**UML: Diagramy tříd**

UML (Unified Modeling Language) je grafický jazyk, který slouží k vizualizaci, navrhování a dokumentaci programů. V UML můžete vymodelovat zhruba třináct typů diagramů (liší se podle verze UML). Například diagram aktivit popisuje chování daného programu. Může vypadat například takto:



My se budeme věnovat diagramu tříd:



**Diagram tříd zobrazuje třídy a vazby mezi nimi.**

**Použití diagramu tříd:**

* objektový návrh programu
* dokumentace
* zpětné inženýrství (snaha zjistit, jak program funguje, pokud se k nám dostane pouze končená aplikace, ale ne kód)

**Třídy**

**Název třídy**



**Atributy**

**ibuty**

**Metody**

Název datového typu, atributy a metody mohou být uvozeny symbolem přístupnosti:
**- private
+ public
# protected**
~ implicitní přístup (package private)

**Vztahy**

V UML je možné mezi třídami i objekty nastavit několik různých typů vazeb.

**Obecné vztahy**

**Závislost**



public class Food {

 // ...

}

public class Animal {

 void Feed(Food food) {

 // ...

 }

}

Živočich (Animal) POUŽÍVÁ ve své metodě pro krmení (feed) potravu (Food).

Jedná se o nejslabší vztah indikující závislost jedné třídy na jiné. Závislost je realizována například použitím druhé třídy jako parametru (nebo lokální proměnné) v některé z metod první třídy.

Tento vztah je znázorněn přerušovanou čarou a jednoduchou šipkou.

**Vztahy mezi datovými typy**

**Generalizace (specializace)**



public class Vehicle {

 // ...

}

public class Automobile : Vehicle {

 // ...

}

Automobil (Automobile) JE dopravní prostředek (Vehicle). Dopravní prostředek je generalizací automobilu, a naopak automobil je specializací dopravního prostředku. Většinou se používá pro vztah dědění.

Tento vztah je znázorněn plnou šipkou s nevyplněnou trojúhelníkovou hlavou.

**Realizace**



public interface ISend {

 public void send();

}

public class UDPSend : ISend {

 public override void send() {

 // ...

 }

}

public class TCPSend : ISend {

 public override void send() {

 // ...

 }

}

Závislý datový typ je implementací jiného datového typu. Závislý typ tedy realizuje předepsané chování jiného datového typu.

Vztah je opět znázorněn šipkou s prázdnou trojúhelníkovou hlavou, ovšem na rozdíl od dědičnosti je implementační šipka pouze čárkovaná.

**Vztahy mezi instancemi dvou a více tříd**

****

**Asociace**



Asociace indikuje vztah mezi instancemi dvou a více tříd.

Asociace je znázorněna plnou čarou.

**Agregace**



public class Professor {

 // ...

}

public class Department {

 private List<Professor> professorList;

 // ..

}

Katedra (Department) MÁ profesory (Professor). Agregace je specifičtější vztah než výše popsaná asociace a volnější vztah než níže popisovaná kompozice. Zánik jednoho objektu neznamená zánik druhého.

Vztah agregace je znázorněn plnou čarou s jednoduchou šipkou. U paty čáry je prázdný kosočtverec.

**Kompozice**



Kompozice je nejsilnějším z trojice vztahů asociace – agregace – kompozice. Odkazovaný objekt je zásadní součástí majitele. Pokud zanikne Faculty (Fakulta), musí nutně zaniknout i Department (Katedra).

Vztah kompozice je opět znázorněn plnou čarou s jednoduchou šipkou. Na rozdíl od agregace je však kosočtverec u paty čáry vyplněný.

Přesné vymezení mezi jednotlivými vztahy není určeno a vždy záleží na rozhodnutí programátora nebo analytika. Striktní dělení vztahů je důležité, pokud budeme z UML diagramu přímo generovat kód.

Vzhledem k tomu, že my kód generovat nebudeme, bohatě postačí, budete-li všude používat asociaci.

Zdroje:

[***http://www.milosnemec.cz/clanek.php?id=199***](http://www.milosnemec.cz/clanek.php?id=199)