Metodika *Architecture First* a její podpora v prostředí *BlueJ++*

Rudolf PECINOVSKÝ
rudolf@pecinovsky.cz

Proč prosazuji metodiku Architecture First

- Technologická signatura
- Je třeba předvídat vývoj
- Co děláme špatně
- Analogie
- Bolesti současných absolventů
- Nejde jen o profesionální programátory
- Učí se vysvětlovat JAK řešit místo CO řešit
- Konstruktivistická teorie učení
- Zásada ranního ptáčete
- Zpátky k singularitě
- Sémantická mezera
- Nevýhody předčasné koncentrace na kód
- Soustředění na detail
- Zpětná vazba od firem

Katedra informačních technologií Kit VŠE Praha

Technologická signatura

- Okamžik, kdy zařízení vyrobené člověkem dokáže **samostatně** navrhnout a vyrobit zařízení, které je složitější než ono samo
- Očekáváno mezi roky 2025 až 2040
- Od tohoto okamžiku přestanou zařízení ke svému dalšímu zdokonalení potřebovat člověka
- Porovnejme to s okamžikem, kdy studenti, o nichž hovoříme, nastoupí do praxe

Je třeba předvídat vývoj

- Tento stav nespadne z nebe,
  programy se ke své dokonalosti
  budou propracovávat postupně

- Nakolik si při návrhu osnov uvědomujeme
  skutečný současný stav
  a očekávatelný nejbližší vývoj?

- Je třeba učit studenty tak,
  aby při nástupu do praxe nezjistili,
  že to, co se ve škole pracně naučili,
  už v danou chvíli umějí počítače lépe

Co děláme špatně

- **Autoři osnov se snaží naučit žáky to, co bylo vhodné umět v době, kdy tito autoři studovali, resp. opouštěli studium**
- **Jediné, co se většinou ve výuce změnilo, jsou použité nástroje a technologie; vlastního předmětu výuky se změny nedotkly**
  - Leckde se ale nezměnily ani ty nástroje
- **Při návrhu osnov bychom měli více předjímat očekávaný vývoj oboru a pokusit se na něj studenty vhodně připravit**

Analogie

- **Učíme budoucí uživatele a programátory jako kdybychom učili kuchaře pro dobu Magdaleny Dobromily Rettigové**
  - Musejí umět správně rozdělat oheň
  - Musejí umět správně vykrmit husu/kachnu/…
  - Musejí umět zvíře stáhnou/oškubat/…
- **Při takovéto výuce pak nezbyde čas na výuku používání moderních nástrojů**
  - Mikrovlnné trouby, indukční ohřev, roboty, …
- **To se musejí naučit v praxi za chodu**

Bolesti současných absolventů

- **Vedoucí programátorských týmů si stěžují, že školy neopouštějí programátoři, ale kodéři, kteří si osvojili si kódování, ale ne paradigma**
  - Umějí zakódovat navržený program, ale neumějí správně navrhnout jeho architekturu
  - Umějí implementovat interfejs, ale neumějí poznat, kdy jej začlenit do navrhované architektury
  - Umějí používat různé frameworky, ale používají je mechanicky, nechápou jejich podstatu
  - Umějí vyjmenovat návrhové vzory, ale opět nechápou jejich filosofické pozadí
  - Atd., atd.

Nejde jen o profesionální programátory

- **Zmíněné problémy se netýkají pouze výchovy budoucích profesionálních programátorů, ale ukazují se už i u dětí**

- **Klasická výuka programování učí děti soustředění se na detail, přičemž přitom občas ztrácejí ze zřetele celek**
  - Děti se soustředí na to, jak zadanou úlohu vyřešit, resp. jak vyřešit některou její část, ale občas jim přitom uniká, co vlastně řeší
  - Často si neuvědomují, že na jednom místě řeší věci, které spolu přímo nesouvisejí – neuvědomují si, že by se každý objekt měl starat především o sebe

