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

**Jan Laštovička**
https://www.inf.upol.cz/lide/jan-lastovicka
jan.lastovicka@upol.cz
17. listopadu 12, 771 46 Olomouc

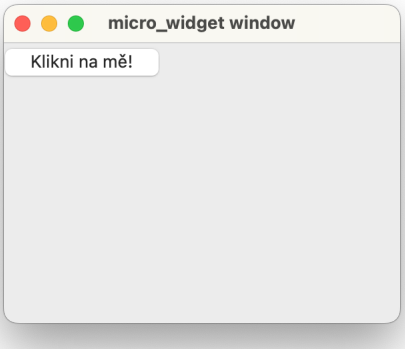
**Úvod do programovacích stylů ◇ poznámky k přednášce**

# 5. Události

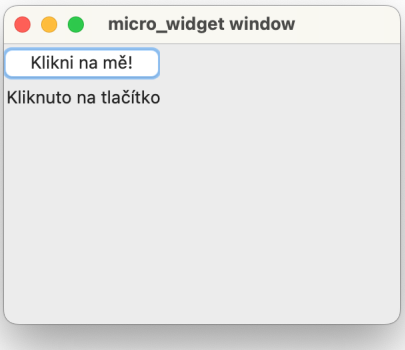
verze z 23. října 2024

## 1 Motivace

Začneme příkladem. Chceme vytvořit tlačítko:




které zobrazí text poté, co se na něj klikne:




Splnit zadání by nyní bylo velmi nepohodlné. Potřebovali bychom vytvořit dvě nové třídy: jednu pro tlačítko (`DisplayTextButton`) a druhou pro skupinu tlačítka s popiskem (`LabeledButton`). Naše tlačítko by mělo vlastnost `labeled_button`, kde hodnotou by byla skupina, ve které se nachází. Při stisku tlačítka by tlačítko

zaslalo skupině zprávu vyžadující zobrazení textu. Třída pro tlačítko by vypadala následovně.

```
class DisplayTextButton(Button):
    def __init__(self):
        super().__init__()
        self.labeled_button = None

    def set_labeled_button(self, labeled_button):
        self.labeled_button = labeled_button
        return self

    def get_labeled_button(self):
        return self.labeled_button

    def button_clicked(self):
        super().button_clicked()
        labeled_button = self.get_labeled_button()
        labeled_button.display_text()
        return self
```

Skupina by vytvořila tlačítko a popisek a tlačítku nastavila sebe jako jeho skupinu. Obsluha zprávy `display_text` by zajistila zobrazení textu.

```
class LabeledButton(Group):
    def __init__(self):
        super().__init__()
        button = DisplayTextButton().set_text("Klikni na mě!")
        button.set_labeled_button(self)
        label = Label().move(0, 30)
        self.set_items([button, label])

    def get_label(self):
        return self.get_items()[1]

    def display_text(self):
        self.get_label().set_text("Kliknuto na tlačítko")
        return self
```

Zbývá vytvořit obsah okna.

```
window = Window()
labeled_button = LabeledButton()
window.set_widget(labeled_button)
```

Přestože řešení je funkční a při stisku tlačítka dojde k zobrazení textu, řešení obsahuje vážnou chybu. Tlačítko je obsažené ve skupině a tedy je jí podřízené, přesto

může ke své skupině přistupovat přímo skrz vlastnost `labeled_button`. Správně by objektový systém měl nabízel obecný mechanizmus, kterým by tlačítko skupinu informovalo, že na něj bylo kliknuto. Skupina by mohla ale nemusela na získanou informaci reagovat. Obecně bychom chtěli, aby objekt mohl informovat o nějaké skutečnosti všechny objekty, které jsou za něj zodpovědné.

Vrátíme se na začátek a začneme definicí třídy pro skupinu:

```
class LabeledButton(Group):
    def __init__(self):
        super().__init__()
        button = Button().set_text("Klikni na mě!")
        label = Label().move(0, 30)
        self.set_items([button, label])

    def get_label(self):
        return self.get_items()[1]

    def display_text(self):
        self.get_label().set_text("Kliknuto na tlačítko")
        return self
```

Všimněme si, že narozdíl od předchozího řešení je tlačítko přímou instancí třídy `Button`. Zobrazení textu má stále na starosti metoda `display_text`.

