2007

2019

Programowanie dla iOS

SWIFT

SWIFT

Funkcje zagnieżdżone

Funkcje zagnieżdżone

- Funkcje mogą się zagnieżdżać

Funkcje zagnieżdżone

- Funkcje mogą się zagnieżdżać
- Funkcja zagnieżdżona ma dostęp do zmiennych funkcji zewnętrznej

Funkcje zagnieżdżone

- Funkcje mogą się zagnieżdżać
- Funkcja zagnieżdżona ma dostęp do zmiennych funkcji zewnętrznej

```
func returnFifteen() -> Int {
    var y = 10
    func add() {
        y += 5
    }
    add()
    return y
}
returnFifteen()
```

```
10
15

15

15
```

Funkcje zagnieżdżone

- Funkcje mogą się zagnieżdżać
- Funkcja zagnieżdżona ma dostęp do zmiennych funkcji zewnętrznej

```
func returnFifteen() -> Int {
    var y = 10
    func add() {
        y += 5
    }
    add()
    return y
}
returnFifteen()
```

10

15

15

15

```
func returnFifteen() -> Int {
    var y = 10
    func add() {
        y += 5
    }
    add()
    add()
    return y
}
returnFifteen()
```

10

(2 times)

20

20

Funkcje są typem pierwszej klasy

Funkcje są typem pierwszej klasy

- Funkcja może być traktowana jak inne typy

Funkcje są typem pierwszej klasy

- Funkcja może być traktowana jak inne typy
- Funkcje mogą przekazywać funkcję

```
func makeIncrementer() -> (Int -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

```
8

(Function)

(Function)
8
```

Funkcje są typem pierwszej klasy

- Funkcja może być traktowana jak inne typy
- Funkcje mogą przekazywać funkcję

```
func makeIncrementer() -> (Int -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

```
8

(Function)

(Function)
8
```

- i być przekazywane jako argument

Funkcje są typem pierwszej klasy

- Funkcja może być traktowana jak inne typy
- Funkcje mogą przekazywać funkcję

```
func makeIncrementer() -> (Int -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

```
8

(Function)

(Function)
8
```

- i być przekazywane jako argument

```
func hasAnyMatches(list: [Int], condition: Int -> Bool) -> Bool {
    for item in list {
        if condition(item) {
            return true
        }
    }
    return false
}
func lessThanTen(number: Int) -> Bool {
    return number < 10
}
var numbers = [20, 19, 7, 12]
hasAnyMatches(numbers, lessThanTen)
```

```
true

(3 times)

[20, 19, 7, 12]
true
```

Funkcje i domknięcia

Funkcje i domknięcia

- Funkcje są szczególnymi przypadkami domknięć (closure)

Funkcje i domknięcia

- Funkcje są szczególnymi przypadkami domknięć (closure)
- Domknięcia to segmenty kodu, które mogą być wywoływane wielokrotnie

Funkcje i domknięcia

- Funkcje są szczególnymi przypadkami domknięć (closure)
- Domknięcia to segmenty kodu, które mogą być wywoływane wielokrotnie
- Domknięcie jest tworzone pomiędzy nawiasami klamrowymi { }

Funkcje i domknięcia

- Funkcje są szczególnymi przypadkami domknięć (closure)
- Domknięcia to segmenty kodu, które mogą być wywoływane wielokrotnie
- Domknięcie jest tworzone pomiędzy nawiasami klamrowymi { }

```
numbers.map({
    (number: Int) -> Int in
    let result = 3 * number
    return result
})
```

```
[60, 57, 21, 36]

(4 times)
(4 times)
```

Funkcje i domknięcia

- Funkcje są szczególnymi przypadkami domknięć (closure)
- Domknięcia to segmenty kodu, które mogą być wywoływane wielokrotnie

- Domknięcie jest tworzone pomiędzy nawiasami klamrowymi { }

```
numbers.map({
    (number: Int) -> Int in
    let result = 3 * number
    return result
})
```

```
[60, 57, 21, 36]

(4 times)
(4 times)
```

```
let mappedNumbers = numbers.map({ number in 3 * number })
println(mappedNumbers)
```

```
(5 times)
"[60, 57, 21, 36]"
```

Funkcje i domknięcia

- Funkcje są szczególnymi przypadkami domknięć (closure)
- Domknięcia to segmenty kodu, które mogą być wywoływane wielokrotnie

- Domknięcie jest tworzone pomiędzy nawiasami klamrowymi { }

```
numbers.map({
    (number: Int) -> Int in
    let result = 3 * number
    return result
})
```

```
[60, 57, 21, 36]

(4 times)
(4 times)
```

```
let mappedNumbers = numbers.map({ number in 3 * number })
println(mappedNumbers)
```

```
(5 times)
"[60, 57, 21, 36]"
```

wersja uproszczona

Obiekty i klasy

Obiekty i klasy

- Klasa jest deklarowana słowem kluczowym class

Obiekty i klasy

- Klasa jest deklarowana słowem kluczowym class
- Wszystko zdefiniowane wewnątrz klasy jest jej elementem składowym (pola, metody, własności)

Obiekty i klasy

- Klasa jest deklarowana słowem kluczowym class
- Wszystko zdefiniowane wewnątrz klasy jest jej elementem składowym (pola, metody, własności)

```
class Shape {
    var numberOfSides = 0
    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

Obiekty i klasy

- Klasa jest deklarowana słowem kluczowym class
- Wszystko zdefiniowane wewnątrz klasy jest jej elementem składowym (pola, metody, własności)

```
class Shape {
    var numberOfSides = 0
    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

```
var shape = Shape()
shape.numberOfSides = 7
var shapeDescription = shape.simpleDescription()
```

```
{numberOfSides 0}
{numberOfSides 7}
"A shape with 7 sides."
```

Konstruktor klasy

Konstruktor klasy

- Metoda zaczynająca się od `init`, nie posiadająca typu

Konstruktor klasy

- Metoda zaczynająca się od `init`, nie posiadająca typu
- Jeżeli obiekt jest kosztowny może posiadać destruktor `deinit`

Konstruktor klasy

- Metoda zaczynająca się od `init`, nie posiadająca typu
- Jeżeli obiekt jest kosztowny może posiadać destruktor `deinit`

```
class NamedShape {
    var numberOfSides: Int = 0
    var name: String

    init(name: String) {
        self.name = name
    }

    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

Dziedziczenie

Dziedziczenie

- Tylko pojedyncze

Dziedziczenie

- Tylko pojedyncze
- Składnia jak w C++/C# - klasa nadrzędna po **:**

Dziedziczenie

- Tylko pojedyncze
- Składnia jak w C++/C# - klasa nadrzędna po **:**

```
class Square: NamedShape {
    var sideLength: Double

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 4
    }

    func area() ->  Double {
        return sideLength * sideLength
    }

    override func simpleDescription() -> String {
        return "A square with sides of length \(sideLength)."
    }
}
let test = Square(sideLength: 5.2, name: "my test square")
test.area()
test.simpleDescription()
```

```
27.04

"A square with sides of length 5.2."

{{numberOfSides 4 name "my test square"}
27.04
"A square with sides of length 5.2."
```

Przeciążanie metod Własności

Przeciążanie metod Własności

- Tylko ze słowem kluczowym override

Przeciążanie metod Własności

- Tylko ze słowem kluczowym override
- Własności mogą mieć getter i setter

Przeciążanie metod Własności

- Tylko ze słowem kluczowym override
- Własności mogą mieć getter i setter

```
class EquilateralTriangle: NamedShape {
    var sideLength: Double = 0.0

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 3
    }

    var perimeter: Double {
        get {
            return 3.0 * sideLength                                  9.3
        }
        set {
            sideLength = newValue / 3.0                              {{numberOfSides 3 name "a triangle"} side...
        }
    }

    override func simpleDescription() -> String {
        return "An equilateral triangle with sides of length \
            (sideLength)."
    }
}
var triangle = EquilateralTriangle(sideLength: 3.1, name: "a triangle")   {{numberOfSides 3 name "a triangle"} side...
println(triangle.perimeter)                                               "9.3"
triangle.perimeter = 9.9                                                  {{numberOfSides 3 name "a triangle"} side...
println(triangle.sideLength)                                              "3.3"
```

