

C++

Forelesning 9, vår 2015
Alfred Bratterud

Effektiv kode med C og C++

Forleøpig Semesterplan Vår 2015

Uke	Dato	Tema	Mappe / Obliger
1			
2	06.jan	Basics	Roulette / Ukesoppg.
3	13.jan	Minne	Roulette / Ukesoppg.
4	20.jan	Minne	Ukesoppgaver
5	27.jan	Objektorientering	Oblig1 Ut
6	03.feb	Objektorientering	
7	10.feb	Objektorientering	Oblig1 Inn
8	17.feb	UNDERVISNINGSFRI	UNDERVISNINGSFRI
9	24.feb	Objektorientering	Ukesoppgaver
10	03.mar	Avlyst (sykdom)	Ukesoppgaver
11	10.mar	Templates - VIDEO	Ukesoppgaver Templates
12	17.mar	Oblig2 + Design Patterns	Oblig 2 Ut
13	24.mar	Design Patterns +++	
14	31.mar	Påske (mulig videoforelesning)	Oblig2 Inn
15	07.apr	Qt	Prøve + prosjektbeskr.
16	14.apr	Undervisning slutt	
17	21.apr		
18	28.apr		Prosjektoppgave Inn
19	05.mai		Mulig Presentasjoner
20	12.mai		
21	19.mai		
22	26.mai	Bachelorppgave 3.kl. 26 mai	
23	02.jun		Mulig Presentasjoner
24	09.jun	Bachelorpresentasjoner ca. her	

I dag:

- * Oblig 2: grafikk med FLTK
- * Oblig 2: gjennomgang av designet
- * Design patterns
 - * Objektorienterte designmønstre: et avansert - og morsomt - emne
 - * Relevant for oblig2 (spørsmålene)
 - * Gir mye uttelling i prosjektoppgave



Macintosh HD

L22046:oblig2 alfrebs\$./fireworks 2>> /dev/null

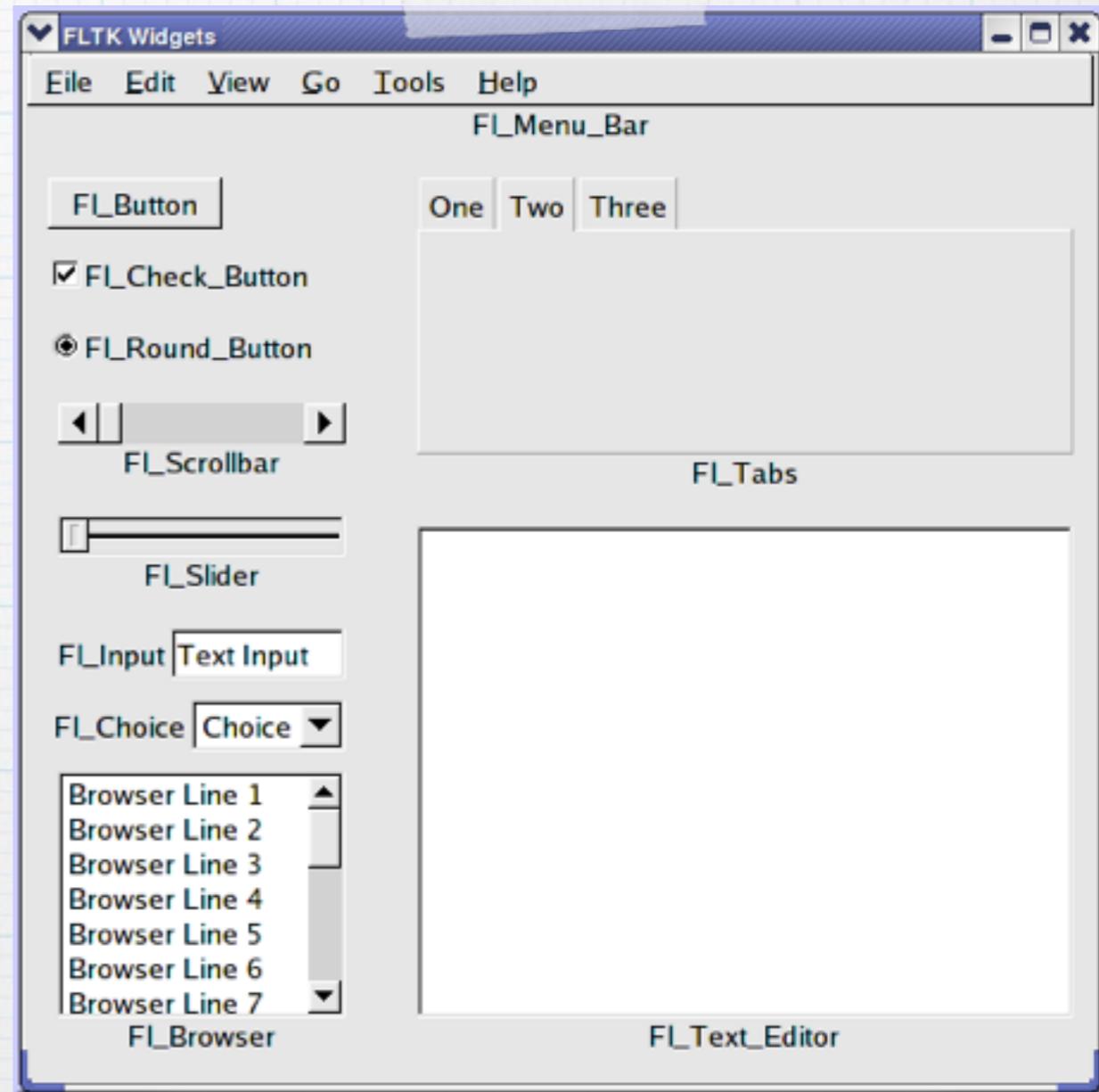
A terminal window titled "oblig2 — bash — 83x32" is displayed against a background of a starry night sky. The window contains the command "L22046:oblig2 alfrebs\$./fireworks 2>> /dev/null". A cursor arrow is visible at the bottom left of the terminal window. The window has standard OS X-style controls (minimize, maximize, close) in the top right corner.

