

Úvod do objektové dekompozice

Petr Adámek

Obsah přednášky

Úvod do objektové dekompozice

- Organizace předmětu
- Osnova předmětu
- Nejdůležitější pravidlo
- Rekapitulace ostatních důležitých zásad
- Základní principy objektové dekompozice

Organizace a osnova předmětu

Organizace předmětu

- <https://is.muni.cz/.../op/OrganizacePredmetu.html>

Osnova předmětu

- <http://is.muni.cz/el/1433/jaro2012/PV168/index.qwarp>

Nejdůležitější pravidlo

KISS

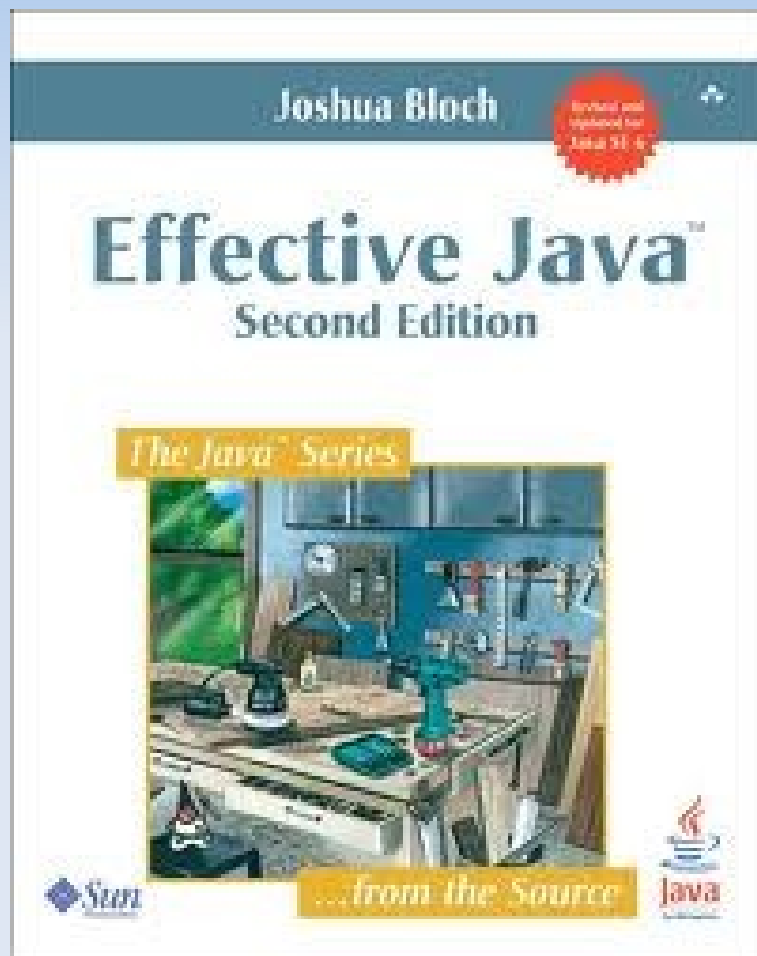


Rekapitulace důležitých zásad

Viz stránka na kore wiki:

- http://kore.fi.muni.cz/.../Jak_vyvijet_programy_v_Javě

Zdroje



Effective Java (2nd Edition)

Joshua Bloch

<http://amazon.com/dp/0321356683/>

Objektová dekompozice

Dekompozice je rozklad problému na dílčí podproblémy, které se pak řeší samostatně. V případě programování se jedná zejména o rozdělení programu do modulů, objektů a metod.

Dobrá dekompozice je důležitou součástí dobrého návrhu programu.

Jak se pozná dobrá dekompozice

Dobře provedená dekompozice:

- Je jednoduchá (vyhýbáme se zbytečně komplikovaným návrhům a neřešíme problémy, které řešit nemusíme).
- Každá komponenta (třída nebo metoda) je co nejjednodušší (ale aby dávala smysl).
- Mezi komponentami je co nejméně závislostí.
- Neobsahuje duplicitní kód
- Využívá výhody zapouzdření (skrývání složitosti).
- Používá správně definovanou hierarchii výjimek.

