PB161
Základy OOP

Tomáš Brukner

Sylabus

- Co je to OOP?
- Jaké jsou základní principy OOP?
- Jak se projevují v C++?

https://cs.wikipedia.org/wiki/Strahovská_knihovna

```
SELECT *
FROM books
WHERE pages < 250
AND student_friendly = True
```

```
struct Book {
  const char* name;
  const char* author;
  unsigned pages;
  bool student_friendly;
};

for(unsigned i = 0; i < books_size; ++i) {
  if (books[i].pages < 250 && books[i].student_friendly) {
    printf("%s: %s", books[i].author, books[i].name);
  }
}
```

```
class Book {
public:
  const std::string name;
  const std::string author;
  const unsigned pages;
  const bool student_friendly;

  unsigned getPageWithRecommendations() const;
};
```

Paradigmata

- Deklarativní
  - SQL, funkcionální jazyky, logické...
- Imperativní
  - Procedurální - C, Pascal...
  - Objektově orientované - C++, Java, Python…

Více: https://en.wikipedia.org/wiki/List_of_programming_languages_by_type

Jedno paradigma?

Jedno paradigma?

- C++ (generic, imperative, object-oriented (class-based), functional)
- ECMAScript (functional, imperative, object-oriented (prototype-based))
- Python (functional, compiled, interpreted, object-oriented (class-based), imperative, metaprogramming, extension, impure, interactive mode, iterative, reflective, scripting)

Více: https://en.wikipedia.org/wiki/List_of_programming_languages_by_type

List of multi-paradigm programming languages

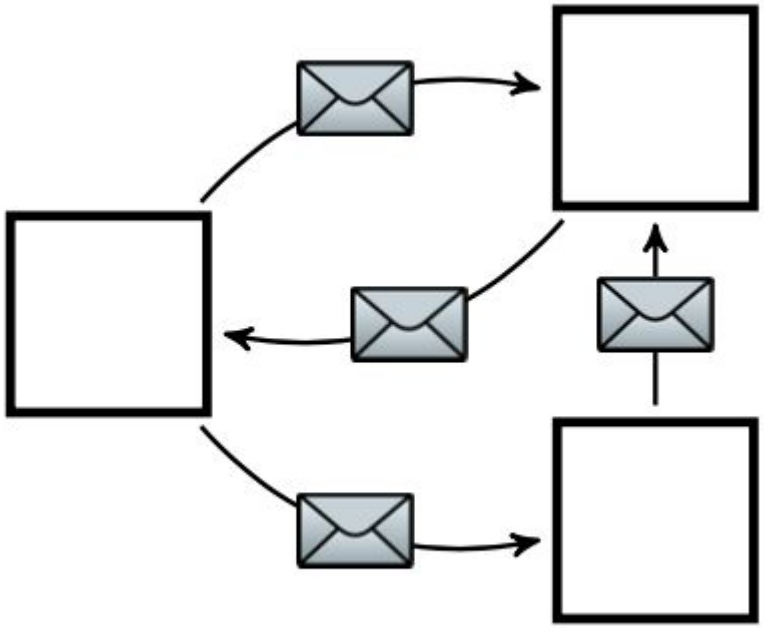
Language	Number of Paradigms	Concurrent	Constraints	Dataflow	Declarative	Distributed	Functional	Metaprogramming	Generic	Imperative	Logic	Reflection	Object-oriented	Pipelines	Visual	Rule-based	Other paradigms
Ada[2][3][4][5][6]	5	Yes[a 1]	No	No	No	Yes	No	No	Yes	Yes	No	No	Yes[a 2]	No	No	No	No
ALF	2	No	No	No	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
AmigaE[citation needed]	2	No	No	No	No	No	No	No	No	Yes	No	No	Yes[a 2]	No	No	No	No
APL	2	No	No	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	No
BETA[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes[a 2]	No	No	No	No
C++	7 (14)	Yes[7][8][9]	Library[10]	Library[11][12]	Library[13][14]	Library[15][16]	Yes	Yes[17]	Yes[a 3]	Yes	Library[18][19]	Library[20]	Yes[a 2]	Yes[21]	No	Library[22]	No
C#	6 (7)	Yes	No	Library[a 4]	No	No	Yes[a 5]	No	Yes	Yes	No	Yes	Yes[a 2]	No	No	No	reactive[a 6]
Chuck[citation needed]	3	Yes	No	No	No	No	No	No	No	Yes	No	No	Yes[a 2]	No	No	No	No
Claire	2	No	No	No	No	No	Yes	No	No	No	No	No	Yes[a 2]	No	No	No	No
Common Lisp (some other paradigms are implemented as libraries)[citation needed]	5	No	No	No	No	No	Yes	Yes	No	Yes	No	Yes	Yes[a 2]	No	No	No	No
Curl	5	No	No	No	No	No	Yes	No	Yes[a 3]	Yes	No	Yes	Yes[a 2]	No	No	No	No
Curry	4	Yes	Yes	No	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
D (version 2.0)[23][24]	6	Yes[a 7]	No	No	No	No	Yes	Yes[25][a 3]	Yes[a 3]	Yes	No	No	Yes[a 2]	No	No	No	No
Dylan[citation needed]	3	No	No	No	No	No	Yes	No	No	No	No	Yes	Yes[a 2]	No	No	No	No
E	3	Yes	No	No	No	Yes	No	No	No	No	No	No	Yes[a 2]	No	No	No	No
ECMAScript[26][27] (ActionScript, E4X, JavaScript, JScript)	4 (5)	partial (promises, native extensions)[a 8]	No	No	No	No	Yes	No	No	Yes	No	Yes	Yes[a 9]	No	No	No	reactive[a 10]
Embarcadero Delphi	3	No	No	No	No	No	No	No	Yes[a 3]	Yes	No	No	Yes[a 2]	No	No	No	No
Erlang	3	Yes	No	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
Elixir	4	Yes	No	No	No	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No

