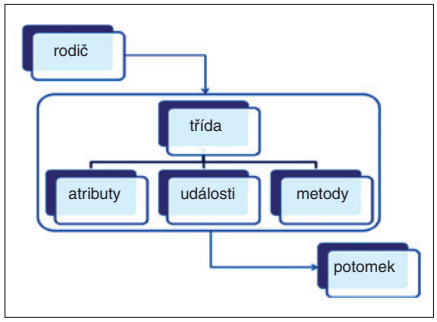
Objektově orientované programování v prostředí Matlab

V článku je popsána syntaxe objektově orientovaného programování v prostředí Matlab. Popis je doplněn ilustrativním příkladem. Článek předpokládá základní obecné znalosti metody objektově orientovaného programování a programovacího jazyka Matlab. Pro seznámení s objektovým programováním je vhodná kniha [1], odkud bylo převzato i české názvosloví objektového programování. Prostředí Matlab je popsáno ve [2].

V programovacím prostředí Matlab je od verze 7.6 (R2008a) zavedena nová syntaxe pro objektově orientované programování [3]. Programovací jazyk prostředí Matlab je od svých počátků procedurální, ačkoliv objektový přístup je patrný při práci s grafikou. Nová syntaxe objektového programování umožňuje snadno definovat třídy (*class*) a jejich atributy (*properties*), události (*events*) a metody (*methods*).



Obr. 1. Model objektového programování

Objektové programování je vhodné používat při práci na větším softwarovém projektu, ale také např. při navrhování automatizovaných systémů. Na rozdíl od procedurálního způsobu programování, kde se používají funkce, se při objektově orientovaném programování používají třídy a jejich instance. Třídu si lze představit jako určitou formu, pomocí které se dělají bábovky – instance třídy (někde též objekty). Každá bábovka bude mít tvar daný formou. Forma může být vyrobena podle jiné formy a zároveň může být použita jako předloha pro výrobu další formy – tato vlastnost se označuje jako dědičnost (*inheritance*). Model objektového programování je znázorněn na *obr. 1*.

Popis syntaxe

Pro definici tříd byla v prostředí Matlab zavedena čtyři nová klíčová slova: *classdef* (definice třídy), *properties* (atributy), *events* (události) a *methods* (metody). U těchto klíčových slov lze použít parametry, kterými je možné např. specifikovat přístup k atributům či metodám. Všechna shora uvedená klíčová slova označují začátek bloku zdrojového kódu. Konec bloku se vyznačuje klíčovým slovem *end*. Jak plyne z *obr. 1*, atributy, události a metody se vkládají do definice třídy. Jedna definice třídy může obsahovat několik bloků s atributy (popř. událostmi, metodami) s různými parametry. Parametry se zapisují do kulatých závorek za dané klíčové slovo ve formátu *nazevParametru* = *hodnota*.

Definice třídy (*classdef*)

Třídu si lze představit jako prototyp, podle kterého se vytvářejí jednotlivé instance třídy. Definice nové třídy se vytváří obdobně jako definice nové funkce do souboru s názvem shodným s názvem třídy/funkce. Ostatní náležitosti jsou od definice funkce již značně odlišné. Uvnitř třídy mohou být definovány bloky atributů, událostí a metod. Třída může využívat dědičnost – být rodičem (*superclass*) či potomkem (*subclass*) jiné třídy. Možnost vytvářet potomky dané třídy lze ovlivnit pomocí parametru *sealed* (logický typ).

Speciálním typem tříd jsou tzv. abstraktní třídy, u nichž nelze vytvořit instanci. Fungují pouze jako rodičovská třída pro dědění. Atributy a metody abstraktní třídy lze definovat pomocí atributu *abstract* (logický typ).

Přímo v prostředí Matlab jsou obsaženy tři abstraktní třídy, které lze využít při tvorbě vlastních tříd. Základní třída *handle* funguje jako rodičovská pro všechny třídy pracující s ukazateli (*handles*) na jednotlivé instance. Třída *handle* definuje metody pro porovnávání ukazatelů (*eq*, *lt*, *gt* atd.), pro vyhledávání instancí a atributů (*findobj*, *findprop*), pro práci s událostmi (*notify*, *addlistener*) a destruktory (*delete* – viz popis metod). Příklad založení nové třídy, která je potomkem abstraktní třídy *handle*, je ukázán na *obr. 2*.

Další abstraktní třída *hgsetget* je potomkem třídy *handle*. Dodefinovává metody *set* a *get*, které jsou hojně využívány při práci s grafickým prostředím. Poslední z abstraktních tříd je třída *dynamicprops*. Je také potomkem třídy *handle* a dodefinovává metodu *addprop*, která umožňuje dynamicky přidávat atributy k existujícím objektům.