Przeciążanie metod Własności

- Tylko ze słowem kluczowym override
- Własności mogą mieć getter i setter

```
class EquilateralTriangle: NamedShape {
    var sideLength: Double = 0.0

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 3
    }

    var perimeter: Double {
        get {
            return 3.0 * sideLength
        }
        set {
            sideLength = newValue / 3.0
        }
    }

    override func simpleDescription() -> String {
        return "An equilateral triangle with sides of length \
            (sideLength)."
    }
}
var triangle = EquilateralTriangle(sideLength: 3.1, name: "a triangle")
println(triangle.perimeter)
triangle.perimeter = 9.9
println(triangle.sideLength)
```

klasa nadrzędna

```
9.3
{{numberOfSides 3 name "a triangle"} side...
{{numberOfSides 3 name "a triangle"} side...
"9.3"
{{numberOfSides 3 name "a triangle"} side...
"3.3"
```

Przeciążanie metod Własności

- Tylko ze słowem kluczowym override
- Własności mogą mieć getter i setter

```
class EquilateralTriangle: NamedShape {
    var sideLength: Double = 0.0

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 3
    }

    var perimeter: Double {
        get {
            return 3.0 * sideLength
        }
        set {
            sideLength = newValue / 3.0
        }
    }

    override func simpleDescription() -> String {
        return "An equilateral triangle with sides of length \
            (sideLength)."
    }
}
var triangle = EquilateralTriangle(sideLength: 3.1, name: "a triangle")
println(triangle.perimeter)
triangle.perimeter = 9.9
println(triangle.sideLength)
```

klasa nadrzędna

nowa wartość

```
9.3

{{numberOfSides 3 name "a triangle"} side...

{{numberOfSides 3 name "a triangle"} side...
"9.3"
{{numberOfSides 3 name "a triangle"} side...
"3.3"
```

Własności

Własności

- Jeżeli własność nie wymaga obliczania, ale wymaga sprawdzenia przed lub po zmianie, można wykorzystać sekcje `willSet` i `didSet`

Własności

- Jeżeli własność nie wymaga obliczania, ale wymaga sprawdzenia przed lub po zmianie, można wykorzystać sekcje `willSet` i `didSet`

```
class TriangleAndSquare {
    var triangle: EquilateralTriangle {
        willSet {
            square.sideLength = newValue.sideLength
        }
    }
    var square: Square {
        willSet {
            triangle.sideLength = newValue.sideLength
        }
    }
    init(size: Double, name: String) {
        square = Square(sideLength: size, name: name)
        triangle = EquilateralTriangle(sideLength: size, name: name)
    }
}
var triangleAndSquare = TriangleAndSquare(size: 10, name: "another test
    shape")
println(triangleAndSquare.square.sideLength)
println(triangleAndSquare.triangle.sideLength)
triangleAndSquare.square = Square(sideLength: 50, name: "larger square")
println(triangleAndSquare.triangle.sideLength)
```

```
{{numberOfSides 3 name "another test sha...
{{{...} sideLength 10.0} {{...} sideLength 10....
"10.0"
"10.0"
{{{...} sideLength 50.0} {{...} sideLength 50....
"50.0"
```

Typy wyliczeniowe

Typy wyliczeniowe

- Deklarowane przez słowo kluczowe **`enum`**

Typy wyliczeniowe

- Deklarowane przez słowo kluczowe **`enum`**

```
enum Rank: Int {
    case Ace = 1
    case Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten
    case Jack, Queen, King
    func simpleDescription() -> String {
        switch self {
        case .Ace:
            return "ace"
        case .Jack:
            return "jack"
        case .Queen:
            return "queen"
        case .King:
            return "king"
        default:
            return String(self.rawValue)                    (2 times)
        }
    }
}
let ace = Rank.Ace                                          (Enum Value)
let aceRawValue = ace.rawValue                              1
```

Typy wyliczeniowe

- Deklarowane przez słowo kluczowe **`enum`**

```
enum Rank: Int {
    case Ace = 1
    case Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten
    case Jack, Queen, King
    func simpleDescription() -> String {
        switch self {
        case .Ace:
            return "ace"
        case .Jack:
            return "jack"
        case .Queen:
            return "queen"
        case .King:
            return "king"
        default:
            return String(self.rawValue)
        }
    }
}
let ace = Rank.Ace
let aceRawValue = ace.rawValue
```

typ bazowy

(2 times)

(Enum Value)
1

Typy wyliczeniowe

- Deklarowane przez słowo kluczowe **`enum`**

```
enum Rank: Int {
    case Ace = 1
    case Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten
    case Jack, Queen, King
    func simpleDescription() -> String {
        switch self {
        case .Ace:
            return "ace"
        case .Jack:
            return "jack"
        case .Queen:
            return "queen"
        case .King:
            return "king"
        default:
            return String(self.rawValue)
        }
    }
}
let ace = Rank.Ace
let aceRawValue = ace.rawValue
```

typ bazowy

(2 times)

(Enum Value)
1

Typem bazowym może być **`Int`**, **`String`** lub typy zmiennoprzecinkowe

Typy wyliczeniowe

Typy wyliczeniowe

- Zmienną wyliczeniową można utworzyć z typu bazowego

```
if let convertedRank = Rank(rawValue: 3) {
    let threeDescription = convertedRank.simpleDescription()
}
```

"3"

Typy wyliczeniowe

- Zmienną wyliczeniową można utworzyć z typu bazowego

```
if let convertedRank = Rank(rawValue: 3) {
    let threeDescription = convertedRank.simpleDescription()
}
```

"3"

- Typ wyliczeniowy może nie mieć typu bazowego

Typy wyliczeniowe

- Zmienną wyliczeniową można utworzyć z typu bazowego

```
if let convertedRank = Rank(rawValue: 3) {
    let threeDescription = convertedRank.simpleDescription()
}
```

"3"

- Typ wyliczeniowy może nie mieć typu bazowego

```
enum Suit {
    case Spades, Hearts, Diamonds, Clubs
    func simpleDescription() -> String {
        switch self {
        case .Spades:
            return "spades"
        case .Hearts:
            return "hearts"
        case .Diamonds:
            return "diamonds"
        case .Clubs:
            return "clubs"
        }
    }
}
let hearts = Suit.Hearts
let heartsDescription = hearts.simpleDescription()
```

"spades"

"hearts"

(Enum Value)
"hearts"

Typy wyliczeniowe

Typy wyliczeniowe

- składowe typu wyliczeniowego mogą mieć skojarzone ze sobą wartości

Typy wyliczeniowe

- składowe typu wyliczeniowego mogą mieć skojarzone ze sobą wartości
- mogą się one różnić w ramach jednego typu

Typy wyliczeniowe

- składowe typu wyliczeniowego mogą mieć skojarzone ze sobą wartości
- mogą się one różnić w ramach jednego typu

```
enum ServerResponse {
    case Result(String, String)
    case Error(String)
}

let success = ServerResponse.Result("6:00 am", "8:09 pm")
let failure = ServerResponse.Error("Out of cheese.")

switch failure {
case let .Result(sunrise, sunset):
    let serverResponse = "Sunrise is at \(sunrise) and sunset is at \
        (sunset)."
case let .Error(error):
    let serverResponse = "Failure...  \(error)"
}
```

(Enum Value)
(Enum Value)

"Failure... Out of cheese."