https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages

Objektový pohled na svět

Objektový pohled na svět - principy

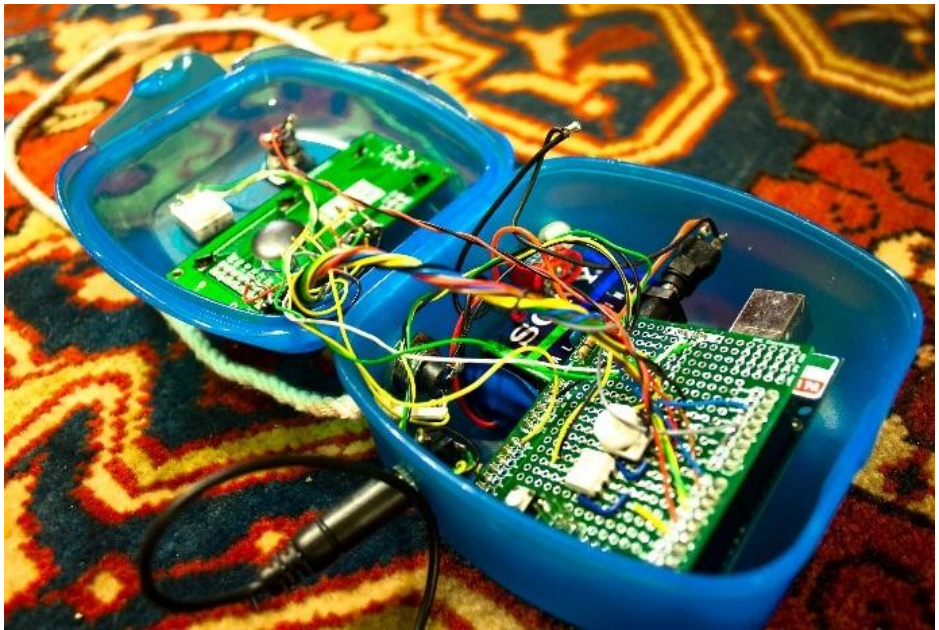
- Svět se skládá z objektů
- Každý objekt má svůj vnitřní stav
- Objekty mezi sebou komunikují pomocí zpráv



Objektový pohled na svět - principy

- Směrem ven je objekt černá skříňka.
- Zaměřujete se pouze na to, jak objekt komunikuje.
- Nemusíte znát vnitřnosti - skrýváte je před zbytkem světa.