Atributy (*properties*)

Do bloku atributů se zapisuje výčet proměnných a konstant dané třídy. Přístup k jednotlivým atributům lze řídit prostřednictvím specifikátorů přístupu. Ty jsou v prostředí Matlab realizovány pomocí parametrů *Access*, *GetAccess* a *SetAccess*. Podle nastavení specifikátorů přístupu mohou být atributy:

- veřejné (*public*): s atributy lze libovolně manipulovat,
- chráněné (*protected*): k atributům lze přistupovat pouze prostřednictvím metod dané třídy nebo jejích potomků,
- soukromé (*private*): k atributům lze přistupovat pouze při použití metod dané třídy.

Dále lze pomocí logických parametrů *Hidden* a *Transient* nastavit viditelnost. Pomocí taktéž logických parametrů *Constant* a *Dependent* je možné nastavit definovatelnost. Úplný výčet parametrů lze najít ve [3].

Ukázka definice bloku atributů je na *obr. 3*. Uvedený blok obsahuje celkem dva atributy.

```
classdef MojeTrida < handle
    ...
end
```

Obr. 2. Definice třídy „MojeTrida“ založené na abstraktní třídě „handle“

```
properties (SetAccess = private)
    prop1 = 10;
    prop2
end
```

Obr. 3. Definice bloku atributů

```
events
    nazevUdalosti
end
```

Obr. 4. Definice bloku událostí

První atribut *prop1* je rovnou definován a je mu přiřazeno číslo 10. Druhý atribut *prop2* je pouze deklarován (bude obsahovat prázdnou matici). Definice obou atributů mohou být změněny pouze při použití metod dané třídy, ale ke čtení jsou přístupné bez omezení.

Události (*events*)

Instance třídy může dát najevo, že nastala určitá událost. Na to mohou reagovat jiné instance (jiných tříd), které na danou událost čekají. U událostí lze opět specifikovat přístup. Okruh tříd, které mohou událost sledovat, lze omezit pomocí parametru *ListenAccess*. Obdobně lze nastavit i okruh tříd, které mohou událost nahlásit, a to parametrem *NotifyAccess*. Oba tyto parametry mohou nabývat stejných hodnot jako parametr *Access* u atributů. Ukázka definice bloku událostí je na *obr. 4*. Blok zde obsahuje pouze jednu událost, která je deklarována pomocí svého názvu *nazevUdalosti*.

Metody pro práci s událostmi lze podědit z abstraktní třídy *handle*. Událost je možné

vyvolat pomocí metody *notify* dané instance, které se předá název události tak, jak je zapsán v bloku událostí (*events* – viz *obr. 6*). Sledování události lze aktivovat pomocí metody *addlistener* (*obr. 7*), která vytvoří novou instanci třídy *event.listener*. Tento objekt kontroluje, zda sledovaný objekt nenahlásil danou událost. Jestliže ano, spustí předem definovanou akci.

Metody (*methods*)

Metody jsou obdoba funkcí a i způsob jejich definice tomu odpovídá. Jednotlivé metody se zapisují jako funkce v bloku *methods*. Začátek metody je uvozen hlavičkou začínající klíčovým slovem *function* a obsahující název metody a výčet vstupních a výstupních proměnných. První vstupní proměnná je ukazatel na instanci, která metodu vyvolala (neplatí u konstuktorů a nezávislých metod). Definice metody se ukončuje klíčovým slovem *end*.

Metody mohou měnit atributy, vyvolávat události či spouštět jiné funkce a metody. Mohou spouštět i metody jiné instance (i jiné třídy), popř. vyvolávat jejich události či měnit atributy.

Také u metody je možné specifikovat přístup pomocí parametru *Access*. Ochranu proti změně definice metody při dědění lze nastavit pomocí tzv. zapečetění (parametr *Sealed*). Nezávislé metody je možné definovat pomocí parametru *Static*. Tyto metody nepracují s instancí, která je vyvolala.

Zvláštní postavení mezi metodami mají konstruktor a destruktor. Konstruktor je metoda, která se spustí při vytváření nové instance třídy. Definuje se jako metoda s názvem shodným s třídou. Konstruktor lze použít k nastavení atributů, kontrole apod. Jestliže má konstruktor vstupní parametry, je nutné je při volání specifikovat. Výstupním parametrem je ukazatel na novou instanci. Destruktor se spouští těsně před vymazáním instance z paměti. Definuje se jako metoda s názvem *delete*. Je-li ukazatel na danou instanci uložen v proměnné, spuštěním destruktoru se vymaže instance, ale proměnná s ukazatelem zůstane zachována. Naopak při vymazání proměnné s ukazatelem pomocí funkce *clear* zůstane instance v paměti zachována. V takovém případě je třeba pro úplné odstranění instance i proměnné s ukazatelem nejdříve spustit destruktor (*delete*) a následně vymazat proměnnou s ukazatelem (*clear*) – celý proces je ukázán na *obr. 8*. Definice vlastního konstruktoru i destruktoru jsou nepovinné.