FLTK

Installasjon og oppsett

FLTK

- * «Fast Light Toolkit» (Uttales visttnok «fulltick»)
- * Enkel, portabel grafikk for C++. Det biblioteket som brukes i boken
- * Finnes mange «mer fancy», men FLTK er helt «standards compliant» og funker på flere platfromer.
- * Alle widgets er «helt hjemmelagede» - så ingen «native look'n feel»
- * Beste alternativ: Qt
 - * Men, bruker en del non-standard innpakning.



FLTK Oppsett

- * FLTK kompileres til et «forhåndskompilert» bibliotek, som vi «linker mot»
 - * Dynamisk linking betyr at koden fra biblioteket ikke blir en del av din binærfil, men «kobles sammen» med biblioteket når koden kjøres
 - * Fordel: Vi slipper å ha kopier av samme kode i mange programmer
 - * Dette er akkurat som «printf» og annen kode fra standardbiblioteket
 - * Statisk linking betyr at koden fra biblioteket ***blir*** en del av vår binærfil.
 - * Fordel: Vi trenger ikke bekymre oss for hvorvidt biblioteker er riktig installert
 - * FLTK støtter begge, «default» hos meg ble dynamisk linking.
 - * Men: Til forskjell fra standardbiblioteket må man oppgi eksplisitt at man vil linke med FLTK, i kompilering- / linkingskommandoen.
- * FLTK kan fint kompileres fra kildekode, og installeres med «Make install»
- * For Ubuntu finnes imidlertid ferdig-kompilert pakke, men det er FLTK 1.3.1 (siste er FLTK 1.3.3)
- * OBS: All installasjon krever administratorrettigheter - så dette går ikke på studssh.

FLTK

- * Installasjon på Ubuntu (der du må jobbe)

- * \$ sudo apt-get install libfltk1.3-dev

- * Linking mot FLTK. Med FLTK 1.3.1 (pakken):

- * g++ -std=c++11 myObject.o ... -lfltk

- * -lfltk - betyr «libfltk». Vi må kanskje oppgi path til biblioteket eksplisitt. Gjøres med «-Lmy/Lib/Path»

- * OBS: Rekkefølgen teller! Les kommandoen som «myObject.o - som avhenger av ... som avhenger av libfltk»

- * Med FLTK 1.3.3 fra kildekode trengte jeg:

- * -L/usr/local/lib -lfltk -lXfixes -lXext -lpthread -ldl -lm -lX11

Bygge FLTK fra kildekode

- * Last ned kildekoden:

```
$ wget http://fltk.org/pub/fltk/1.3.3/fltk-1.3.3-source.tar.gz
```

- * Pakk ut:

```
$ tar -xvf fltk_zipfile.tar.gz
```

- * I mappen som ble laget:

```
$ ./configure && make  
$ sudo make install
```

- * «Spør FLTK» om feks. linking-argumenter, paths etc.:

```
$ fltk-config
```

Oblig2 - idéen

- * I stedet for en «naiv algoritme» skal vi tenke helt objektorientert:
 - * Et **fyrverkeri** - slik vi ***ser det*** består av...
 - * flere **raketter**, som består av ...
 - * mange **dotter...**
 - * som er **animert**
 - * **det vil si...?**
 - * som er en (matematisk) **vektor**
 - * **det vil si...?**
 - * som **har en farge**
 - * **det vil si...?**

Oblig2 - idéen

* I stedet for en «naiv algoritme» skal vi tenke helt objektorientert:

* Et **fyrverkeri** - slik vi *ser det* består av...

* flere **raketter**, som består av ...

* mange **dotter...**

* som er **animert**

* det vil si...?

operator++ hvert tidsintervall

* som er en (matematisk) **vektor**

* det vil si...?

* som har en **farge**

* det vil si...?

Oblig2 - idéen

* I stedet for en «naiv algoritme» skal vi tenke helt objektorientert:

* Et **fyrverkeri** - slik vi *ser det* består av...

* flere **raketter**, som består av ...

* mange **dotter...**

* som er **animert**

* det vil si...?

operator++ hvert tidsintervall

* som er en (matematisk) **vektor**

* det vil si...?

Fart og retning

* som har en **farge**

* det vil si...?

Oblig2 - idéen

* I stedet for en «naiv algoritme» skal vi tenke helt objektorientert:

* Et **fyrverkeri** - slik vi *ser det* består av...

* flere **raketter**, som består av ...

* mange **dotter...**

* som er **animert**

* det vil si...?

operator++ hvert tidsintervall

* som er en (matematisk) **vektor**

* det vil si...?

Fart og retning

* som har en **farge**

* det vil si...?

en int, med «R», «G» og «B»

Oblig2 - idéen

- * I stedet for en «naiv algoritme» skal vi tenke helt objektorientert:
 - * Et **fyrverkeri** - slik vi ***ser det*** består av...
 - * flere **raketter**, som består av ...
 - * mange **dotter...**
 - * som er **animert**

Kan «kjøre» fyrverkeriet ved å si...?
kalle «++» på alle **rakettene** -
som igjen kaller det på alle
dottene.

Trenger en **timer** for å styre
tiden.

operator++ hvert tidsintervall

vektor

Fart og retning

en int, med «R», «G» og «B»

Oblig2 - idéen

- * I stedet for en «naiv algoritme» skal vi tenke helt objektorientert:
 - * Et **fyrverkeri** - slik vi ***ser det*** består av...
 - * flere **raketter**, som består av ...
 - * mange **dotter...**
 - * som er **animert**
 - * det vil si...?
 - * som er en (matematisk) **vektor**
 - * det vil si...?
 - * som **har en farge**
 - * det vil si...?