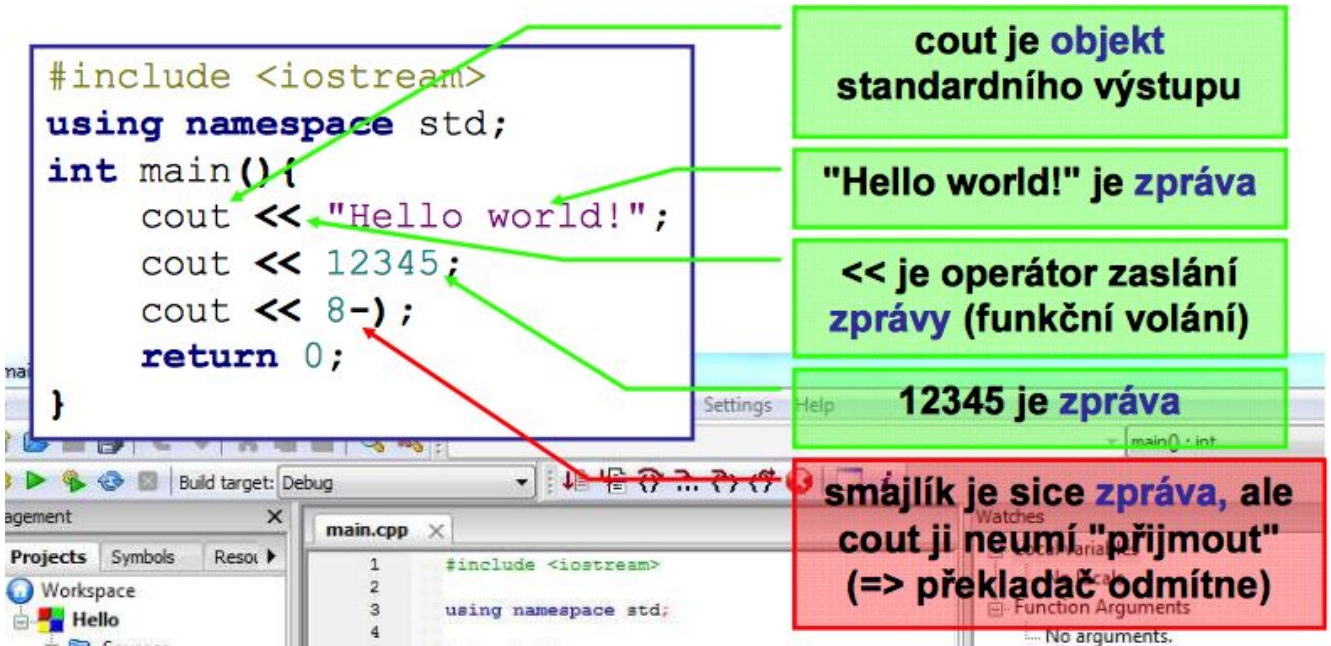
→ **Zapouzdření**



Objektový pohled na svět - principy

- Jedinou povolenou komunikací jsou zprávy.
- Samotná zpráva je objektem.
- Objekt musí přijmout zprávu, které rozumí, a odmítnout tu, které nerozumí.

→ **Abstrakce**



Objektový pohled na svět - principy

- Různé objekty mohou umět zpracovat stejnou zprávu.
- Reakce se ale mohou liši.
- Z pohledu odesílatele je důležité chování, samotné objekty jsou zaměnitelné.

