Vâce o Objective C

V minulÝm dâlu jsme se seznÖmili se systÝmem objekt a ukÖzali jsme si vechny zÖkladnâ sluby jazyka Objective C. Nynâ se seznÖmâme se zbytkem konstrukcâ, je Objective C nabâzâ; akoli ÖdnÖ z nich nenâ pro programovÖnâ bezpodmânen nutnÖ -- jednoduchÝ testovacâ progrÖmky jste si snadno mohli vyzkouet u s vyuitâm slueb, popsan÷ch minule -- dokÖâ programÖtorovi v÷razn usnadnit ivot.

NeobjektovÖ rozâenâ

Jazyk Objective C je uren pedevâm pro prÖci s objekty; neobjektov÷ch rozâenâ v nm proto mnoho nenalezneme. Ta, je zde jsou, jsou vak velmi pâjemnÖ. Prv÷m z nich je monost pouâvat komentÖ "//" stejn jako v C++ (to ji nepâmo vyplynulo z pâklad v minulÝm dâlu, kde byly takovÝ komentÖe pouâvÖny). Druh÷m je standardizace typu a hodnot pro logickÝ (boolovskÝ) promnnÝ:€*ani by byl naruen standardnâ pâstup jazyka C*, slouâ jako logick÷ typ typ `BOOL`€a odpovâdajâcâ hodnoty jsou `YES`€a `NO`. Standardnâ headery prost definujâ

```
typedef int BOOL;
#define YES€1
#define NO€0
```

pâpadn v jazyce C ekvivalentnâ `typedef enum {NO, YES} BOOL`, jeho v÷hodou je, e konstanty `YES`€a `NO` jsou znÖmy i na órovni debuggeru.

Velmi ikovn÷m rozâenâm je direktiva `#import`. Ta funguje tÝm stejn, jako klasickÖ direktiva `#include`; peklada ale zajistâ, e kad÷ zdrojov÷ soubor se bude peklÖdat *nejv÷e jednou*. V Objective C si proto meme uetit nepohodlnÝ obklÖdÖnâ kadÝho hlavikovÝho souboru direktivami typu

```
#ifndef _STDIO_H_
#define _STDIO_H_
...
#endif
```

Je snad trochu spornÝ, zda mezi neobjektovÖ rozâenâ meme adit *novÝ typy, hodnoty a identifikÖtory*: krom typ id a Class a hodnot nil a Nil, je ji znÖme z minulÝho dâlu, nabâzâ Objective C€nÖsledujâcâ typy a hodnoty:

Typy:

`SEL`	vnitnâ representace zprÖvy
`IMP`	metoda (pâm÷ ukazatel na metodu, pouâvan÷ pro statick÷ pâstup)

IdentifikÖtory

`id self`	v implementaci metody reprezentuje objekt, kter÷ metodu zpracovÖvÖ
`id super`	ditto, ale jeho metody jsou vyhledÖvÖny v rodiovskÝ tâd
`SEL€_cmd`	v implementaci metody reprezentuje zprÖvu, je metodu vyvolala

Typ `SEL` reprezentuje zprÖvu a je definovÖn jako celoâselnÖ hodnota, na kterou je zprÖva intern peloena. Spolu s direktivou `@selector`, je zprÖvy pevÖdâ na tento typ, umouje zprÖvy uklÖdat do promnn÷ch, vzÖjemn porovnÖvat a podobn. Typ `IMP`€ vlastn nenâ niâm jin÷m, ne ukazatelem na funkci, a vyuâvÖ se v tch zcela v÷jimen÷ch pâpadech, kdy potebujeme volat metodu rychleji, ne prostednictvâm mechanismu zprÖv. UkÖzky praktickÝho pouitâ naleznete v pâkladech na CD; totÝ platâ i pro vechny ostatnâ konstrukce.

Poznamenejme, e pro dosaenâ statickÝ typovÝ kontroly srovnatelnÝ s C++ nabâzâ Objective C monost pouâvat na mâst typu `id` konstrukci "ukazatel na tâdu" s v÷znamem "objekt danÝ tâdy nebo jejâho ddice". Je vhodnÝ zdraznit, e jde pouze o statickou, pekladovou kontrolu -- na v÷sledn÷ program to nemÖ vbec Ödn÷ vliv, ten bude fungovat stejn dobe (nebo stejn patn) jako kdybychom vude dsledn pouâvali `id`.

Dâky tomu e `self`, `super` a `_cmd` jsou *identifikÖtory* a ne klâovÖ slova (jako je tomu nap. v nedomylenÝm C++), meme je bez jak÷chkoli problÝm pedefinovat; peklada Objective C proto bez problÝm peloâ 'obyejn÷ cÝkov÷' program, ve kterÝm je pouita nap. promnnÖ jmÝnem `self`.

Pâstup k promnn÷m

PromnnÝ objektu mohou b÷t k dispozici pouze jeho vlastnâm metodÖm, nebo i metodÖm vech jeho ddic, nebo -- ve v÷jimen÷ch pâpadech, kdy z njakÝho dvodu musâme rezignovat na objektovÝ programovÖnâ a vyuâvat statickÝ programÖtorskÝ techniky -- mohou b÷t promnnÝ pâstupnÝ z jakÝhokoli óseku kídu. Monosti pâstupu k promnn÷m jsou ureny pouitâm jednÝ ze tâ direktiv:

`@private` promnnÝ jsou pâstupnÝ pouze metodÖm objektu samotnÝho;
`@protected` promnnÝ jsou pâstupnÝ i ddicm (tento pâstup je standardnâ nepouijeme-li Ödnou z direktiv);
`@public` promnnÝ jsou pâstupnÝ komukoli.