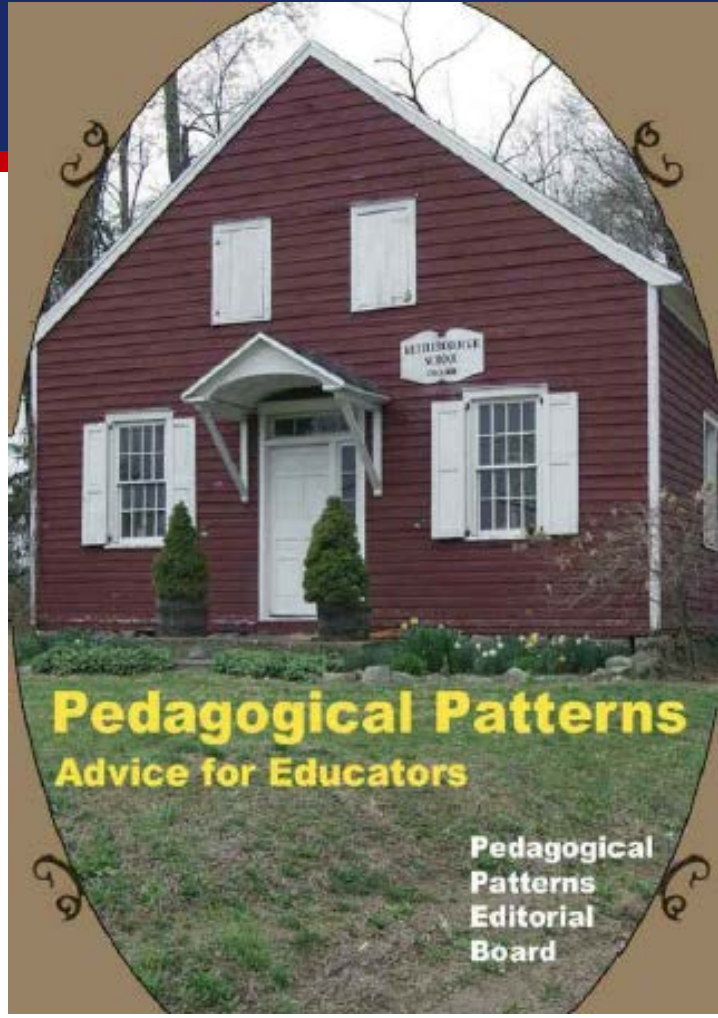
Učí se vysvětlovat JAK řešit místo CO řešit

- **Prakticky všechny základní kurzy učí imperativní způsob programování**
- **Současné programování přitom přechází od imperativního k deklarativnímu, které je lidskému způsobu myšlení bližší**
- **Bohužel, zatím většinou chybějí nástroje pro výuku tohoto stylu řešení problémů**
  - Problém je, že se o vývoji takovýchto nástrojů prozatím ani nemluví

Konstruktivistická teorie učení

- **Veškeré naše nové poznatky se konstruují z toho, co jsme se naučili před tím**
  - Pochopení a interpretace nových poznatků jsou závislé na poznatcích předchozích a na míře zkušenosti s jejich používáním
    - Věřící a nevěřící interpretují často týž jev naprosto rozdílně přičemž každý je zcela přesvědčen o své pravdě
- **Špatně či nešikovně vysvětlené základy výrazně ovlivní veškerou nadstavbu**
  - Někdy lze výsledek výrazně ovlivnit pouhou změnou pořadí vykládaných témat

Zásada ranního ptáčete



- **Když se návrhové vzory prosadily v programování, definovali je někteří badatelé i pro pedagogiku**

- **Zásada ranního ptáčete (Early Bird Pattern):**
  To, co považujeme v daném oboru za nejdůležitější, bychom měli probrat na počátku výuky, anebo alespoň co nejdříve
  - Současná výuka programování začíná výkladem toho, co už nyní dokáží různé generátory kódu, a ne toho, jak tyto generátory použít, co chceme zakódovat

