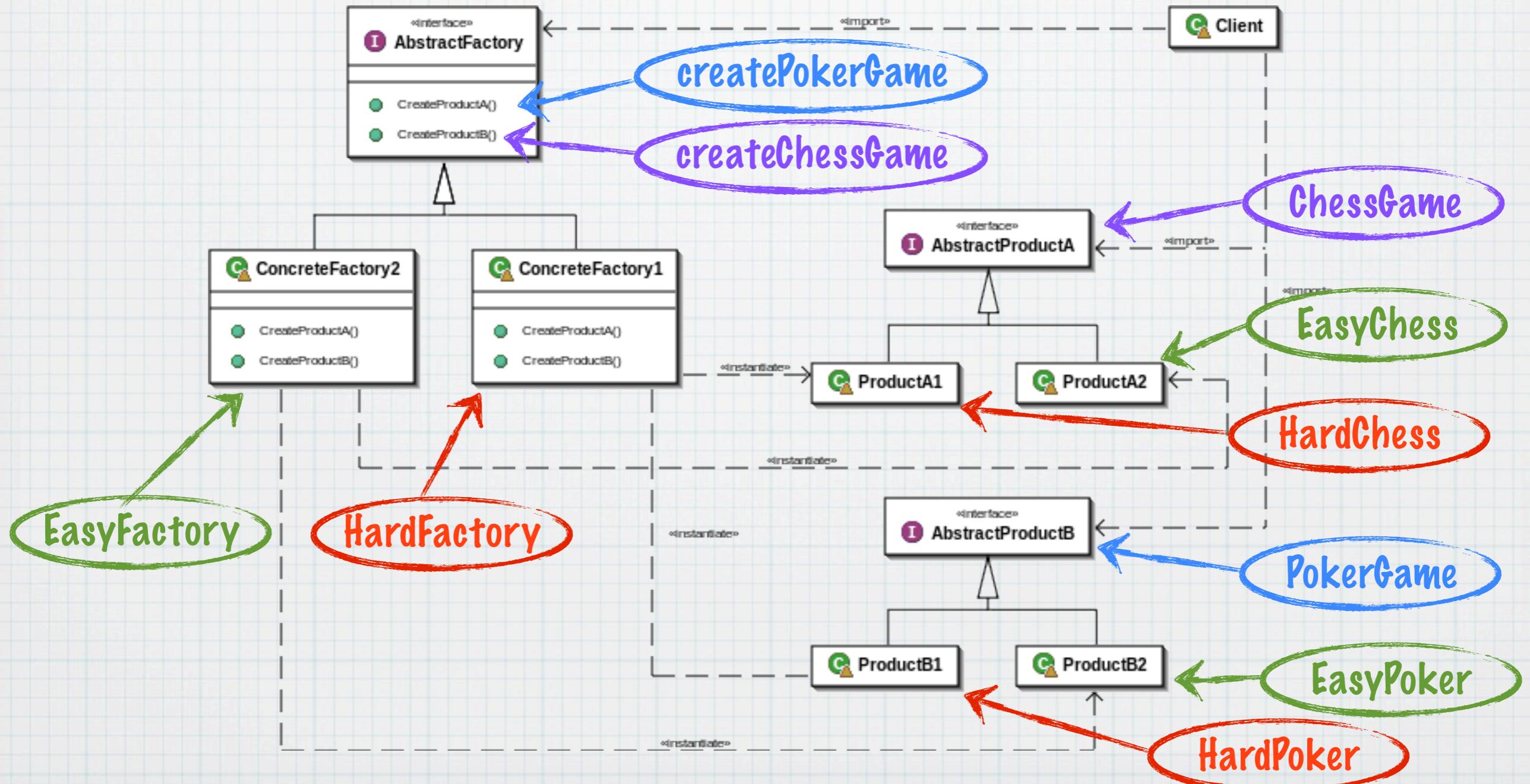


- * Bruk når:
 - En klasse “oppfører seg forskjellig” avhengig av tilstand
- * Erstatter lange, grisete kondisjonaler
- * Implementerer gjerne hver tilstand som singleton