V příkladu definice bloku metod na *obr. 5* jsou definovány celkem dvě metody. První metoda *MojeTrida* je konstruktor – má název shodný s názvem třídy. Při vytvoření nové instance této třídy se tedy do atributu *prop2* uloží náhodné číslo. Ukazatel na tuto nově vznikající instanci je zde označen jako *hObject* a je výstupním parametrem konstruktoru. Druhá metoda *nazevMetody* nemá výstupní parametry. Prvním a v tomto případě jediným vstupním parametrem je ukazatel na instanci, jejíž metoda je volána.

```
methods
    function hObject = MojeTrida
        hObject.prop2 = rand;...
    end
    function nazevMetody(hObject)
        ...
    end
end
```

Obr. 5. Definice bloku metod

Příklad

K ilustraci předchozího výkladu následuje krátký příklad. Uvažujme situaci, kdy jsou v sérii k vypínači V zapojeny dvě žárovky označené Z1 a Z2. Při sepnutí vypínače se žárovky rozsvítí, při rozepnutí zhasnou.

Na *obr. 6* je úplný zdrojový kód souboru *vypinac.m*, který definuje třídu *vypinac* obsahující jeden atribut (*sepnuto*), jednu událost (*prepnuti*) a jednu metodu (*pouzit*). Atribut *sepnuto* je logického datového typu a jeho hodnota určuje, zda je vypínač sepnutý nebo rozepnutý. Hodnotu atributu *sepnuto* lze měnit pouze pomocí metod třídy *vypinac*. Jestliže někdo vypínač použije (sepne/rozepne), je negován atribut *sepnuto* a nahlášena událost *prepnuti*.

```
classdef vypinac < handle
    properties (SetAccess=private),
        sepnuto = false;
    end
    events, prepnuti, end
    methods
        function pouzit(hObject)
            hObject.sepnuto = ~hObject.sepnuto;
            hObject.notify('prepnuti');
        end
    end
end
```

Obr. 6. Zdrojový kód souboru „vypinac.m“

```
classdef zarovka < handle
    properties, sviti; hPrepnuti; end
    methods
        function hObject = zarovka(hVypinac)
            hObject.sviti = hVypinac.sepnuto;
            hObject.hPrepnuti = addlistener(hVypinac,'prepnuti',@hObject.zmena);
        end
        function zmena(hObject,src,evtdata)
            hObject.sviti = src.sepnuto;
        end
        function delete(hObject)
            hObject.hPrepnuti.delete;
        end
    end
end
```

Obr. 7. Zdrojový kód souboru „zarovka.m“

Definice třídy žárovka je uvedena na *obr. 7*. Tato třída obsahuje dva atributy (*sviti*, *hPrepnuti*) a tři metody (*zarovka*, *zmena*, *delete*), z nichž první je konstruktor a poslední destruktor. Při vytvoření nové instance třídy *zarovka* je žárovce přiřazen vypínač, atribut *sviti* se nastaví shodně se stavem vypínače a zároveň se vytvoří objekt třídy *event.listener* (kontrolor událostí), který bude hlídat událost *prepnuti* u vypínače. Ukazatel kontroloru události se uloží do atributu *hPrepnuti*. Při přepnutí vypínače bude tedy aktivována jeho událost *prepnuti*. Na to zareaguje kontrolor a spustí metodu *zmena* dané žárovky. Tato metoda má tři vstupní argumenty – ukazatel na žárovku, ukazatel na vypínač, který vyvolal událost, a ukazatel na kontrolor, který metodu spustil. Destruktor *delete* má za úkol v případě mazání instance třídy *zarovka* smazat i instanci kontroloru události.

```
V = vypinac;
Z1 = zarovka(V);
Z2 = zarovka(V);
V.prepnout
Z2.delete
clear('Z2')
```

Obr. 8. Příklad: vytvoření instancí a volání metod

Po úspěšném definování metod lze přikročit k vytvoření jednotlivých instancí (*obr. 8*). Nejdříve se vytvoří instance třídy *vypinac* a ukazatel se uloží do proměnné *V*. Následně jsou vytvořeny dvě instance třídy *zarovka* (*Z1* a *Z2*), které budou obě propojeny s vypínačem V. Při spuštění metody *prepnout* vypínače V se změní stav obou žárovek podle aktuálního stavu vypínače. Nakonec je ukázáno spuštění destruktoru žárovky Z2 a následné vymazání ukazatele na tuto žárovku z paměti řídicího systému.

Literatura:

[1] KEOGH, J. – MARIO, G.: *OOP bez předchozích znalostí*. Brno, Computer Press, 2006, 220 s., ISBN 80-251-0973-9.

[2] DOŇAR, B. – ZAPLATÍLEK, K.: *Matlab pro začátečníky*. Praha, BEN – technická literatura, 2003, 152 s., ISBN 80-7300-175-6.

[3] *MathWorks – Matlab and Simulink for Technical Computing* [on-line]. 2010 [cit. 2010-04-01]. Dostupné z <www.mathworks.com>.

Ing. Jan Hrubeš,
Ústav biomedicínského inženýrství,
Fakulta elektrotechniky a komunikačních technologií VUT v Brně
(hrubes@feec.vutbr.cz)