Java pro začátečníky (13) - Abstraktní třídy a rozhraní

Minule jsme si ukázali, jakým způsob lze v Javě specifikovat podtypy. Tentokrát budeme postupovat opačným směrem – představíme si rozhraní a abstraktní třídy, což jsou konstrukty, které slouží ke generalizaci (zobecnění) tříd.

Abstraktní třídy

Stejně jako v minulém dílu si představme, že máme firmu, kde pracují zaměstnanci mnoha různých profesí (ředitele, sekretářky, uklízečky...). Tito zaměstnanci sdíli určité generické chování, které je společné všem profesím, například všichni zdraví na potkání své kolegy.

Zaměstnanci se ale v určitých aspektech liší. Všichni sice pracují (metoda *work()*), ale každá profese dělá něco trochu jiného.Minule jsme tuto situaci řešili tak, že jsme vytvořili třídu obecného zaměstnance, ze které dědili všichni specifičtí činitelé.

Tentokrát ovšem v naší firmě pracují pouze specifičtí zaměstnanci – to znamená, že neexistuje (nemůže existovat) žádná instance obecné profese. Z tohoto hlediska by bylo velmi nešikovné implementovat metody společné nadtřídy (když stejně nebudou nikdy využity). Zároveň by toto nijak nezamezilo konstrukci onoho obecného zaměstnance, který v naší firmě nemůže pracovat (z definice příkladu).

Klíčové slovo abstract

Pro řešení této situace použijeme klíčové slovo *abstract*. Pokud toto slovo vepíšeme do hlavičky třídy, tak z ní nelze vytvářet instance (tím zamezíme vytvoření instance obecného zaměstnance).

Důležitější je ale jeho použití u metod abstraktních tříd. U takto označených metod můžeme vynechat jejich tělo (a napsat místo něj pouze středník), čímž deklarujeme, že tuto metodu musí mít implementovanou všichni neabstrakní potomci. Tohoto později využijeme při polymorfním volání nad jednotlivými objekty (*vizte minulý díl*).

Rozhraní (interface)

Druhou velmi podobnou konstrukcí jsou rozhraní (klíčové slovo *interface*). Základním rozdílem je, že interface obsahuje pouze konstanty a metody bez těla (v tomto případě je neoznačujeme slovem *abstract*). Rozhraní je kontrakt, který specifikuje operace, které má třída splňovat, a který se již nezabývá tím, jak toho bude konkrétně dosaženo.

Velkou výhodou rozhraní oproti abstraktním třídám je, že každá třída může implementovat až mnoho rozhraní, avšak vždy maximálně jednu třídu.

Zatímto pro dědění tříd (a dědení interfaců mezi sebou) využíváme v hlavičce klíčové slovo *extends*, tak pro implementaci rozhraní používáme slovo *implements*.

Konvence

Pro pojmenovávání rozhraní neexistuje žádná široce uznávaná konvence. Časté je využívání přídavných jmen pro rozhraní (*Serializable*, *Movable* atp.), což ale nelze v mnoha případech dodržet (neexistuje přídavné jméno, které by onen kontrakt vystihovalo). Další častou konvencí je využívání přípon *Interface* a *Iface*. Poslední častou variantou je nepoužít žádnou příponu pro rozhraní, ale pojmenovat třídu implementující rozhraní jeho názvem s příponou *Impl* (rozhraní: *Server*, třída: *ServerImpl*).

Kdy použít abstraktní třídu a kdy rozhraní?

Velmi častou otázkou je, kdy je vhodné použít abstraktní třídu a kdy rozhraní. Ačkoliv jsou si tyto dva konstrukty velmi podobné, tak mají poněkud odlišné využití a navzájem se velmi dobře doplňují. Odpovědí je proto, že buď dává jasný smysl pouze jeden z konstruktů, případně, že použijeme oba dva.