Zpátky k singularitě

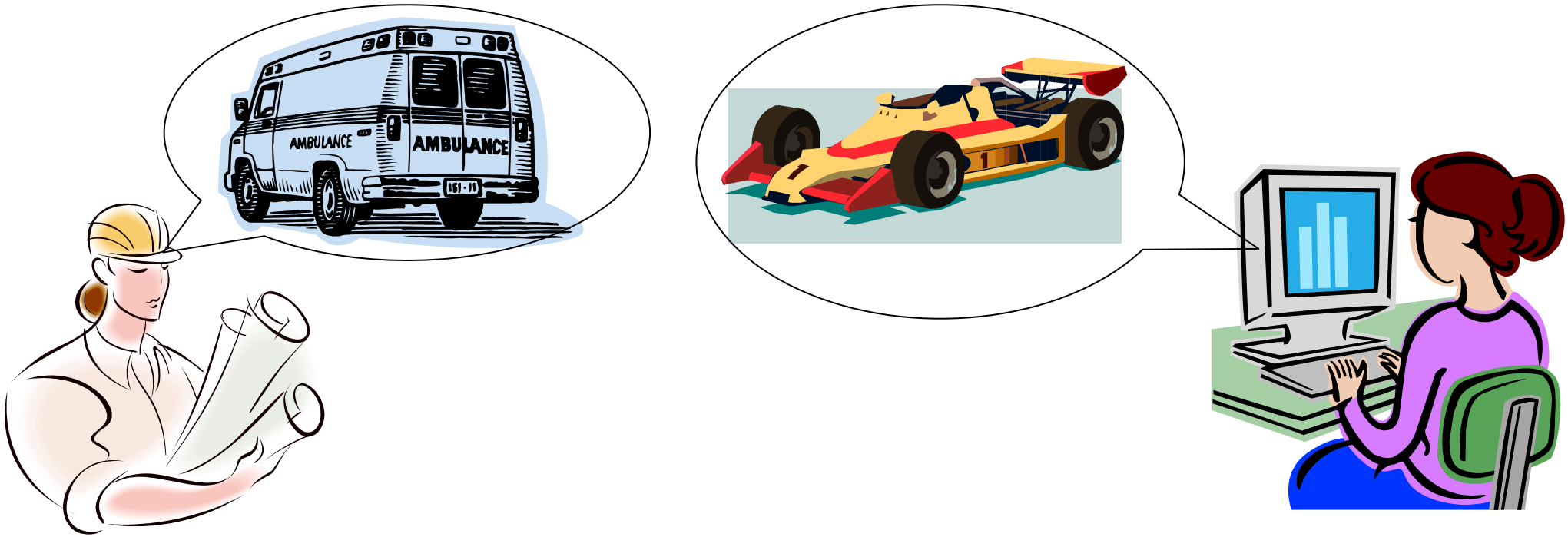
- **Víme-li, že počítač bude postupně přebírat větší a větší část kódování, měli bychom se při výuce soustředit na výklad oblastí, které od nás budou počítače přebírat později**
  - Protože se tím zvedá důležitost těchto oblastí, měli bychom je učit co nejdříve
- **V současnosti se ukazuje, že jako poslední začne přebírat návrh architektury => měli bychom se při výkladu soustředit na něj**
- **V dalším kole pak přijde návrh zadání**

Sémantická mezera

- **Když učíme studenty dělat operace, které za ně může stejně dobře nebo i lépe dělat nějaký stroj, měli bychom k tomu mít pádný důvod**
  - Doposud musel algoritmy vymýšlet člověk, protože stroj za něj většinu vymyslet nedokázal
  - Nepříjemným důsledkem tohoto postupu je to, že se při tom studenti odnaučí řešit problémy tak, jak to je pro ostatní lidi přirozené
  - Když programátor dostane za úkol něco naprogramovat, začne přemýšlet zcela jinak, začne se pohybovat v jiném mentálním prostoru než jeho zákazník – objeví se sémantická mezera

Nevýhody předčasné koncentrace na kód

- Soustředění se na kód vede k zanedbání výuky návrhu složitých aplikací
- Takto vychovaný student si pak neumí ani objednat program, který potřebuje



