

# Polymorfi og abstrakte klasser

---

Forelesning 7,  
Effektiv kode med C og C++, vår 2014  
Alfred Bratterud

Pensum: Kap.14

# Agenda:

- \* Copy- og move-constructorer
- \* Årv så langt
- \* Polymorfi
- \* Abstrakte klasser

# Copy- og move- constructor

- \* Hva skjer her: `myclass a(), b(); a=b;`
  - \* Innholdet i b \*kopieres inn\* i a
  - \* Hva hvis myclass inneholder pekere?
- \* En "copy constructor" er en constructor som tar sin egen type som argument
  - \* `myclass(myclass& c){/*...kopier alle data*/}`
  - \* I den kan du definere "deep copy" - dvs. kopi av dynamisk minne
  - \* Alle klasser har en default copy-constructor
- \* Copy er litt tregt... C++11 introduserer "move constructorer", som flytter innholdet fra b inn i a, og "nøller ut" b. Flyttingen gjør at vi slipper kopiering.
  - \* `myclass(myclass&& c){/*...flytt alle data*/}`
- \* For å kunne få `a=b;` til å gjøre "deep copy" eller "move", må du implementere assignment-operatoren, og la den gjøre kopiering eller "swap"

# Demo:

---

`copy_constructor.cpp`  
`(move_constructor.cpp)`

Eksempler:

<http://www.cplusplus.com/articles/y8hv0pDG/>  
<http://en.cppreference.com/w/cpp/language/>

# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
- \* Multippel arv gir mulighet for "Mixin"
- \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
- \* Kan vi legge flere typer objekter i samme container?
- \* Kan vi få ulike varianter av en baseklasse til å oppføre seg forskjellig- selv om vi betrakter dem som "base-type"?

# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
- \* Ja - med abstrakte klasser og multippel arv
  - Multippel arv gir mulighet for Maxim
- \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
- \* Kan vi legge flere typer objekter i samme container?
- \* Kan vi få ulike varianter av en baseklasse til å oppføre seg forskjellig- selv om vi betrakter dem som "base-type"?

# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
- \* Ja - med abstrakte klasser og multippel arv
- \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
  - Ja - hvis de har felles baseklasse
- \* Kan vi legge flere typer objekter i samme container?
- \* Kan vi få ulike varianter av en baseklasse til å oppføre seg forskjellig - selv om vi betrakter dem som "base-type"?

