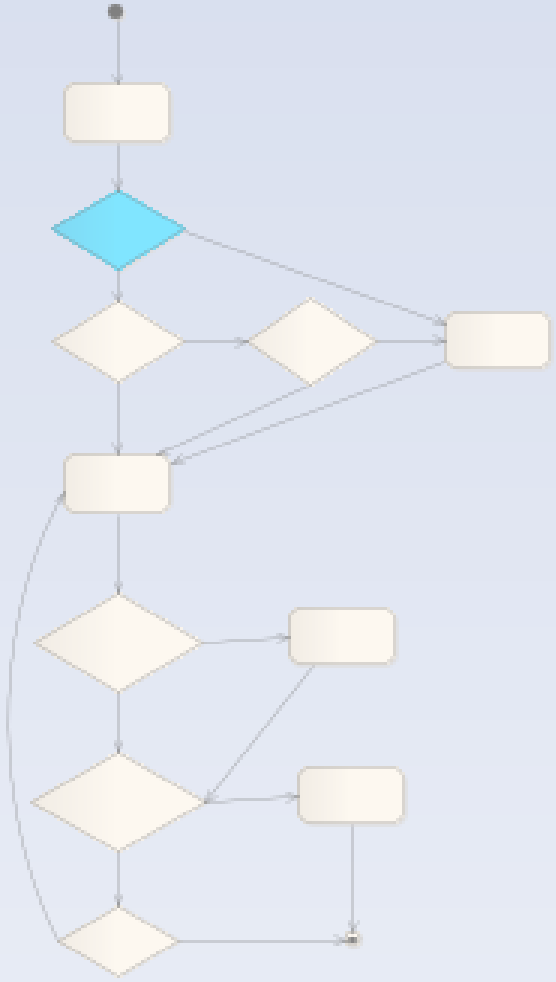
Principy OOP při tvorbě aplikací v JEE



Michal Čejchan

Témata přednášky

- Principy OOP - připomenutí
- Úvod - co nás vede k používání OOP
- Reálný svět - jak (ne)používáme OOP
- Nedostatky na úrovni programovacích jazyků
- Způsob návrhu aplikace v objektovém jazyce
- Java EE aplikace

Základní principy OOP (připomenutí)

- Encapsulation
- Polymorphism
- Abstraction - understandability
- Inheritance
- Modularization - loose coupling
- Composition - structured design

Co nás vede k používání OOP?

- Efektivita programování
- Zábava
- Přehlednější a hezčí kód
- Lépe udržovatelný kód
- Nevzniká tolik duplicit

Nevyužívání OOP

- Praxe: převážná většina aplikací je psána pomocí objektových jazyků, ale velké procento kódu je napsáno neobjektově. Proč?
- Programátoři používají objektové jazyky, ale programují neobjektově.
- Důvody:
  - závislost na technologii
  - závislost na frameworku
  - závislost na standardizovaných firemních postupech
  - použití návrhových vzorů
  - programátor z nějakého důvodu nechce použít objektový přístup
  - nedostatečná znalost OOP
  - jazyk sám o sobě má nedostatky

Nedostatky na úrovni programovacích jazyků

- Ačkoliv jsou principy OOP obecně uznávány za platné, programovací jazyky se liší v jejich interpretaci
- Objective C - chybí privátní metody
- Java - Nemožnost rozšířit některý ze základních objektů jako String o další metody.

  Např. chceme přidat metodu leftZeros ("34", 4) -> "0034"
- Konstrukty jako anotace nebo používání reflexe řeší problémy, které přesahují možnosti syntaxe Java.
- Zavádění neobjektových syntax jako např. “method reference” nebo “closures”

```
{ String => int } parseInt = Integer#parseInt(String);

int x = parseInt.invoke("42");
```

Dva pohledy na návrh aplikace

- Data Driven Design
- Responsibility Driven Design

Data Driven Design

- Data aplikace jsou hlavním prvkem, okolo který je opřený celý návrh.
- Analýza zpravidla začíná návrhem databázové struktury.
- Vrstvy + data: data protékají aplikací a přetvářejí se do požadovaného výstupu
- Vede na procedurální návrh
- Obecně je považován za postup, který vede na špatný objektový návrh na rozdíl od Responsibility Driven Design

Responsibility Driven Design

- někdy také zaměňovaný s Domain Driven Design (širší pojem)
- vede na OOP návrh
- inspirovaný client-server modelem
- zaměřuje se na kontrakt mezi objekty zodpovězením otázek:
- Jaké akce má tento objekt na starosti
- Jaké informace objekt sdílí svému okolí
- v idealizované formě: zapouzdření doménových objektů, které se starají samy o sebe včetně perzistence atd.

Realita v Java EE

- Aplikace v objektovém jazyce: posílání zpráv mezi objekty
- Aplikace v Java EE: čtení atributů objektů (objekty reprezentující entity nebo neperzistentní data), práce s daty je soustředěna v objektech typu manager, facade, controller, helper atd.
- Aplikace není designována v OOP, ale v Java EE

Realita v Java EE

- EJB: Stateful vs. Stateless beany
  - Stateless EJB si neudržují žádný vnitřní stav, což je základní předpoklad pro objektové programování.
  - Použití stateful EJB má své limity vyplývající z použité technologie; jejich použití se proto snažíme omezit nebo se ho vyvarovat.
  - Řešením je používat EJB pouze jako vstupní facade, která není zahrnuta v objektovém modelu.
- Propojení modulů přes RMI/WS technologie, kde jsou požadavky na coarse-grained rozhraní.
  - stejně jako v předchozím případě - používat jenom jako facade
- Používání UML pro modelování aplikace má vliv na výslednou podobu programu.