Soustředění na detail

- **Pro správný návrh kódu je důležité mít dobře navržený mentální model subjektů a objektů zpracovávané domény a jejich interakcí**
  - Chápat jejich obecné chování a specifikace jeho projekce do vyvíjeného programu
- **Programátoři vychovaní k v hladině kódu budou při vývoji programů stále přemýšlet v hladině kódu a budou mít problémy s návrhem správné architektury**
  - Mimo jiné budou mít problémy s pochopením objektových modelů nejrůznějších knihoven a frameworků, které budou používat

Zpětná vazba od firem

- Přichází řada zpětných vazeb od podniků,
  že ze škol přicházejí studenti opojení
  představou o své dokonalosti,
  ale když přejdou od školních úloh
  k těm praktickým, zjišťují,
  že je škola naučila něco jiného, než to,
  co doopravdy potřebují

Metodika Architecture First

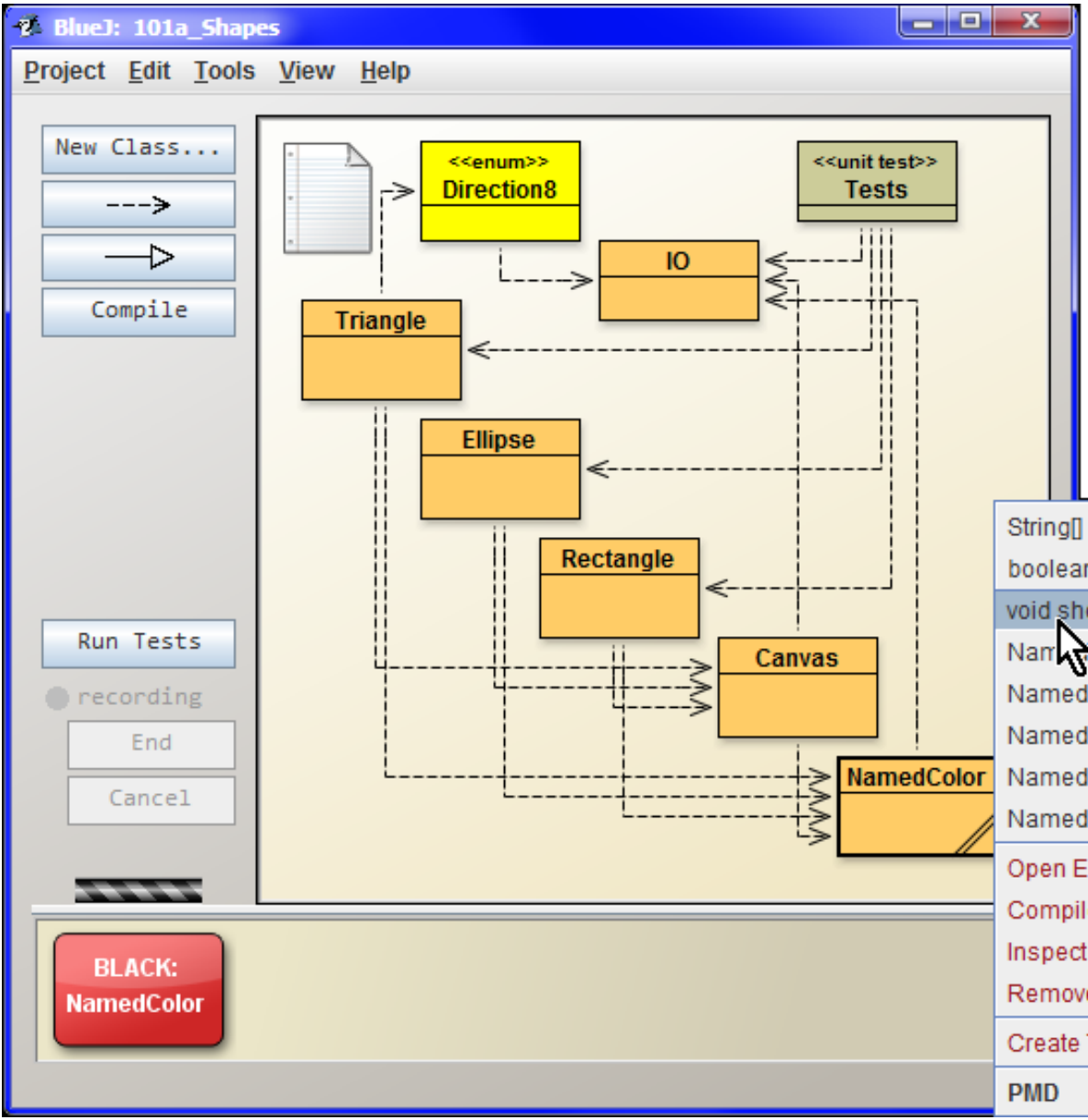
- 1. etapa – základní myšlenky
- Úvodní projekt
- Kdo má a kdo nemá problémy
- Zavedení pojmů rozhraní a interfejs
- Zdokonalení projektu
- Pokračování v interaktivním režimu
- Shrnutí interaktivního režimu
- Výhody práce v interaktivním režimu
- Pokračování pro budoucí programátory