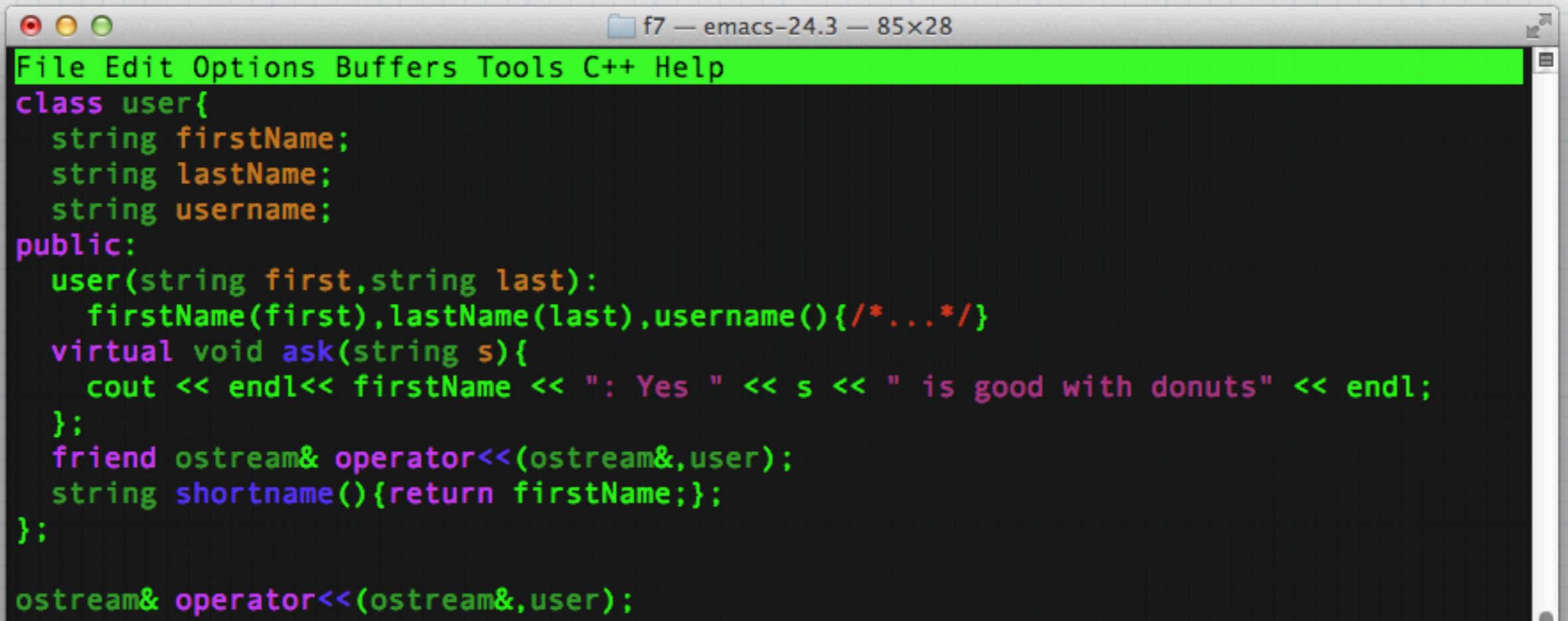
# Arv

- \* En klasse kan arve egenskaper fra en annen klasse.
- \* Hovedmotiv med arv:
  - \* Mindre kode: Færre duplikater, færre feil.
  - \* Eksempler: String med stringsplit, egne exceptions etc. map med "keys"-vector etc.
  - \* Ja - med abstrakte klasser og multippel arv
  - \* Kan vi oppnå det samme som "interface" i Java, altså å "tvinge" utviklere til å implementere visse funksjoner?
    - Ja - hvis de har felles baseklasse
  - \* Kan vi legge flere typer objekter i samme container?
    - Ja - med polymorfi
  - \* Kan vi få ulike varianter av en baseklasse til å oppføre seg forskjellig- selv om vi betrakter dem som "base-type"?

# Polymorfi

- \* “Poly” betyr mange, “morf” betyr “form”
- \* Polymorfi er når et kall på en funksjon **f** i en instans **B**, sender kallet nedover i “årv”, til subklassen **S**.
- \* For at det skal skje må **f** være deklarert **virtual** og definert både i **B** og i **S**. Definisjonen i subklassen er da en “**override**” (kan/bør spesifiseres i C++11)
- \* Hvis man ikke bruker **virtual** kan **B** og **S** fint ha funksjonen **f** men den er da ikke “overridet” og funker kun når man har en peker av subtype.

