Zbytečně časná výuka dědičnosti je škodlivá

Úvod

Metodika *Design Patterns First*

Metodika *Design Patterns First* rozděluje výuku do tří kurzů: základního, pokračovacího, nadstavbového. V této části bych naznačil základní strukturu těchto kurzů. V následujících částech pak podrobně rozeberu pouze základní kurz, abych na něm ukázal, co všechno je možno vyložit a naprogramovat, aniž bychom museli používat dědičnost implementace.

Základní kurz

Základní kurz rozděluje celý výklad do tří etap:

1. V první etapě studenti neprogramují. Pracuje se v interaktivním režimu, v němž uživatel vystupuje jako jeden z objektů aktuálního programu a posílá ostatním objektům zprávy. V této etapě se studenti seznámí se všemi základními principy OOP aniž by byla jejich pozornost rozptylována nějakými syntaktickými pravidly, která je třeba dodržet.

   V této etapě také poznají první návrhové vzory, i když prozatím ještě stále z hlediska jejich vnějších projevů a vlivu na strukturu a chování programu. Při analýze návrhových vzorů se současně začnou seznamovat s různými účely a uplatněními rozhraní.
2. V druhé etapě se studenti seznámí se základy syntaxe používaného jazyka a naučí se definovat jednoduché třídy, výčtové typy a rozhraní. Naučí se aplikovat jednodušší návrhové vzory ve svých programech a na jejich příkladech pak mohou si mohou dále osvojovat význam a účel definice rozhraní a způsoby jeho optimálního zapojení do celého projektu. Probírá se v ní dědičnost rozhraní, ale neprobírá se dědičnost tříd.

   Tato etapa se koncentruje pouze na syntaxi objektových konstrukcí a tím pádem se vůbec nezmiňuje o konstrukcích algoritmických (podmíněné příkazy, přepínače, cykly). Ani se o nich zmiňovat nemusí, protože příklady jsou koncipovány tak, aby demonstrovaly objektovou složku řešení a prohlubovaly její pochopení. Použití algoritmických konstrukcí při jejich řešení potřeba.
3. Ve třetí etapě se předpokládá, že studenti již zvládli základy syntaxe objektových konstrukcí s výjimkou dědičnosti tříd a začnou proto řešit složitější problémy vyžadující propracovanější způsoby kooperace jednotlivých zúčastněných objektů a vzhledem k rostoucí složitosti projektů a jejich propracovanější architekturu.

   Při řešení stále složitějších problémů se tak naučí pracovat s kontejnery a mezi nimi pak především s dynamickými kontejnery, které jsou v Javě označovány jako kolekce. Přitom se také seznámí s balíčky, základními algoritmickými konstrukcemi a výjimkami. Výjimky se však prozatím naučí pouze vyhazovat.

U většiny probíraných témat se studenti v tomto kurzu seznámí pouze se základní koncepcí dané konstrukce a jejím typickým použitím. Systematický výklad všech probraných konstrukcí je zařazen až do pokračovacího kurzu

Jak již bylo řečeno, základní kurz se důsledně vyhýbá výkladu dědičnosti tříd, jejíž špatné pochopení a z toho plynoucí i špatné následné užití vede k mnohým problémům. Snaží se proto studenty seznámit s alternativními technikami a architekturami, které umožňují řešit řadu problémů elegantněji a efektivněji než prosté použití dědičnosti, po kterém většinou sáhnou ti, kteří se s ní seznámí příliš brzy.

Pokračovací kurz

Na základní kurz navazuje pokračovací kurz, který předpokládá, že studenti již zvládli základy syntax použitého programovacího jazyka s výjimkou některých speciálních konstrukcí (např. zachytávání výjimek, práce s vlákny, serializace apod.). I tento kurz bude rozdělen do několika etap.

1. Kurz začne zopakováním látky základního kurzu a její aplikací při vývoji nějakého středně složitého projektu. Při vývoji tohoto projektu se postupně narazí na několik problémů, které budou vyřešeny v následujících etapách v rámci výuky příslušného tématu.
2. Na toto opakování naváže výklad dědičnosti implementace a na ní založené dědičnosti tříd. Při té příležitosti se znovu vrátíme k parametrizovaným typům a typovým parametrům a probereme je poněkud podrobněji, aby studenti byli schopni definovat vlastní parametrizované typy. Po výkladu dědičnosti implementace bude následovat podrobný výklad výjimek a práce s nimi včetně výkladu definice vlastních výjimek. Druhou etapu zakončí výklad složitěji definovaných výčtových typů a možností jejich použití. Tím kurz dokončí výklad hlavních syntaktických pravidel a jejich použití.
3. Následující etapa probírá některé součásti standardní knihovny (např. knihovnu kolekcí, práci s proudy, časovač…). Nesoustředí se však na to, jak se daná součást používá (i když i na to dojde řeč), ale především na to, jak je navržena. Především ukazuje na použití návrhových vzorů a dodření některých zásad správného návrhu.