→ **Polymorfismus**

http://www.micheltriana.com/blog/2011/02/03/polymorphism-back-to-school

```
c = a + b;
3 = 1 + 2;
"ahoj" = "a" + "hoj";
```

Objektový pohled na svět - pravidla

- Pokud by všechny objekty byly unikátní, musel by být každý objekt velmi pracně definován.
- Proto je možnost stavět jeden objekt na nějakém jiném.
- A to buď z pohledu znovuvyužití implementace, anebo doimplementování chování.

→ **Dědičnost**

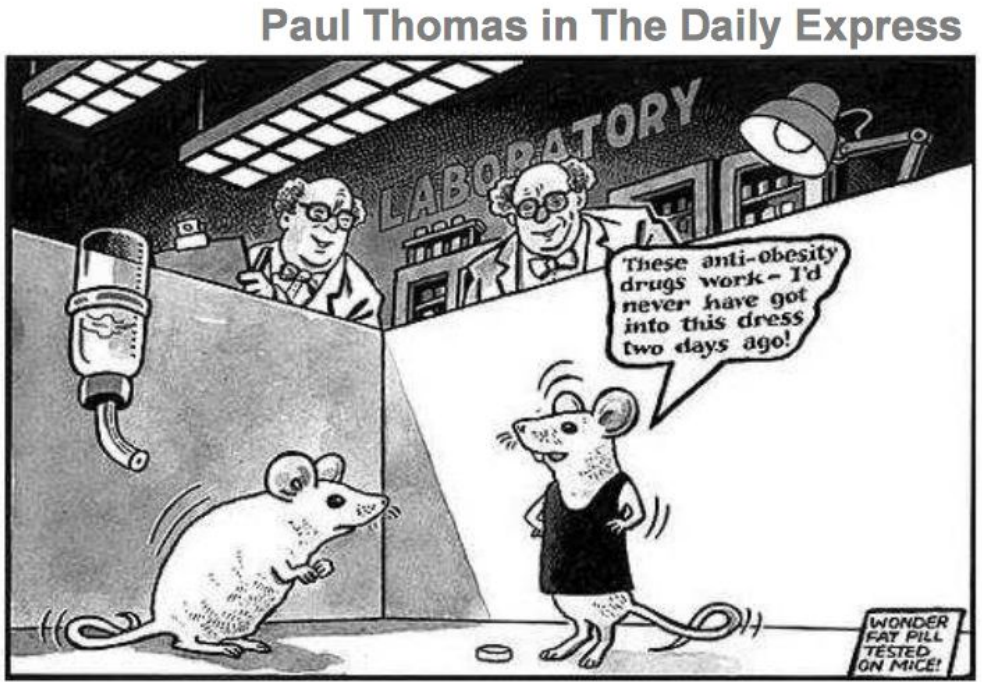
Principy OOP

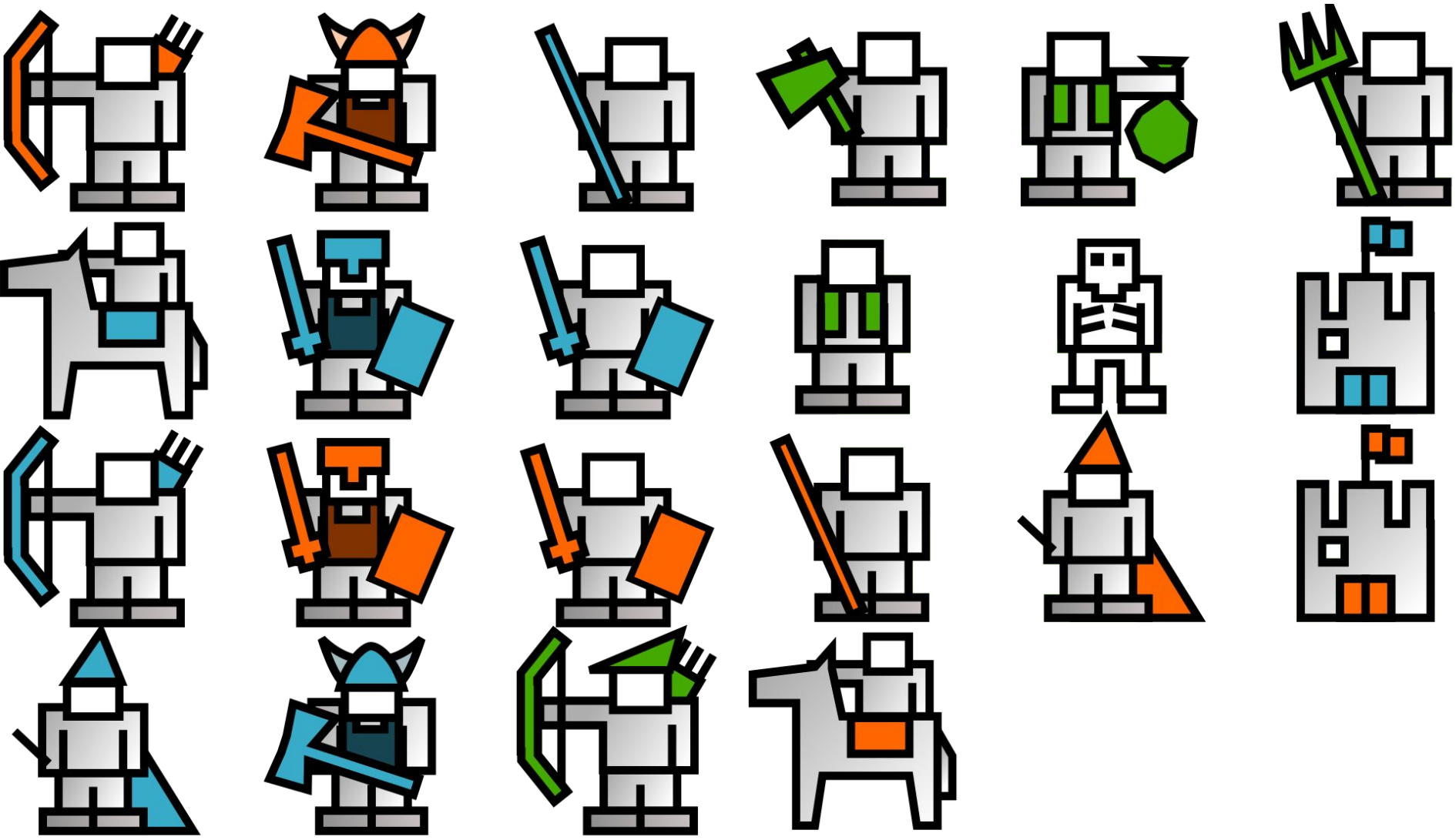
- Zapouzdření
- Abstrakce
- Polymorfismus
- Dědičnost

OOP příklady

Laboratoř

V laboratoři jsou pěstované různé druhy zvířat. V tuto chvíli pouze myši a pavouci. Zvířata se pohybují po ohraničeném prostoru, ve kterém se nachází potrava. Pokud se k ní dostanou, sežerou ji a vyrostou v závislosti na druhu zvířete.

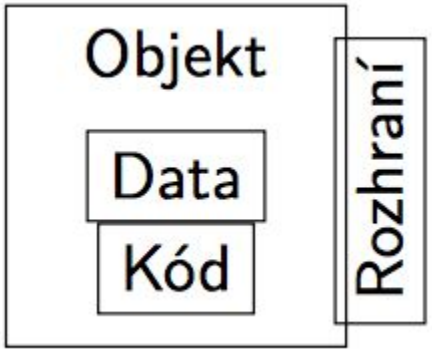




OOP v C++

Zapouzdření

- Na jednom místě jsou data (*atributy*) i kód (*metody*).
- Objekty jsou *instance tříd*, posílání zpráv *volání metod.*
- Klíčová slova **public**, **private** a **protected**



Proč se v C++ používají i veřejné atributy oproti jiným jazykům?

