22. Metaprogramowanie

1 Informacje o obiektach

Język Ruby umożliwia dostęp do informacji związanych z definicją klasy:

- `Object#class` — zwraca klasę obiektu,
- `Class#superclass` — zwraca nadklasę klasy,
- `Mod#ancestors` — zwraca wszystkie klasy nadrzędne i wmieszane moduły,
- `Object#instance_of?` — czy obiekt jest instancją danej klasy?,
- `Object#kind_of?` — czy obiekt jest instancją danej klasy lub jej podklasy?,
- `Object#respond_to?` — czy obiekt odpowiada na metodę podaną jako argument,
- `Object#public_methods` — zwraca wszystkie metody publiczne, z parametrem `false`, tylko te zdefiniowane bezpośrednio w obiekcie (nie odziedziczone),
- `Object#send` — wywołuje metodę o podanej nazwie i argumentach,
- `Object#instance_variable_defined?` — sprawdza czy obiekt ma zdefiniowany atrybut o podanej nazwie,
- `Object#instance_variable_get` — zwraca wartość atrybutu klasy o podanej nazwie.

Zadanie 01 Korzystając z `irb` przetestuj powyższe metody na różnych obiektach, w tym na obiektach klas zdefiniowanych w ramach poprzednich zajęć.

2 Singletony

Metody (zachowanie) obiektu jako instancji klasy jest zdeterminowane przez definicję klasy. Jeżeli chcemy, aby konkretna instancja miała dodatkowe metody (zachowanie) w wielu innych językach programowania konieczne jest stworzenie nowej klasy. W Rubym możliwe jest dodanie metod singletonowych do pojedynczego obiektu.

```
1 obj_1 = Object.new       # dwa obiekty dowolnej klasy
2 obj_2 = Object.new
3
4 def obj_1.extra_method  # zdefiniowanie metody dla obiektu obj_1
5   "Extra method!"
6 end
7
8 obj_1.extra_method       # => "Extra method!"
9 obj_2.extra_method       # => NoMethodError
```

Alternatywny zapis wykorzystujący definicję klasy prototypowej:

```ruby
class << obj_1
  def extra_method
    "Extra method!"
  end
end
```

3 Otwarte klasy

Klasy w języku Ruby, nawet te wbudowane, można dynamicznie modyfikować, np. dodając nowe metody lub redefiniując istniejące.

```ruby
class String
  def palindrome?
    self.reverse == self
  end
end

"kajak".palindrome?   # => true
```

Zadanie 02 Rozszerz klasę `Array` o metody `sum` oraz `mean`, które zwrócą odpowiednio: sumę elementów w tablicy i ich średnią arytmetyczną.

```ruby
tab = [1, 2, 3, 4, 5]
tab.sum   # => 15
tab.mean  # => 3
```

4 Dynamiczne definiowanie metod

W języku Ruby metody można definiować dynamicznie za pomocą metody o nazwie `#define_method`, która przyjmuje dwa argumenty: nazwę metody jako symbol oraz blok (odpowiadający ciału tworzonej metody):

```ruby
class Foo
  define_method(:bar){ |arg| puts "Bar with argument: #{arg}!" }
end

foo = Foo.new
foo.bar("baz")
```

Zadanie 03 Korzystając z poniższej tablicy kolorów, rozszerz klasę `String` o metody umożliwiające wypisywanie tekstu w zadanym kolorze. Na przykład kod:

```ruby
puts "Ala ma " + "kota".red + " i " + "psa!".blue
```

wypisze na ekran tekst *„Ala ma kota i psa!"*, w którym fragment *kota* zostanie wyświetlony w kolorze czerwonym, a *psa!* w niebieskim. Metody odpowiedzialne za kolorowanie zdefiniuj dynamicznie.

```
COLORS = {
  :black => 30,
  :red => 31,
  :green => 32,
  :yellow => 33,
  :blue => 34,
  :magenta => 35,
  :cyan => 36,
  :white => 37
}
```

Kolor napisu w konsoli linuxowej można zmienić poprzez znaki specjalne. Fragment kodu:

```
puts "\033[0;31mAla ma kota\033[0m"
```

spowoduje wypisanie na ekran napisu w kolorze czerwonym. Decyduje o tym liczba 31, pozostałe symbole dla każdego koloru są takie same.

Uwaga! Powyższe informacje dotyczące kolorowania są odpowiednie tylko dla konsoli linuxowej. W przypadku systemu Windows w celach testowych można wypisać napis w postaci:

```
"Ala ma [kota (red)] i [psa! (blue)]"
```

4.1 `method_missing`

Gdy na obiekcie wywoływana jest niezdefiniowana metoda, Ruby wywołuje automatycznie metodę `Object#method_missing`, której domyślna implementacja wyrzuca wyjątek `NoMethodError`.

Zadanie 04 Dopisz do klasy `Hash` mechanizm, który pozwoli odwoływać się do kluczy typu `Symbol` lub `String` tak, jakby to były zwykłe metody.

```
hsh = { :foo => 1, 'bar' => 2 }
puts hsh.foo  # => 1
puts hsh.bar  # => 2
puts hsh.baz  # => NoMethodError
```