Základní programovací paradigmata

# Přednáška 4. Dědičnost

Jan Laštovička

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

# Obsah

# Opakující metody

# Opakující metody

Například:

```
def get_color(self):
    return self.color

def set_color(self, color):
    self.color = color
    return self
```

# Opakující metody

Například:

```
def get_color(self):
    return self.color

def set_color(self, color):
    self.color = color
    return self
```

V třídách:

- `Point`
- `Circle`
- `Polygon`

# Opakující se definice atributů

# Opakující se definice atributů

Například:

```
self.color = "black"
```

# Opakující se definice atributů

Například:

```
self.color = "black"
```

V třídách:

- `Point`
- `Circle`
- `Polygon`

# Nemáme obecný typ grafických objektů

Metoda třídy `Picture`:

```
def check_item(self, item):
    if not (isinstance(item, Point)
            or isinstance(item, Circle)
            or isinstance(item, Polygon)
            or isinstance(item, Picture)):
        raise ValueError("Invalid picture element type.")
```

# Podobnost tříd `Polygon` a `Picture`

Atribut `vertices` je podobný atributu `items`:

```
class Picture:
    def __init__(self):
        self.items = []


class Polygon:
    def __init__(self):
        self.vertices = []
```

# Podobnost tříd `Polygon` a `Picture`

# Podobnost tříd `Polygon` a `Picture`

- podobné způsoby transformace

# Podobnost tříd `Polygon` a `Picture`

- podobné způsoby transformace
- například posun

# Podobnost tříd `Polygon` a `Picture`

- podobné způsoby transformace
- například posun

`Polygon`:

```
def move(self, dx, dy):
    for vertex in self.get_vertices():
        vertex.move(dx, dy)
    return self
```

# Podobnost tříd `Polygon` a `Picture`

- podobné způsoby transformace
- například posun

`Polygon`:

```
def move(self, dx, dy):
    for vertex in self.get_vertices():
        vertex.move(dx, dy)
    return self
```

`Picture`:

```
def move(self, dx, dy):
    for item in self.get_items():
        item.move(dx, dy)
    return self
```

# Obsah

# Princip dědičnosti

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek
- přímá instance

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek
- přímá instance

Objekt:

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek
- přímá instance

Objekt:

- přímou instancí jedné třídy $C$

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek
- přímá instance

Objekt:

- přímou instancí jedné třídy $C$
- instancí třídy $C$ a jejích předků

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek
- přímá instance

Objekt:

- přímou instancí jedné třídy $C$
- instancí třídy $C$ a jejích předků

Třída získá (zdění):

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek
- přímá instance

Objekt:

- přímou instancí jedné třídy $C$
- instancí třídy $C$ a jejích předků

Třída získá (zdění):

- definice všech atributů předků

# Princip dědičnosti

*Umožníme vytvářet potomky třídy.*

- třídy tvoří stromovou hierarchii
- přímý potomek
- předek
- přímý předek
- přímá instance

Objekt:

- přímou instancí jedné třídy $C$
- instancí třídy $C$ a jejích předků

Třída získá (zdění):

- definice všech atributů předků
- definice všech metod předků

# Třída Shape

# Třída Shape

- grafický objekt

# Třída Shape

- grafický objekt

```
class Shape:
    def __init__(self):
        self.color = "black"
        self.thickness = 1
        self.filled = False
```

# Metody pro vlastnosti třídy Shape

Shape:

```
def get_color(self):
    return self.color

def set_color(self, color):
    self.color = color
    return self

def get_thickness(self):
    return self.thickness

def set_thickness(self, thickness):
    self.thickness = thickness
    return self
```

# Metody pro vlastnosti třídy Shape

Shape:

```
def get_filled(self):
    return self.filled

def set_filled(self, filled):
    self.filled = filled
    return self
```

# Pravidlo *is-a*

# Pravidlo *is-a*

*Je-li třída $D$ potomkem třídy $C$, pak věta „každé $D$ je $C$“ musí dávat smysl.*

# Pravidlo *is-a*

*Je-li třída $D$ potomkem třídy $C$, pak věta „každé $D$ je $C$“ musí dávat smysl.*

Splňuje:

# Pravidlo *is-a*

*Je-li třída $D$ potomkem třídy $C$, pak věta „každé $D$ je $C$“ musí dávat smysl.*

Splňuje:

- Každý bod *je* grafický útvar. (Every point *is a* shape.)