Zapouzdření

- Omezení toho, co vnější svět může udělat → ochrana proti chybám
- Nutí vás program rozvrhnout do nezávislých částí → vnějších a vnitřních
- Umožňuje změnit implementační detaily bez ohledu na uživatele.

```
class Counter {
public:
  unsigned getCount() const;
private:
  void setCount();
protected:
  unsigned calculateCount() const;
};
```

Abstrakce

- Lze pracovat s ideální představou, nikoliv s implementačními detaily.
- *Jak souvisí se zapouzdřením?*

- **Datová abstrakce**
  - Můžete přistupovat k datům, aniž byste věděli, kde a jak fyzicky jsou.
  - *Paměť, disk, server; JPG, GIF, BMP…*
- **Funkční abstrakce**
  - Nemusíte vědět, jak se nějaká akce udělá; jde vám o to ji vyvolat.
  - *Uložení obrázku.*

```
class Image {
public:
  Color getPixel(unsigned x, unsigned y) const;
  void putPixel(unsigned x, unsigned y, Color color);

  void save();
};
```

Abstrakce

- Důležité je **vnější chování**.
- To definuje **rozhraní.**
- Co do něj patří?

public **private** **protected**

Polymorfismus

- Různé chování na základě typu.
- Umožňuje se objektům *zachovat různě* v závislosti na typu.
- *Jak to souvisí s abstrakcí?*

- Přetěžování metod
  - `c = a + b;`
- Typově parametrizovaný
  - `vector<int>, vector<bool>`
- Podtypy
  - `out << "Vypis - ale kam?";`

Polymorfismu - podtypy

- Předem je nadefinováné **rozhraní - interface**
- Nějaká jiná třída může toto rozhraní naplnit - **dodat mu chování.**
- Vnější pozorovatel může tuto třídu používat **skrz interface**

```
class Image { //interface
public:
  virtual Color getPixel(unsigned x, unsigned y) const = 0;
  virtual void putPixel(unsigned x, unsigned y, Color color) = 0;
  virtual void save() = 0;
};
```

Dědičnost

- Způsob realizace podtypů.
- Zároveň umožňuje znovu použít kód *(o tom příště).*
- Umožňuje vytvořit třídu, který lze referovat **referencí nadřazeného typu**.
- *Jak to souvisí s polymorfismem?*

```
class GIF : public Image {
public:
  virtual Color getPixel(unsigned x, unsigned y) const override { /* omitted */ }
  virtual void putPixel(unsigned x, unsigned y, Color color) override { /*... */ }
  virtual void save() override { /* omitted */ }
};

GIF gif("img.gif");
Image &img = gif;
img.putPixel(0, 0, BLACK);
img.save();
```

```
class Image {
public:
   virtual Color getPixel(unsigned x, unsigned y) const = 0;
   virtual void putPixel(unsigned x, unsigned y, Color color) = 0;
   virtual void save() = 0
};
class GIF : public Image {
public:
   virtual Color getPixel(unsigned x, unsigned y) const override {}
   virtual void putPixel(unsigned x, unsigned y, Color color) override {}
   virtual void save() override {}
};
class JPG: public Image { /* omitted */ };
class BMP: public Image { /* omitted */ };

GIF gif("img.gif");
JPG jpg("img.jpg");
BMP bmp("img.bmp");
Image &img = gif; //or jpg, or bmp
img.putPixel(0, 0, BLACK);
img.save();
```