Instanci třídy vložíme do okna:

```
window = Window()
labeled_button = LabeledButton()
window.set_widget(labeled_button)
```

Chceme zajistit, aby se objektu `labeled_button` zaslala zpráva `display_text` poté, co uživatel klikne na tlačítko. Tlačítko ale nemá přímý přístup k instanci třídy `LabeledButton`, ve které je obsaženo. Přidáme objektu možnost informovat všechny objekty, které za něj mají zodpovědnost o nějaké skutečnosti.

Ovládací prvky a okna souhrnně nazýváme **objekty uživatelského rozhraní**. Nejprve si definujeme vztah **býti přímo zodpovědný** mezi objekty uživatelského rozhraní. Okno je přímo zodpovědné za svůj obsah. Skupina je přímo zodpovědná za své prvky. Například za popisek z předchozího příkladu přímo zodpovídá skupina `labeled_button`, za kterou je přímo zodpovědné okno `window`.

Dále si zavedeme vztah **býti zodpovědný** mezi objekty objekty uživatelského rozhraní. Každý objekt je zodpovědný sám za sebe. Pokud je *`object1`* přímo zodpovědný za objekt *`object2`*, pak je za něj zodpovědný. Nakonec pokud je *`object1`* zodpovědný za *`object2`* a *`object2`* je zodpovědný za *`object3`*, pak je i *`object1`* zodpovědný za *`object3`*. Například za popisek z předchozího příkladu je zodpovědný on sám, skupina `labeled_button` a okno `window`.

Pokud objekt potřebuje informovat všechny objekty, které jsou za něj zodpovědné, o nějaké skutečnosti **zašle událost**. Konkrétněji objekt *sender* zašle událost *event* s argumenty *arg1*, *arg2*, ... Událost se nejprve zpracovává v objektu *sender*.

Zpracování události v objektu *object* probíhá ve dvou krocích:

1. Pokud objekt *object* rozumí zprávě *event*, pak je mu zaslána zpráva *event* s argumenty *sender*, *arg1*, *arg2*, ...
2. Pokud je za objekt *object* přímo zodpovědný objekt *object2*, pak pokračuje zpracování události v objektu *object2*.

Odesílatel události se tedy stane prvním argumentem zaslané zprávy. Například tlačítko zašle událost `ev_button_click` v případě, že na něj uživatel klikne. Jména událostí vždy začínají písmeny `ev` (což je zkráceně *event* – událost).

K zajištění zobrazení textu v našem příkladě stačí reagovat na událost `ev_button_click`. Do třídy `LabeledButton` doplníme její obsluhu:

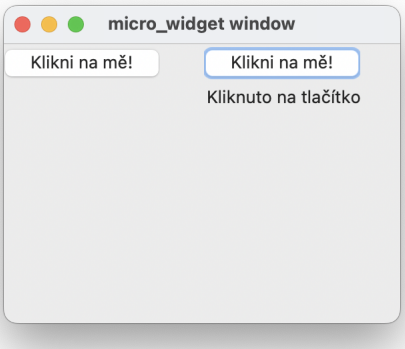
```
def ev_button_clicked(self, sender):
    self.display_text()
```

Pokud klikneme na tlačítko s textem `Klikni na mě!`, zašle se událost `ev_button_click`, tlačítko nerozumí zprávě `ev_button_click`, ale je za něj přímo zodpovědná skupina `labeled_button`, kde pokračuje zpracování události. Objekt `labeled_button` rozumí zprávě `ev_button_click`, proto je mu zaslána. To způsobí zavolání výše uvedené metody. Protože je za objekt `labeled_button` přímo zodpovědné okno `window`, pokračuje zpracování zprávy sem. Okno ale nerozumí zprávě `ev_button_click` a ani za něj není nikdo zodpovědný. Proto zde zpracování události končí.