Jak se pozná dobrá dekompozice

Další tipy pro dekompozici

- Pokud je to možné, používejte neměnitelné třídy.
- Pro definici typů, služeb a potenciálně vyměnitelných nebo obecných komponent používejte rozhraní.
- Mezi komponentami je co nejméně závislostí.
- Omezte použití dědičnosti (často je vhodnější kombinace kompozice nebo agregace a delegování).
- Klíčem je využití existujících komponent (knihovny, Java Core API, jakarta-commons, frameworky, apod.)

Jak na dekompozici

Snažíme se identifikovat jednotlivé části problému, které je vhodné modelovat pomocí tříd. Obvykle narazíme na:

- třídy s charakterem entit (Student, Car, Course, apod.);
- třídy reprezentující operace či sady operací (StudentCatalog, QueryParser, StorageManager).

Entita je třída reprezentující nějaký objekt, který existuje v problémové domény. Objekt může být konkrétní (auto, osoba, faktura) i abstraktní (úkol, dovednost, předmět).

Každá entita by měla mít (pouze):

- ID (primární klíč);
- příslušné atributy a k nim get/set metody;
- bezparametrický konstruktor;
- metody equals() a hashCode();
- vhodná bývá i metoda toString(), případně compareTo().

Entity - důležité zásady

Při vytváření dodržujeme následující zásady:

- ID (primární klíč) by mělo být syntetické, nevolíme atributy z problémové domény (např. rodné číslo, RZ).
- Až na výjimky nevytváříme jiný než bezparametrický konstruktor.
- Entita slouží jako schránka na data, nikdy nesmí obsahovat aplikační logiku.

Příklad entity

Person

Závislosti mezi komponentami

Třída A závisí na třídě B pokud:

- Metody třídy A používají třídu nebo rozhraní B
- Třída A rozšiřuje třídu B
- Třída A implementuje třídu B

Potom třídu A nelze použít bez třídy B. Podobně to funguje i na ostatních úrovních (metody, komponenty, moduly, apod.)

Problémy se závislostmi

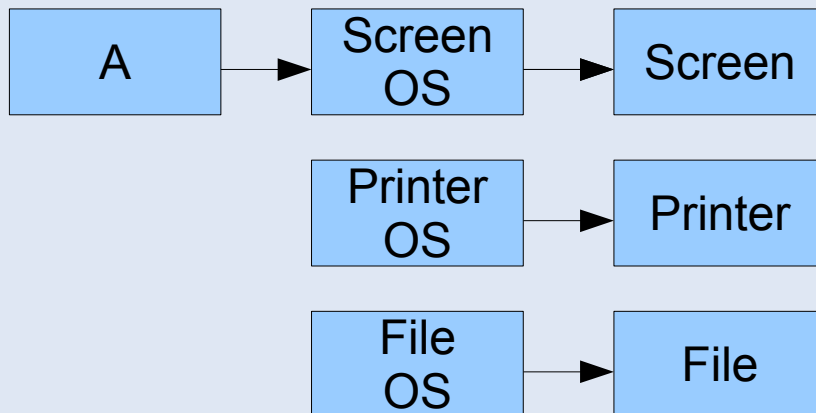
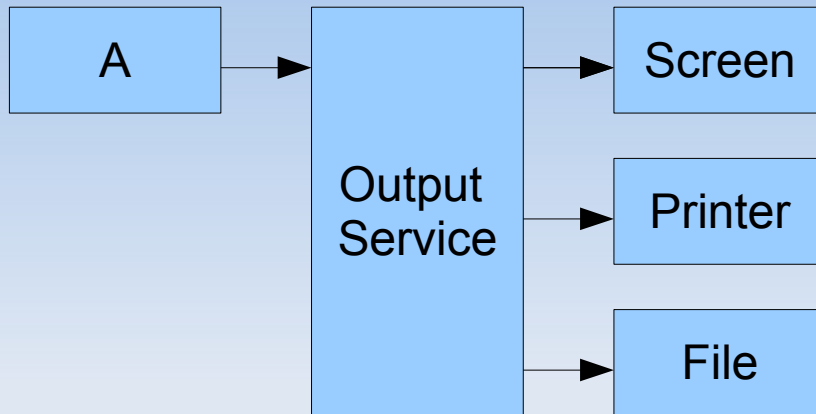
Závislosti:

- komplikují údržbu kódu (změny jsou složité);
- brání znovupoužitelnosti kódu;
- indikují chybu v dekompozici (pokud je jich moc).