Principy OOP

- Zapouzdření
- Abstrakce
- Polymorfismus
- Dědičnost

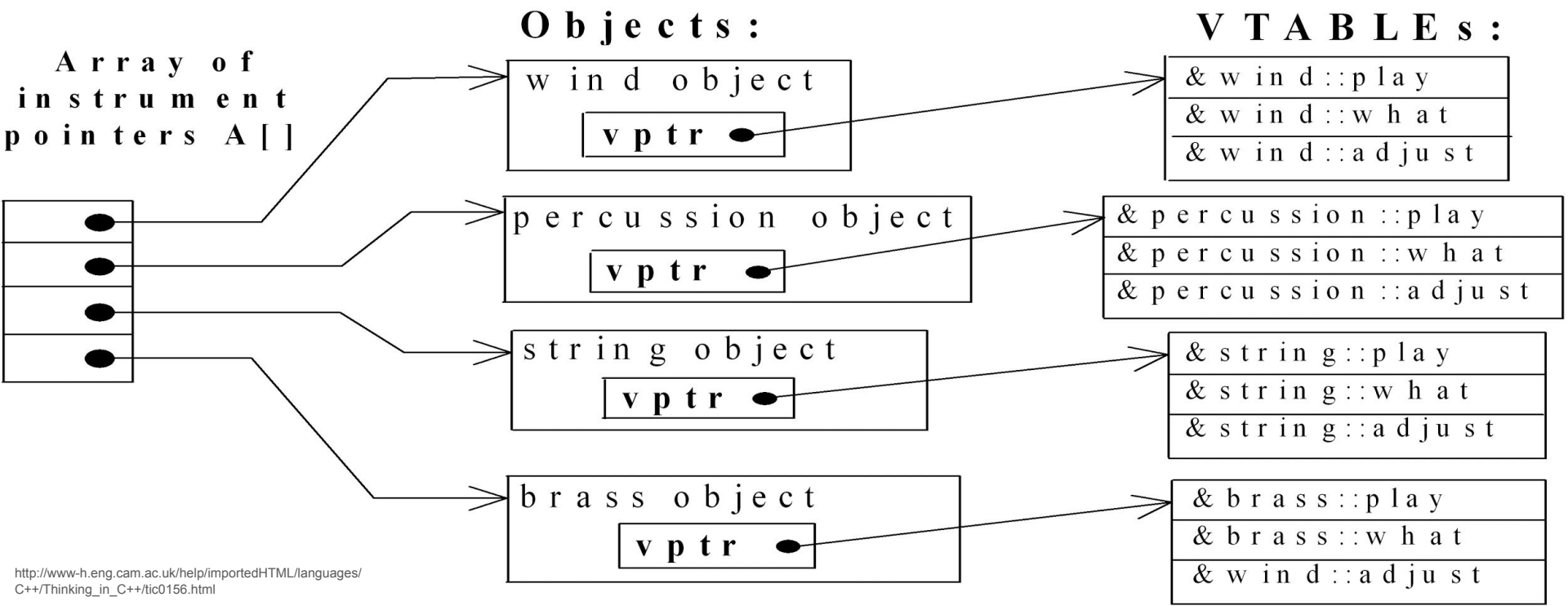
Proč ne čistý OOP jazyk?

Magické “virtual”

- Překladač musí vědět, co má zavolat za metodu.
- Normálně to ví - v asembleru bude přímo adresa funkce, která se má zavolat.
- To je **brzká vazba**; již v době překladu je známo, která implementace se zavolá.
- https://godbolt.org/g/BZccoj

Magické “virtual”

- Každý potomek má tabulku virtuálních funkcí, do které se za běhu program podívá, kterou implementaci má vlastně zavolat.
- To je **pozdní vazba**; implementace se určuje až za běhu programu.
- https://godbolt.org/g/EFpiUV



http://www-h.eng.cam.ac.uk/help/importedHTML/languages/C++/Thinking_in_C++/tic0156.html

Včasná vs pozdní vazba

Včasná vazba	**Pozdní vazba**
Implementace známá v době překladu	Implementace určena až za běhu
V asm přímo adresa funkce	Nutné se podívat do vtable
Rychlejší	Pomalejší
https://godbolt.org/g/BZccoj	https://godbolt.org/g/EFpiUV

Další informace k dědičnosti

- Mohu dědit několikrát?
- Mohu mít některé metody definované v předkovi?
- Mohu se nějak dostat k metodám potomka?
- Mohu dědit jinak, než **public**?
- Jaké jsou alternativy k dědičnosti?

Díky za pozornost