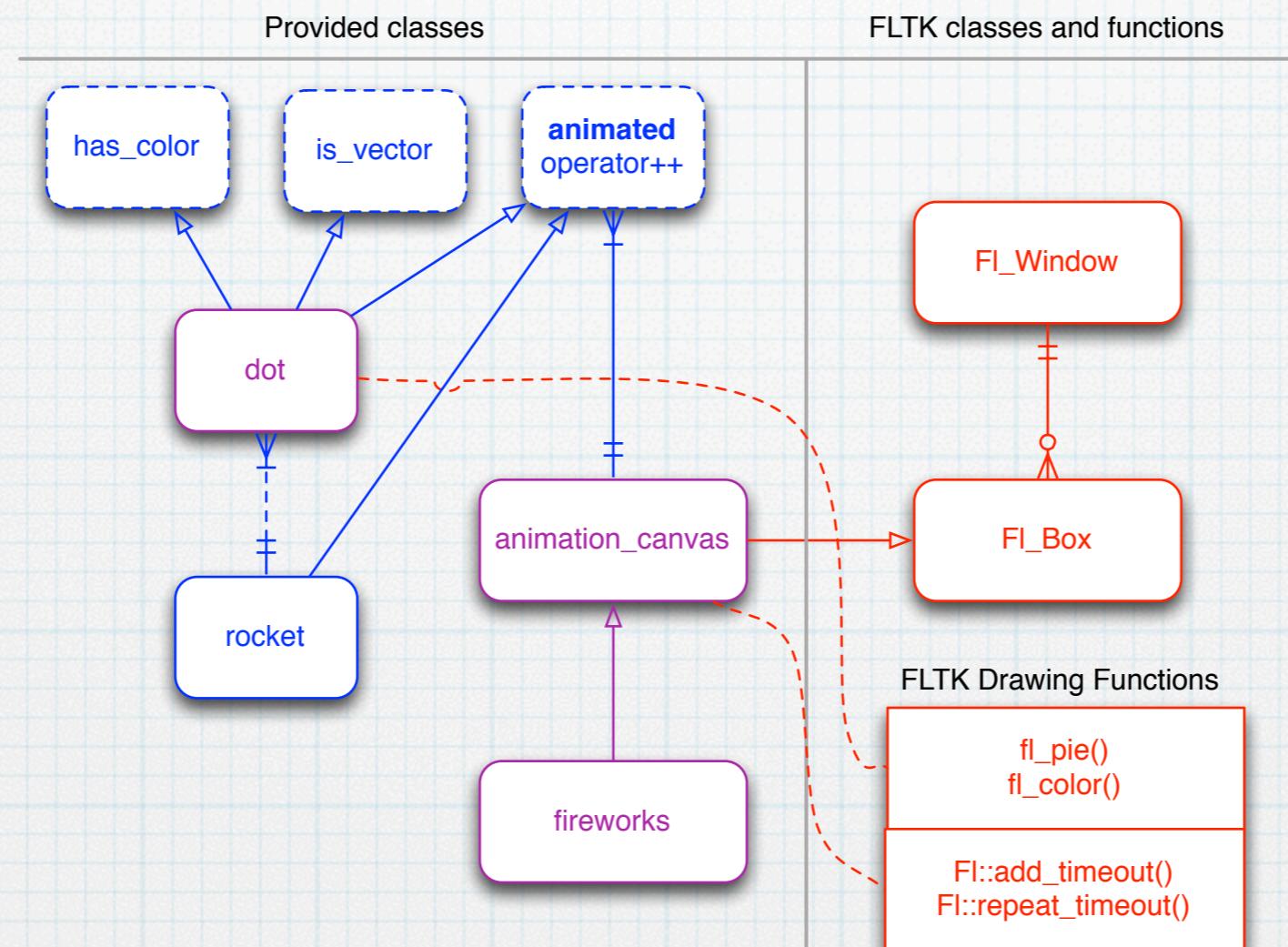
Fart og retning

Og en posisjon, x,y

Ny posisjon:
 $x += \sin(\text{retning})$;
 $y += \cos(\text{retning})$;