# Polymorfi

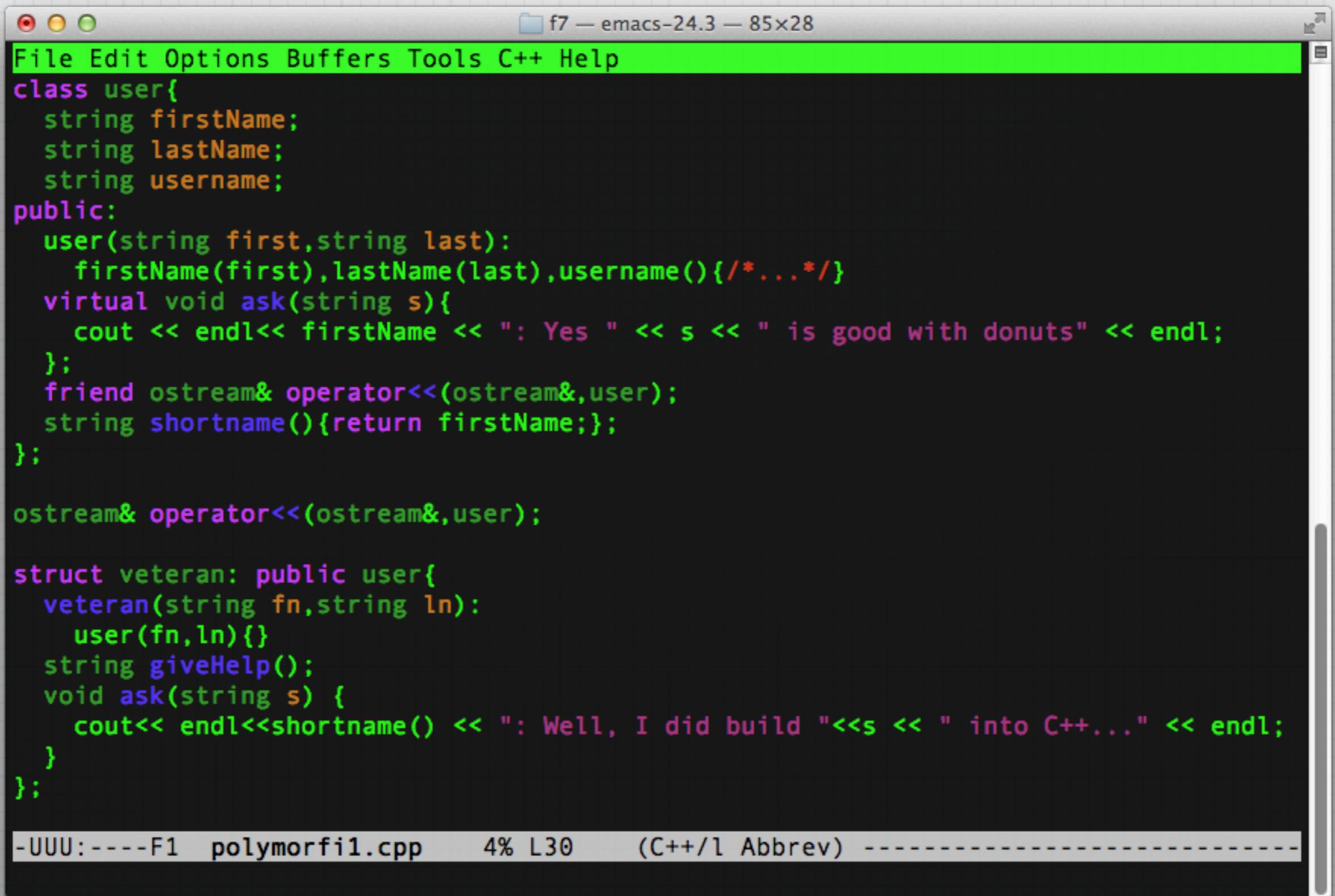


A screenshot of an Emacs window titled "f7 — emacs-24.3 — 85x28". The window contains C++ code for a "user" class. The code defines a constructor that initializes first and last names, a virtual member function "ask" that prints a message about donuts, a friend operator for output streams, and a member function "shortname" that returns the first name.

```
File Edit Options Buffers Tools C++ Help
class user{
    string firstName;
    string lastName;
    string username;
public:
    user(string first,string last):
        firstName(first),lastName(last),username()/*...*/
    virtual void ask(string s){
        cout << endl<< firstName << ": Yes " << s << " is good with donuts" << endl;
    };
    friend ostream& operator<<(ostream&,user);
    string shortname(){return firstName;};
};

ostream& operator<<(ostream&,user);
```

# Polymorfi



The screenshot shows an Emacs window titled "f7 — emacs-24.3 — 85x28". The buffer contains C++ code demonstrating polymorphism. The code defines a class `user` with private members `firstName`, `lastName`, and `username`. It has a constructor `user(string first, string last)` and a `virtual void ask(string s)` method. The `ask` method prints a message based on the user's name. A friend `operator<<(ostream&, user)` is defined to output the user's `shortname()`. A struct `veteran` inherits from `user` and overrides the `ask` method to print a different message. The file is named "polymorfi1.cpp".

```
File Edit Options Buffers Tools C++ Help
class user{
    string firstName;
    string lastName;
    string username;
public:
    user(string first, string last):
        firstName(first), lastName(last), username() /*...*/
    virtual void ask(string s){
        cout << endl << firstName << ": Yes " << s << " is good with donuts" << endl;
    };
    friend ostream& operator<<(ostream&, user);
    string shortname(){return firstName;};
};

ostream& operator<<(ostream&, user);

struct veteran: public user{
    veteran(string fn, string ln):
        user(fn, ln){}
    string giveHelp();
    void ask(string s) {
        cout << endl << shortname() << ": Well, I did build "<< s << " into C++..." << endl;
    }
};
```

-UUU:----F1 polymorfi1.cpp 4% L30 (C++/l Abbrev) -----

# Polymorfi

- \* Man kan bruke nøkkelordet **virtual** også i subklasser, men ikke nødvendig. **override** anbefales i stedet (C++11) - da kan kompilatoren hjelpe deg.
- \* Polymorfi funker for pekere, og referanser
- \* Ulempen med å bruke kun en referanse er at man gjerne har brukt new for å lage pekeren - mindre tydelig at man må "delete" en referanse
- \* Hvis man ikke bruker **virtual** kan B og S fint ha funksjonen f men den er da ikke "overridet" og funker kun når man har en peker av subtype.
- \* Klasser med virtuelle funksjoner bør ha virtuelle destruktører...
- \* OBS: Virtuelle funksjoner har litt overhead (vtab)

