

# EPI: Interfejs Graficzny 2011/2012

## Laboratorium nr 2

### Programowanie obiektowe

Aleksander Pohl

9 listopada 2011

# Klasa temperatura

Funkcjonalności:

- ▶ określanie temperatury w dowolnej skali
- ▶ konwersja temperatury do napisu
- ▶ odczytanie wartości i jednostki temperatury
- ▶ konwersja z jednej skali do innej
- ▶ dodawanie i odejmowanie temperatur
- ▶ zwielokrotnianie temperatury
- ▶ porównywanie temperatur
- ▶ definicja „zera bezwzględnego”

# Klasa Temperature – sposób użycia

```
in_celsius = Temperature.new(36.6, :c)
in_celsius.value
in_celsius.scale
in_celsius.scale = :z

in_kelvins = in_celsius.to_kelvin

invalid = Temperature.new(36.6, :z)
multiplied = in_celsius * 10
invalid = Temperature.new(-300, :c)
sum = in_celsius + in_kelvins
if(in_celsius > in_kelvins)
  # ...
end
zero = Temperature.absolute_zero
puts zero
```

*#=> 36.6*  
*#=> :c*  
*# niemożliwe - wewnętrzna*  
*# struktura jest ukryta*  
*# jednostka trzymana jest*  
*# "wewnątrz"*  
*# można np. rzucić wyjątek*  
*# zachowuje skalę*  
*# można np. rzucić wyjątek*  
*# da poprawny wynik*  
*# da poprawny wynik*

*# posiada określoną jednostkę*  
*# można określić domyślną*  
*# reprezentację np. '0 K'*

## Klasa Temperature – użycie w irb

- ▶ plik temperature.rb

```
# encoding: utf-8
class Temperature
  # ciało klasy
end
```

- ▶ irb
- ▶ `$.unshift "."`
- ▶ `require 'temperature'`
- ▶ zmieniamy temperature.rb
- ▶ `load 'temperature.rb'`

# Konstruktor i konwersja na napis

```
class Temperature
  # Initialize the temperature with +value+ and +scale+.
  def initialize(value,scale)
    @value = value
    @scale = scale
  end

  # Default string representation of the temperature.
  def to_s
    "%.2f %s" % [@value, @scale.to_s.upcase]
  end
end

temperature = Temperature.new(36.6,:c)
puts temperature           #=> 36.60 C
```

# Dostęp do atrybutów

```
class Temperature
  # Returns the value of the temperature.
  def value
    @value
  end

  # Returns the scale of the temperature.
  def scale
    @scale
  end
end

temp = Temperature.new(6, :c)
temp.value          #=> 6
```

Krócej:

```
class Temperature
  attr_reader :value, :scale
end
```

Poza tym: attr\_writer, attr\_accessor

# Definiowanie metod

```
class Temperature
  # Converts the temperature to Kelvin scale.
  def to_kelvin
    Temperature.new(convert(@scale, :k, @value), :k)
  end

  # Converts the temperature to Celsius scale.
  def to_celsius
    Temperature.new(convert(@scale, :c, @value), :c)
  end

  # Converts the temperature to Fahrenheit scale.
  def to_fahrenheit
    Temperature.new(convert(@scale, :f, @value), :f)
  end
end
```

## Definiowanie metod – cd.

```
class Temperature
  # Converts (in place) the temperature to Kelvin scale.
  def to_kelvin!
    @value = convert(@scale, :k, @value)
    @scale = :k
    self
  end
  # Converts (in place) the temperature to Celsius scale.
  def to_celsius!
    @value = convert(@scale, :c, @value)
    @scale = :c
    self
  end
  # Converts (in place) the temperature to Fahrenheit scale.
  def to_fahrenheit!
    @value = convert(@scale, :f, @value)
    @scale = :f
    self
  end
end
```

## Definiowanie metod – cd.

```
class Temperature
  ABSOLUTE_ZERO = -273.15 # in Celsius
  CELSIUS_FACTORS = {:c => 1.0, :f => 1.8, :k => 1.0}
  CELSIUS_OFFSETS = {:c => 0, :f => 32, :k => -ABSOLUTE_ZERO}

  protected
  def convert(from,to,value)
    factor(from,to) * value + offset(from,to)
  end
  def factor(from,to)
    return CELSIUS_FACTORS[to] if from == :c
    return (1 / CELSIUS_FACTORS[from]) if to == :c
    factor(from,:c) * factor(:c,to)
  end
  def offset(from,to)
    return CELSIUS_OFFSETS[to] if from == :c
    return (- CELSIUS_OFFSETS[from] * factor(from,to)) if to == :c
    offset(from,:c) * factor(:c,to) + offset(:c,to)
  end
end
```

# Używanie własnych metod

```
temp = Temperature.new(10, :c)
puts temp.to_celsius      #=> 10.00 C
puts temp.to_kelvin      #=> 283.15 K
puts temp                 #=> 10.00 C
puts temp.to_fahrenheit  #=> 50.00 K
```

```
temp = Temperature.new(10, :f)
puts temp.to_celsius!    #=> 12.22 C
puts temp                #=> 12.22 C
puts temp.to_kelvin!    #=> 260.93 K
puts temp.to_fahrenheit! #=> 10.00 F
puts temp                #=> 10.00 F
```

