KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Michal Krupka
krupka.inf.upol.cz
michal.krupka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 3 ◇ poznámky k přednášce

7. Události

verze z 16. listopadu 2020

1 Jednoduché obsloužení vstupu z myši

Vstup z myši řešíme rovněž obsluhou zpětných volání. Nejprve naprogramujeme instalaci příslušného zpětného volání knihovny `micro-graphics` ve třídě `window` a zvážíme, co by měla okna v reakci na vstup z myši dělat.

V dokumentaci ke knihovně `micro-graphics` se dozvíme, že pro případy, kdy uživatel klikne do okna, knihovna dává k dispozici zpětné volání typu `:mouse-down`. Knihovna je volá se čtyřmi argumenty: oknem knihovny, do kterého uživatel klikl, symbol označující tlačítko myši (může být `:left`, `:center`, `:right`), které použil, a dvě souřadnice v okně bodu, na který klikl (podrobnosti v dokumentaci ke knihovně).

Příklad: test zpětného volání `mouse-down`

Zpětné volání můžeme otestovat takto:

```
CL-USER 1 > (setf w (mg:display-window))
#<MG-WINDOW 40200C3D43>

CL-USER 2 >
(mg:set-callback
  w
  :mouse-down (lambda (w button x y)
                (format t "~%Tlačítko: ~s. Souřadnice: (~s,~s)"
                        button x y)))
NIL
```

Nyní můžeme zkusit klikat do okna.

Příklad: instalace zpětného volání `:mouse-down`

Rozšiřme tedy definici třídy `abstract-window` o reakci na zpětné volání `:mouse-down`. Uděláme to jednoduše. Definujeme novou metodu `window-mouse-down` třídy `abstract-window`, jejímž úkolem bude reagovat na uživatelovo kliknutí do okna. Ve třídě `abstract-window` tato metoda nebude zatím dělat nic. Metodu `install-callbacks` třídy `abstract-window` rozšíříme o instalaci zpětného volání, které zašle zprávu `window-mouse-down` oknu.

```
(defmethod install-mouse-down-callback ((w abstract-window))
  (mg:set-callback
   (slot-value w 'mg-window)
   :mouse-down (lambda (mgw button x y)
                 (declare (ignore mgw))
                 (window-mouse-down
                  w
                  button
                  (move (make-instance 'point) x y))))
  w)

(defmethod install-callbacks ((w abstract-window))
  (install-display-callback w)
  (install-mouse-down-callback w))
```

Jak vidíme, zpráva `window-mouse-down` bude zasílána se dvěma argumenty: symbolem označujícím stisknuté tlačítko myši a bodem posunutým na místo, kam uživatel klikl. Prozatím prázdná definice metody:

```
(defmethod window-mouse-down ((w abstract-window) button position)
  w)
```

Příklad: test klikání

Po vyhodnocení definic z předchozího příkladu (samozřejmě s načtenou knihovnou `micro-graphics` a zdrojovým kódem z předchozí kapitoly) můžeme zpětné volání otestovat. Napíšeme třídu `click-window-1`, jejíž instance budou obsahovat kolečko, které se vždy po kliknutí levým tlačítkem přesune pod myš. Použijeme také funkci `make-test-circle` z předchozí přednášky.

```
(defun make-test-circle ()
  (move (set-radius (set-thickness (set-color
                                    (make-instance 'circle)
                                    :darkslategrey)
                                   5)
                    55)
        148
        100))

(defclass test-window-1 (window)
  ())

(defmethod initialize-instance ((w click-window-1) &key)
  (call-next-method)
```

```
  (set-shape w (make-test-circle)))

(defmethod window-mouse-down
           ((w click-window-1) button position)
  (when (eql button :left)
    (let ((c (center (shape w))))
      (move (shape w)
            (- (x position) (x c))
            (- (y position) (y c)))))
  w)
```

Metodu `window-mouse-down` třídy `click-window-1` by bylo jednodušší napsat tak, že by se přímo nastavily souřadnice středu kruhu na souřadnice bodu `position`:

```
(defmethod window-mouse-down
           ((w click-window-1) button position)
  (when (eql button :left)
    (set-x (set-y (center (shape w))
                  (y position))
           (x position)))
  w)
```

Čtenář, který se zabýval úkoly k předchozí přednášce„ ví, že to bohužel zatím nejde.