Nadstavbový kurz

Nadstavbový kurz se věnuje některým náročnějším tématům, kterým se nákladní kurzy vyhnuly. Začne výkladem návrhu uživatelského rozhraní, na něž naváže výklad základů programování vláken. Přitom se využije toho, že se na práci s GUI dá řada dobře vysvětlit a demonstrovat řada otázek a problémů souvisejících s prací s vlákny.

Nadstavbový kurz pak pokračuje výkladem důležitých zásad objektově orientovaného návrhu a učí studenty, jak program navrhnout tak, aby byl nejenom efektivní, ale především co nejsnáze spravovatelný a modifikovatelný.

1. etapa: interaktivní režim

Jak jsem byl řekl, v první etapě základního pracujeme v interaktivním režimu, v němž uživatel vystupuje jako objekt programu, který může posílat ostatním objektům zprávy.

Objekty a třídy

Na začátku výuky studentům vysvětlujeme základní principy objektově orientovaného programování. Pak jim ukážeme vývojové prostředí *BlueJ*, s nímž budeme v základním kurzu pracovat, a předvádíme dříve vysvětlené principy na úvodním projektu.

Ukážeme studentům, že v OOP je objektem opravdu vše, co je možno označit podstatným jménem. V úvodním projektu se např. setkají s tím, že jako objekty vystupují barvy nebo směry. Při té příležitosti je seznáme i s pojmem třídy a vysvětlíme jim, že třída je speciální druh objektu, který umí vytvářet nové objekty, které označujeme jako instance dané třídy.

Předvedeme jim, jak můžeme třídu požádat o vytvoření své instance. Přitom se hned přesvědčí, že práce se třídami je velmi podobná práci s instancemi a nebudou mít pak v budoucnu tolik problému s rozlišováním členů (atributů, metod a typů) třídy a členů instancí.

Tvorba vlastní třídy

Po úvodním seznámení studentům ukážeme, jak je možno v interaktivním režimu vytvořit vlastní třídy a jak je možno za pomoci použitého vývojového prostředí definovat její chování. Prostřednictvím této třídy studenti nepřímo vytvoří svůj první program, který mohou uložit a na příštích lekcích dále zdokonalovat.

Následně jim na aktuálním projektu předvedeme tří základní pilíře OOP, jimiž jsou identita, skládání a zapouzdření.

Identita objektu a jeho rozhraní

Ukážeme jim, že každá zpráva musí mít konkrétního adresáta, který se sám rozhodne, jak na zprávu zareaguje. Předvedeme jim, jak pošleme různým objektům stejnou zprávu a každý z nich na tuto zprávy zareaguje vlastním způsobem. Při té příležitosti vysvětlíme zkušenějším studentům, kteří se již s výkladem OOP setkali, že polymorfizmus je vlastně pouze důsledkem identity.

Při té příležitosti současně vysvětlíme, co je to rozhraní objektu a co je to implementace. Při tom současně řekneme, že část kódu, která definuje reakci objektu na zprávu, se nazývá metoda a že programátoři často místo o posílání zpráv hovoří o volání metod. Kód metody označujeme jako implementaci zprávy daným objektem. Přidáme, že správný program před svým okolím svoji implementaci skrývá. V tuto chvíli, kdy studenti opravdu nic netuší o implementaci, se jim skrývání implementace poměrně dobře předvádí.

Skládání

Na aktuálním projektu můžeme dobře vysvětlit i pojem skládání. Nahlédneme do nitra instancí a ukážeme, že objekty jsou složeny z jiných objektů, přičemž odkazy na tyto své součásti mají uloženy v atributech. Vysvětlíme si, že součástí skrývání implementace je i skrývání atributů a předvedeme že naprostá většina všech atributů je deklarována jako soukromé vlastnictví svých majitelů.

Při té příležitosti jim současně vysvětlíme, že moderní programovací jazyky neumožňují přímou práci s objekty, ale povolují pouze práci s odkazy na objekty, čímž zvyšují spolehlivost programů, produktivitu programátorské práce a v neposlední řadě činí program průzračnějším.

Zkušenějším studentům, kteří již prošli kurzy OOP, pak vysvětlíme, že dědičnost je vlastně pouze speciální případ skládání, kdy s objektem svého předka převezme potomek i jeho rozhraní.

Zapouzdření

Na aktuální aplikaci naznačíme i princip zapouzdření, i když zde nám studenti musejí víc věřit, protože to, že kód u zpracovávaná data jsou opravdu pohromadě, si nemohou „osahat“.

Při té příležitosti jim vysvětlíme, že správně aplikované zapouzdření nám usnadní skrývání implementace a současně je upozorníme, že řada učitelů tyto dva termíny nerozlišuje a obě dvě vlastnosti označuje společným termínem *zapouzdření*.