# Demo:

---

polymorf1.cpp

# Abstrakte klasser

- \* En abstrakt klasse er en klasse som ikke kan instansieres direkte (kun via subtype)
- \* I C++ blir en klasse abstrakt hvis den har minst en **virtual** funksjon som ikke kan implementeres i baseklassen
- \* Det vanlige er å definere funksjonen som "pure virtual" ved å nekte den en kropp, slik:  
**virtual void ask(string question) = 0;**
- \* En annen løsning er å gjøre en virtuell funksjon "protected".
- \* Abstrakte klasser er nyttige når det å snakke om instanser av klassen blir "for generelt" til å gi mening. Feks:
  - \* "fileFormatCheker" (hvilet format?)
  - \* "connection" (hva slags?)

# Abstrakte klasser



```
f7 — emacs-24.3 — 65x25
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){}
    virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};

class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html",HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}

-UUU:---F1  abstrakte_klasser.cpp  Bot L24      (C++/l Abbrev) --
```

# Abstrakte klasser

```
f7 — emacs-24.3 — 65x25
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){}
    virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};
class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html",HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}

-UUU:---F1  abstrakte_klasser.cpp  Bot L24  (C++/l Abbrev) --
```

"Pure Virtual"

Typet argument - Bra design!

# Abstrakte klasser

The screenshot shows an Emacs window titled "f7 — emacs-24.3 — 65x25" displaying C++ code. The code defines an abstract base class `fileFormatValidator` with a pure virtual function `valid`, and a derived class `HTMLvalidator` that inherits from it and provides a concrete implementation. The code also includes a main function that creates an `HTMLvalidator` object and checks its validity.

```
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){}
    virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};
class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html",HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}
```

Three callout bubbles highlight specific parts of the code:

- A blue bubble points to the `virtual bool valid()=0;` line with the text "Pure Virtual".
- A blue bubble points to the `HTMLvalidator` constructor with the text "Typet argument - Bra design!".
- A blue bubble points to the `HTMLvalidator` declaration with the text "Riktig instansiering og bruk".

# Abstrakte klasser

```
f7 — emacs-24.3 — 65x25
File Edit Options Buffers Tools C++ Help
class fileFormatValidator{
    string filename;
public:
    fileFormatValidator(string filename):filename(){}
    virtual bool valid()=0;
};

enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};
class HTMLvalidator : public fileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : fileFormatValidator(filename){/*...*/}
    virtual bool valid(){cout << "Probably not. "; return false;}
};

int main(){
    HTMLvalidator validator("index.html",HTML5);
    cout << "Valid HTML5: " << validator.valid() << endl;
    //fileFormatValidator v("myFile.ext"); //ERROR: Abstract!
}
```

“Pure Virtual”

Typet argument - Bra design!

Ugyldig: abstrakt klasse!