2 Zpravení grafického objektu o kliknutí

Zatím jsme řešili odezvu na kliknutí pouze na úrovni okna. Nyní se podíváme, jak lze reagovat na kliknutí na konkrétní grafický objekt v okně. Ve shodě s principem samostatnosti by okno mělo postupovat tak, že o kliknutí zpraví přímo objekt, na který uživatel kliknul, a nechá vhodnou reakci na něm.

Příklad: testování zásahu

Při klikání do okna je především třeba umět zjistit, do které části okna uživatel klikl. K tomu si zavedeme zprávu `contains-point-p`, která slouží k tzv. *hit-testingu* (*testování zásahu*). Jako argument bude mít bod a grafický objekt by na ni měl odpovědět, zda tento bod leží uvnitř něj.

Než si ukážeme implementaci některých jejích metod, řekneme si jasněji, jaký *kontrakt* mají metody zprávy dodržovat. Už jsme si řekli, že argumentem zprávy je bod (to by tedy byla *prekondice* pro metodu ve třídě `shape`; potomci by ji mohli nějak vhodně rozšiřovat, ale my to dělat nebudeme). Výsledek zprávy *postkondici* budeme formulovat poněkud volněji, ale jasně: zpráva by měla vracet *Pravdu* (hodnotu různou od `nil` právě když testovaný bot leží v grafické reprezentaci objektu,

tedy uvnitř oblasti, kterou v okně objekt pokrývá. (Všimněte si, že tím jsme dodali podmínky na metodu `draw` třídy `shape` a jejích potomků.)

Implementace metod zprávy si ukážeme u tříd `shape`, `circle` a `picture`. Ve třídě `shape` nemáme dost informací na to, abychom mohli metodu `contains-point-p` úplně implementovat. Abychom dodrželi kontrakt, musíme zabránit, aby se metoda vůbec dala úspěšně zavolat.

```
(defmethod contains-point-p ((shape shape) point)
  (error "Method has to be rewritten."))
```

U třídy `circle` se rozhodujeme podle vlastnosti `filledp` a pak porovnáme vzdálenost testovaného bodu od středu kruhu s poloměrem. Vzdálenost bodů zjistíme pomocnou funkcí `point-dist` (najdete ji ve zdrojovém kódu).

```
(defmethod contains-point-p ((circle circle) point)
  (let ((dist (point-dist (center circle) point))
        (half-thickness (/ (thickness circle) 2)))
    (if (filledp circle)
        (<= dist (radius circle))
      (<= (- (radius circle) half-thickness)
          dist
          (+ (radius circle) half-thickness)))))
```

U třídy `picture` zjistíme, zda testovaný bod leží v některém podobrázku:

```
(defmethod contains-point-p ((pic abstract-picture) point)
  (find-if (lambda (item)
             (contains-point-p item point))
           (items pic)))
```

Zjistit, zda bod leží v polygonu, není jednoduché. Příslušná metoda to nechává na knihovně `micro-graphics`, která má k tomuto účelu implementovanou funkci.

Příklad: možné cíle klikání

Pomocí zprávy `contains-point-p` nyní můžeme najít grafický objekt v okně, do kterého uživatel klikl. Z hledání ovšem budeme chtít některé objekty vyloučit. Většinou to budou instance třídy `abstract-picture`. Pokud bychom vyjímečně chtěli učinit i obrázek cílem kliknutí, budeme to signalizovat hodnotou vlastnosti, kterou si za tímto účelem zavedeme.

Bude to vlastnost `solidp`. Jejím účelem bude informovat, zda je objekt určen jako příjemce kliknutí myši. Objekty, které nejsou obrázky, budou tuto vlastnost

vždy mít nastavenu na *Pravdu*, u obrázků bude hodnota vždy *Nepravda*, potomci třídy `abstract-picture` to ale budou moci změnit. Definice metod pro vlastnost `solidp` je tedy jednoduchá:

```
(defmethod solidp ((shape shape))
  t)

(defmethod solidp ((pic abstract-picture))
  nil)
```

Vlastnost `solidp` je hezkou ukázkou vlastnosti, jejíž hodnota nezávisí na konkrétní instanci, ale pouze na její třídě. Je pouze ke čtení a nemusí být uložena v žádném slotu (obojí mohou potomci změnit).

Příklad: nalezení objektu pod myší

Nyní se podíváme, jak okno zjistí, na který objekt uživatel klikl. Půjde o objekt vnořený libovolně hluboko v hierarchii obrázků, který má hodnotu vlastnosti `solidp` rovnou *Pravdě*. Hledání proběhne ve dvou krocích. Nejprve shromáždíme do seznamu všechny objekty v okně s nastavenou vlastností `solidp` a pak v něm najdeme první, který obsahuje bod, na nějž uživatel klikl.