# Metody operatorowe

```
class Temperature
  # Adds +other+ to the temperature. The result has the
  # same scale as left operand.
  def +(other)
    Temperature.new(self.value + other.value *
                    factor(other.scale,@scale),@scale)
  end
  # Subtracts +other+ from the temperature. The result has the
  # same scale as left operand.
  def -(other)
    Temperature.new(self.value - other.value *
                    factor(other.scale,@scale),@scale)
  end
  # Multiplies the temperature by given +factor+.
  def *(factor)
    Temperature.new(self.value * factor,@scale)
  end
end
puts Temperature.new(10,:c) + Temperature.new(20,:k) #=> 30.00 C
puts Temperature.new(10,:c) * 10                    #=> 100.00 C
```

# Porównywanie wartości

```
class Temperature
  include Comparable
  def <=>(other)
    self.to_kelvin.value <=> other.to_kelvin.value
  end
end
```

```
Temperature.new(10, :c) <=> Temperature.new(20, :c)    #=> -1
Temperature.new(10, :c) <=> Temperature.new(20, :f)    #=> 1
Temperature.new(10, :c) < Temperature.new(20, :f)     #=> false
Temperature.new(10, :c) >= Temperature.new(20, :f)    #=> true
Temperature.new(-273.15, :c) == Temperature.new(0, :k) #=> true
```

# Metody i zmienne klasowe

```
class Temperature
  @@absolute_zero = Temperature.new(0, :k)

  # Returns true if the temperature is an absolute zero.
  def zero?
    self == @@absolute_zero
  end

  # The absolute zero.
  def Temperature.absolute_zero
    @@absolute_zero
  end
end

zero = Temperature.absolute_zero
puts zero           #=> 0 K
puts zero.to_celsius  #=> -273.15 C
puts zero.zero?      #=> true
puts Temperature.new(10, :c).zero?  #=> false
```

# Poprawki w konstruktorze

```
class Temperatura
  VALID_SCALES = [:c, :k, :f]
  # Initialize the temperature with +value+ and +scale+.
  def initialize(value, scale)
    unless VALID_SCALES.include?(scale)
      raise "Invalid scale '#{scale}'."
    end
    @value = value
    @scale = scale
    if scale == :k
      if value < 0
        raise "Temperature below absolute zero."
      end
    else
      self.to_kelvin
    end
  end
end

Temperature.new(36.6, :z) #=> RuntimeError: Invalid scale 'z'
Temperature.new(-10, :k) #=> RuntimeError: Temperature below absolute zero
```

# Specyfikacja w postaci testów

```
$.unshift "."
require 'temperature'
require 'minitest/autorun'

describe Temperature do
  before do
    @temp1 = Temperature.new(36, :c)
  end

  it "should have proper value" do
    @temp1.value.must_equal 36
  end

  it "should have proper scale" do
    @temp1.scale.must_equal :c
  end
end
```

## Dostępne metody weryfikacji poprawności

- ▶ `must_be_empty`
- ▶ `must_be_close_to`
- ▶ `must_be_instance_of`
- ▶ `must_be_nil`
- ▶ `must_be_same_as`
- ▶ `must_be_silent`
- ▶ `must_be_equal`
- ▶ `must_match`
- ▶ `must_raise`
- ▶ `wont_be`
- ▶ `wont_be_empty`
- ▶ `wont_be_close_to`
- ▶ ...

# Temperatura

- ▶ skonwertuj wartość 10 F do Celsjuszy i Kelvinów
- ▶ sprawdź, która z wartości jest większa: 10 stopni Fahrenheita czy 10 stopni Celsjusza
- ▶ sprawdź ile w Kelvinach wynosi 10 stopni Fahrenheita dodać 10 stopni Celsjusza
- ▶ napisz testy weryfikujące poprawność definicji klasy Temperature:
  - ▶ konwersja
  - ▶ dodawanie, odejmowanie, mnożenie
  - ▶ porównywanie
  - ▶ wartość zera bezwzględnego
  - ▶ niemożność stworzenia niepoprawnej temperatury

# Waluty

- ▶ Załóżmy, że mamy stworzyć system "Portfel walutowy", który realizuje następujące funkcjonalności
  - ▶ pozwala przeliczać ilość pieniędzy w jednej walucie na ilość pieniędzy w innej walucie
  - ▶ pozwala wykonywać proste operacje arytmetyczne na pieniądzach w portfelu (dodawania, odejmowanie)
- ▶ zidentyfikuj przykładowe wartości, które mogą pojawić się w systemie
- ▶ zidentyfikuj klasy, które dobrze opisują te wartości
- ▶ określ jakie atrybuty i metody powinny posiadać te klasy
- ▶ zaimplementuj te klasy
- ▶ napisz testy jednostkowe dla tych klas

## Więcej zadań

- ▶ try ruby  
<http://tryruby.org>
- ▶ przykładowe zadania  
[apohllo.pl/dydaktyka/interfejsy-graficzne/zadania-ruby](http://apohllo.pl/dydaktyka/interfejsy-graficzne/zadania-ruby)

# Pytania

PYTANIA?