Příklad

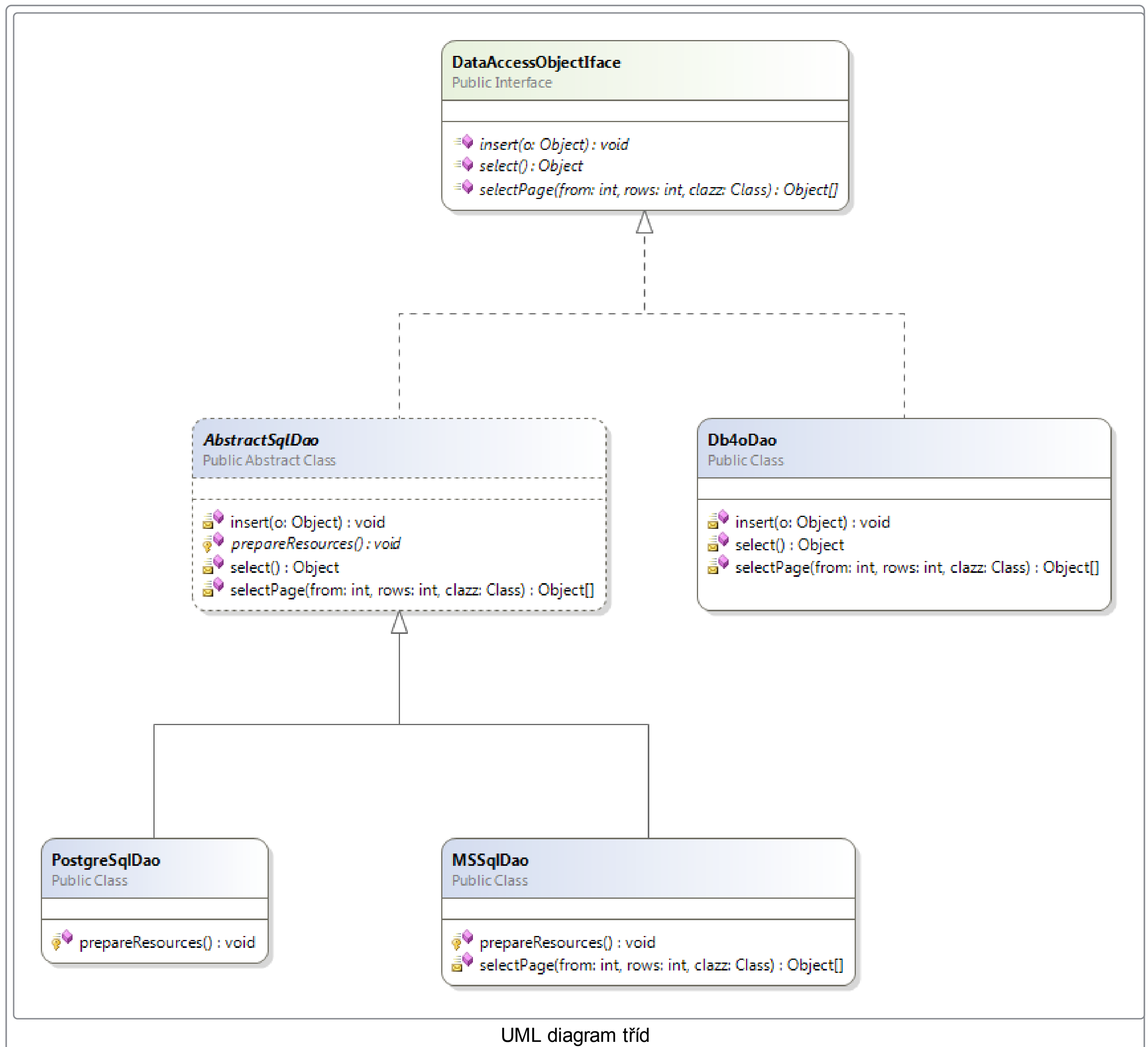
Abychom si dnešní látku procvičili, tak si ukážeme jeden příklad z reálného světa. Příklad sice bude velmi zjednodušený, ale jasně nám ukáže základní myšlenku rozhraní a abstraktních tříd.

Mějme dva typy databází – klasické *SQL* databáze (*Oracle*, *PostgreSQL*, *MySQL*, *MS SQL Server* atp.) a objektové databáze (např. *db4o*). Program chceme postavit tak, abychom mohli implementace (v tomto případě operací *select* a *insert*) jednoduše zaměňovat (pomocí polymorfismu).

Hierarchie tříd

Z logiky věci vyplývá, že na vrcholu naší hierarchie bude rozhraní definující operace *save* a *insert*. Z něj pak budou dědit dvě větve tříd. První větev bude určena pro *SQL* databáze a jejím vrcholem bude abstraktní třída implementující příkazy *select* a *insert* obecným způsobem platným pro značnou část relačních databází. Z této třídu budou dědit jednotlivé implementace pro specifické databáze (a překryjí případně generické chování vlastní implementací).

Druhou větví budou objekty určené přímo pro jednotlivé objektové databáze. Ty bohužel žádnou abstraktní třídu mít nemohou, jelikož nemají žádný standardizovaný dotazovací jazyk.



UML diagram tříd

– Pavel Mička
Zakladatel a administrátor stránek Algoritmy.net