Jestlie z njakÝho dvodu musâme rezignovat na objektov÷ pâstup, meme takÝ zâskat neomezen÷ pâstup k promnn÷m kterÝhokoli objektu pomocâ direktivy `@defs`.

Kategorie

PrimÖrnâ óel kategoriâ je umonit rozloenâ implementace jednÝ sloitÝ tâdy do nkolika zdrojov÷ch soubor. Kategorie mÖ interface i implementaci obdobnÝ standardnâm, avak na mâst nadâzenÝ tâdy je jmÝno kategorie v zÖvorkÖch. Kategorie samozejm neme definovat vlastnâ promnnÝ; mÖ vak voln÷ pâstup k promnn÷m, definovan÷m v zÖkladnâm rozhranâ tâdy.

Dejme tomu, e mÖme nÖsledujâcâ tâdu:

```
@interface Xxx:Yyy
-aaa;
-bbb;
-ccc;
@end
```

vetn odpovâdajâcâ implementace

```
@implementation Xxx
-aaa { ... }
-bbb { ... }
-ccc { ... }
@end
```

Pokud by pro nÖs bylo z jakÝhokoli dvodu v÷hodnÝ oddlit od sebe implementace tchto tâ metod do samostatn÷ch celk, mohli bychom stejn dobe pouât zÖkladnâ tâdy a dvou kategoriâ -- z hlediska prÖce s tâdou `Xxx` a jejâmi objekty by se nezmnilo vbec nic:

```
@interface Xxx:Yyy // zÖkladnâ tâda
-aaa;
@end
@interface Xxx (KategorieProMetoduB)
-bbb;
@end
@interface Xxx (AProMetoduCcc)
-ccc;
@end
```

Obdobn by samozejm byla rozdlena i implementace.

Kategorie navâc umoujâ *doplovat nebo mnit ji existujâcâ tâdy*:€dejme tomu, e bychom chtli, aby *libovoln÷ objekt* dokÖzal reagovat na zprÖvu where jmÝnem poâtae, na kterÝm bâ proces, v rÖmci nho objekt existuje. V Objective C€nenâ nic jednoduâho -- prost implementujeme kategorii

```
@interface NSObject (ReportWhere)
-(NSString*)where;
@end

@implementation NSObject (ReportWhere)
-(NSString*)where
{
€€return [[NSProcess Info processInfo] hostName];
}
@end
```

Jakmile mÖme kategorii hotovou, meme novou slubu zcela voln pouâvat u kterÝhokoli objektu.

Protokoly

Protokol v zÖsad nenâ niâm jin÷m, ne seznamem metod; pouâvÖ se jako spolen÷ prvek pro specifikaci tâd, kterÝ majâ mât spolenÝ metody, ale nejsou strukturÖln pâbuznÝ (âm nahrazuje implementan i programÖtorsky obtânou vâcenÖsobnou ddinost C++ v tom jedinÝm pâpad, kdy mla jak÷si smysl).

Protokol je definovÖn obdobn jako interface, neme vak samozejm obsahovat promnnÝ. Protokoly vak mohou mât svou vlastnâ ddinost. Namâsto direktivy `@interface` je zde pouita direktiva `@protocol`. KonkrÝtnâ pâklad naleznete opt na CD.

Ostatnâ

Objective C nabâzâ jet dv direktivy, `@class` a `@encode`. PrvÖ z nich prost informuje peklada o existenci tâdy danÝho jmÝna, a slouâ pro dopednÝ reference:

```
@class Xxx;
@interface Yyy
-(Xxx*)xxx;
@end
@interface Xxx
-(Yyy*)yyy;
@end
```

Direktiva `@encode` slouâ pro dynamickou specifikaci typu, a v praxi se tÝm vbec nepouâvÖ (protoe pln objektov÷ systÝm dynamickÝ typy vlastn nepotebuje -- namâsto nich se pouâvajâ objekty, je si typovou informaci nesou implicitn v sob); jejâ podrobn÷ popis si proto meme odpustit.

Pâklady

Pro lepâ ilustraci je na CD ada bohat komentovan÷ch pâklad jednoduch÷ch program:

Pâklad 1: vyuitâ peddefinovan÷ch tâd (knihovny tâd NeXTstepu);
Pâklad 2: tvorba vlastnâ tâdy;
Pâklad 3: ddinost a vklÖdÖnâ objekt;
Pâklad 4: dynamickÝ rozpoznÖnâ tâdy;
Pâklad 5: sklÖdÖnâ objekt a dynamickÝ rozpoznÖnâ zprÖv;
Pâklad 6: sklÖdÖnâ objekt a dynamickÝ rozpoznÖnâ zprÖv -- jinÖ varianta;
Pâklad 7: mechanismus klient/server;
Pâklad 8: statick÷ pâstup k objektm.

Shrnutâ

Dokonili jsme strun÷ popis jazyka Objective C; ti, kdo majâ jeho peklada k dispozici (jako GNU€C je k dispozici na libovolnÝ platform, od Mac OS€X pes vechny varianty unixu a po DOS€i Windows) v nm mohou psÖt libovolnÝ testovacâ programy.

Pât se u zaneme bavit o skuten÷ch vlastnostech prostedâ Cocoa:€ukÖeme si mechanismus tvorby a zÖniku objekt a podobn.

-oc-