Objektový návrh v Javě

Obsah

Zásady objektového návrhu

Zásady I. - Srozumitelnost

- Základní biblí každého programátora v Javě je kniha *Java efektivně, aneb 57 zásad softwarového experta*.
- Primárním cílem je vždy **jednoduchý a přehledný kód**. Jednoduchost a přehlednost by většinou měla mít přednost před efektivitou.
- Dodržujeme **konvence pro zápis kódu** - letitým, ale stále platným dokumentem jsou Code Conventions for the Java(TM) Programming Language [http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html].
- Volíme **srozumitelné identifikátory**, snažíme se dodržet zavedené konvence - čím častěji a více lidmi je identifikátor používán, tím musí být zřetelnější, co představuje.
- Snažíme se psát kód tak, aby nebyly potřeba komentáře.

Zásady II. - Znovupoužití kódu

- Vyhýbáme se **duplicitnímu kódu**. Redundance je zlo, které vede k horší přehlednosti a udržovatelnosti kódu.
- V maximální možné míře **využíváme knihovny** (zejména Java Core API), nevynalézáme znovu kolo. Totéž - kvalita nebo přinejmenším spolehlivost známých knihoven bývá vyšší, než jsme v rychlosti schopni napodobit sami. Čas věnovaný vyhledání a zvládnutí existující knihovny se oproti psaní vlastního kódu bohatě vrátí.

Zásady III. - Zapouzdření

- Důsledně uplatňujeme princip **zapouzdření**. Pozor na nechtěné porušení tohoto principu zpřístupněním vnitřních modifikovatelných instancí - např. *vracením hodnot atributů jako modifikovatelných kolekcí!*
- Třídy **nemodifikovatelných** objektů (immutable, jako např. String, Integer, Character) jsou bezpečné a vhodné - např. i proto, že objekty jsou bez rizika znovupoužitelné a to i souběžně z více vláken.

- Řetězce vždy porovnáváme vždy metodou **equals**. *Porovnání na identitu (==) se nehodí skoro nikdy.*
- Když překrýváme metodu **equals**, nesmíme zapomenout překrýt **hashCode**. *Děláme to i v případě, že objekty (my sami) nebudeme vkládat do hašovacích struktur (HashMap, HashSet)!*

Zásady IV. - Objektový návrh, dědění

- Dědičnost používáme s mírou a pouze tehdy, jde-li skutečně o vztah **generalizace-specializace**, tedy nikoli jen jako pomůcku pro programátora na úsporu psaní kódu. Pozor na občas se vyskytující chybné příklady jako `Point` a jeho podtřída `Circle`, kde se u kruhu sice využije bod jako jeho střed, ale rozhodně konceptuálně neplatí, že by kruh byl speciálním případem bodu (spíše snad naopak, ale i to je zbytečné modelovat děděním).
- V mnoha případech se požadovaného výsledku dá dosáhnout flexibilnějšími metodami (**agregace, delegování**, vhodné návrhové vzory, oddělení funkcionality do jiných tříd).

Zásady V. - Výjimky

- Výjimky slouží **pouze k ošetření chyb** či jiných nestardardních situací a nikdy by se neměly používat jako běžný prostředek pro řízení toku programu.
- Výjimka by měla mít vhodnou **chybovou hlášku** a současně nést i přesnou strojově zpracovatelnou informaci, co se stalo (např. chybný parametr, který výjimku způsobil, by měl být součástí objektu výjimky - ovšem pozor v případech, kdy se objekty výjimek - často aniž bychom si to uvědomili - někde "pamatují", např. v záznamu (logu) o činnosti programu.
- Pokud výjimka vznikla jako reakce na jinou výjimku, měla by také mít správně nastavenou svoji **příčinu** (cause).
- Výjimka by měla být vhodným způsobem **ošetřena**. Naprostým minimem je `printStackTrace`.

Zásady VI. - Kontrakty

- Jako typ proměnných, atributů, parametrů a návratových hodnot metod se snažíme používat co **nejobecnější** typy, nejlépe rozhraní. Vzpoměňme přitom na jiné situace v reálném i IT životě - jako standard se uchytily vždy ty jednoduché, nikoli dokonalé věci! I zde vládne Occamova břitva - použijme to stávající, nezavádějme nic, co není nezbytně potřeba.
- Každá metoda by měla mít dobře definovaný kontrakt - co očekává, co dělá, co vrací. Plus u metod chovajících se podle stavu objektu, i tyto kontextové informace.
- Programujeme defenzivně a vždy kontrolujeme **dodržení kontraktu** metody. Nepředpokládejme více, než je nezbytně nutné, kontrolujme dodržení kontraktu ze strany klienta - tzn. např. validujme parametry, vyhazujme běhové výjimky...
- Zásadou je **včasné selhání** - "fail early", ihned, jak se přijde na chybu, kvůli níž nelze pokračovat, skutečně nepokračujme. Diagnostika chyb je pak snažší, chyba se lépe lokalizuje a následky lépe řeší.

Z širšího pohledu...

Co musíme znát?

1. Znát Java Core API i další knihovny

2. Používat vhodné nástroje a dát si práce s jejich zvládnutí
3. Znát a používat návrhové vzory
4. Provádět vhodnou dekompozici
5. Základním pravidlem je rozdělení problému na co nejmenší podproblémy, které ještě má cenu řešit samostatně - raději více menších tříd, více jednodušších metod. Snadné vodítko: maximem jsou dva vnořené cykly, resp. cyklus a větvení. Cokoli je složitejší, dekomponujeme do více metod. Důležité je potom správně modelování dat kvůli předávání parametrů mezi těmito metodami. Java bohužel na rozdíl od Pascalu nemá úplnou blokovou strukturu...
6. Minimalizovat míru závislostí
7. Používat vhodnou metodiku (např. Extrémní programování)
8. Provádět automatizované testování