Struktury

Struktury

- Są podobne do klas

Struktury

- Są podobne do klas
- Przekazywane zawsze przez wartość (klasy przez referencję)

Struktury

- Są podobne do klas
- Przekazywane zawsze przez wartość (klasy przez referencję)
- Słowo kluczowe `struct`

Struktury

- Są podobne do klas
- Przekazywane zawsze przez wartość (klasy przez referencję)
- Słowo kluczowe `struct`

```
struct Card {
    var rank: Rank
    var suit: Suit
    func simpleDescription() -> String {
        return "The \(rank.simpleDescription()) of \
            (suit.simpleDescription())"
    }
}
let threeOfSpades = Card(rank: .Three, suit: .Spades)
let threeOfSpadesDescription = threeOfSpades.simpleDescription()
```

```
"The 3 of spades"

(Enum Value), (Enum Value)
"The 3 of spades"
```

Protokoły

Protokoły

- Odpowiednik interfejsów

Protokoły

- Odpowiednik interfejsów
- Słowo kluczowe `protocol`

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    mutating func adjust()
}
```

Protokoły

- Odpowiednik interfejsów
- Słowo kluczowe `protocol`

metoda będzie zmieniać obiekt

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    mutating func adjust()
}
```

Protokoły

- Odpowiednik interfejsów
- Słowo kluczowe `protocol`

metoda będzie zmieniać obiekt

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    mutating func adjust()
}
```

- Mogą być adaptowane (implementowane) przez klasy struktury i wyliczenia

Protokoły

- Odpowiednik interfejsów
- Słowo kluczowe `protocol`

metoda będzie zmieniać obiekt

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    mutating func adjust()
}
```

- Mogą być adaptowane (implementowane) przez klasy struktury i wyliczenia

```
class SimpleClass: ExampleProtocol {
    var simpleDescription: String = "A very simple class."
    var anotherProperty: Int = 69105
    func adjust() {
        simpleDescription += "  Now 100% adjusted."
    }
}
var a = SimpleClass()
a.adjust()
let aDescription = a.simpleDescription

struct SimpleStructure: ExampleProtocol {
    var simpleDescription: String = "A simple structure"
    mutating func adjust() {
        simpleDescription += " (adjusted)"
    }
}
var b = SimpleStructure()
b.adjust()
let bDescription = b.simpleDescription
```

```
{simpleDescription "A very simple class." a...
{simpleDescription "A very simple class.  N...
"A very simple class.  Now 100% adjusted."

{simpleDescription "A simple structure"}
{simpleDescription "A simple structure (adj...
"A simple structure (adjusted)"
```

Protokoły

Protokoły

- Protokoły mogą być używane jak każdy inny typ

Protokoły

- Protokoły mogą być używane jak każdy inny typ

```
let protocolValue: ExampleProtocol = a
println(protocolValue.simpleDescription)
println(protocolValue.anotherProperty)  // Uncomment to see the error
```

{simpleDescription "A very simple class. N...
"A very simple class. Now 100% adjusted."

Typy generyczne

Typy generyczne

- Podobnie jak w innych językach deklarowane za pomocą < >

Typy generyczne

- Podobnie jak w innych językach deklarowane za pomocą < >
- Generyczne mogą być funkcje, metody, klasy, typy wyliczeniowe i struktury

Typy generyczne

- Podobnie jak w innych językach deklarowane za pomocą < >
- Generyczne mogą być funkcje, metody, klasy, typy wyliczeniowe i struktury

```
func repeat<Item>(item: Item, times: Int) -> [Item] {
    var result = [Item]()
    for i in 0..<times {
        result.append(item)
    }
    return result
}
repeat("knock", 4)
```

```
0 elements
(4 times)
["knock", "knock", "knock", "knock"]
["knock", "knock", "knock", "knock"]
```

Typy generyczne

- Podobnie jak w innych językach deklarowane za pomocą < >

- Generyczne mogą być funkcje, metody, klasy, typy wyliczeniowe i struktury

```
func repeat<Item>(item: Item, times: Int) -> [Item] {
    var result = [Item]()
    for i in 0..<times {
        result.append(item)
    }
    return result
}
repeat("knock", 4)
```

```
0 elements
(4 times)
["knock", "knock", "knock", "knock"]
["knock", "knock", "knock", "knock"]
```

```
// Reimplement the Swift standard library's optional type
enum OptionalValue<T> {
    case None
    case Some(T)
}
var possibleInteger: OptionalValue<Int> = .None
possibleInteger = .Some(100)
```

```
(Enum Value)
(Enum Value)
```

Typy generyczne

Typy generyczne

- Typ generyczny może stawiać wymagania, np. co do realizacji protokołów z użyciem słowa kluczowego `where`

Typy generyczne

- Typ generyczny może stawiać wymagania, np. co do realizacji protokołów z użyciem słowa kluczowego **`where`**

```
func anyCommonElements <T, U where T: SequenceType, U: SequenceType, T.
    Generator.Element: Equatable, T.Generator.Element == U.Generator.
    Element> (lhs: T, rhs: U) -> Bool {
    for lhsItem in lhs {
        for rhsItem in rhs {
            if lhsItem == rhsItem {
                return true                                              true
            }
        }
    }
    return false
}
anyCommonElements([1, 2, 3], [3])                                        true
```

Rozszerzenia

Rozszerzenia

- Dodawanie nowej funkcjonalności do istniejących klas

Rozszerzenia

- Dodawanie nowej funkcjonalności do istniejących klas
- Słowo kluczowe `extension`

Rozszerzenia

- Dodawanie nowej funkcjonalności do istniejących klas
- Słowo kluczowe `extension`

```
extension Int: ExampleProtocol {
    var simpleDescription: String {
        return "The number \(self)"
    }
    mutating func adjust() {
        self += 42
    }
}
println(7.simpleDescription)
```

"The number 7"

"The number 7"

Przeciążanie operatorów

Przeciążanie operatorów

- Można przeciążać operatory dla własnych typów

Przeciążanie operatorów

- Można przeciążać operatory dla własnych typów

```
struct Vector2D {
    var x = 0.0, y = 0.0
}

extension Vector2D {
    static func + (left: Vector2D, right: Vector2D) -> Vector2D {
        return Vector2D(x: left.x + right.x, y: left.y + right.y)
    }
}
```

Przeciążanie operatorów

- Można przeciążać operatory dla własnych typów

```
struct Vector2D {
    var x = 0.0, y = 0.0
}

extension Vector2D {
    static func + (left: Vector2D, right: Vector2D) -> Vector2D {
        return Vector2D(x: left.x + right.x, y: left.y + right.y)
    }
}
```

```
extension Vector2D {
    static func == (left: Vector2D, right: Vector2D) -> Bool {
        return (left.x == right.x) && (left.y == right.y)
    }
    static func != (left: Vector2D, right: Vector2D) -> Bool {
        return !(left == right)
    }
}
```

Dodawanie operatorów

Dodawanie operatorów

- Można stworzyć własne operatory

Dodawanie operatorów

- Można stworzyć własne operatory

```
extension Vector2D {
    static prefix func +++ (vector: inout Vector2D) -> Vector2D {
        vector += vector
        return vector
    }
}

var toBeDoubled = Vector2D(x: 1.0, y: 4.0)
let afterDoubling = +++toBeDoubled
// toBeDoubled now has values of (2.0, 8.0)
// afterDoubling also has values of (2.0, 8.0)
```

Rozszerzenia

Rozszerzenia

- Dodawanie własności

Rozszerzenia

- Dodawanie własności

```
extension Double {
    var km: Double { return self * 1_000.0 }
    var m: Double { return self }
    var cm: Double { return self / 100.0 }
    var mm: Double { return self / 1_000.0 }
    var ft: Double { return self / 3.28084 }
}
let oneInch = 25.4.mm
print("One inch is \(oneInch) meters")
// Prints "One inch is 0.0254 meters"
let threeFeet = 3.ft
print("Three feet is \(threeFeet) meters")
// Prints "Three feet is 0.914399970739201 meters"
```

Rozszerzenia

- Dodawanie własności

```
extension Double {
    var km: Double { return self * 1_000.0 }
    var m: Double { return self }
    var cm: Double { return self / 100.0 }
    var mm: Double { return self / 1_000.0 }
    var ft: Double { return self / 3.28084 }
}
let oneInch = 25.4.mm
print("One inch is \(oneInch) meters")
// Prints "One inch is 0.0254 meters"
let threeFeet = 3.ft
print("Three feet is \(threeFeet) meters")
// Prints "Three feet is 0.914399970739201 meters"
```

```
let aMarathon = 42.km + 195.m
print("A marathon is \(aMarathon) meters long")
// Prints "A marathon is 42195.0 meters long"
```

Rozszerzenia

Rozszerzenia

- Dodawanie konstruktora

Rozszerzenia

- Dodawanie konstruktora

```
struct Size {
    var width = 0.0, height = 0.0
}
struct Point {
    var x = 0.0, y = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
}
```

Struktura

Rozszerzenia

- Dodawanie konstruktora

```
struct Size {
    var width = 0.0, height = 0.0
}
struct Point {
    var x = 0.0, y = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
}
```

Struktura

```
let defaultRect = Rect()
let memberwiseRect = Rect(origin: Point(x: 2.0, y: 2.0),
                          size: Size(width: 5.0, height: 5.0))
```

Domyślne konstuktory

Rozszerzenia

- Dodawanie konstruktora

```
struct Size {
    var width = 0.0, height = 0.0
}
struct Point {
    var x = 0.0, y = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
}
```