# Demo:

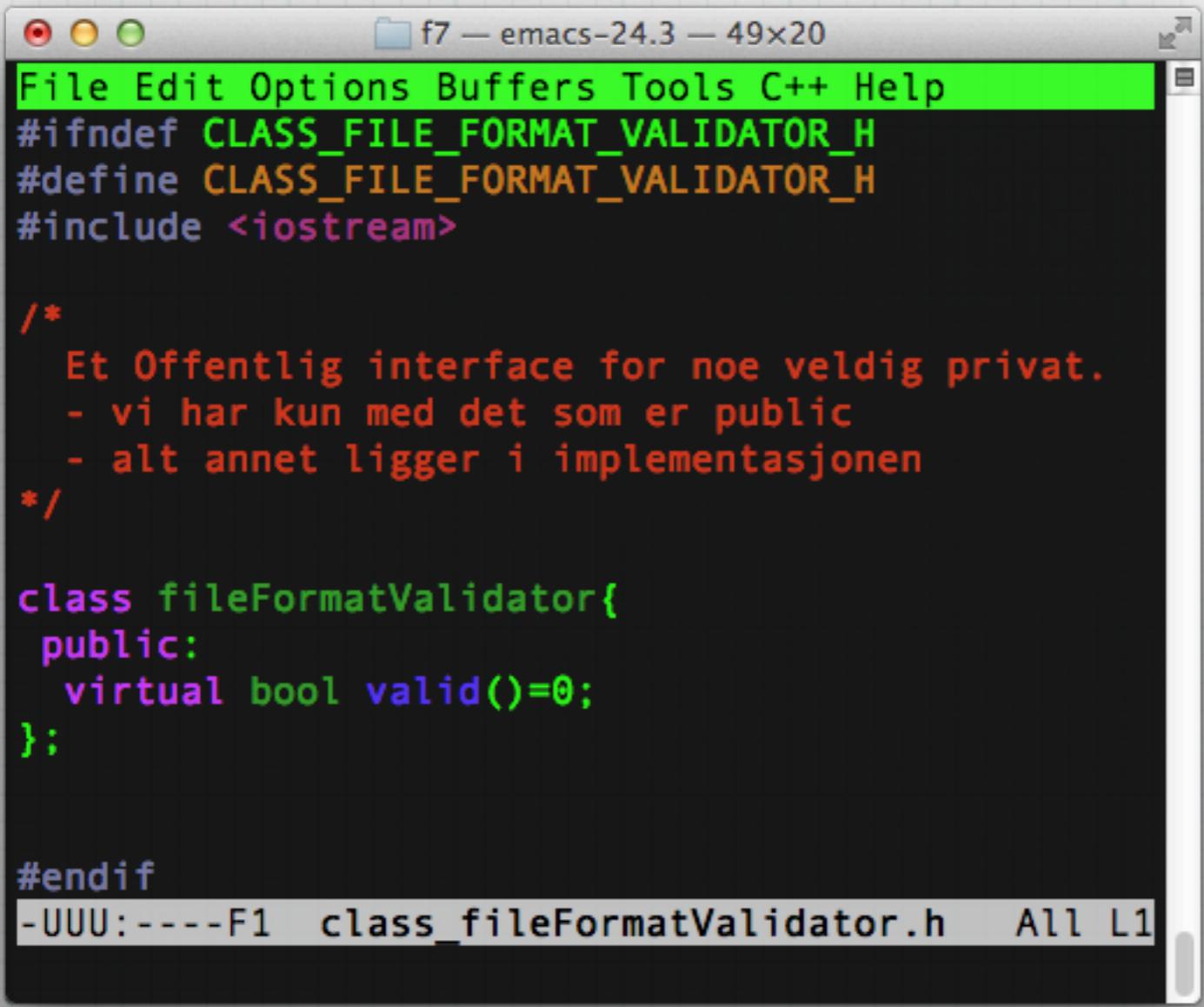
---

abstrakte\_klasser.cpp

# Abstrakte klasser som interface

- \* Abstrakte klasser kan brukes som "interface" i Java:
  - \* Effektivt sett er de en "liste" av ting som må være med
  - \* En klasse kan arve mange (multippel arv)
  - \* Trenger abstrakte klasser noe privat?
    - \* Ikke nødvendigvis - men noen medlemmer er ofte nødvendig (feks. "filename", "username" etc.)
    - \* Og: man kan gjerne implementere funksjoner i en abstrakt klasse. De kan bare ikke brukes før du har en subklasse. Nyttig?
    - \* Ja: feks. funksjoner som "toString" kan gjerne ha en "default"
  - \* Hvis man er "microsoft" og kun ønsker å vise frem det som er public?
  - \* Da kan man lage en abstrakt baseklasse over den første, som bare har med det som er public. Resten kan puttes i implementasjonen.

# Abstrakte klasser som interface



The screenshot shows an Emacs window titled "f7 — emacs-24.3 — 49x20". The buffer contains the following C++ code:

```
File Edit Options Buffers Tools C++ Help
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
#include <iostream>

/*
Et Offentlig interface for noe veldig privat.
- vi har kun med det som er public
- alt annet ligger i implementasjonen
*/
class fileFormatValidator{
public:
    virtual bool valid()=0;
};

#endif
-UUU:----F1  class_fileFormatValidator.h  All L1
```

