DUM č. 13 v sadě

28. Inf-4 Jednoduchá hra Had ve Flashi (ActionScript)

Autor: Robert Havlásek

Datum: 14.05.2013

Ročník: 5AV

Anotace DUMu: Flash - teorie: Funkce. Předávání parametrů do funkce. Funkce jako metoda objektu.\n

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Funkce

Kus kódu, který se v programu vyskytuje několikrát, je ideální napsat zvlášť do tzv. funkce.

Funkci lze zapsat do programu úplně samostatně, ale může být též navázána na konkrétní objekt (být jeho součástí, „být tím, co objekt umí“), v tom případě ji nazýváme metoda. Již existující metodu existujícího objektu lze též předefinovat, přiřazením (zapsáním) jiné funkce. Psaní objektů s metodami bude popsáno níže.

Samostatná funkce

Zápis funkce do programu:

```
function jmeno_funkce(parametr:typ, parametr:typ):typ_navratove_hodnoty {
//zde se pise vlastní kod, co se ma provest
}
```

Jako jméno funkce může sloužit libovolný identifikátor, obvyklý úzus říká, že obsahuje-li více slov, je první písmeno každého dalšího slova psáno velkým písmenem, typicky dlouhyNazevNejakeFunkce.
Parametry do funkce nemusíme psát (definice `function jmeno(){ … }` je platná a běžně se používá, v našem programu tak máme definovanou `smrt()`).
Pokud parametry napíšeme, při volání funkce musíme zadat hodnoty stejného typu.
Funkce vrací jednu svoji hodnotu, její typ je zadán za dvojtečkou. Hodnotu, kterou vrátí, řekneme na konci kódu funkce pomocí parametru příkazu return. Nechceme-li žádnou hodnotu vracet, použijeme return bez argumentu; typ vracené hodnoty je pak Void.

Například:

```
function secti(a:Number, b:Number):Number {
  var s:Number;
  s=a+b
  return(s);
  }
//volani funkce:
trace(secti(3,5));
```

Napsali jsme funkci, která (po zavolání uvedeném níže) si do svých parametrů a, b vezme čísla 3 a 5, zavede novou lokální proměnnou s, do ní přiřadí a+b a následně příkazem return skončí s návratovou hodnotou s, která je rovněž typu číslo.
Parametry a, b se chovají též jako „lokální proměnné“, po skončení funkce se jejich hodnota zapomene. Popis lokální platnosti proměnných naleznete v DUMu č. 3 této sady, strana 4.

Pedagogická poznámka: V ActionScriptu 3.0 je nově přidána možnost „volitelných parametrů“ – napíšeme-li při definici do parametrů do některých pozic defaultní hodnoty, při volání pak danou pozici nemusíme využít (a použije se ona defaultní hodnota).
Například:

```
function pricti(a:Number, b:Number=1):Number {
  var s:Number;
  s=a+b
  return(s);
  }
//volani funkce:
trace(pricti(3)); // vrati 4
trace(pricti(3,2)); // vrati 5
```

My zde ale popisujeme verzi Macromedia Flash 8 s ActionScriptem 2.0, kde tato možnost ještě není. Obvykle ji studentům zmíním, aby věděli, že ve vyšších verzích ji mohou použít.

Při volání funkce je možné zadat i víc hodnot, než je počet argumentů zadaných v definici – hodnoty se předají, jen k nim nemáme přístup pomocí běžně pojmenovaných argumentů, ale pouze pomocí pole `arguments`, kde jsou všechny parametry uvedeny. Například:

```
function sectivse():Number {
  var i:Number;
  var s:Number = 0;
  for (i=0; i<arguments.length; i++) {s=s+arguments[i];}
  return(s);
}
//volani funkce
trace(sectivse(5,3,1,0,10));  // vrati 19
```

Zde jsme funkci žádné parametry při deklaraci neporučili, přesto při volání pět parametrů uvedeme, k nim přistupujeme for-cyklem, přičítáme je do proměnné s, kterou na konci vrátíme jako výstupní hodnotu funkce.

Kromě pole `arguments` lze občas použít i proměnnou `callee`, která odkazuje na právě prováděnou funkci, resp. `caller`, která odkazuje na funkci, která prováděnou funkci zavolala.

Předávat nemusíme jen čísla, ale i objekty, tak, jak je to v případě funkce drawCircle v DUMu č. 12:

```
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
```

Zde jsme kromě tří čísel předali i celý MovieClip (resp. odkaz na něj) – zde platí, že vlastnosti MovieClipu, které změníme uvnitř funkce (např. nakreslené čáry do něj) v objektu už zůstanou. (Srovnejte s číselným parametrem, např. x, který funguje jen jako vstupní parametr, tedy, změníme-li jej uvnitř funkce, změněná hodnota se zapomene…)

Funkce jako součást objektu – jeho metoda

Správně napsaný objektově orientovaný program by měl obsahovat pouze objekty a jejich metody, tedy funkce, které říkají, „co objekt umí“.
Vezmeme-li pravidlo zapouzdření do úplného extrému, nesmíme ani na vlastnost objektu sáhnout pouhým přiřazením, ale měli bychom pro ni mít funkci na změnu, tedy špatně je:

```
_root.obdelnik._width = 70;
```

zatímco správně je nadefinovat si metodu objektu obdélník (nazvěme ji zmensirku), která vlastnost mění, přitom do ní můžeme napsat i ošetření vstupu či jiné podmínky pro změnu:

```
_root.obdelnik.zmensirku = function (naco:Number):Void {
  // podminky, osetreni vstupu, napr. (naco>0), (naco<Stage.width-this._x)
  this._width=naco;
  return;  //zadnou hodnotu nevracime
  }
```

kterou kdykoliv zavoláme jako:

```
_root.obdelnik.zmensirku(70);
```

Namísto typu Void a prázdného return se občas hodí použít návratovou hodnotu typu Boolean – funkce bude vracet true při provedené změně či false při jakékoliv chybě. (Nebo naopak.)

Případně můžeme použít návratovou hodnotu typu Number – funkce bude vracet 0 při provedené změně či nenulové číslo znamenající kód chyby (proč změna nebyla provedena), který popíšeme v dokumentaci funkce.

V předchozí ukázce jsme viděli přiřazení nové funkce do objektu, „objekt jsme tedy naučili něco nového“. Stejným způsobem lze i předefinovat již existující metodu, např.:

```
_root.obdelnik.onEnterFrame = function () {trace("Probehl onEnterFrame objektu "+this._name) }
```

Bez ohledu na to, jaký kód `on(EnterFrame) {cokoliv}` bude v panelu Actions daného obdélníka při návrhu zapsán, po provedení výše uvedeného příkazu se přepíše na trace.

Uvedené definování či měnění obsahu funkce za běhu programu se hodí pro dynamicky generované objekty a též při totální změně chování objektu (typicky u vyšších levelů naší hry).