Struktura

```
let defaultRect = Rect()
let memberwiseRect = Rect(origin: Point(x: 2.0, y: 2.0),
                          size: Size(width: 5.0, height: 5.0))
```

Domyślne konstuktory

```
extension Rect {
    init(center: Point, size: Size) {
        let originX = center.x - (size.width / 2)
        let originY = center.y - (size.height / 2)
        self.init(origin: Point(x: originX, y: originY), size: size)
    }
}
```

Dodany konstruktor

Rozszerzenia

- Dodawanie konstruktora

```
struct Size {
    var width = 0.0, height = 0.0
}
struct Point {
    var x = 0.0, y = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
}
```

Struktura

```
let defaultRect = Rect()
let memberwiseRect = Rect(origin: Point(x: 2.0, y: 2.0),
                          size: Size(width: 5.0, height: 5.0))
```

Domyślne konstuktory

```
extension Rect {
    init(center: Point, size: Size) {
        let originX = center.x - (size.width / 2)
        let originY = center.y - (size.height / 2)
        self.init(origin: Point(x: originX, y: originY), size: size)
    }
}
```

Dodany konstruktor

```
let centerRect = Rect(center: Point(x: 4.0, y: 4.0),
                      size: Size(width: 3.0, height: 3.0))
// centerRect's origin is (2.5, 2.5) and its size is (3.0, 3.0)
```

Wywołanie

Rozszerzenia

Rozszerzenia

- Dodawanie metody

Rozszerzenia

- Dodawanie metody

```
extension Int {
    func repetitions(task: () -> Void) {
        for _ in 0..<self {
            task()
        }
    }
}
```

Rozszerzenia

- Dodawanie metody

```
extension Int {
    func repetitions(task: () -> Void) {
        for _ in 0..<self {
            task()
        }
    }
}
```

```
3.repetitions {
    print("Hello!")
}
// Hello!
// Hello!
// Hello!
```

Rozszerzenia

Rozszerzenia

- Dodawanie indeksu

Rozszerzenia

- Dodawanie indeksu

```
extension Int {
    subscript(digitIndex: Int) -> Int {
        var decimalBase = 1
        for _ in 0..<digitIndex {
            decimalBase *= 10
        }
        return (self / decimalBase) % 10
    }
}
746381295[0]
// returns 5
746381295[1]
// returns 9
746381295[2]
// returns 2
746381295[8]
// returns 7
```

Rozszerzenia

Rozszerzenia

- Dodawanie typu zagnieżdżonego

Rozszerzenia

- Dodawanie typu zagnieżdżonego

```
extension Int {
    enum Kind {
        case negative, zero, positive
    }
    var kind: Kind {
        switch self {
        case 0:
            return .zero
        case let x where x > 0:
            return .positive
        default:
            return .negative
        }
    }
}
```

Rozszerzenia

- Dodawanie typu zagnieżdżonego

```
extension Int {
    enum Kind {
        case negative, zero, positive
    }
    var kind: Kind {
        switch self {
        case 0:
            return .zero
        case let x where x > 0:
            return .positive
        default:
            return .negative
        }
    }
}
```

```
func printIntegerKinds(_ numbers: [Int]) {
    for number in numbers {
        switch number.kind {
        case .negative:
            print("- ", terminator: "")
        case .zero:
            print("0 ", terminator: "")
        case .positive:
            print("+ ", terminator: "")
        }
    }
    print("")
}
printIntegerKinds([3, 19, -27, 0, -6, 0, 7])
// Prints "+ + - 0 - 0 + "
```

Wyjątki

Wyjątki

- Aby zgłosić wyjątek musimy zadeklarować typ zgodny z protokołem Error

Wyjątki

- Aby zgłosić wyjątek musimy zadeklarować typ zgodny z protokołem Error

```
enum PrinterError: Error {
    case outOfPaper
    case noToner
    case onFire
}
```

Wyjątki

Wyjątki

- Do zgłoszenia wyjątku służy instrukcja throw, a słowo kluczowe throws informuje, że funkcja może zgłosić wyjątek

Wyjątki

- Do zgłoszenia wyjątku służy instrukcja throw, a słowo kluczowe throws informuje, że funkcja może zgłosić wyjątek

```
func send(job: Int, toPrinter printerName:
          String) throws -> String {
    if printerName == "Never Has Toner" {
        throw PrinterError.noToner
    }
    return "Job sent"
}
```

Wyjątki

Wyjątki

- Do przechwycenia wyjątku służy instrukcja do-catch i instrukcja try

Wyjątki

- Do przechwycenia wyjątku służy instrukcja do-catch i instrukcja try

```
do {
    let printerResponse = try send(job: 1040,
            toPrinter: "Bi Sheng")
    print(printerResponse)
} catch {
    print(error)
}
```

Wyjątki

- Do przechwycenia wyjątku służy instrukcja do-catch i instrukcja try

```
do {
    let printerResponse = try send(job: 1040,
            toPrinter: "Bi Sheng")
    print(printerResponse)
} catch {
    print(error)
}
```

```
var vendingMachine = VendingMachine()
vendingMachine.coinsDeposited = 8
do {
    try buyFavoriteSnack(person: "Alice", vendingMachine: vendingMachine)
} catch VendingMachineError.invalidSelection {
    print("Invalid Selection.")
} catch VendingMachineError.outOfStock {
    print("Out of Stock.")
} catch VendingMachineError.insufficientFunds(let coinsNeeded) {
    print("Insufficient funds. Please insert an additional \(coinsNeeded) coins.")
}
```

Wyjątki

Wyjątki

- Instrukcja try ma swoją wersję opcjonalną try?

Wyjątki

- Instrukcja try ma swoją wersję opcjonalną try?

```swift
func someThrowingFunction() throws -> Int {
    // ...
}

let x = try? someThrowingFunction()

let y: Int?
do {
    y = try someThrowingFunction()
} catch {
    y = nil
}
```

Wyjątki

Wyjątki

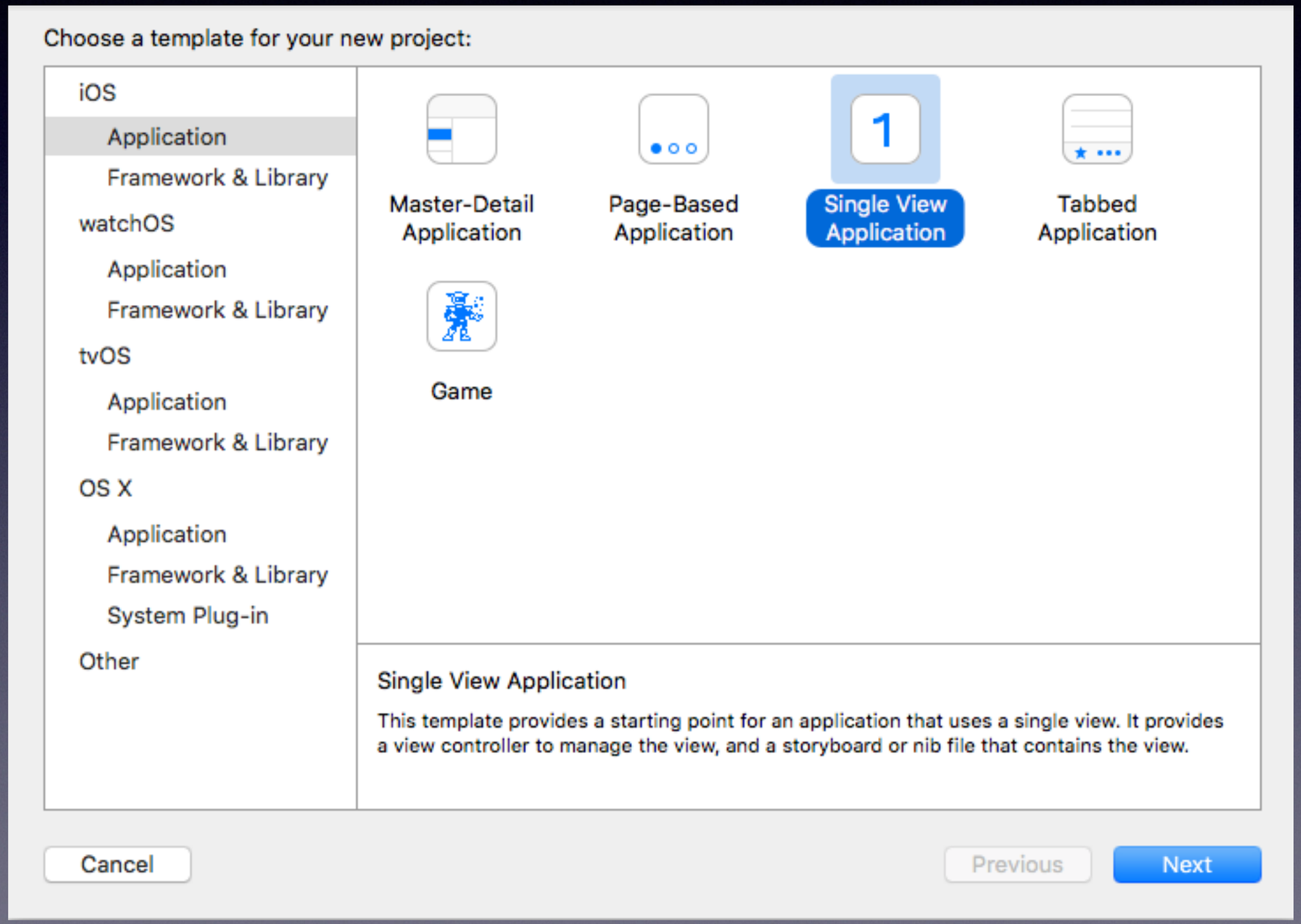
- Zgłoszenie wyjątku można zablokować instrukcją try!