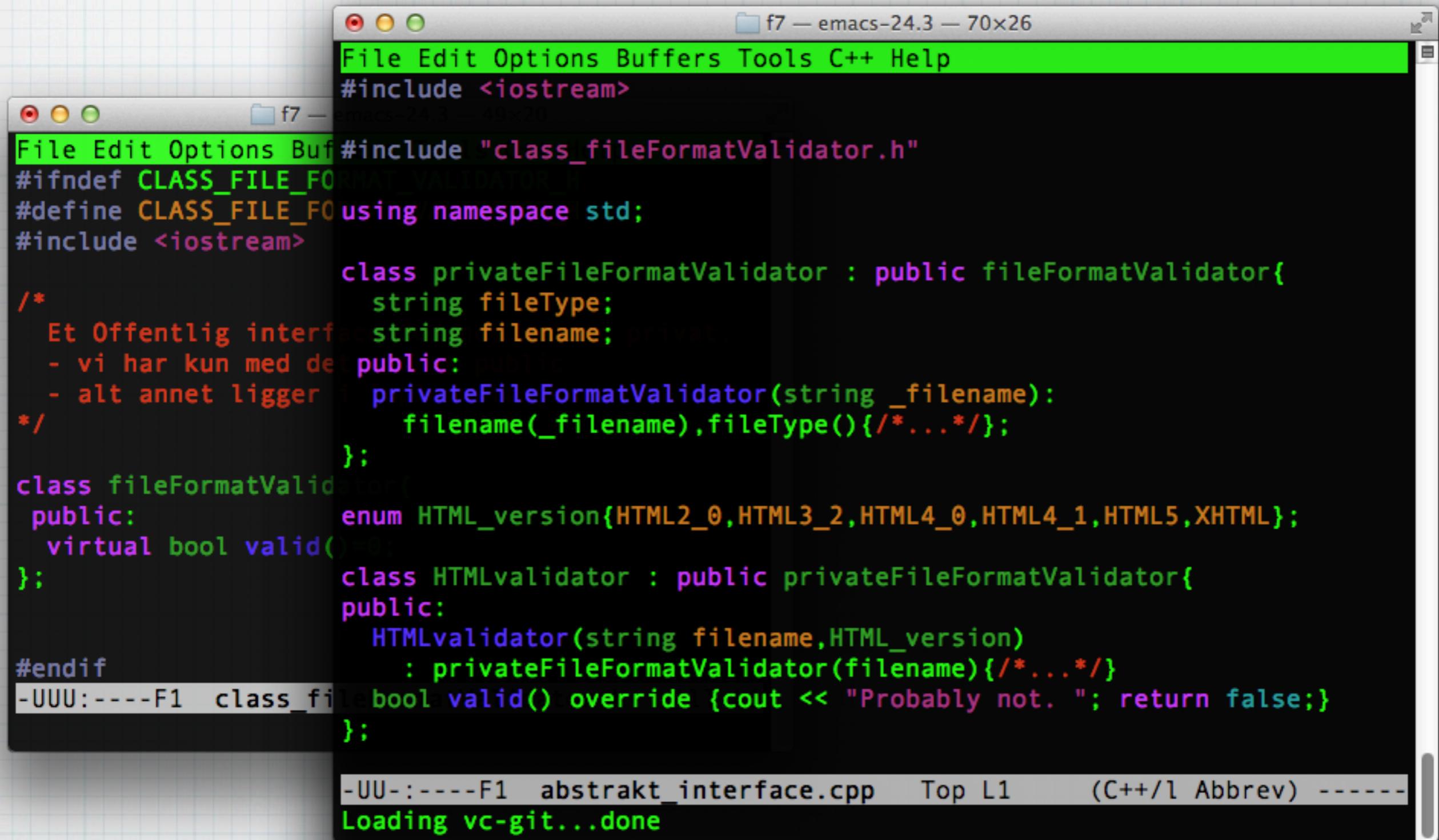
Hvis man er "Microsoft"  
vil man kanskje eksponere  
minst mulig av koden.

Men: Vi vil at folk skal  
kunne bruke bibliotekene  
våre, så vi må ha en  
headerfil

Løsning:  
Enda et nivå av arv

VIS VS SVIN I SÅNOM

# Abstrakte klasser som interface

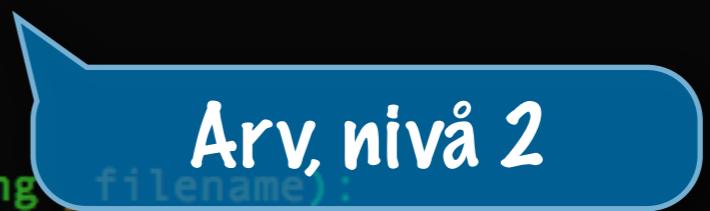


The screenshot shows an Emacs window with two buffers. The title bar indicates it's 'f7 — emacs-24.3 — 70x26'. The top buffer contains the following C++ code:

```
#include <iostream>
#include "class_fileFormatValidator.h"
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
using namespace std;
#include <iostream>
class privateFileFormatValidator : public fileFormatValidator{
/*
    string fileType;
    Et Offentlig interfas string filename; PRIVATE
    - vi har kun med den public:
    - alt annet ligger i privateFileFormatValidator(string _filename):
        filename(_filename),fileType()/*...*/;
*/
}
class fileFormatValidator{
public:
    enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};
    virtual bool valid()=0;
};
class HTMLvalidator : public privateFileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : privateFileFormatValidator(filename){/*...*/}
};
#endif
-UUU:----F1  class_fileFormatValidator.h  Top L1      (C++/l Abbrev) -----
-UUU:----F1  abstrakt_interface.cpp  Top L1      (C++/l Abbrev) -----
Loading vc-git...done
```

The bottom buffer is empty.