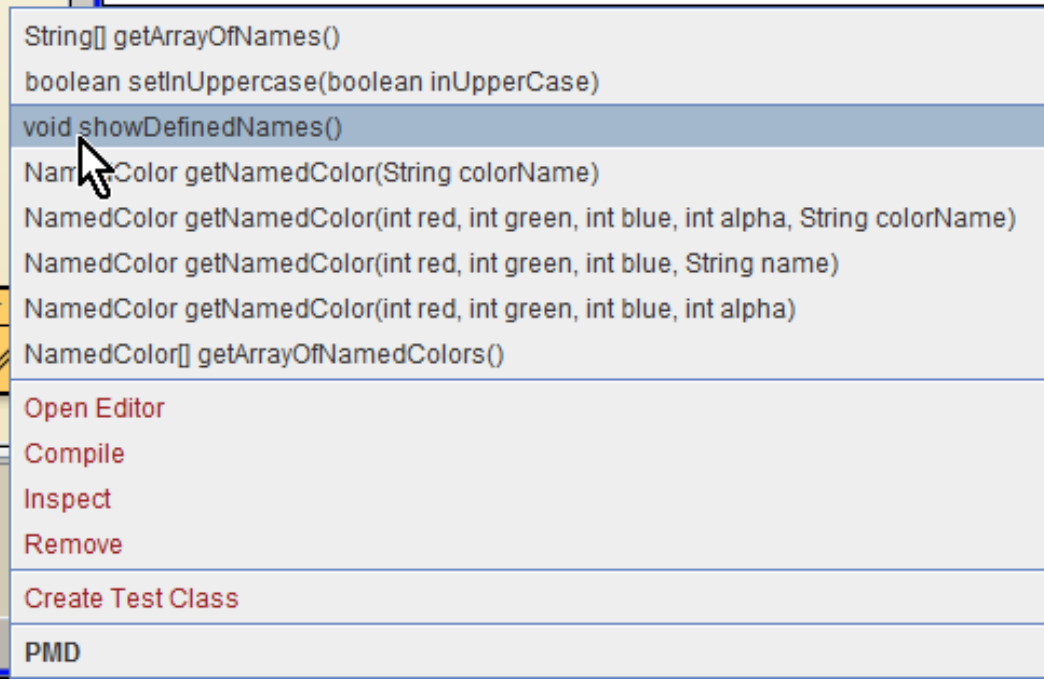
Katedra informačních technologií Kit VŠE Praha

1. etapa – základní myšlenky

- **Výuka začíná v interaktivním režimu prací s objekty a třídami**
  - Student vystupuje jako jeden z objektů programu, který si vyměňuje zprávy s ostatními objekty
- **Pracuje se pouze s diagramem tříd a objektů**
  - Studenti přistupují k aplikaci z architektonického pohledu bez jakékoliv znalosti kódu
  - Vstřebávají, že objekt je cokoliv, co lze označit podstatným jménem =>
    jako s objekty pracujeme i s abstraktními pojmy: směr, barva, krása, motivace, ...
  - I třída je objekt