# Pravidlo *is-a*

*Je-li třída $D$ potomkem třídy $C$, pak věta „každé $D$ je $C$“ musí dávat smysl.*

Splňuje:

- Každý bod *je* grafický útvar. (Every point *is a* shape.)
- Každý kruh *je* grafickým útvarem.

# Pravidlo *is-a*

*Je-li třída $D$ potomkem třídy $C$, pak věta „každé $D$ je $C$“ musí dávat smysl.*

Splňuje:

- Každý bod *je* grafický útvar. (Every point *is a* shape.)
- Každý kruh *je* grafickým útvarem.

Nesplňuje:

# Pravidlo *is-a*

*Je-li třída $D$ potomkem třídy $C$, pak věta „každé $D$ je $C$“ musí dávat smysl.*

Splňuje:

- Každý bod *je* grafický útvar. (Every point *is a* shape.)
- Každý kruh *je* grafickým útvarem.

Nesplňuje:

- Každá volant *není* automobil.

# Pravidlo *is-a*

*Je-li třída $D$ potomkem třídy $C$, pak věta „každé $D$ je $C$“ musí dávat smysl.*

Splňuje:

- Každý bod *je* grafický útvar. (Every point *is a* shape.)
- Každý kruh *je* grafickým útvarem.

Nesplňuje:

- Každá volant *není* automobil.
- Každý grafický útvar *není* bod.

# Určení přímého předka

```
class name(parent):
    def __init__(self):
        super().__init__()
        ⋮
```

*name*: jméno nové třídy
*parent*: jméno existující třídy

# Třída Point a třída Circle

```
class Point(Shape):
    def __init__(self):
        super().__init__()
        self.x = 0
        self.y = 0
```

```
class Circle(Shape):
    def __init__(self):
        super().__init__()
        self.center = Point()
        self.radius = 1
```

# Třída CompoundShape

# Třída CompoundShape

- objekt složený z jiných objektů

# Třída `CompoundShape`

- objekt složený z jiných objektů
- společný předek `Polygon` a `Picture`

# Třída CompoundShape

- objekt složený z jiných objektů
- společný předek `Polygon` a `Picture`

```
class CompoundShape(Shape):
    def __init__(self):
        super().__init__()
        self.items = []
```

# Třídy Polygon a Picture

# Třídy Polygon a Picture

Polygon:

# Třídy `Polygon` a `Picture`

`Polygon`:

- přejmenujeme vlastnost `vertices` na `items`

# Třídy `Polygon` a `Picture`

`Polygon`:

- přejmenujeme vlastnost `vertices` na `items`
- změna není zpětně kompatibilní

# Třídy Polygon a Picture

Polygon:

- přejmenujeme vlastnost `vertices` na `items`
- změna není zpětně kompatibilní

```
class Polygon(CompoundShape):
    def __init__(self):
        super().__init__()
        self.closed = True
```

# Třídy Polygon a Picture

Polygon:

- přejmenujeme vlastnost `vertices` na `items`
- změna není zpětně kompatibilní

```
class Polygon(CompoundShape):
    def __init__(self):
        super().__init__()
        self.closed = True
```

```
class Picture(CompoundShape):
    def __init__(self):
        super().__init__()
```

# Obsah

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>>
```

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>> c = Circle()
```

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>> c = Circle()
>>>
```

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>> c = Circle()
>>> isinstance(c, Circle)
```

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>> c = Circle()
>>> isinstance(c, Circle)
True
>>>
```

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>> c = Circle()
>>> isinstance(c, Circle)
True
>>> isinstance(c, Shape)
```

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>> c = Circle()
>>> isinstance(c, Circle)
True
>>> isinstance(c, Shape)
True
```

# Přepisování metod

- Objekt může být instancí více tříd:

```
>>> c = Circle()
>>> isinstance(c, Circle)
True
>>> isinstance(c, Shape)
True
```

- Více metod stejného názvu:

  Shape:

```
def move(self, dx, dy):
    return self
```

  Circle:

```
def move(self, dx, dy):
    self.get_center().move(dx, dy)
    return self
```

# Přepisování metod

- Jakou obsluhu zprávy vybrat?

```
>>>
```

# Přepisování metod

- Jakou obsluhu zprávy vybrat?

```
>>> c.move(5, 5)
```

# Přepisování metod

- Jakou obsluhu zprávy vybrat?