Oblig2 - design

Fireworks class hierarchy



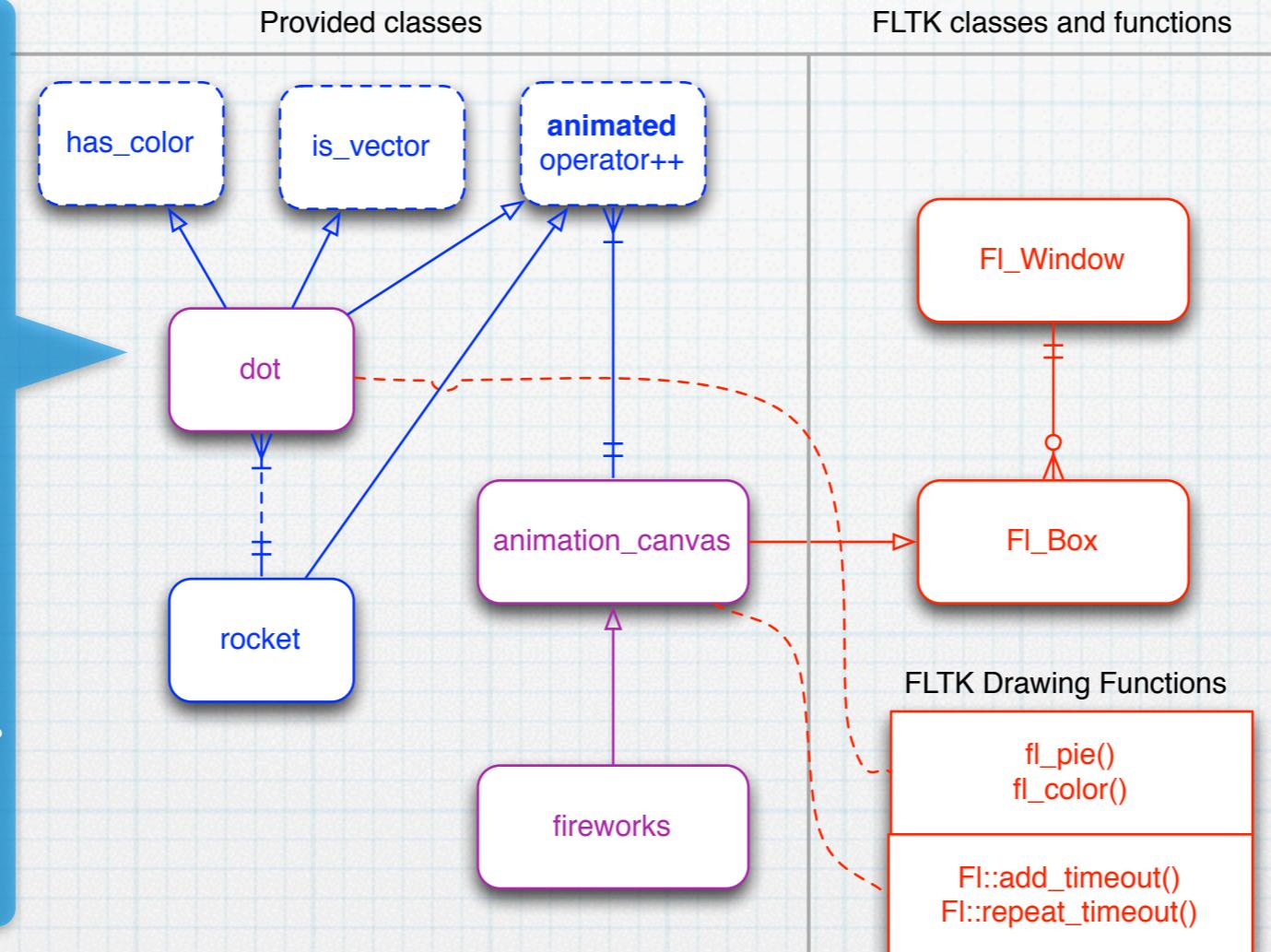
Oblig2 - design

Fireworks class hierarchy

I tillegg:

«Abstract Dot-factory»
for å lage ulike typer «dotter».

Skifte dot-factory: helt nytt fyrverkeri.

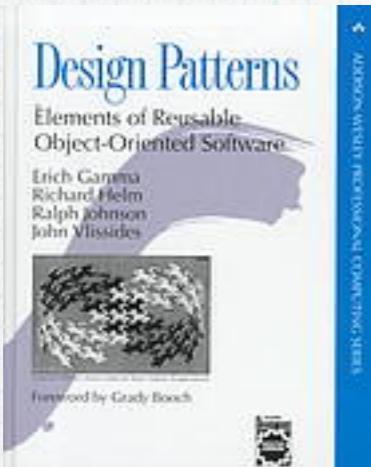


Design Patterns

- * 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?

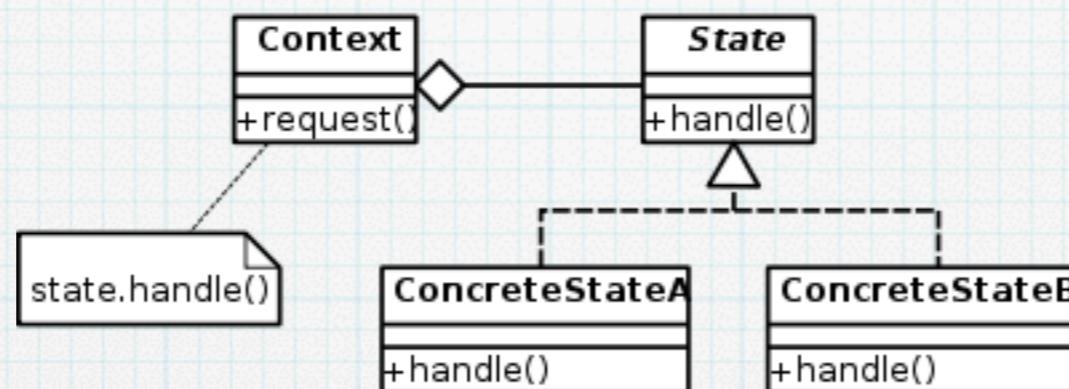
Demo!