Úvodní projekt





- **Třída není čirá abstrakce, ale pouze zvláštní druh objektu**

Kdo má a kdo nemá problémy

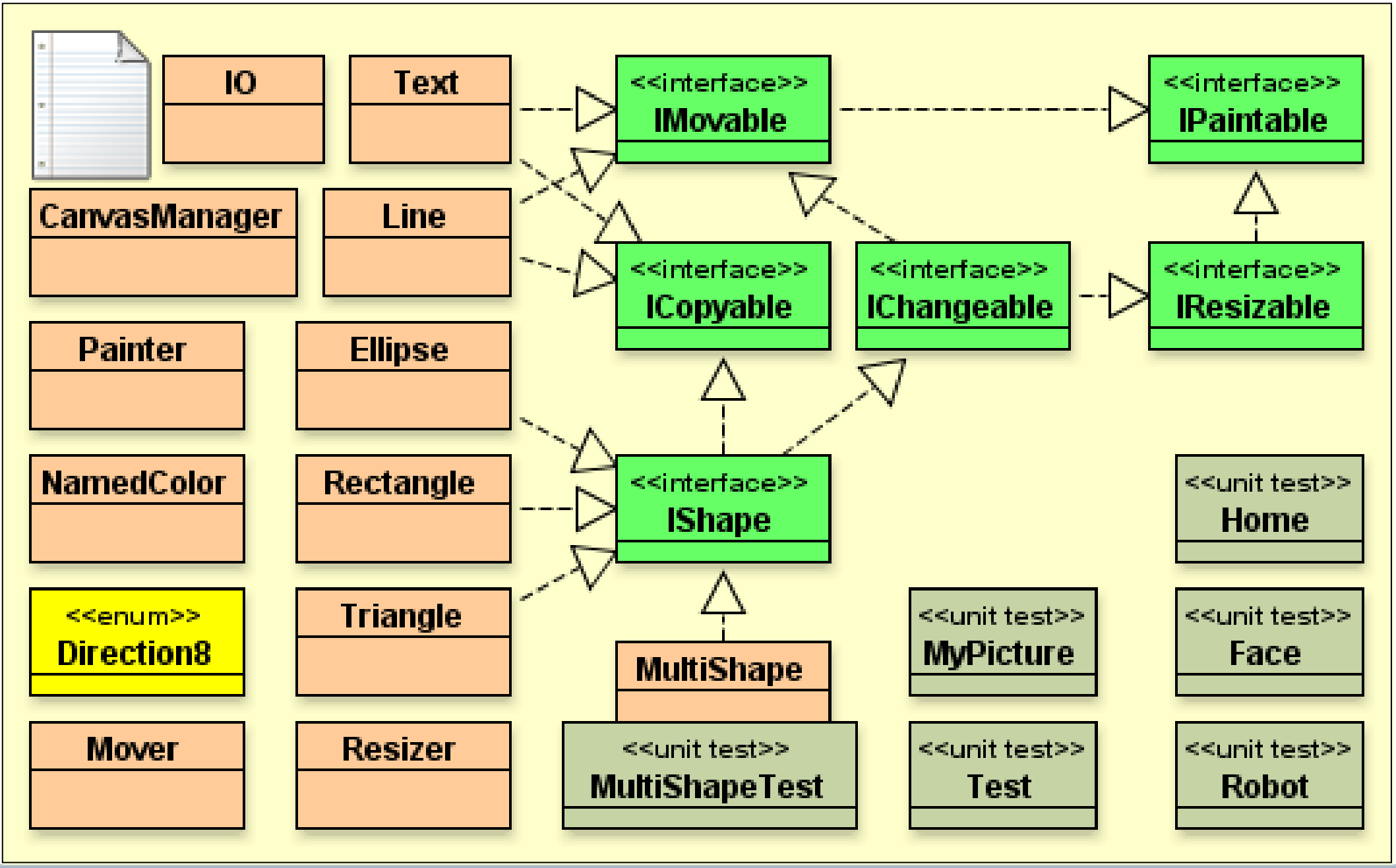
- 12leté děti ani začínající studenti
  (tj. studenti bez předchozích zkušeností
  s programováním) nemívají s prací
  v interaktivním režimu žádné problémy