```
>>> c.move(5, 5)
```

*Vykoná se metoda definovaná pro nejspecifičtější třídu.*

# Přepisování metod

- Jakou obsluhu zprávy vybrat?

```
>>> c.move(5, 5)
```

  *Vykoná se metoda definovaná pro nejspecifičtější třídu.*

- metoda `move` definovaná třídou `Circle`

# Přepisování metod

- Jakou obsluhu zprávy vybrat?

```
>>> c.move(5, 5)
```

*Vykoná se metoda definovaná pro nejspecifičtější třídu.*

- metoda `move` definovaná třídou `Circle`

*Pokud třída definuje metodu, kterou zdědila, říkáme, že ji přepisuje.*

# Přepisování metod

- Jakou obsluhu zprávy vybrat?

```
>>> c.move(5, 5)
```

*Vykoná se metoda definovaná pro nejspecifičtější třídu.*

- metoda `move` definovaná třídou `Circle`
  *Pokud třída definuje metodu, kterou zdědila, říkáme, že ji přepisuje.*
- třída `Circle` přepisuje metodu `move`

# Vlastnost items

CompoundShape:

```
def get_items(self):
    return self.items[:]

def set_items(self, items):
    for item in items:
        self.check_item(item)
    self.items = items[:]
    return self

def check_item(self, item):
    raise NotImplementedError(
        "Method check_item of CompoundShape must be rewritten.")
```

# Vlastnost items

Polygon:

```
def check_item(self, item):
    if not isinstance(item, Point):
        raise ValueError("Items of polygon must be points.")
    return self
```

Picture:

```
def check_item(self, item):
    if not isinstance(item, Shape):
        raise ValueError("Items of picture must be shapes.")
```

# Geometrické transformace složených grafických objektů

CompoundShape:

```
def move(self, dx, dy):
    for item in self.get_items():
        item.move(dx, dy)
    return self

def rotate(self, angle, center):
    for item in self.get_items():
        item.rotate(angle, center)
    return self

def scale(self, coeff, center):
    for item in self.get_items():
        item.scale(coeff, center)
    return self
```

# Kreslení ve třídě Shape

Shape:

```
def set_mg_params(self, mg_window):
    mg.set_param(mg_window, "foreground", self.get_color())
    mg.set_param(mg_window, "thickness", self.get_thickness())
    mg.set_param(mg_window, "filled", self.get_filled())
    return self

def do_draw(self, mg_window):
    return self

def draw(self, mg_window):
    self.set_mg_params(mg_window)
    self.do_draw(mg_window)
    return self
```

# Volání zděděné metody

# Volání zděděné metody

- metoda $A$ přepisuje metodu $B$

# Volání zděděné metody

- metoda $A$ přepisuje metodu $B$

V těle $A$ můžeme zavolat $B$:

```
super().message(a1, ..., an)
```

*message*: jméno zprávy
*a1*, ...,*an*: hodnoty

# Volání zděděné metody

- metoda $A$ přepisuje metodu $B$

V těle $A$ můžeme zavolat $B$:

```
super().message(a1, ..., an)
```

*message*: jméno zprávy
*a1*, ...,*an*: hodnoty

Například `Polygon`:

```
def set_mg_params(self, mg_window):
    super().set_mg_params(mg_window)
    mg.set_param(mg_window, "closed", self.get_closed())
```

# Obsah

# Inicializace instancí

# Inicializace instancí

- definice inicializace instance (`__init__`) je obyčejná metoda

# Inicializace instancí

- definice inicializace instance (`__init__`) je obyčejná metoda
- zpráva `__init__` se pošle po vytvoření objektu

# Inicializace instancí

- definice inicializace instance (`__init__`) je obyčejná metoda
- zpráva `__init__` se pošle po vytvoření objektu
- nutné vždy nejdříve volat zděděnou metodu: `super().__init__()`

# Inicializace instancí

- definice inicializace instance (`__init__`) je obyčejná metoda
- zpráva `__init__` se pošle po vytvoření objektu
- nutné vždy nejdříve volat zděděnou metodu: `super().__init__()`
- ukázky použití v příkladech

`__init__` v třídě `Picture` můžeme odstranit:

```
def __init__(self):
    super().__init__()
```

# Ukázky

# Zdroje

- Michal Krupka: materiály k předmětu Paradigmata programování 3