Wyjątki

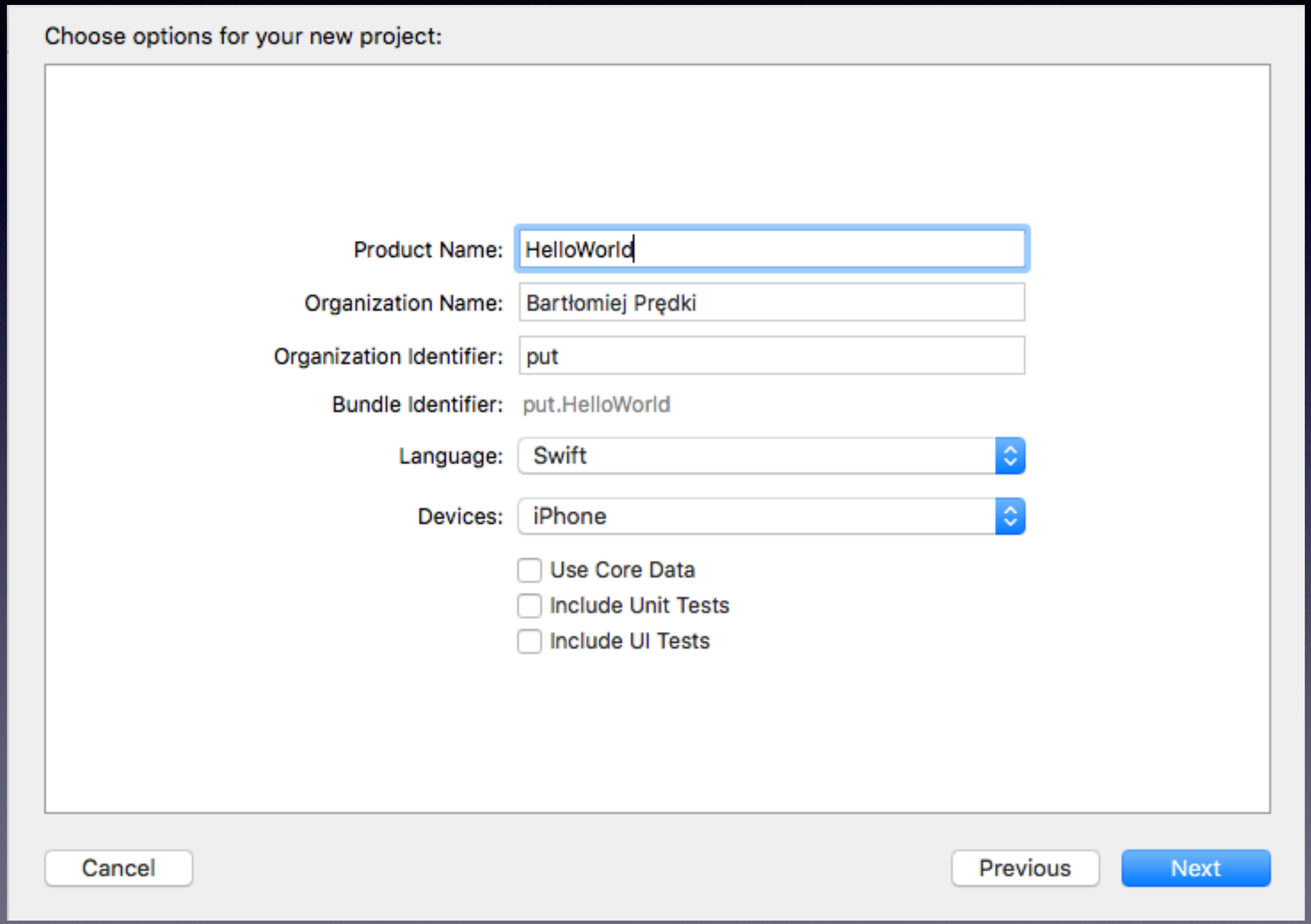
- Zgłoszenie wyjątku można zablokować instrukcją try!

```
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```

Hello World 2015



Hello World


Choose options for your new project:
Product Name: HelloWorld
Organization Name: Bartłomiej Prędki
Organization Identifier: put
Bundle Identifier: put.HelloWorld
Language: Swift
Devices: iPhone
Use Core Data
Include Unit Tests
Include UI Tests
Cancel
Previous
Next

Hello World

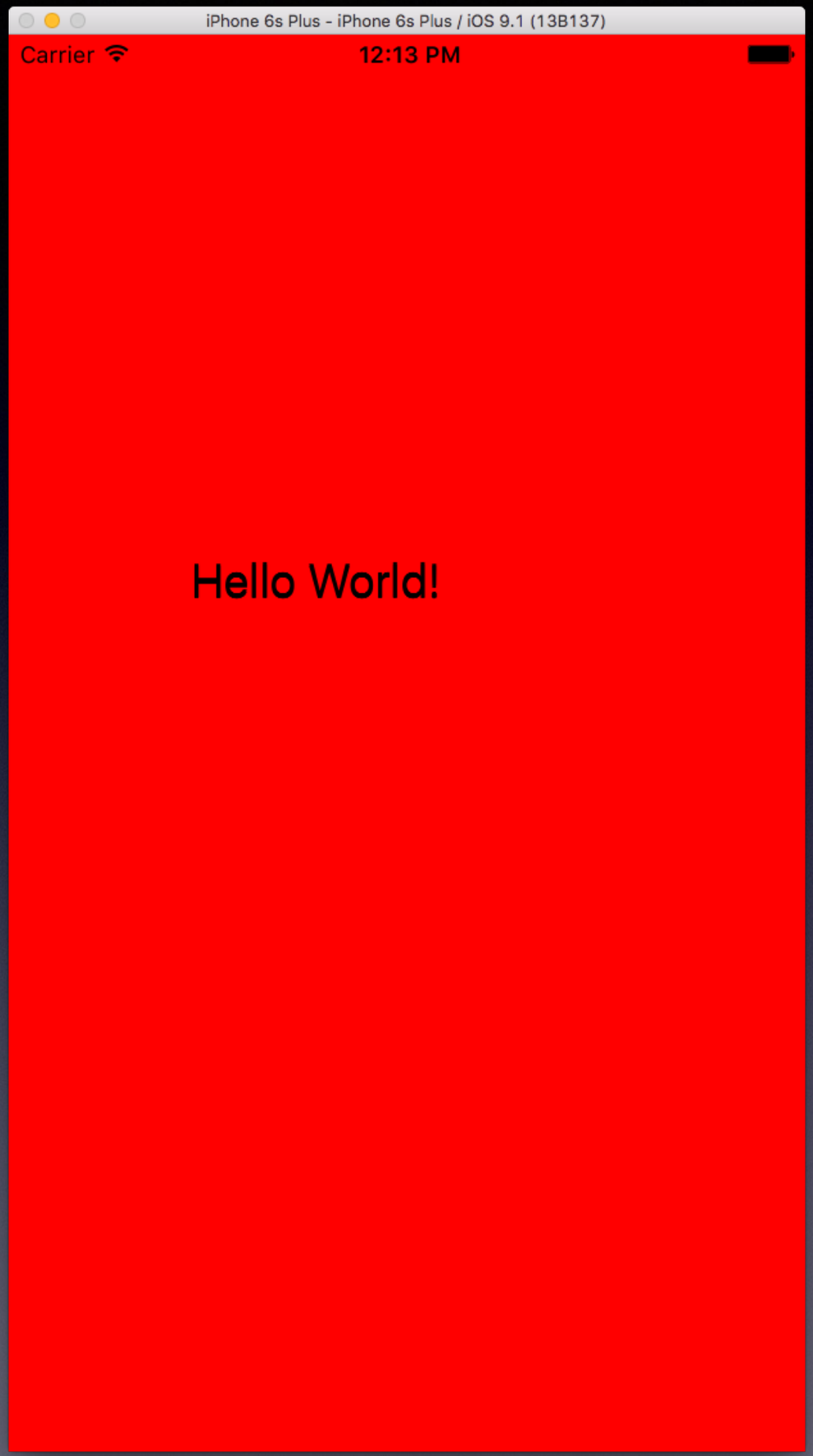
```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        self.view.backgroundColor=UIColor.redColor()
        let label = UILabel(frame: CGRectMake(0, 0, 200, 21))
        label.center = CGPointMake(160, 284)
        label.shadowColor=UIColor.blackColor()
        label.textAlignment = NSTextAlignment.Center
        label.font=UIFont.systemFontOfSize(24)
        label.text = "Hello World!"
        self.view.addSubview(label)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```


