GRASP - Příklad

Přednáška 9, LS 2013/2014

Modelový příklad e-shop

- **Register** – představuje celý sub-systém evidence objednávek a plateb, udržuje přehled všech objednávek

- **ProductCatalog** – katalog produktů obsahující informace o všech prodávaných produktech

- **ProductSpecification** – obsahuje informace o daném typu produktu jako je například název, katalogové číslo a cena

- **Sale** – konkrétní objednávka (obsahuje seznam položek objednávky, její stav a případně odkaz na přijatou platbu za tuto objednávku)

Modelový příklad e-shop

- **SaleLineItem** – položka objednávky (počet kusů daného typu produktu)
- **Payment** – příchozí platba (přijatá částka)
- **PaymentsHistory** – evidence příchozích plateb

Modelový příklad e-shop

- Předchozí dva slides popisovaly současný stav systému.
- Ukážeme si jak pomocí principů **GRASP** přiřadit další odpovědnosti těmto třídám.

Vytvoření nové objednávky

- Vytvoření instance třídy **Sale** představující objednávku.
- Zařazení instance **Sale** do evidence v **Register.**
- Atributy instance **Sale** musí být inicializovány na správné hodnoty.
- Prvním krokem bude určení toho, který objekt bude pro událost vytvoření objednávky fungovat jako **controller** – tedy **objekt, který přijme prvotní zprávu o požadavku na provedení dané činnosti.**

Vytvoření nové objednávky

- Přijetí impulsu - zprávy = GRASP princip *Controller*.
- Podle GRASP je nejvhodnějším kandidátem objekt, který představuje celý sub-systém v našem případě tedy objekt **Register**.
- **Register** v tomto případě slouží jako **Facade** a podporuje princip *Indirection*.

Vytvoření nové objednávky

- Dalším krokem je určení toho, kdo bude zodpovědný za **vytvoření instance Sale.**

- Vytváření instancí = princip *Creator*.

- Princip *Creator* doporučuje přiřadit zodpovědnost za vytváření instancí třídě, která agreguje nebo zaznamenává vytvářené objekty.

- Princip *Creator* také doporučuje, aby instance vytvářel objekt, který k tomu má všechny dostupné informace (*Information expert*).

- **Register** přijímá požadavek, agreguje instance **Sale** a má všechny dostupné informace.

Vytvoření nové objednávky

- Dalším krokem je zařazení instance **Sale** do evidence.

- Zařazení instance **Sale** do evidence provede objekt **Register** sám hned po jejím vytvoření.

Vytvoření nové objednávky

- Posledním krokem je přiřadit zodpovědnost za inicializaci atributů nově vytvořené objednávky.

- Inicializace bude mimo jiné obsahovat vytvoření prázdné kolekce položek objednávky.

- Uplatníme princip *Creator* a tuto zodpovědnost přiřadíme třídě, která tuto kolekci obsahuje – tedy samotné třídě **Sale**.

- Ostatní parametry objednávky jako například její číslo a podobně by pak dle principu *Information expert* měl nastavit objekt, který k tomu má dostupné informace.

- V našem případě je to objekt **Register**.

Přidání položek objednávky

- Vytvoření instance třídy **SaleLineItem** představující jednu položku objednávky.

- Přidání instance **SaleLineItem** do příslušné instance Sale.

- Atributy instance **SaleLineItem** musí být inicializovány na správné hodnoty (počet a reference na **ProductSpecification**)

Přidání položek objednávky

- Opět musíme určit *controller* – tedy bod, kde náš subsystém obdrží instrukce k přidání položky do objednávky, kterým bude opět objekt **Register**.

- Zodpovědnost za vytvoření instance **SaleLineItem** přiřadíme podle principu *Creator* třídě, která tyto položky agreguje – tedy třídě **Sale**.

- Třída **Sale** zároveň provede i přidání nové instance **SaleLineItem** do svého seznamu položek objednávky.