# Abstrakte klasser som interface

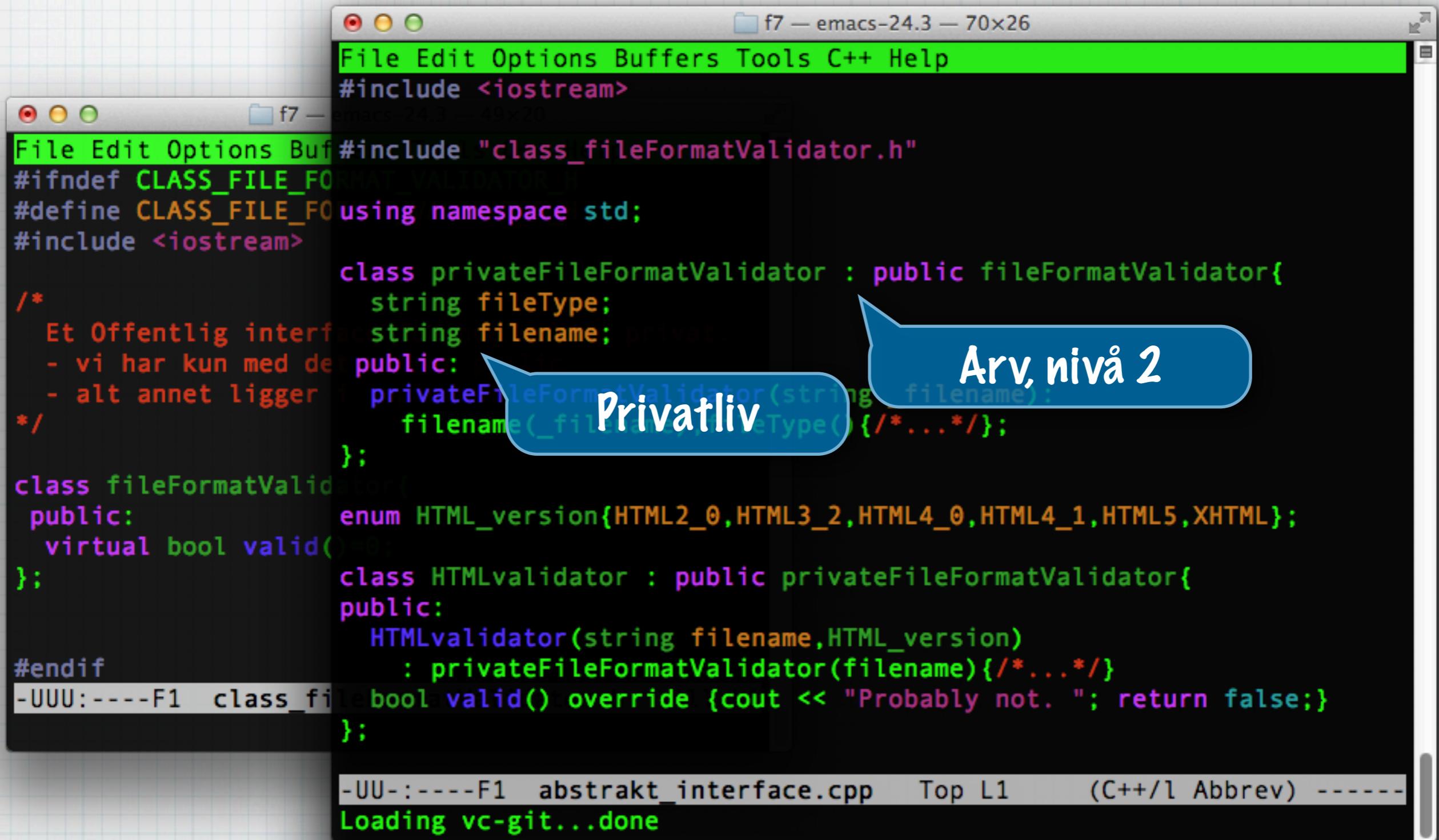


```
f7 — emacs-24.3 — 70×26
File Edit Options Buffers Tools C++ Help
#include <iostream>
# include "class_fileFormatValidator.h"
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H_
#define CLASS_FILE_FO using namespace std;
#include <iostream>
    class privateFileFormatValidator : public fileFormatValidator{
/*
        string fileType;
    Et Offentlig interf string filename; PRIVATE
    - vi har kun med den public: PUBLIC
    - alt annet ligger i privateFileFormatValidator(string _filename):
        filename(_filename),fileType()/*...*/;
*/
    };
class fileFormatValidator{
public: enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};
    virtual bool valid()=0;
};

#endif
-UUU:----F1  class_fileFormatValidator.h
-UU:----F1  abstrakt_interface.cpp  Top L1      (C++/l Abbrev) -----
Loading vc-git...done
```