Zásadní chybou jsou cyklické závislosti:

- nikdy se nesmí objevit;
- vždy jsou důsledkem špatné dekompozice;
- vždy se dají odstranit.

Příklad problematických závislostí



Refaktoring

Refaktoring je změna struktury kódu, aniž by došlo ke změně funkčnosti.

Málokdy se podaří vše navrhnout správně hned napoprvé, případně může dojít v průběhu vývoje ke změně či upřesnění požadavků. Proto je občas potřeba strukturu kódu změnit.

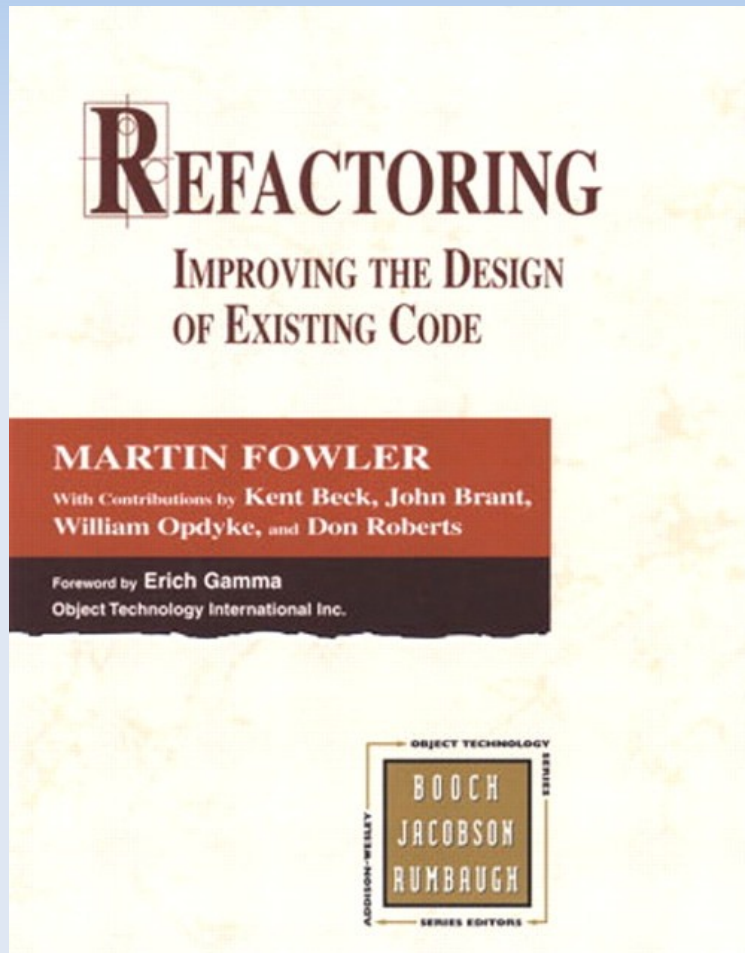
Řada agilních metodik (TDD, XP, apod.) používá refaktoring jako jeden ze základních nástrojů.

Refaktoring – jak na to

Při refaktoringu nikdy nepřidáváme novou funkcionalitu, pouze měníme strukturu kódu.

Klíčem k úspěchu jsou jednotkové testy

Zdroje



Refactoring: Improving the Design of Existing Code

Martin Fowler, Kent Beck,
John Brant, William Opdyke,
Don Roberts

<http://amazon.com/dp/0201485672/>

Otázky?