iPhone 6s Plus - iPhone 6s Plus / iOS 9.1 (13B137)
Carrier
12:13 PM
Hello World!

Pierwsza aplikacja w Swift

Pierwsza aplikacja

Pierwsza aplikacja

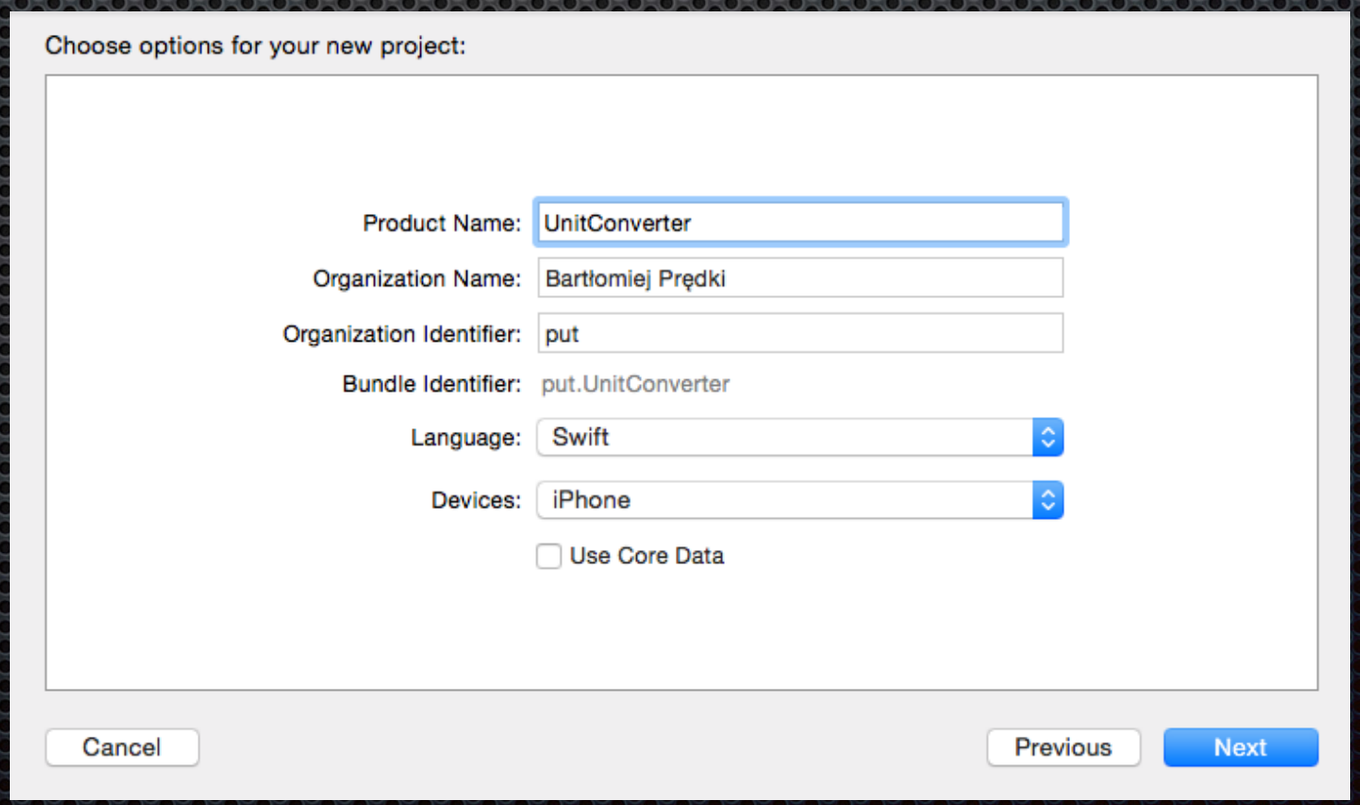
- Prosty konwerter wartości ze stopni Celsjusza na Fahrenheita

Pierwsza aplikacja

- Prosty konwerter wartości ze stopni Celsjusza na Fahrenheita
- Szablon Single View Application

Pierwsza aplikacja

- Prosty konwerter wartości ze stopni Celsjusza na Fahrenheita
- Szablon Single View Application



UnitConverter 〉 iPhone 6 | UnitConverter: **Ready** | Today at 12:35

UnitConverter

UnitConverter
2 targets, iOS SDK 8.3

- UnitConverter
  - AppDelegate.swift
  - ViewController.swift
  - Main.storyboard
  - Images.xcassets
  - LaunchScreen.xib
  - Supporting Files
- UnitConverterTests
- Products

UnitConverter | General | Capabilities | Info | Build Settings | Build Phases | Build Rules

Identity

Bundle Identifier: put.UnitConverter
Version: 1.0
Build: 1
Team: None

Deployment Info

Deployment Target: 8.3
Devices: iPhone
Main Interface: Main
Device Orientation: Portrait, Upside Down, Landscape Left, Landscape Right
Status Bar Style: Default
Hide status bar

App Icons and Launch Images

App Icons Source: AppIcon
Launch Images Source: Use Asset Catalog
Launch Screen File: LaunchScreen

Embedded Binaries

Add embedded binaries here

Linked Frameworks and Libraries

Name | Status

Add frameworks & libraries here

Identity and Type

Name: UnitConverter
Location: Absolute
UnitConverter.xcodeproj
Full Path: /Users/bpredki/Dysk Google/Projects/UnitConverter/UnitConverter.xcodeproj

Project Document

Project Format: Xcode 3.2-compatible
Organization: Bartłomiej Prędki
Class Prefix:

Text Settings

Indent Using: Spaces
Widths: 4 Tab, 4 Indent
Wrap lines

Source Control

Repository: UnitConverter
Type: Git
Current Branch: Branches Unavailable
Version: --
Status: No changes
Location: /Users/bpredki/Dysk Google/Projects/UnitConverter/UnitConverter.xcodeproj

View Controller - A controller that supports the fundamental view-management model in iOS.

Navigation Controller - A controller that manages navigation through a hierarchy of views.

Table View Controller - A controller that manages a table view.