singleton.cpp

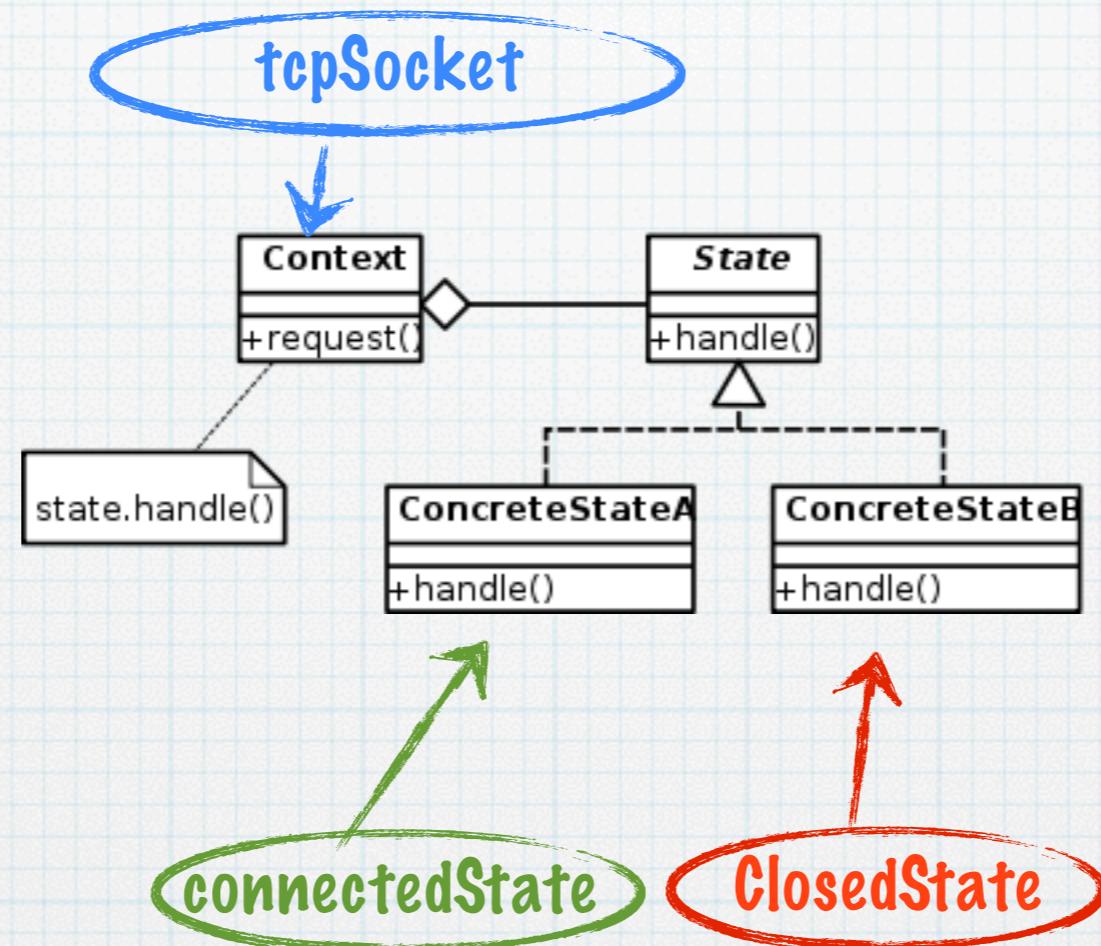
Fordeler med singleton kontra "vanlig" static?

- * 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

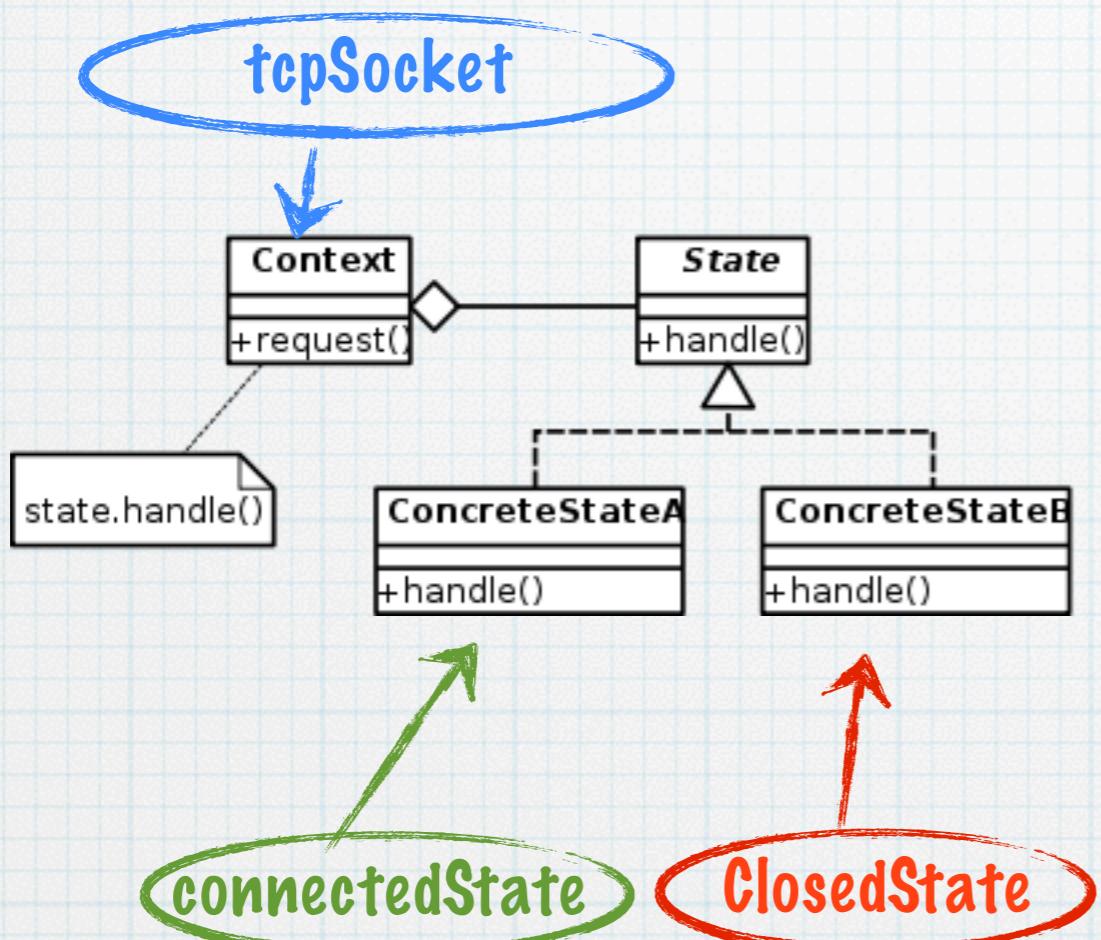
Erstatter:

`void handle()`

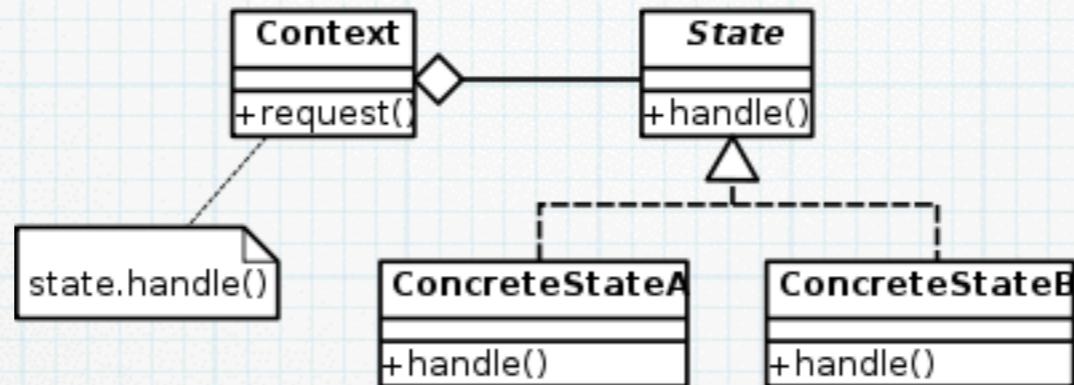
`if (state==connected){
 // do stuff ...
}`

