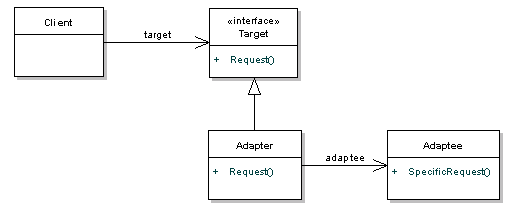
# Návrhový vzor adaptér

Již podle názvu tohoto návrhového vzoru můžeme odhadnout hlavní problematiku, na kterou je tento návrhový vzor specializován. V reálném životě si toto můžeme představit jako cestovní adaptér do zásuvek. Tento problém je změna rozhraní třídy na rozhraní jiné, nebo další v pořadí. Adaptér umožní spolupráci třídám, které by vzhledem k rozdílným rozhraním jinak nespolupracovaly. Způsob implementace je tak, že mezi tyto 2 třídy vložíme třídu adapter. V jazyce Java se lze s příkladem adaptéru jako návrhového vzoru nejčastěji setkat při práci s tzv. obalovými třídami (wrapper). Tyto wrappery přizpůsobují rozhraní primitivních datových typů tak, aby s nimi bylo zacházeno jako s objektovými typy.



**Client** - pro něj přizpůsobujeme, využívá rozhraní **Target**

**Target -** definuje rozhraní, které vyžaduje **Client**

**Adaptee (= adaptovaný, přizpůsobovaný) -** poskytuje požadované operace, jiné rozhraní než Target, často převzatý kód, který nelze měnit

**Adapter -** Přizpůsobuje **Adaptee** na interface **Target**

Tento návrhový vzor můžeme implementovat třemi způsoby. Každý tento způsob se hodí na něco jiného.

První způsob je **univerzální adaptér**. Univerzální adaptér použijeme v situaci, kdy klient požaduje po třídách, aby implementovaly určité rozhraní. Adaptér je definován jako společný předek pro všechny tyto třídy a v něm jsou implementovány implicitní verze všech metod požadovaných rozhraním. Nevýhodou Univerzálního adaptéru je fakt, že se dá použít pouze v případě, kdy definujeme novou třídu (Adaptér), která nemusí být součástí žádné dědické hierarchie.

Druhý způsob je **Adaptér obsahující adaptovaný objekt**. Při tomto způsobu implementace již adaptér není definován jako univerzální, nýbrž jako jednoúčelový. Výhoda a hlavní důvod použití této implementace je ten, že není třeba zasahovat do adaptované třídy. Příkladem takto definovaného adaptéru jsou obalové třídy, které obalí hodnotu primitivního typu jako atribut instance některého z potomků třídy Object.

Třetí způsob je **Adaptér jako potomek adaptované třídy**. Při této definici adaptéru je potřeba zařídit, aby byl adaptér potomkem adaptované třídy a zároveň implementoval požadované rozhraní. Všechny metody požadované rozhraním jsou pak definovány přes volání metod adaptované třídy. Takto implementovaný adaptér je již čistě jednoúčelový. Tento způsob se využívá v situaci, kdy je potřeba začlenit instance třídy současně do dvou rámců, přičemž každý rámec vyžaduje implementaci jiného rozhraní.

## Příklady užití

Nejčastěji se tento návrhový vzor užívá jako wrapper. Tento wrapper se staví okolo databází (lze poté vyměnit databázi bez jediné změny v naší aplikaci) nebo okolo webových služeb, které mají často složitá API.

Další využití je u programovacího jazyku Java a jeho grafickém rozhraní. Rozhodneme se používat novou grafickou knihovnu Swing, ale musíme zajistit kompatibilitu s předchozí verzí grafické knihovny AWT.

Další příklad užití. Tento příklad pojednává o přehrávači. Přehrávač může přehrávat pouze soubory .mp3. My ale chceme aby přehrával i soubory .mp4 a .vlc. Proto implementujeme do našeho přehrávače rozšířený, který přehrává tyto soubory defaultně.

Například aplikace pracuje s údaji o akciích. Stahuje data z více poskytovatelů dat v XML a vytváří grafy. V nové verzi jsme se rozhodli vylepšit aplikaci a použít nějakou knihovnu inteligentní analýzy. Zde však nastává chyba, analytická knihovna pracuje pouze s daty ve formátu JSON.

Implementovaný příklad je na téma účetní systém. Seznam zaměstnanců je třeba převést do formátu, který odpovídá formátu účetního systému. Tento příklad je převzatý z internetu.

Zdroj: http://www.dotnettricks.com/learn/designpatterns/adapter-design-pattern-dotnet