Přidání položek objednávky

- Přiřazení zodpovědnost za nastavení atributů **SaleLineItem**.
- *Controller* je **Register**, ale instance se vytváří v **Sale.**
- Proto je třeba, aby při volání příslušné metody třídy **Sale** objekt **Register** předal informaci o počtu kusů.
- Pro inicializaci **SaleLineItem** musíme získat referenci na **ProductSpecification**.
- **Register** obdrží pouze informaci o katalogovém čísle produktu, který má být přidán.
- Musíme tedy podle tohoto katalogového čísla získat příslušnou instanci obsahující dodatečné informace (například cenu) - **nová zodpovědnost**.

Přidání položek objednávky

- Zodpovědnost za získání instance **ProductSpecification** dle katalogového čísla přiřadíme podle principu *Information expert.*

- Potřebné informace o všech typech produktů má v našem příkladu objekt **ProductCatalog** – bude to tedy jeho zodpovědnost.

Přidání položek objednávky

- Na přidání položky do objednávky se podílejí dva objekty, **Register** a **Sale**. Který z nich by však měl být zodpovědný za spolupráci s **ProductCatalog**?

- Aby mohl některý z těchto objektů požádat **ProductCatalog** o nalezení instance **ProductSpecification**, musí pro něj být *viditelný*.

- Musíme tedy rozhodnout, kterému objektu tuto *viditelnost* poskytneme a v jaké podobě.

Přidání položek objednávky

- Aplikujeme další hledisko - Low-Coupling.
- Z hlediska Low-Coupling se jako nejvhodnější kandidát jeví **Register.**
- V případě zvolení **Sale** bychom totiž museli zajistit, aby pro **Sale** byl **ProductCatalog** *viditelný*.
- Lze zajistit nastavením reference na **ProductCatalog** do každé **Sale**, což by nutně musel provádět **Register.**
- Protože v obou případech by vzniklo provázání **Register-ProductCatalog**, je vhodnější takové řešení, kde nevznikne zbytečné provázání **Sale-ProductCatalog**.

Výpočet celkové ceny objednávky

- Kdo by měl mít zodpovědnost za vrácení celkové ceny objednávky?

- Podle *Information expert* by to měla být **Sale** sama.

- Toto rozhodnutí podporuje i *Low-Coupling* – vnitřní objekty **SaleLineItem** není nutné předávat nikam ven a tím zvyšovat provázání.

- **Sale** svoji celkovou cenu spočítá jako součet celkových cen jednotlivých **SaleLineItem**.

Výpočet celkové ceny objednávky

- Odlišné řešení nám však doporučuje princip *High cohesion*.

- Pokud výpočet celkové ceny přidáme do třídy **Sale**, bude tato třída dělat více věcí a bude tak mít menší soudržnost.

- Řešením by tedy bylo výpočet ceny oddělit do samostatné třídy např. **PriceCalculation** představující *Pure fabrication*, která by řešila pouze výpočet.

- Musíme tedy zvážit, zda snížení soudržnosti třídy **Sale** je dostatečným důvodem, proč zvýšit provázání a zavést *Pure fabrication* třídu.

Výpočet celkové ceny objednávky

- Obdobně jako u **Sale** i **SaleLineItem** by měla umět svoji celkovou cenu spočítat sama, protože je pro tento úkol *Information expert*.

- Pro výpočet stačí prostě provést součin počtu kusů a ceny za jeden kus uvedené v **ProductSpecification**, který **SaleLineItem** *vidí* ve svém atributu.

- Toto uspořádání podporují i principy *Protected variations* a *Polymorphism*.

- Pokud by vznikly případy, kdy je cena položky počítána jinak – například množstevní sleva.

- Stačilo by vytvořit podtřídu **SaleLineItem** obsahující upravený algoritmus výpočtu. Takováto změna by se **jiných částí systému nedotkla.**

Přijetí platby za objednávku

- Náš subsystém obdrží informaci o přijetí platby za objednávku s daným číslem a zaplacenou částku.