Vytvoření seznamu podobjektů objektu s nastavenou vlastností `solidp` (včetně případně objektu samého):

```
(defmethod solid-shapes ((shape shape))
  (if (solidp shape)
      (list shape)
    (solid-subshapes shape)))

(defmethod solid-subshapes ((shape shape))
  (error "Method has to be rewritten."))

(defmethod solid-subshapes ((shape abstract-picture))
  (mapcan 'solid-shapes (items shape)))
```

Poslední metoda používá funkci `mapcan`, kterou zatím neznáte. Funkce pracuje jako funkce `mapcar`, ale jako výsledek očekává seznam seznamů, které pak spojí do jednoho seznamu. Efekt volání

```
(mapcan function list)
```

je zhruba stejný jako

```
(apply 'append (mapcar function list))
```

Metoda `solid-subshapes` třídy `shape` nebude nikdy volána pro instanci, pro kterou je hodnota `solidp` *Pravda*. V případě, že v nějaké třídě může hodnota `solidp` být *Neravda*, musí třída přepsat i metodu `solid-subshapes`. Tak to děláme ve třídě `picture`.

Nyní je již jednoduché napsat metodu `window-mouse-down` třídy `abstract-window`. Pokud metoda najde objekt s nastavenou vlastností `solidp`, na který uživatel klikl, pošle mu (v metodě `mouse-down-inside-shape`) zprávu `mouse-down`. Pokud ne, pošle oknu zprávu `mouse-down-no-shape`.

```
(defmethod find-clicked-shape ((w abstract-window) position)
  (when (shape w)
    (find-if (lambda (shape) (contains-point-p shape position))
             (solid-shapes (shape w)))))

(defmethod mouse-down-inside-shape
           ((w abstract-window) shape button position)
  (mouse-down shape button position)
  w)

(defmethod mouse-down-no-shape
           ((w abstract-window) button position)
  w)

(defmethod window-mouse-down ((w abstract-window) button position)
  (let ((shape (find-clicked-shape w position)))
    (if shape
        (mouse-down-inside-shape w shape button position)
      (mouse-down-no-shape w button position))))
```

Příklad: `mouse-down`, první verze

Grafické objekty nemusejí dělat v reakci na kliknutí nic. Metoda `mouse-down` ve třídě `shape` tedy bude zatím prázdná.

```
(defmethod mouse-down ((shape shape) button position)
  shape)
```

Příklad: `click-circle`, první verze

Jeden z příkladů k této přednášce je třída `click-circle`, která definuje metodu `mouse-down` tak, aby kolečko po kliknutí změnilo barvu. Prozatímní definice je

tato (funkce `color:make-rgb` je k dispozici v LispWorks, vytvoří barvu o daných komponentách):

```
(defun random-color ()
  (color:make-rgb (random 1.0)
                  (random 1.0)
                  (random 1.0)))


(defclass click-circle (circle) ())

(defmethod mouse-down ((circ click-circle) button position)
  (set-color circ (random-color)))
```

3 Události

Na minulé přednášce jsme definovali zprávy `ev-changing` a `ev-change` jejich metody ve třídě `abstract-window`, pomocí nichž grafické objekty informují své okno o změně. Zprávy jsme nazvali *událostmi*. Proti obecným zprávám se události vyznačují následujícími zvláštnostmi:

1. Událost objekt posílá výhradně svému oknu — a obecněji svému majiteli, jak si řekneme příště.
2. Účelem události je majitele o něčem informovat, nikoli mu něco přikazovat.
3. Objekt tedy nesmí očekávat, že a jak majitel na přijetí události zareagoval (nemusel udělat vůbec nic).

Stejný princip se nám bude nyní hodit i na klikání do objektu. O kliknutí necháme objekt informovat jeho okno prostřednictvím nové události `ev-mouse-down`.

Jména událostí

Kvůli odlišení událostí od ostatních zpráv musí jejich jméno vždy začínat předponou `ev-`.