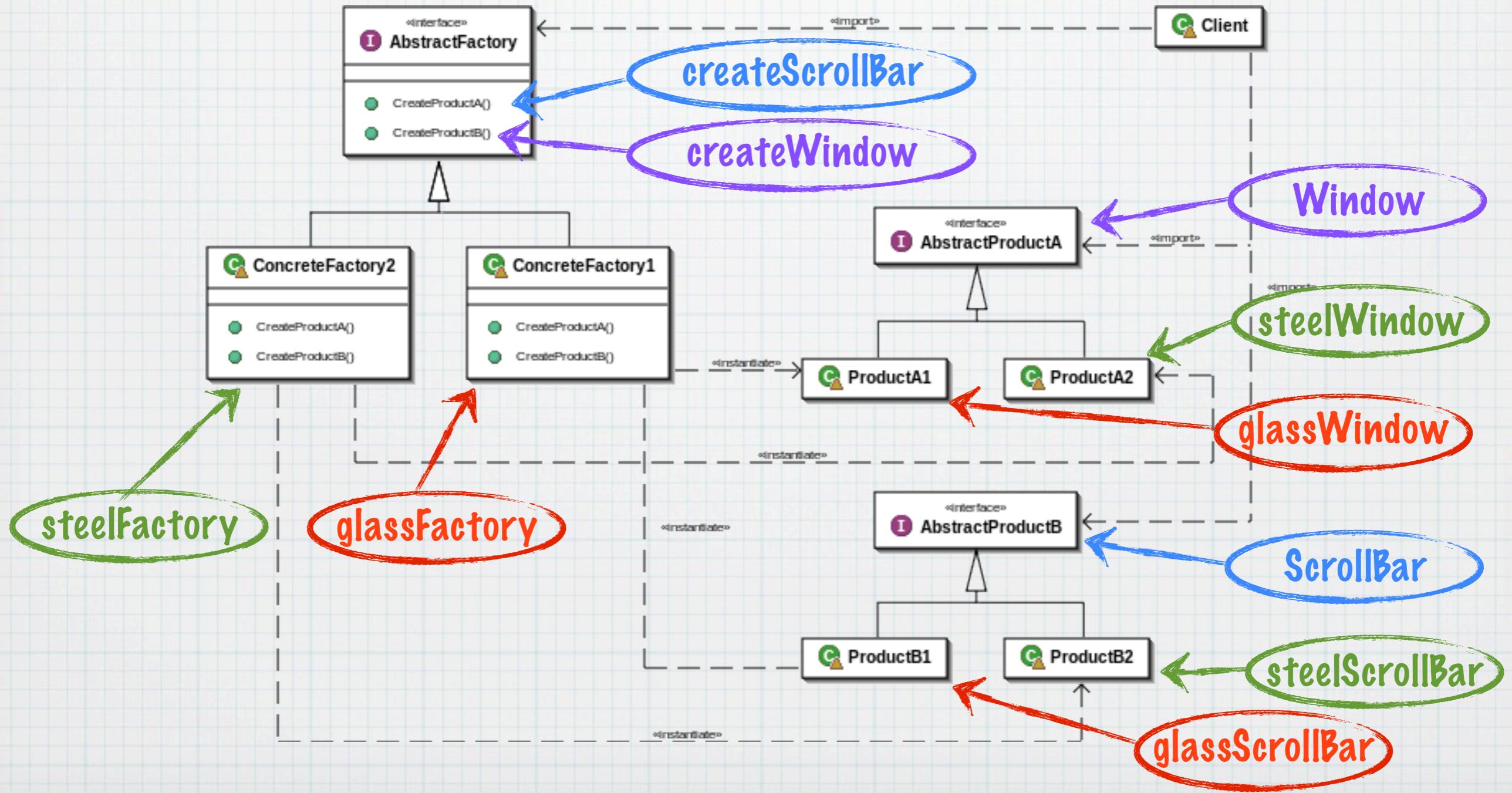
Mye brukt: Abstract Factory



Mye brukt: Abstract Factory



Mye brukt: Abstract Factory



Abstract Factory

- * En "Abstract factory" bruker en "concrete factory" til å lage instanser.
- * Instansene er polymorfe subklasser av "abstract products".
- * Effekt: ved å skifte "concrete factory" skifter man subtype på et helt sett av objekter.
- * Concrete factories bruker "Factory methods", eget pattern, også kalt "virtual constructors".
- * Dermed kan vi lage factories som kun override visse av factory-metodene, mens resten blir i base-class.

Abstract Factory

- * En "Abstract factory" bruker en "concrete factory" til å lage instanser.
- * Instansene er polymorfe subklasser av "abstract products".
Anvendelse for "Fireworks"?
- * Vurdér det i oblig 2, tekstspersmål
subtype på et helt sett av objekter.
- * Concrete factories bruker "Factory methods", eget pattern, også kalt "virtual constructors".
- * Dermed kan vi lage factories som kun overrider visse av factory-metodene, mens resten blir i base-class.

Demo!

Dot factories
(custom.cpp - legges ut først etter oblig2)

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

Maaange patterns

AbstractFactory

Builder

Flyweight

Proxy

FactoryMethod

Prototype

Adapter

Bridge

Iterator

Facade

Command

Composite

Memento

Thread Pool

State

Decorator

Chain of Responsibility

Strategy

AbstractFactory

Maaange patterns

AbstractFactory

Builder

Flyweight

Adapter

Facade

Memento

Proxy

FactoryMethod

Prototype

Mer i neste uke!

Command

State

Composite

Thread Pool

Strategy

Decorator

Chain of Responsibility

AbstractFactory