Konwerter

Konwerter

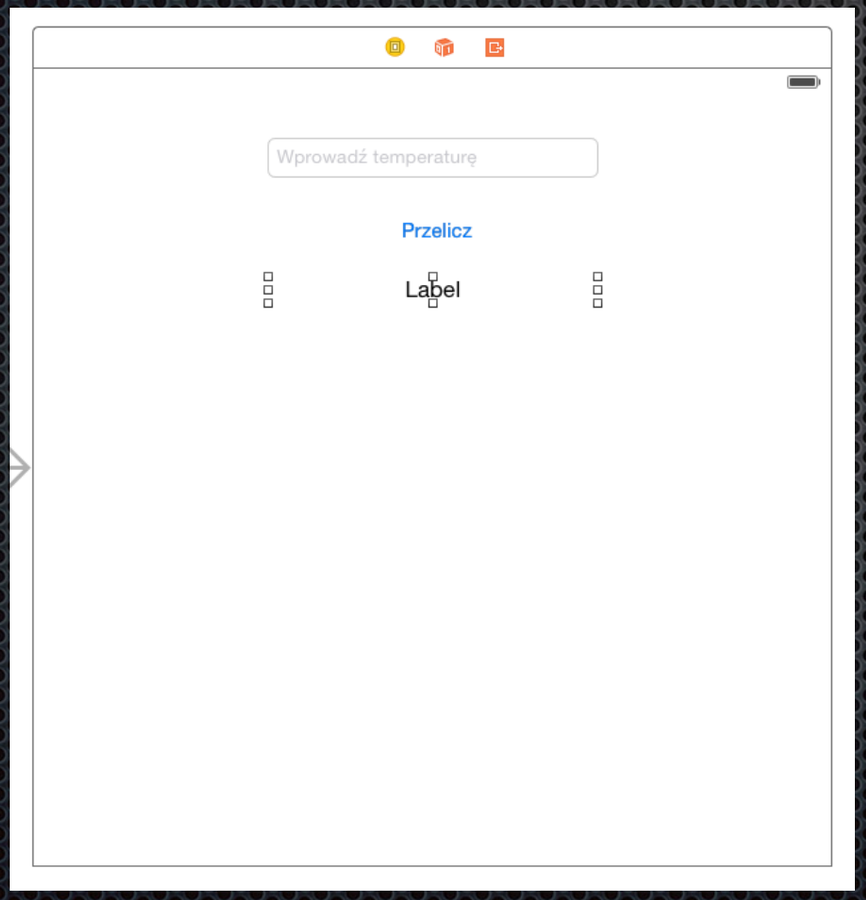
- Otwieramy scenorys

Konwerter

- Otwieramy scenorys
- Dodajemy pole tekstowe, guzik i etykietkę

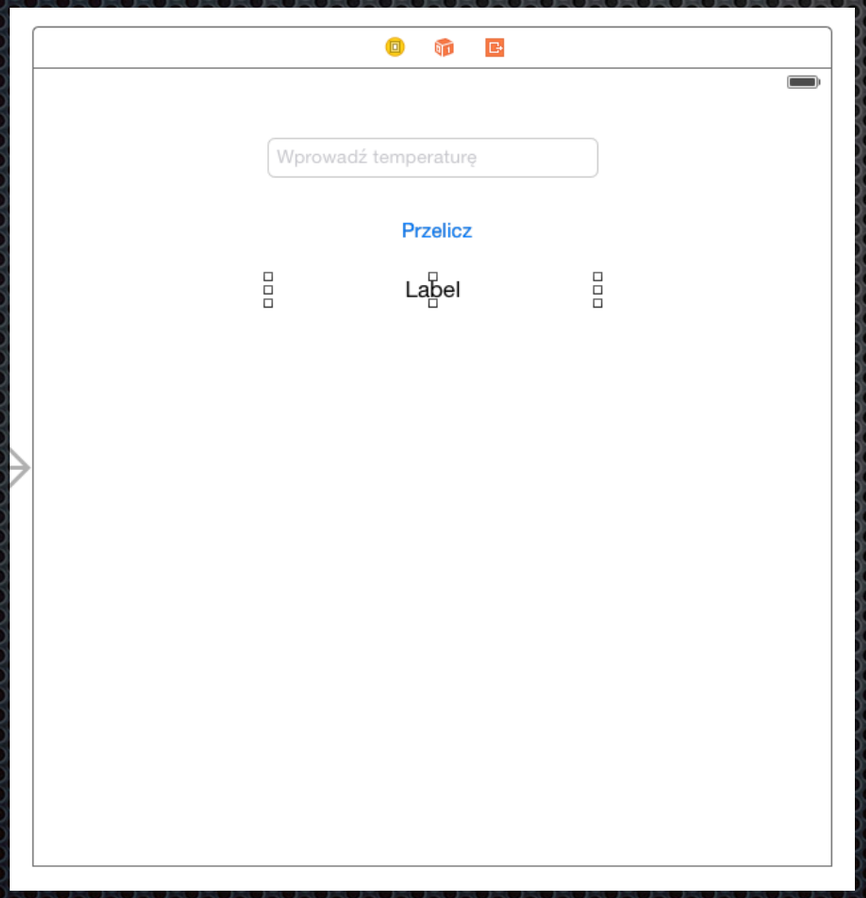
Konwerter

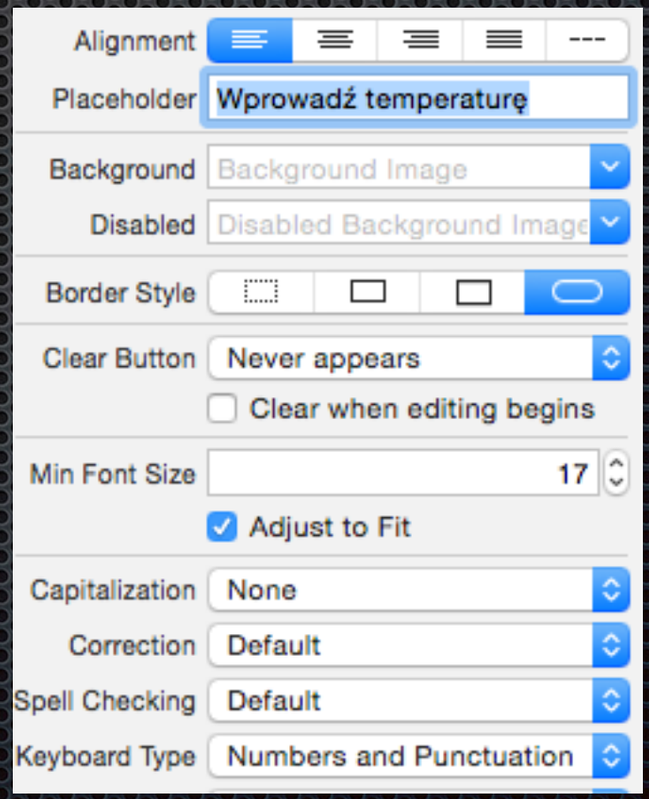
- Otwieramy scenorys
- Dodajemy pole tekstowe, guzik i etykietkę



Konwerter

- Otwieramy scenorys
- Dodajemy pole tekstowe, guzik i etykietkę





Konwerter

Konwerter

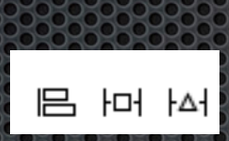
- Ustawienia Auto Layout

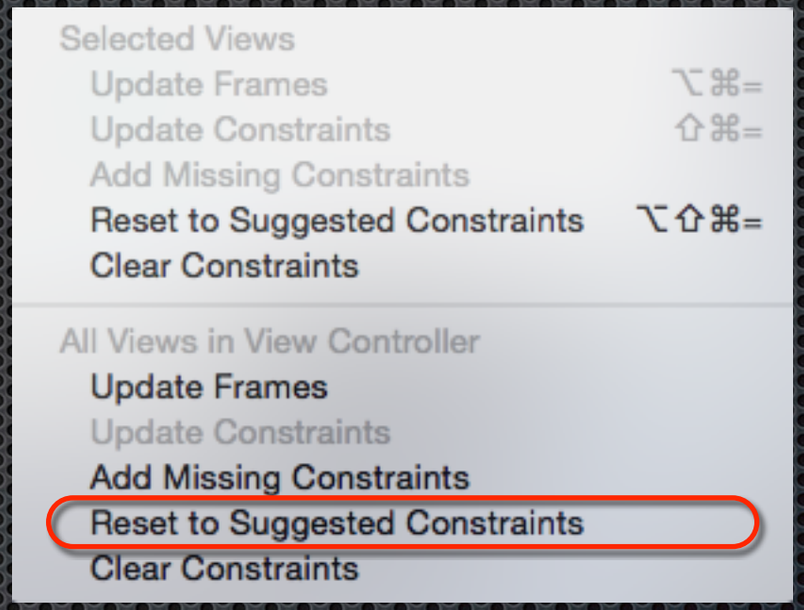
Konwerter

- Ustawienia Auto Layout

Konwerter

- Ustawienia Auto Layout





Klasa AppDelegate

Klasa AppDelegate

```swift
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationWillResignActive(application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary
            interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the
            transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method
            to pause the game.
    }

    func applicationDidEnterBackground(application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information
            to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user
            quits.
    }

    func applicationWillEnterForeground(application: UIApplication) {
        // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on
            entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously
            in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    }


}
```