- Problémy s interaktivním režimem
  zato občas mívají programátoři,
  protože se neumějí oprostit od myšlení v kódu
  a přenést se do hladiny problému
  bez znalosti toho,
  jak je navržené řešení zakódováno

Zavedení pojmů rozhraní a interfejs

- **Již v interaktivním režimu se žáci seznamují s pojmem rozhraní a konstrukcí `interface`**
  - Na VŠ v 2. cvičení, mladší o trochu později
  - Učí se její význam v programech
  - Učí se, kde a proč ji použít v návrhu
  - Včetně dědičnosti rozhraní
- **Seznamují se s návrhovými vzory**
  - Použitými v používané aplikaci
    - *Knihovní třída* , *Jedináček* , *Výčtový typ, Multiton, Jednoduchá tovární metoda*
  - Umožňujícími řešení vyskytnuvších se problémů
    - *Služebník, Prostředník, Pozorovatel*, *Inverze závislostí*

Zdokonalení projektu



Pokračování v interaktivním režimu

- **Zavádějí se funkcionální konstrukce (metoda či jiná část kódu je také objekt)**
- **Zavádějí se další návrhové vzory**
  - *Přepravka, Stav, Tovární metoda*
- **Seznamují se s dalšími programovými konstrukcemi:**
  - Deklarativní programování (datovody)
  - Dědění implementace
  - Jmenné prostory / balíčky
  - …

Shrnutí interaktivního režimu

- **Studenti nevidí konkrétní kód, pouze posílají objektům zprávy se svými požadavky a vlastní kód vytváří zabudovaný generátor kódu**

- **Každá programová konstrukce je jim vysvětlována na jako reakce na potřebu co nejvěrněji reprezentovat simulovaný svět**
  - Na počátku výuky se jim vysvětlí, že každý program je simulací reálného či virtuálního světa a programové konstrukce slouží pouze k usnadnění této simulace v konkrétním kódu

Výhody práce v interaktivním režimu 1/2

- **Nutí studenty přemýšlet v termínech řešeného problému a téměř se oprostit od požadavků použitého kódu**
  - Neučí se programovat, ale řešit problémy
- **Lze jej využít i při výuce těch, kteří se nechtějí stát programátory**
- **Můžeme záhy řešit i složitější projekty**
  - Svět kolem studentů je složitý => nemívají problém s tím, že jeho model je také složitý, jenom musí ten svět rozumně reprezentovat

