INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

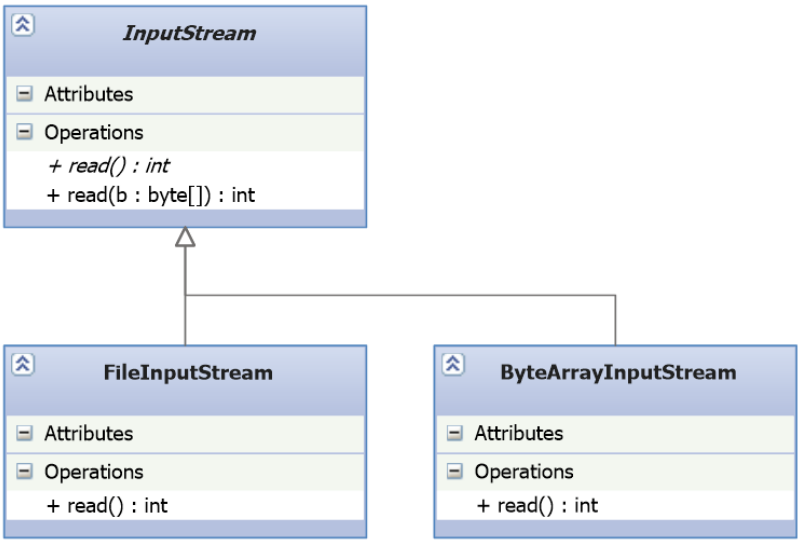
Projekt „Popularizace vědy a výzkumu ČVUT“, reg. č. CZ.1.07/2.3.00/35.0021

Objektově orientované programování (10): Abstraktní třídy

Důležitým prvkem objektově orientovaného programování jsou tzv. *abstraktní třídy*. Tyto struktury jsou nedocenitelné při tvorbě dobře strukturovaného, přehledného a snadno rozšiřitelného kódu.

Abstraktní třídy jsou takové třídy, které deklarují alespoň jednu tzv. *abstraktní metodu*, což je taková metoda, která musí být definovaná ve třídě, která je potomkem deklarující abstraktní třídy. Z toho důvodu tedy ani není možné vytvořit instanci abstraktní třídy, pouze jejích (neabstraktních) potomků.

Definice zní trochu divoce, proto si tento koncept vysvětlíme na příkladu. Pokud jste někdy v Javě pracovali se soubory, nejspíš jste se setkali s třídou `FileInputStream`, která poskytuje metody pro čtení ze souborů. Pokud se podíváte na její zdrojový kód, zjistíte, že třída je potomkem abstraktní třídy `InputStream`, která mimo jiné deklaruje abstraktní metodu `read()` a definuje metodu `read(byte b[])`. Účelem metody `read()` (tedy varianty bez parametrů) je přečtení jednoho bajtu z datového proudu. Nicméně o jaký proud konkrétně jde, již abstraktní třída `InputStream` nedefinuje. Proto je tato metoda a tedy i celá třída označena jako *abstraktní*. Druhá zmíněná metoda `read(byte b[])`, má za úkol pokud možno naplnit bajtové pole `b`. Protože třída `InputStream` deklaruje abstraktní metodu pro čtení jednoho bajtu, může je metoda `read(byte b[])` pro naplnění pole `b` použít. Metoda `read(byte b[])` tedy nemusí být abstraktní a může být společná pro všechny potomky třídy `InputStream`, což jsou například `FileInputStream` (čtení ze souborů), `ByteArrayInputStream` (čtení z paměti, konkrétně bajtového pole), `ObjectInputStream` (čtení celých objektů pomocí deserializace) či `AudioInputStream` (čtení datového toku s audiem).



Obrázek 1 Příklad použití abstraktní třídy v Javě (schéma je značně zjednodušeno).

Na obrázku 1 je znázorněn značně zjednodušený UML diagram popisovaného příkladu. Jak si můžete všimnout, abstraktní metody a třídy jsou v něm vyznačené pomocí kurzívy.

Výhodou takového řešení je možnost sjednocení kódu u tříd, které v důsledku vykonávají různé činnosti. V popisovaném příkladu to je možnost čtení ze souboru, paměti nebo třeba sítě s minimálními (případně žádnými) úpravami zdrojového kódu.

Dalším praktickým příkladem využití abstraktní třídy je při formátování výstupu z programu. Každý výstupní formát bude potomkem společné abstraktní třídy, která bude definovat společné části kódu. Nicméně konkrétní formát výstupu budou již definovat jednotlivé třídy.

V příštím díle si popíšeme, co jsou a k čemu slouží tzv. *rozhraní*.