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

`...
}`



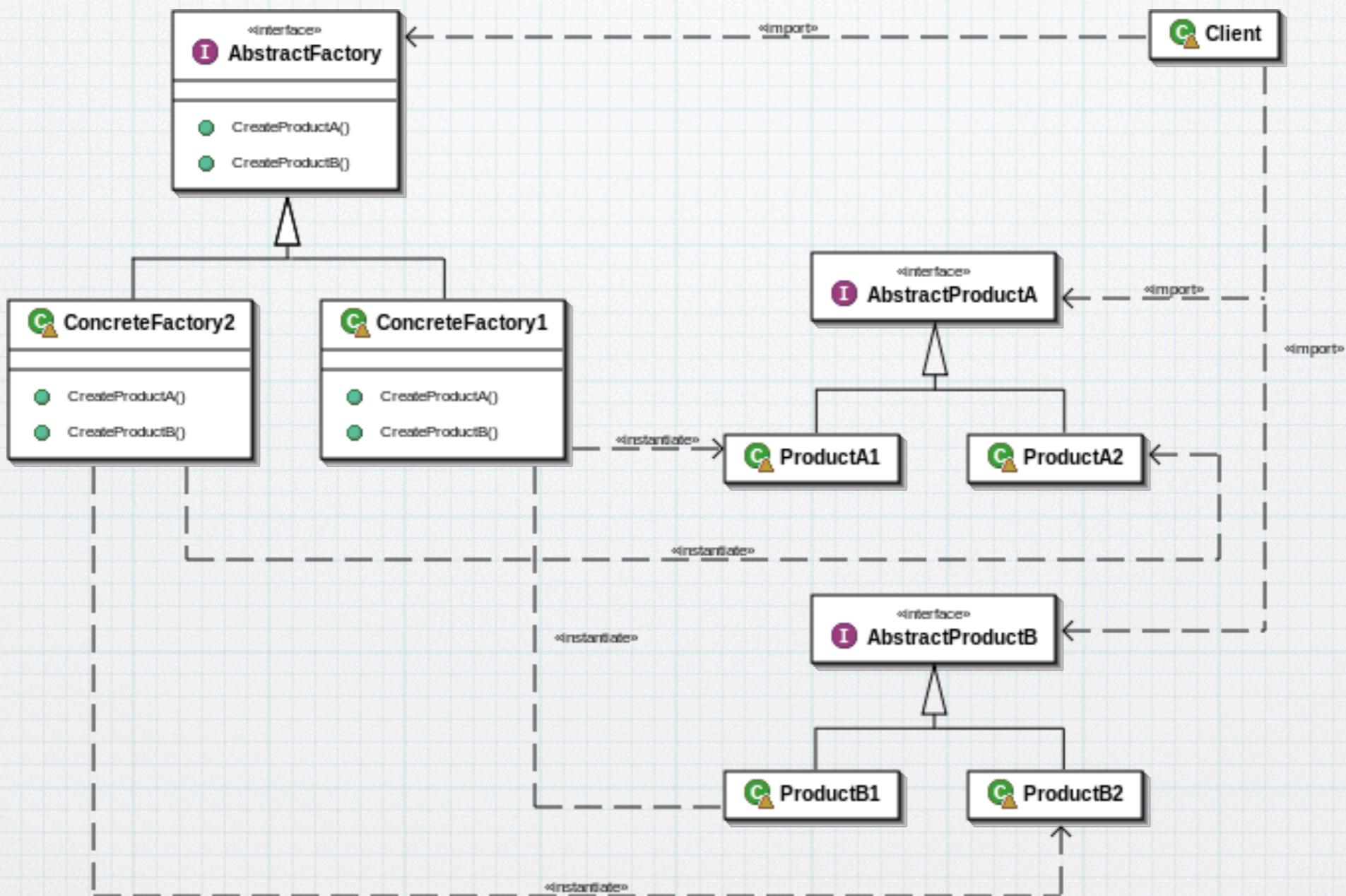
Mer avansert: State



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

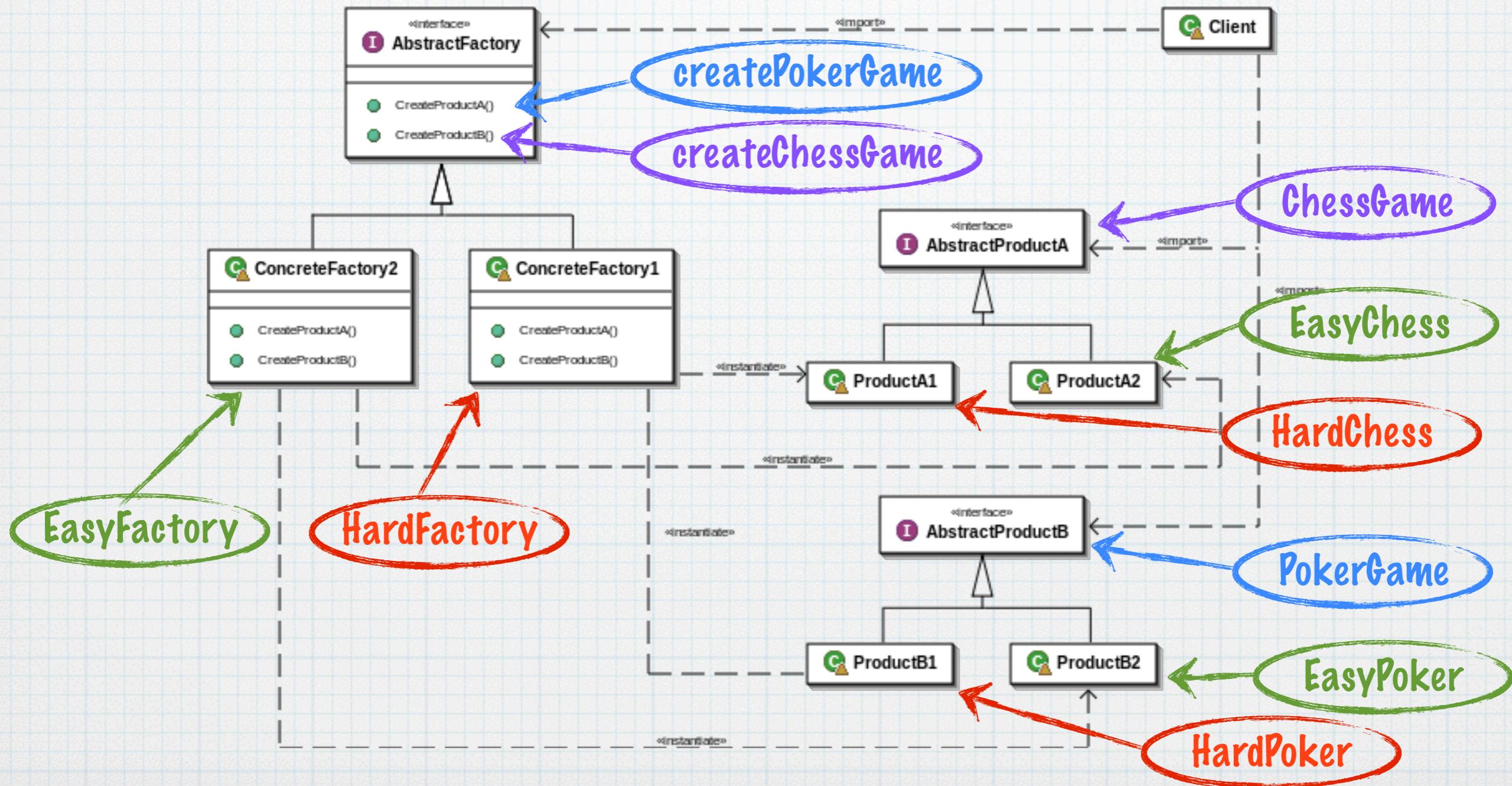
Mye bruk:

Abstract Factory



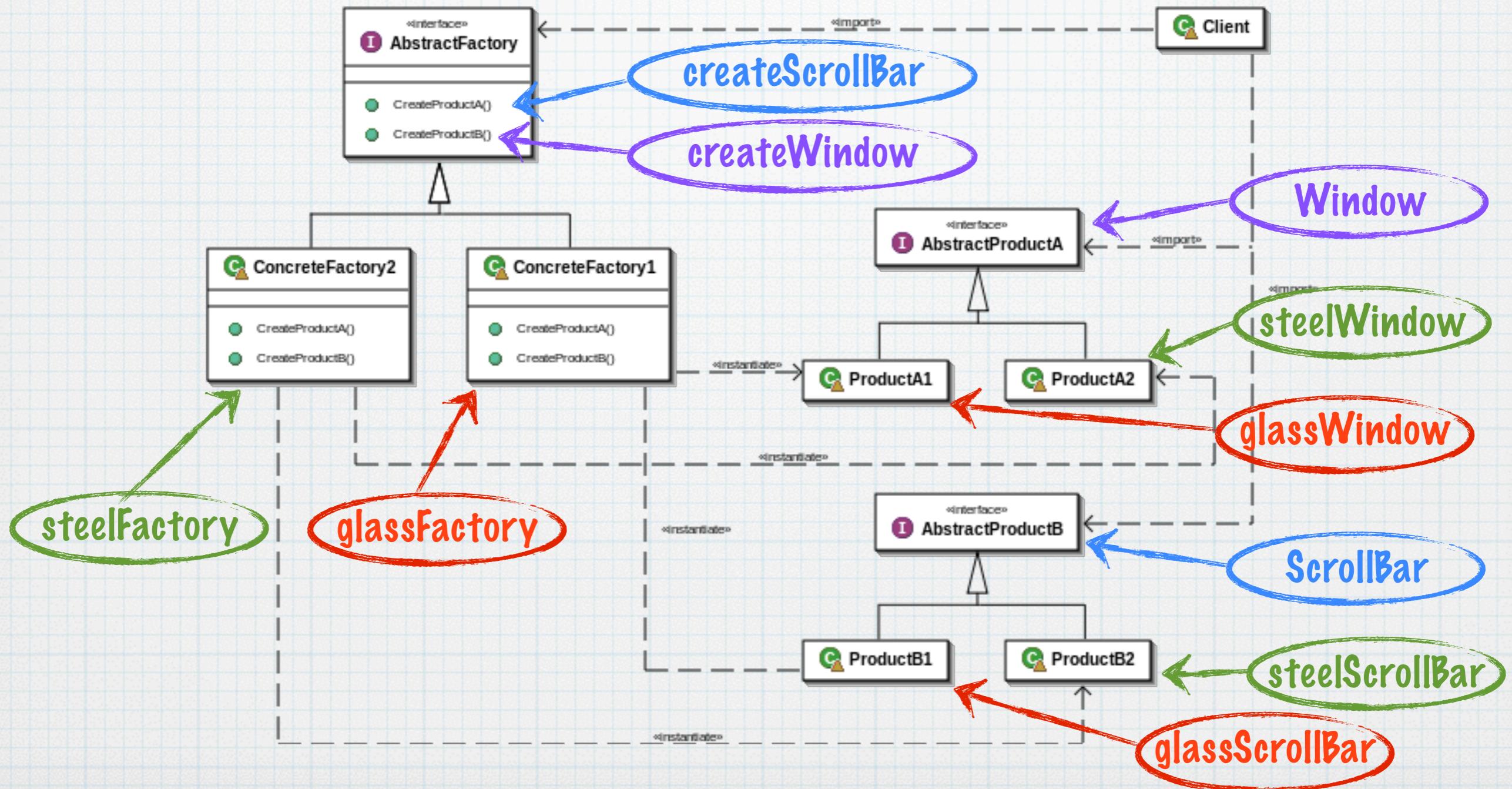
Mye bruk:

