Dědičnost

Z FAV wiki

Základním stavebním blokem OOP je dědičnost. Dědění je způsob deklarace třídy tak, že využijeme jinou třídu jako předka. Naše děděná třída tedy může

Převzít metody a atributy předka Definovat nové metody a atributy Přepsat metody a atributy předka

Ve většině jazyků jsou všechny třídy implicitně děděny od třídy Object. Ve většině jazyků (snad kromě C++) zle dědit pouze od jedné třídy. Pro další dědění lze použít rozhraní.

V Javě dědíme pomocí klíčového slova extends, tedy

Class B extends A { }

V jiných jazycích (C++, C#, ...) používáme dvojtečku, přičemž první třída za dvojtečkou je obvykle dědění, ostatní jsou implementace rozhraní (i když v C++ je to jedno, tam můžeme implementovat i dědit od více tříd)

S pojmem dědění úzce souvisí i pojem Polymorfismus. Ten nám umožňuje nejen hiearchii objektů rozšiřovat, ale také ji využívat opačným směrem, tedy dědí-li třída B od třídy A, můžeme použít konstrukci (přiřazení)

B b = new B(); A a = b;

To je extrémně důležité pro ADT, kdy vytvoříme strukturu pro A, a můžeme do ní ukládat objekty A i B.

Otázka je, pokud voláme metody A.Metoda, a metoda B tuto metodu má také, co se děje? Pro snažší pochopení řekněme, že máme typ pointeru a typ objektu. V předchozí ukázce je a typu A a b typu B, ale typ pointeru a je A a typ objektu (pod tímto pointerem) a je B.

Některé jazyky podporují Virtualitu metod. Virtuální metoda je metoda, která (je li překryta) je volána vždy z typu objektu, který překrývá. Pokud překryta není (nebo pokud je volána z třídy, která tuto virtuální metodu obsahuje), volá se ona. Voláme-li tedy a.Metoda() z předchozího příkladu, zavolá se ve skutečnosti b.Metoda(), protože a ukazuje na typ b. Java má všechny metody virtuální.

Druhou možností je použít nevirtuální metody. Taková metoda nemůže být překryta, nicméně může být znovu vytvořena (předefinována řekněme) pro daná objekt. V praxi to znamená, že je volána z typu Pointeru který na objekt ukazuje, tedy voláme li a.Metoda() zavoláme opravdu a.Metoda().

Za poslední zmínku stojí Kovariance typů. Ta znamená, že je li typ kovariantní, lze vzájemně přiřazovat obalující typy, tedy lze provést např.

pokud B extends A:

List<A> a = new List**();**

tedy je li B potomkem A, je li List<B> potomkem List<A>. Tento mechanismus však ve většině jazyků nefunguje, a je nutné vytvořit List<A> a naskládat do něj objekty B, a při výběru je jeden po druhém přetypovat zpět na B, jelikož List<A> vrací objekty typu A.