-

ZKOUŠKY
a jejich písemná část

Co bude obsahem písemky

Navrhnout, jak by se úloha dala řešit.

Nemusí obsahovat žádný zdrojový kód

(i když popis (pseudo)kódem může být kratší a přesnější než slovy).

V odpovědi popište

0) upřesnění zadání, pokud je potřeba

1) postřehy

2) zdůvodněnou volbu algoritmu

3) reprezentaci dat

4) dekompozici programu

5) diskusi

VŠECHNY TYTO ČÁSTI JSOU DŮLEŽITÉ!

Příklad zadání - ANKETA

Navrhněte program, který zpracuje výsledky studentského hodnocení učitelů.

Vstup:

1) soubor obsahující na každém řádku
   učitel – předmět – hodnocení
2) seznam kateder
   učitel – katedra

Výstup:

1) soubor učitel – předmět – průměrné hodnocení – odchylka
   setříděný podle učitel – předmět
2) dtto, setříděný podle katedra – učitel – předmět

Příklad zadání - ANKETA

Velikosti a omezení:

Počet učitelů: stovky

Počet předmětů: tisíce

Počet hlasovacích lístků: desetitisíce

Velikost paměti: 1kB

Velikost disku: neomezeně

Čas: přiměřeně (minuty až hodiny)

Generické metody a třídy

- jeden a více typových parametrů
- možnost omezit pomocí where
- where:
  - interface
  - class
  - struct
  - rodičovská třída
  - ...apod.

Dekompozice, Objektový návrh, Obrázky a UML, guruové a rady starších

Proč?

Jediná jistota je, že program bude potřeba (z)měnit.

Potřebujeme psát program tak, aby bylo možné ho (snadno, často, dlouhodobě) upravovat.

Tomu pomáhá, když se rozdělí na (co nejvíce nezávislé) části (funkce, moduly, objekty), které komunikují pomocí domluveného rozhraní (interface) (a nijak jinak).

Uživatel/klient potřebuje znát jen rozhraní, všechno ostatní se může měnit.

Obrázky

UML2 = Unified Modeling Language

Domluvený systém obrázků („diagramů").

https://www.omg.org/spec/UML/

Obrázky - Diagram tříd (class diagram)

= třídy a vztahy mezi objekty a mezi třídami

Vztahy (např.):

- Asociace: Student ---studuje---> Předmět
- Agregace: Auto je částí objektu Událost (ale nepatří mu)
- Kompozice: Auto má čtyři Kola (a patří mu)
- Generalizace: Kočka je zobecněním třídy Tygr (ISA-vztah)

Diagram stavů (state machine diagram)