Abstract Factory



Mye bruk:

Abstract Factory



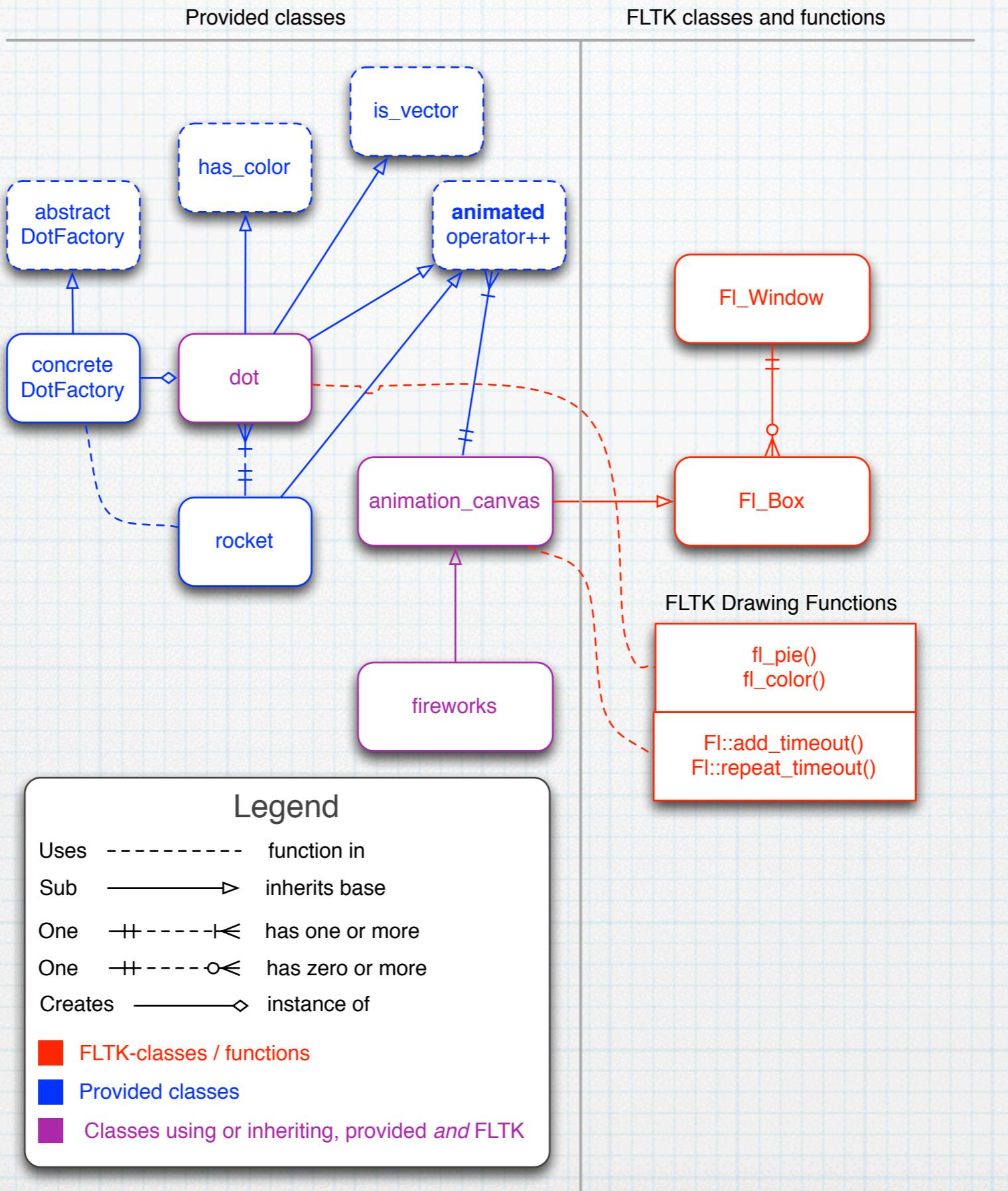
Abstract Factory

- * 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 overrider 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"?
- * 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 overrider visse av factory-metodene, mens resten blir i base-class.

Fireworks class hierarchy - with Factory



Vår “dotFactory”

- * 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

Strategy

AbstractFactory

Chain of Responsibility

Maaange patterns

AbstractFactory

Builder

Flyweight

Proxy

FactoryMethod

Prototype

Adapter

Mer i neste uke!

Facade

Command

State

Composite

Memento

Thread Pool

Strategy

Decorator

Chain of Responsibility

AbstractFactory