Událost `ev-mouse-down` se bude posílat oknu se třemi argumenty: odesílatelem události, symbolem určujícím stisknuté tlačítko myši a bodem určujícím pozici (v souřadnicích okna), na kterou uživatel klikl. Nová metoda `mouse-down` třídy `shape` by tedy vypadala takto:

```
(defmethod mouse-down ((shape shape) button position)
  (when (window shape)
    (ev-mouse-down (window shape) shape button position)))
```

K události `ev-mouse-down` a k vysvětlení, proč ji chceme z metody `mouse-down` posílat, se ještě za chvíli vrátíme.

Teď se ještě na události podívejme koncepčně. Připomeňme naši metodu `change` třídy `shape` z minulé přednášky:

```
(defmethod change ((shape shape))
  (when (window shape)
    (ev-change (window shape) shape))
  shape)
```

Když ji porovnáme s výše uvedenou metodou `mouse-down`, shledáme, že jsou v něčem podobné. Obě posílají nějakou událost, rozdíl je jen v tom, že každá jinou. Zavedeme tedy navíc jednu vrstvu abstrakce, což se nám vyplatí (jak to bývá) později i z jiných důvodů. Napíšeme metodu `send-event`:

```
(defmethod send-event ((s shape) event &rest event-args)
  (when (window s)
    (apply event (window s) s event-args))
  s)
```

a metody `mouse-down` a `change` upravíme:

```
(defmethod mouse-down ((shape shape) button position)
  (send-event shape 'ev-mouse-down button position))

(defmethod change ((shape shape))
  (send-event shape 'ev-change))
```

Stejně opravíme i metodu `changing`:

```
(defmethod changing ((shape shape))
  (send-event shape 'ev-changing))
```

Posílání událostí v metodách `mouse-down`, `changing` a `change`

Metody `mouse-down`, `changing` a `change` vždy posílají i příslušnou událost `ev-mouse-down`, `ev-changing` a `ev-change`. Pokud jsou metody v potomcích třídy `shape` přepsány, musí tedy vždy volat zděděnou metodu.

Nyní ještě ke slíbenému vysvětlení k události `ev-mouse-down`. Reakci na kliknutí do grafického objektu můžeme nyní naprogramovat dvěma způsoby:

1. Přepsat metodu `mouse-down` třídy objektu.
2. Přepsat metodu `ev-mouse-down` třídy okna.

První možnost jsme využili u třídy `click-circle`. Kolečko mění barvu v reakci na kliknutí. Je to jeho vlastnost jakožto instance třídy `click-circle`. Není důvod tuto funkčnost přenášet na okno přes událost — ono by to ani obecně nešlo, protože podle charakteristiky událostí, kterou jsme si uvedli, se nesmíme spoléhat na to, jaká bude reakce okna na událost.

Druhá možnost je využita v příkladě třídy `circle-with-arrow-window`, který máte k dispozici k této přednášce. Zde objekt (polygon ve tvaru šipky) funguje čistě jako tlačítko. Jak víme z reality, tlačítka nevědí o tom, jaká akce se po stisknutí má spustit. Pouze o stisknutí informují. Výhodou tohoto přístupu je, že když chceme použít grafický objekt jako tlačítko, nemusíme pro ně definovat novou třídu. Stačí definovat třídu pro okno, ve kterém se objekt nachází.

Více se o událostech dozvíme na příští přednášce.

Otázky a úkoly na cvičení

1. Napište třídu `polygon-window`, která bude potomkem třídy `abstract-window`. Klikáním do okna se bude vytvářet nový polygon: po každém kliknutí levým tlačítkem se přidá do polygonu vrchol, po kliknutí pravým tlačítkem se všechny vrcholy smažou. Třída bude přepisovat metodu `window-mouse-down` bez volání zděděné metody.
2. Definujte třídu `circle-with-arrows-window`, jejíž instance budou obsahovat kolečko a dvě šipky: doleva a doprava. Kliknutí na šipku posune kolečko směrem, kterým šipka ukazuje.
3. Definujte potomka `delete-item-picture-window` třídy `window` s následující funkčností: pokud je `shape` okna obrázek a uživatel klikne na nějaký jeho prvek, prvek se vypustí ze seznamu `items` obrázku. Můžete předpokládat, že obrázek neobsahuje další podobrázky.
4. Využijte vhodného zpětného volání knihovny `micro-graphics` ke sledování pohybu myši. Objekt, do kterého uživatel vjede kurzorem, vygeneruje událost `ev-mouse-enter`, objekt, ze kterého kurzor vyjede, událost `ev-mouse-leave`. V tomto příkladě můžete upravovat existující třídy naší grafické knihovny (zejména to zřejmě budou třídy `abstract-window` a `shape`).