Tlačítko s popiskem se chová nezávisle. Je tedy možné vytvořit okno obsahující dvě instance třídy `LabeledButton`:

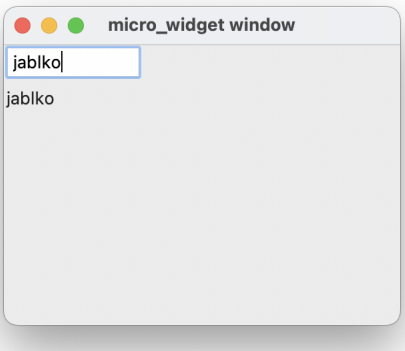
```
window = Window()
button1 = LabeledButton()
button2 = LabeledButton().move(150, 0)
buttons = Group().set_items([button1, button2])
window.set_widget(buttons)
```

Kliknutí na pravé tlačítko neovlivní levé tlačítko:




Textové pole zašle událost `ev_entry_change` poté, co došlo k jeho změně. Změnu mohl způsobit jak uživatel editací pole, tak program zasláním zprávy `set_text`.

Představme si, že chceme změnit popisek tak, aby kopíroval změnu textového pole:




Vytvoříme si třídu pro skupinu obsahující textové pole i popisek:

```
class LabeledEntry(Group):
    def __init__(self):
        super().__init__()
        entry = Entry()
        label = Label().move(0, 30)
        self.set_items([entry, label])

    def get_entry(self):
        return self.get_items()[0]

    def get_label(self):
        return self.get_items()[1]
```

Přidáme do ní metodu, zajišťující kopírování textu z pole do popisku:

```
def ensure_label_text(self):
    entry = self.get_entry()
    label = self.get_label()
    label.set_text(entry.get_text())
```

Nyní stačí přidat obsluhu události `ev_entry_change`:

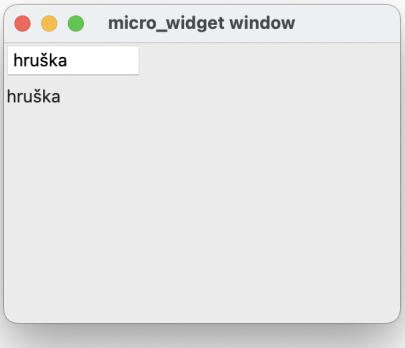
```
def ev_entry_change(self, sender):
    self.ensure_label_text()
```

Program spustíme vytvořením instance třídy a vložením do okna:

```
window = Window()
labeled_entry = LabeledEntry()
window.set_widget(labeled_entry)
```

Text popisku i pole bude stejný i po programové změně textu pole:

```
labeled_entry.get_entry().set_text("hruška")
```




Každý objekt uživatelského rozhraní rozumí zprávě *`object`*`.send_event(`*`event`*`,` *`arg1`*`,` *`arg2`*`,` ...), která zašle událost *`event`* s argumenty *`arg1`*, *`arg2`*, ...

Můžeme například vytvořit třídu pro textové pole, které bude zasílat událost `ev_entry_empty` při každé změně textu na prázdný řetězec:

```
class EmptyTextEntry(Entry):
    def ev_entry_change(self, sender):
        if self.get_text() == "":
            self.send_event("ev_entry_empty")
```

Vytvoříme třídu obsluhující událost `ev_entry_empty`:

```
class LoggingEmpty(Group):
    def ev_entry_empty(self, sender):
        print("Entry is empty")
```

Po vytvoření instancí:

```
window = Window()
entry = EmptyTextEntry()
group = LoggingEmpty().set_items([entry])
window.set_widget(group)
```

napsání a smazání obsahu pole vytiskne:

```
Entry is empty
```

Nahlásí se i změna provedená programem:

```
>>> entry.set_text("A")
<EmptyTextEntry object at 0x10d307fd0>
>>> entry.set_text("")
Entry is empty
<EmptyTextEntry object at 0x10d307fd0>
```

Všimněte si, že první změna textu pole na `A` nebyla hlášena.

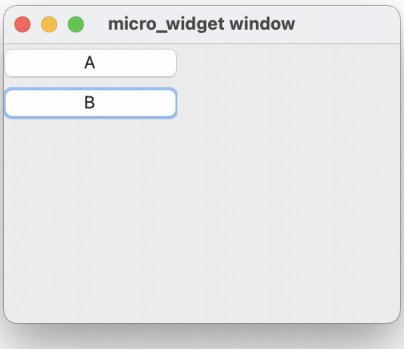
V souboru `omw2.pdf` se nalézá druhý manuál k objektové knihovně pro uživatelské rozhraní `omw`. Oproti prvnímu manuálu jsou zde navíc popsány události knihovny.