`interface`

Na závěr interaktivní části se vrátíme k pojmu rozhraní a vysvětlíme si, že některé programovací jazyky (Java mezi nimi) zavádějí speciální druh datového typu označovaného `interface`. Tento datový typ deklaruje pouze rozhraní a nemá žádnou implementaci. Na aktuálním projektu si pak ukážeme, jak třída, která definuje všechny metody požadované rozhraním, může implementovat zadané rozhraní a jaké to má pro ni východy. Předvedeme si návrhový vzor služebník a ukážeme si jeho aplikaci na příkladu plynulého přesunu objektů.

Poté vysvětlíme některé nevýhody použitého řešení a ukážeme, jak zvýšit tvárnost celého řešení. Vysvětlíme studentům princip dědičnosti typů a na příkladech dědičnosti rozhraní jim předvedeme, jak lze využít některé její výhody i v interaktivním režimu.

Návrhové vzory

V průběhu celého výkladu v interaktivním režimu upozorňujeme na aplikace různých návrhových vzorů. Nejprve na návrhové vzory, které jsou použity v základním projektu, při představování rozhraní pak na návrhový vzor *Služebník*. Na závěr interaktivní části je seznámíme se vzory *Prostředník* (*Mediator*) a *Pozorovatel* (*Observer*) a ukážeme si, jak po jejich aplikaci ztratí používaný projekt většinu ze svých nepříjemných vlastností.

2. etapa: základy kódování

V druhé etapě studenti definují svoji první třídu, u níž budou nejprve analyzovat a posléze doplňovat a vylepšovat její kód.

Nejprve s nimi znovu projdeme všechna témata, které jsme probírali v interaktivním režimu, a ukážeme si, jak se příslušné prvky kódují a jak se používají. Při tom si znovu připomeneme význam rozhraní a tentokrát již můžeme na konkrétním kódu ukázat, které části kódu řeší problematiku rozhraní a které mají na starosti implementaci.

Při probírání atributů jim vysvětlíme rozdíl mezi atributem a vlastností a zopakujeme základní konvence pro definice přístupových metod. Ukážeme jim, že vlastnosti mohou, ale také nemusí být kryty atributy.

Připomeneme studentům datovou konstrukci `interface` a ukážeme jim, že je to vlastně zakódované rozhraní. Znovu zopakujeme implementaci rozhraní třídou a ukážeme jim, jak ji realizovat. Při té příležitosti seznámíme s tím, že rozhraní má vlastně dvě tváře, signaturu a kontrakt, a ukážeme jim, jak lze kontrakt zapsat v dokumentačních komentářích.

Výkladem komentářů opouštíme oblast probranou v interaktivním režimu a pokračujeme ve výkladu konstrukcí, s nimiž se studenti doposud nesetkali. Nejprve jim vysvětlíme význam a použití návrhového vzoru *Přepravka* (*Crate*, *Transport Object*) a poté s nimi probereme rozdíl mezi hodnotovými a odkazovými objektovými typy. Naučíme je definovat metodu `equals(Object)`, i když prozatím ještě bez použití podmíněného příkazu, což u běžných hodnotových typů naštěstí nebývá problém.

Před závěrečným projektem druhé etapy se studenty důkladně probereme mechanizmus zavádění tříd a konstrukce jejich instancí včetně statických i instančních inicializačních bloků, což je oblast, kterou řada učebnic zmiňuje jenom letmo a studenti pak těžko zjišťují, proč se některý program chová tak podivně.

Celou problematiku nejprve předvedeme za pomoci kontrolních tisků, načež studenty seznámíme s ladícím nástrojem (debuggerem), který je součástí použitého vývojového prostředí, a znovu jim vše předvedeme „zevnitř“.

Na závěr této etapy studentům vysvětlíme, jak je možno vytvořit samostatnou aplikaci a vytvoříme společně hru, kterou mohou provozovat na všech počítačích, na nichž je instalovaná Java, resp. platforma, na níž běží jazyk a projekty, které při výkladu OOP používáme.

3. etapa: pokročilé kódování

Jak jsem již řekl, třetí etapa je zaměřena na výklad a procvičení pokročilejších konstrukcí objektově orientovaného programování a spolu s nimi i strukturovaných konstrukcí, tj. podmíněných příkazů, přepínačů a cyklů.

Výklad začíná vysvětlením práce s balíčky, resp. jmennými prostory (záleží na platformě). Seznámíme je s problémy, s nimiž se dříve potýkaly rozsáhlejší projekty, a vysvětlíme jim koncepci balíčků (jmenných prostorů), která tyto problémy z velké části řeší. Pak rozdělíme náš nabobtnalý projekt do balíčků a v dalším výkladu se soustředíme vždy jen na balíček, ve kterém se nacházejí třídy a rozhraní, s nimiž chceme pracovat.