Návrhové vzory

- Příklad neobjektových návrhových vzorů
  - Facade
  - Data Transfer Object
- Příklad objektových návrhových vzorů
  - Command
  - Template Method

Data Transfer Object

- Návrhový vzor, který odděluje data od business logiky
- Vznikají objekty ekvivalentní pascalovskému typu “record”
- Následně se objevuje kód typu:

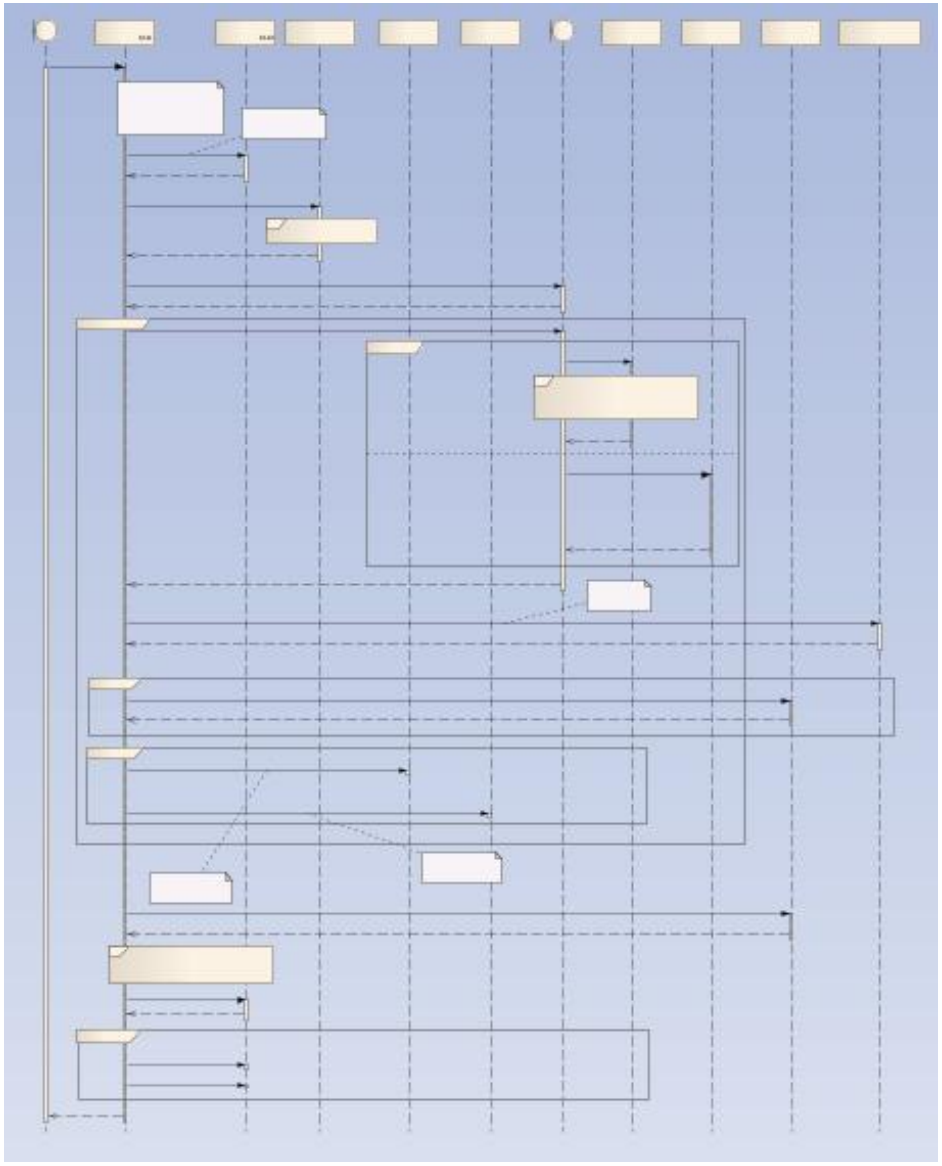
```
FooObject.Property.Item(Index).Method(Parameter 1, Parameter2).Property =
    BarObject.Method(Parameter 1).Property.ToSomeObjectType()
```

- Pro použití vyžaduje rozsáhlou znalost vnitřní funkcionality
- Nefunguje zde zapouzdření. Objekt poskytuje veškeré atributy navenek.

Objektový model ovlivněný funkční specifikací

- Programátor je ovlivněn formou zadání,
- Analýza může popisovat systém na jiném stupni abstrakce, např. pomocí use-case nebo sekvenčních diagramů. Tato část analýzy nic neříká o objektovém modelu, který by měl vzniknout.

Zadání ve formě use-case nebo sekvenčního diagramu



Modelování v UML

- UML neposkytuje dostatečné nástroje pro použití návrhových vzorů.
- Pokud analytik navrhuje použití návrhového vzoru, musí to sdělit jinak.
- Modelování jednotlivých tříd a jejich vztahů je časově náročné a model není snadno udržovatelný.
- Proto popisujeme funkcionalitu až na úrovni větších celků, které si programátor rozdělí do tříd.
- Vyšší požadavky na “kázeň” programátora

Shrnutí – OOP v praxi

- Pokud chcete používat OOP, je dobré si uvědomit omezení, která přináší různé technologie.
- Pokaždé, když budete přidávat novou metodu nebo atribut do třídy, položte si otázku: Patří to sem? Jaká je zodpovědnost této třídy? Jestliže nedokážete zodpovědět jednoduše, je něco špatně.
- Pište dokumentaci k třídám. Zdokumentovaná třída má jasně vymezenou zodpovědnost a je snazší rozhodnout, co do ní opravdu patří.
- Není samozřejmostí, že použití objektového jazyka vede na výslednou aplikaci v OOP.

Diskuse