# Otázky a úkoly na cvičení

1. Vytvořte třídu `Buttons`, která může obsahovat tlačítka. Zařiďte, aby instance třídy vytiskla text každého tlačítka, které obsahuje, v moment, kdy na něj uživatel klikne. Nemusíte řešit kontrolu obsahu instancí třídy `Buttons`. Například kód:

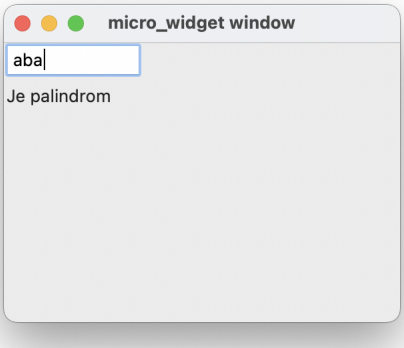
   ```
   window = Window()
   button1 = Button().set_text("A")
   button2 = Button().set_text("B").move(0, 30)
   buttons = Buttons().set_items([button1, button2])
   window.set_widget(buttons)
   ```

   Vytvoří dvě tlačítka:

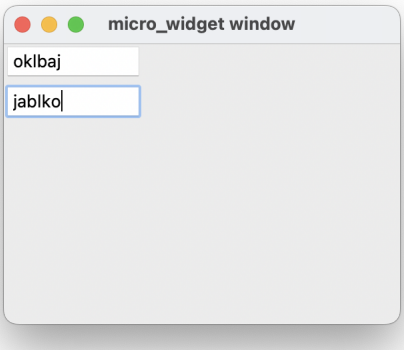



Klik na spodní tlačítko způsobí tisk písmena `B`.

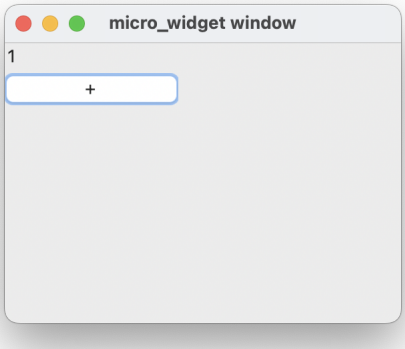
2. Vytvořte textové pole pod kterým bude popisek informující o tom, zda je text v poli palindromem. Například:



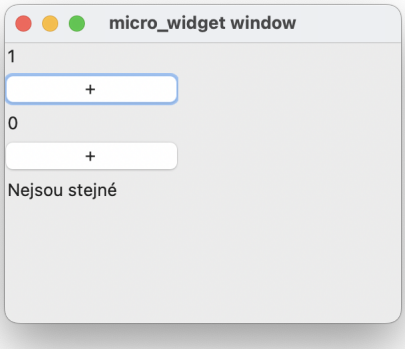

3. Vytvořte dvě textová pole, kde obsah jednoho je vždy pozpátku čtený obsah druhého. Musí být možné změnit text prvního i druhého pole. Například:



4. Vytvořte počítadlo, které při kliku na tlačítko inkrementuje svoji hodnotu:




5. Stvořte dvě počítadla s popiskem, který bude hlásit, zda jsou hodnoty počítadel stejné:




   Text popisku musí být správě i v případě, že hodnotu počítadla změníme programově. Například pokud by v proměnné `counter2` bylo dolní počítadlo z předchozího obrázku, pak nastavení hodnoty následujícím výrazem musí způsobit změnu textu popisku.

   ```
   counter2.set_value(1)
   ```

6. Vytvořte třídu pro položku formuláře, která bude kontrolovat vstup uživatele při psaní. Například na zadání telefonního čísla, emailu, ...

7. Vytvořte formulář dle vaší volby s tlačítkem pro odeslání. Například formulář pro zadání osobních informací. Položky formuláře se budou kontrolovat, až při kliku na tlačítko pro odeslání. Pokud jsou uživatelem zadané hodnoty v pořádku, údaje z formuláře se vytisknou do konzole.