Arv, nivå 2

# Abstrakte klasser som interface



```
f7 -- emacs-24.3 -- 70x26
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H_
#define CLASS_FILE_FO using namespace std;
#include <iostream>
    class privateFileFormatValidator : public fileFormatValidator{
/*
        string fileType;
    Et Offentlig interfas string filename; private
    - vi har kun med den public:
    - alt annet ligger i privateFileFormatValidator(string filename)
        filename(_filename,fileType()); /*...*/
*/
    };
class fileFormatValidator{
public:
    enum HTML_version{HTML2_0,HTML3_2,HTML4_0,HTML4_1,HTML5,XHTML};
    virtual bool valid()=0;
};

#endif
-UUU:----F1  class_fileFormatValidator.h
-UU:----F1  abstrakt_interface.cpp  Top L1      (C++/l Abbrev) -----
Loading vc-git...done
```

Arv, nivå 2

Privatliv

# Abstrakte klasser som interface

The screenshot shows an Emacs window with two buffers. The top buffer, titled 'f7 - emacs-24.3 - 70x26', contains the following C++ code:

```
#include <iostream>
#include "class_fileFormatValidator.h"
#ifndef CLASS_FILE_FORMAT_VALIDATOR_H
#define CLASS_FILE_FORMAT_VALIDATOR_H
using namespace std;
#include <iostream>
class privateFileFormatValidator : public fileFormatValidator{
/*
    string fileType;
    string filename; private
- vi har kun med den public:
- alt annet ligger i private
    filename(_filename,fileType());
*/
};
class fileFormatValidator{
public:
    enum HTML_version{HTML_3_2,HTML4_0,HTML4_1,HTML5,XHTML};
    virtual bool valid()=0;
};
class HTMLvalidator : public privateFileFormatValidator{
public:
    HTMLvalidator(string filename,HTML_version)
        : privateFileFormatValidator(filename){/*...*/}
};
#endif
-UUU:----F1 class_fileFormatValidator.h
bool valid() override {cout << "Probably not. "; return false;}
};

-UU:----F1 abstrakt_interface.cpp   Top L1      (C++/l Abbrev) -----
Loading vc-git...done
```

Annotations in the code:

- A callout bubble points to the line `privateFileFormatValidator : public fileFormatValidator{` with the text "Arv, nivå 2".
- A callout bubble points to the line `private` with the text "Privatliv".
- A callout bubble points to the line `HTML_version{HTML_3_2,HTML4_0,HTML4_1,HTML5,XHTML};` with the text "Arv, nivå 3".

# Abstrakte klasser som interface

OBS:

I kurset skal alle klasser defineres i egne headerfiler. Her har vi gjort det enkelt, for eksemplets skyld.

OBS OBS:

Vi oppmuntrer ikke til denne doble arven i kurset, men det er nødvendig/nyttig for noen.

```
f7 — emacs-24.3 — 70×26
File Edit Options Buffers Tools C++ Help
#include <iostream>
#include "class_fileFormatValidator.h"
FORMAT_VALIDATOR_H
using namespace std;

class privateFileFormatValidator : public fileFormatValidator{
    string fileType;
    string filename; private:
public:
    privateFileFormatValidator(string filename, string fileType) {/*...*/}
};

enum HTML_version{HTML3_2, HTML4_0, HTML4_1, HTML5, XHTML};
HTMLvalidator(
    HTMLvalidator(string filename, HTML_version)
        : privateFileFormatValidator(filename){/*...*/}
    bool valid() override {cout << "Probably not. "; return false;}
};

-UU-----F1  abstrakt_interface.cpp  Top L1      (C++/l Abbrev) -----
Loading vc-git...done
```

Arv, nivå 2

Privatliv

Arv, nivå 3

# Demo:

---

abstrakt\_interface.cpp  
class\_fileFormatValidator.h

---