- Zpracování platby pak bude zahrnovat následující kroky
  - Vytvoření instance **Payment**
  - Nastavení instance **Payment** na hodnotu přijaté částky
  - Přiřazení instance **Payment** k příslušné instanci **Sale**
  - Ověření zda přijatá částka odpovídá ceně objednávky
  - Uložení instance **Payment** do **PaymentsHistory**

Přijetí platby za objednávku

- Opět bude jako *controller*, který obdrží informaci o provedené platbě, fungovat objekt **Register**.

- Prvním krokem je vytvoření instance třídy **Payment**.

- Podle principu *Creator* máme dva kandidáty. Prvním je objekt **Sale**, který bude **Payment** obsahovat. Druhým je objekt **Register**, který má inicializační data (přijatou částku).

- Mezi těmito dvěma možnostmi se tedy musíme rozhodnout dle dalších principů. Musíme zajistit, aby zvolená možnost co nejlépe splňovala *Low coupling*, *High cohesion* a *Protected variations*.

Přijetí platby za objednávku

- Pokud se objekt **Payment** bude vytvářet v **Sale**, objekt **Register** o **Payment** vůbec nemusí vědět, což podporuje *Low coupling*. **Sale** o **Payment** musí vědět v každém případě, protože jej obsahuje.

- Instanci **Payment**, ale musíme také uložit do **PaymentsHistory**. Musíme tedy vyřešit i otázku, kdo vytvořený objekt **Payment** do evidence uloží.

- Opět se zde potýkáme s problémem *viditelnosti*. Pokud by instance **Payment** byla vytvářena v **Sale** musela by **Sale** mít viditelnost na **PaymentsHistory**.

- Vzniklo by tedy provázání **Sale-PaymentsHistory**. Vhodnější by tedy bylo pokud by **Payment** do **PaymentsHistory** uložil **Register**.

- **Register** by s **PaymentsHistory** byl provázán v každém případě, protože by musel zajistit předání viditelnosti na **PaymentsHistory** do **Sale** po jejím vytvoření.

Přijetí platby za objednávku

- Vznikne nám tedy buď provázání **Sale-PaymentsHistory** (pokud bude instanci **Payment** vytvářet **Sale**) nebo **Register-Payment** (pokud instanci **Payment** vytvoří **Register**).

- Dle *Protected variations* se jeví jako menší zlo provázání **Register-Payment**.

- Pokud bychom totiž chtěli do budoucna přidat další objekty, které potřebují z nějakého důvodu obdržet **Payment**, bylo by je vždy nutné provázat jak s **Register**, tak se **Sale**.

Přijetí platby za objednávku

- Poslední krok a tím je ověření **zda přijatá částka odpovídá ceně objednávky.**

- Princip *Information expert* říká, že zpracování by měla provést třída, která k tomu má dostupné informace.

- Zde je však nutné porovnat hodnotu uloženou v **Payment** (přijatá částka) s celkovou cenou objednávky, kterou zná **Sale**.

- Která z těchto dvou tříd by tedy toto ověření měla provést?

Přijetí platby za objednávku

- *Poslední krok a tím je ověření* ***zda přijatá částka odpovídá ceně objednávky.***

- Odpověď nám dá princip *Low coupling*.

- Třída **Sale** již nyní má viditelnost na **Payment** a je s ní tedy provázána.

- Naopak to ale neplatí a třída **Payment** je na **Sale** nezávislá.

- Pokud by ověření měla provést **Payment** bylo by nutné zajistit, aby měla viditelnost na **Sale**, a tím přidat další provázání.

Literatura

- Převzato ze seriálu: http://www.zdrojak.cz/serialy/principy-objektove-orientovaneho-navrhu/

- Robert C. Martin: Čistý kód, Computer Press a.s., 2009, ISBN-978–80–251–2285–3