= stavy a přechody mezi nimi

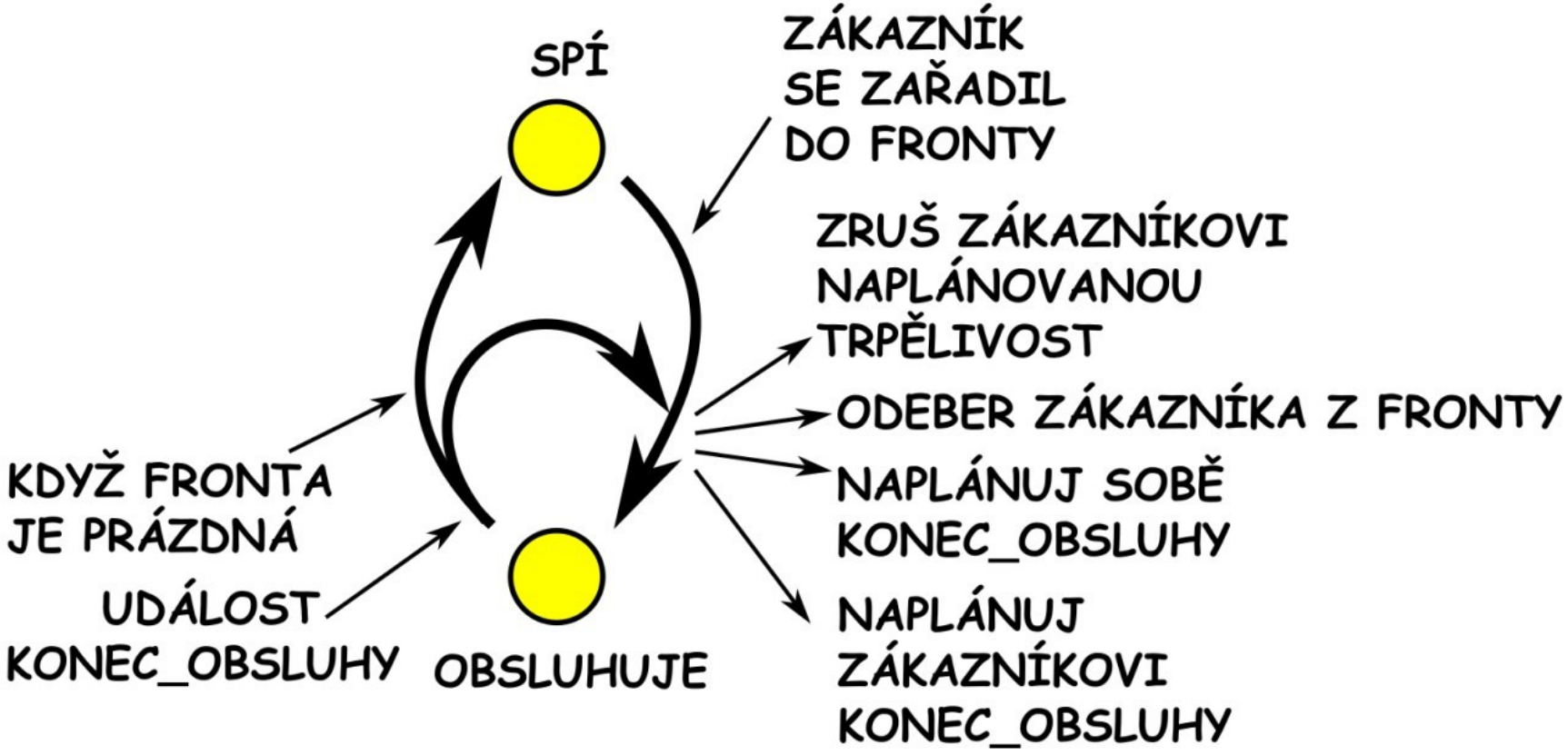


Diagram stavů (state machine diagram)

Příklady v UML:

https://www.omg.org/spec/UML/2.5.1/PDF str. 378 / 796

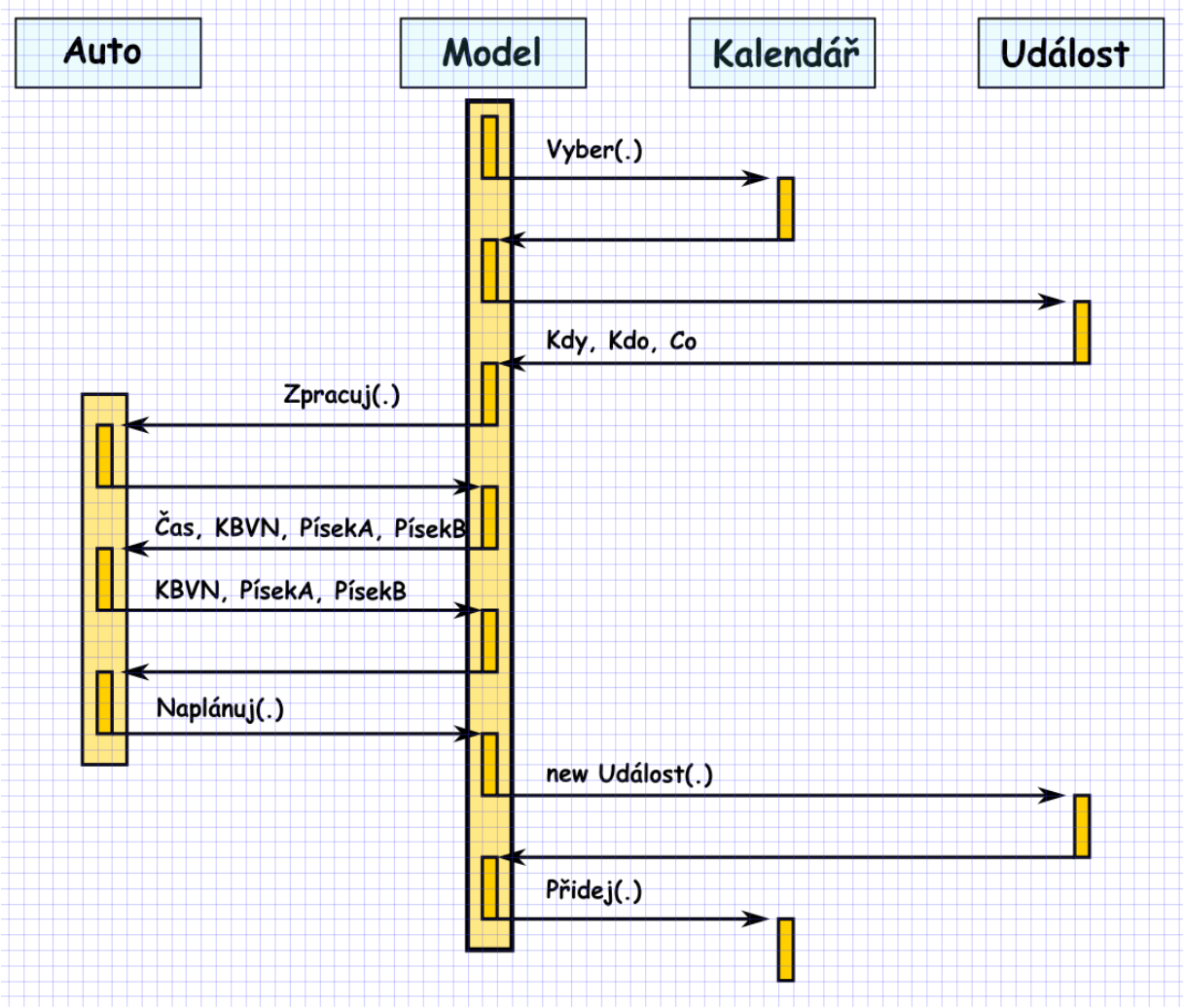
, protože

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without permission of the copyright owner.

Sekvenční diagram

= průběh spolupráce objektů, shora dolů běží čas



Pravidla (aka nevyžádané rady)

existuje řada pravidel; tady jsou některá z nich.

Princip jedné zodpovědnosti
(Single Responsibility Principle)

Jeden objekt by měl mít na starosti jednu věc.

Princip otevřenosti a zavřenosti
(Open–closed principle)

Třída by měla být otevřená pro rozšiřování (třeba tak, že odvodíme novou třídu, do které přidáme data a funkce), ale uzavřená ve smyslu, že má definitivní rozhraní, které mohou ostatní používat a které se nebude měnit.

Pravidla (aka nevyžádané rady) – 2

Zákon substituce
(Liskov substitution principle)

Pokud je někde potřeba objekt typu A, musí být místo něj možné použít i objekt typu B, pokud B je odvozený od A (neboli opravdu platí že Tygr je taky Kočka).

Princip oddělených rozhraní
(Interface segregation principle)

Klient by neměl být nucen záviset na metodách, které nepoužívá, takže místo velkého společného rozhraní je lepší mít několik oddělených malých rozhraní určených pro různé klienty (třída v C# může implementovat více různých rozhraní).

Pravidla (aka nevyžádané rady) – 3

Princip obrácení závislosti
(Dependency Inversion Principle)

Vyšší části by neměly záviset na (konkrétních) nižších částech, ale na abstrakci, protože implementace se mění často, abstrakce by měla být trvalejší. Konkrétnější musí záviset na abstraktnějším.