Výhody práce v interaktivním režimu 2/2

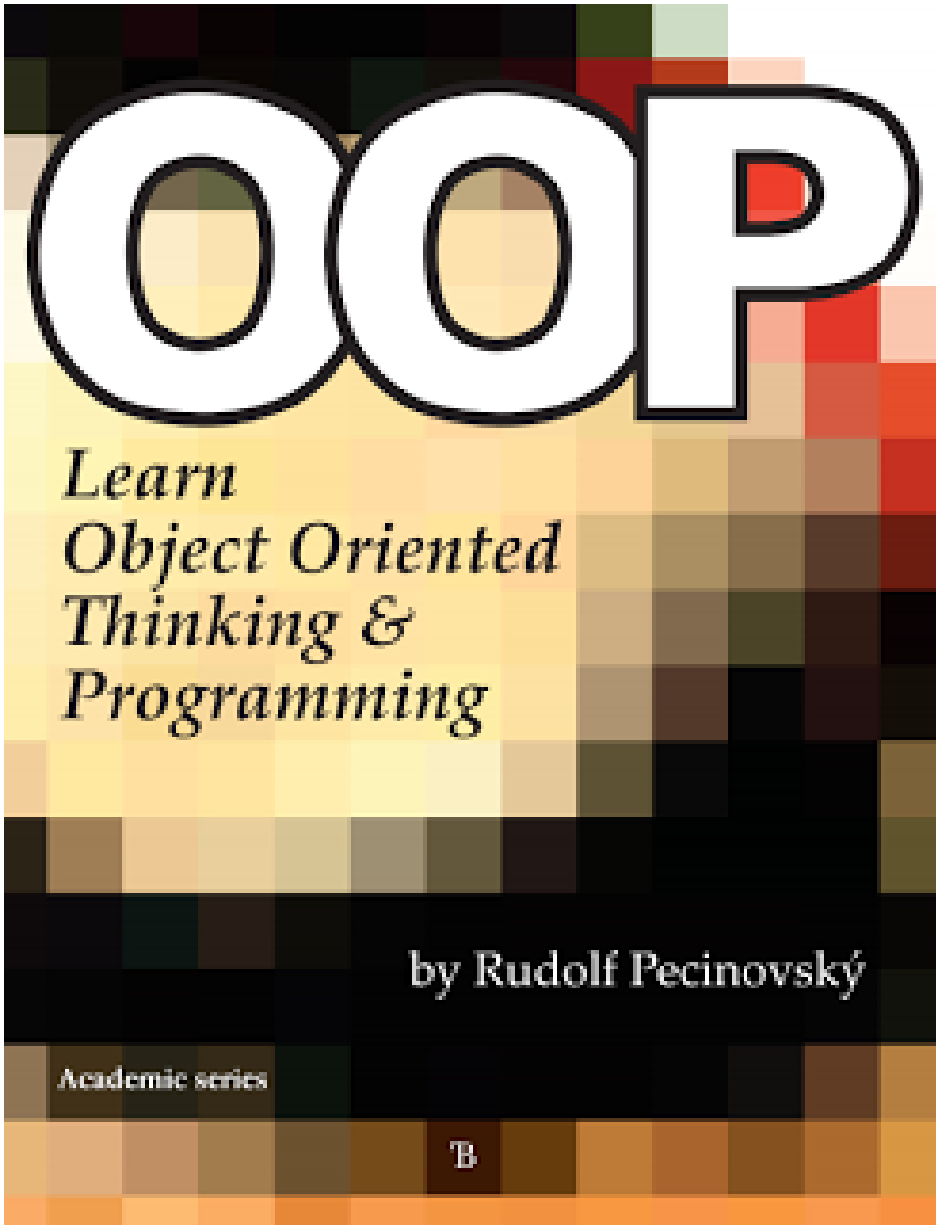
- **Téměř nezávisí na jazyku a platformě => v následující etapě můžeme pokračovat některým z široké množiny jazyků**
  - Prozatím existuje generátor kódu pouze pro Javu, ale není problém vytvořit jeho ekvivalent
  - Vytvoříme-li generátor pro některý z dynamických jazyků (Smalltalk, Groovy, ...), tak se jeho možnosti výrazně zvětší

Pokračování pro budoucí programátory

- **Práci v interaktivním režimu ukončíme, když vyčerpáme možnosti zabudovaného generátoru kódu**
- **Ve 2. etapě procházíme kód vytvořený v interaktivním režimu generátorem**
  - Učíme se syntaktická pravidla na známém kódu
  - Znovu opakujeme použité konstrukce pro jejich lepší osvojení
- **Ve 3. etapě se učíme zakódovat věci, které jsou prozatím mimo schopnosti použitého generátoru kódu**

Učebnice dostupná zdarma

- **OOP - Learn Object Oriented Thinking & Programming**
  - Eva & Tomas Bruckner Publishing © 2013
  - ISBN 978-80-904661-8-0 (paper)
  - ISBN 978-80-904661-9-7 (PDF)
  - Anglická verze učebnice OOP – Naučte se myslet a programovat objektově
- **Učebnice pro středoškoláky psaná jako rozhovor**
- **Soustředí se na to, jak program navrhnout**
  - Není to učebnice jazyka, jazyk je pouze nástroj



Děkuji za pozornost

Rudolf Pecinovský
mail: rudolf@pecinovsky.cz
ICQ: 158 156 600