= pravidla „SOLID“

Pravidla (aka nevyžádané rady) – 4

Deméteřin zákon (Law of Demeter)

Funkce FFF() ve třídě TTT by měla volat pouze:

1. funkce třídy TTT
2. funkce objektů předaných jako parametry funkci FFF
3. funkce objektů vytvořených funkcí FFF
4. funkce objektů patřících třídě TTT
5. globální funkce

Protipříklad:

```
seznam.Vyber().Split()[0].ToItem().Name.ToString()
```

Pravidla (aka nevyžádané rady) – 5

DRY (Don't Repeat Yourself)

Každá informace by měla být v programu jen jednou – jediný zdroj pravdy (Single Source Of Truth). Pokud je ji potřeba ve více formách, je správné ji generovat z jednoho zdroje.

Opak se nazývá WET (Write Everything Twice!).

High cohesion, Low coupling…
Věci, které spolu souvisí, by měly být blízko = high cohesion, ale měly by být provázané jen zlehka (přes interface) = low coupling.

...a mnoho dalších pravidel.

Pravidla (aka nevyžádané rady) – 6

ALE:

cílem není dodržování pravidel,
cílem je udržovatelný kód!

```
public class Komplex
{
    public double Re { get; set; }
    public double Im { get; set; }
}
```

Návrhové vzory

Myšlenka, že v mnoha programech se opakují stejné (pod)problémy a že by bylo možné vydestilovat vzorová řešení těchto podproblémů.

Původní kniha:

Design Patterns:
Elements of Reusable Object-Oriented Software

autorů Gamma, Helm, Johnson a Vlissides,
přezdívaných „Gang of Four (GoF)".

Učí se v předmětu NPRG024 Návrhové vzory.

Zvěřinec OOP jazyků

- Python (1991)
- C++ (1985)
- Java (1995)
- C# (2000)

Simula67 (1967), Smalltalk (1971),
Object Pascal (1989), Visual Basic (1991)
JavaScript (1995), PHP (1995)...a další

Zvěřinec OOP jazyků 2

Specifika, zvláštnosti, úchylky:

- C# a Java pouze objektově
- C++ pracuje přímo s pamětí
- C#, Java, PY: garbage collector
- PY nemá statické typování
- PY nemá private/protected
- PY, Java ani C++ nemají properties
- ...

Zvěřinec OOP jazyků 3

Specifika, zvláštnosti, úchylky:

- Java nemá var-parametry
- Java class = stejnojmenný soubor
- ...

Všechny jazyky se vyvíjejí a přebírají to, co se jinde osvědčilo.

Hygiena programování

(praktické tipy)

1. Oddělovat různé činnosti

1. analýza
2. design
3. GUI design
4. kódování
5. texty, grafika, zvuky
6. testování (psaní testů)
7. ladění
8. dokumentace

   resources, šablony výstupů…

2. Pracovní deník

1. ujasnění si, CO budu dělat teď
2. rozhovor (kačenka)
3. záznam pro soustředění na jednu věc
   (viz ToDo-list)
4. pozdější dohledání (...a viz verse)
5. sledování, kolik času mi co trvá

3. ToDo-list, bug-list, issue tracker

* //TODO ve zdrojovém kódu
* deník, konec nebo vyhledatelná značka
* samostaný soubor
* software (GitHub, Trello, GitLab)

4. Správa versí

* CSV, SVN (Subversion),
  Hg (Mercurial), Git
* možnost návratu k minulým versím
* sledování rozdílů

I při práci na jednom počítači.
I pro ne-programátorské projekty.

5. Automatické testy, TDD

* unit-testy
    1. Napsat testy
    2. Spustit testy a ujistit se, že selžou
    3. Napsat kód
    4. Ladit kód, dokud nesplní testy
    5. Refaktorovat kód
    6. Opakovat
* testování funcionalit
* výkonnostní testy
* regresní testy i dnes funguje, co fungovalo včera
* Průběžná integrace (CI)