Klasa ViewController

Klasa ViewController

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

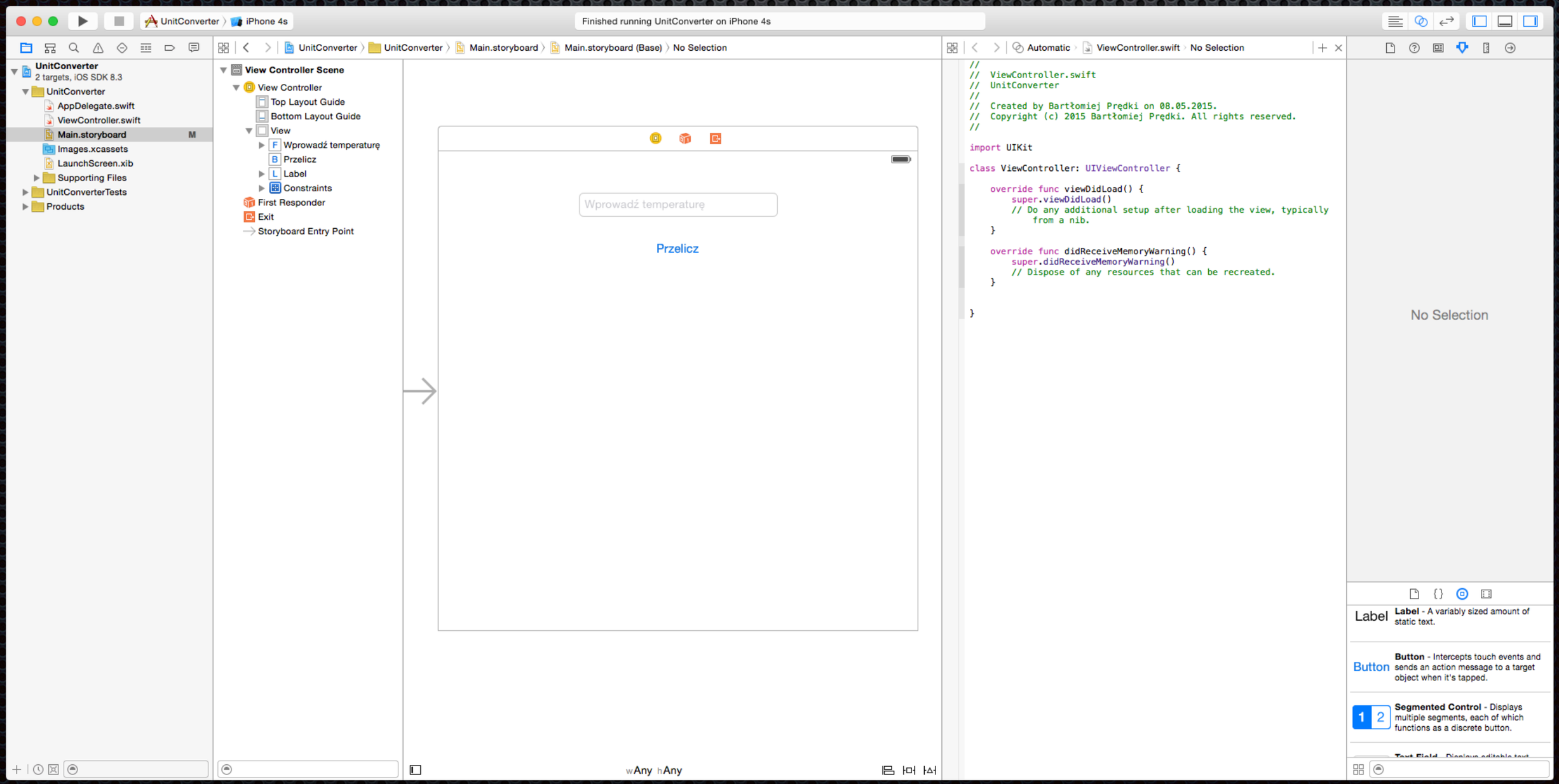
Konwerter

Konwerter

- Tworzymy gniazdka

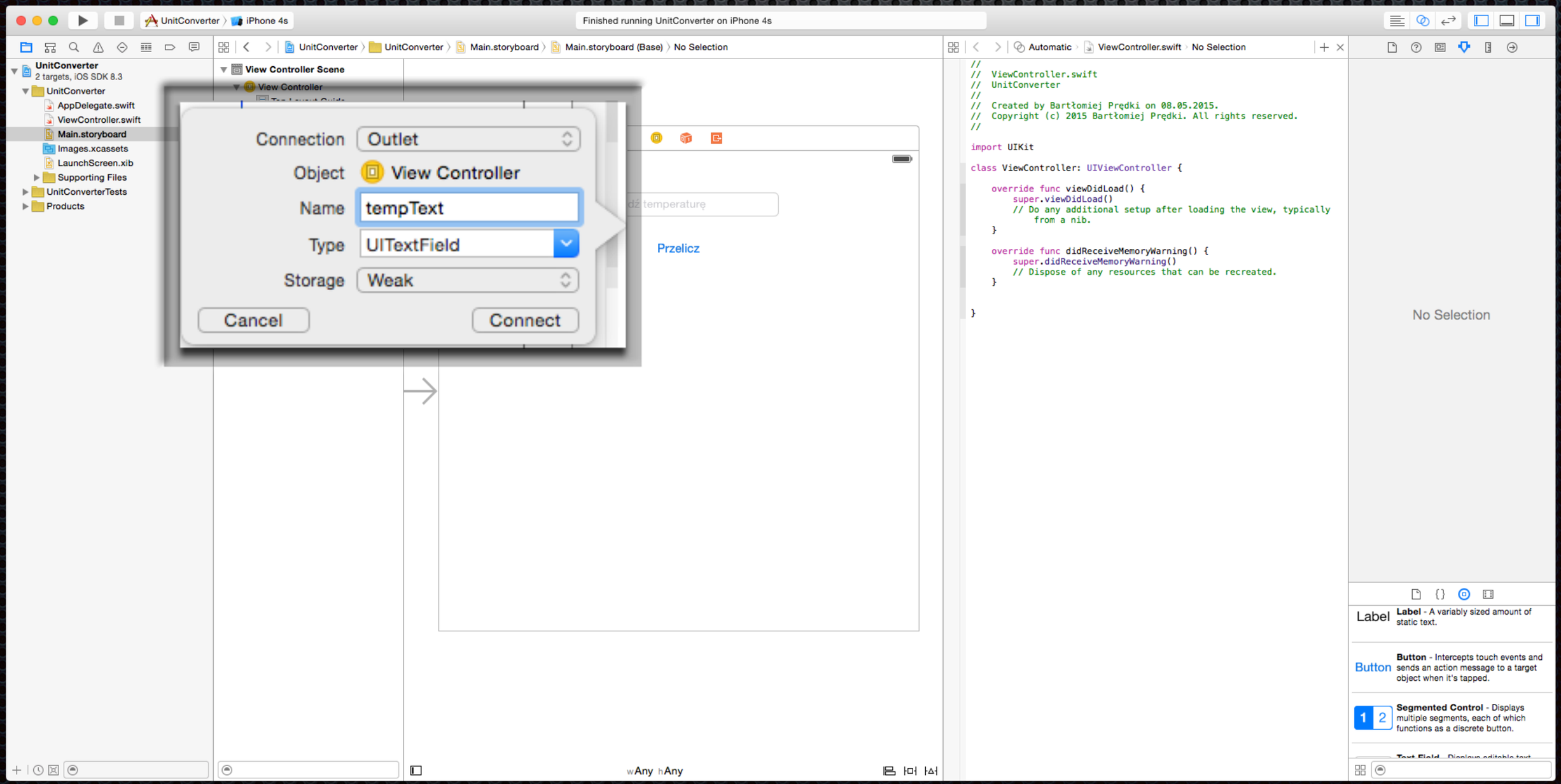
Konwerter

- Tworzymy gniazdka



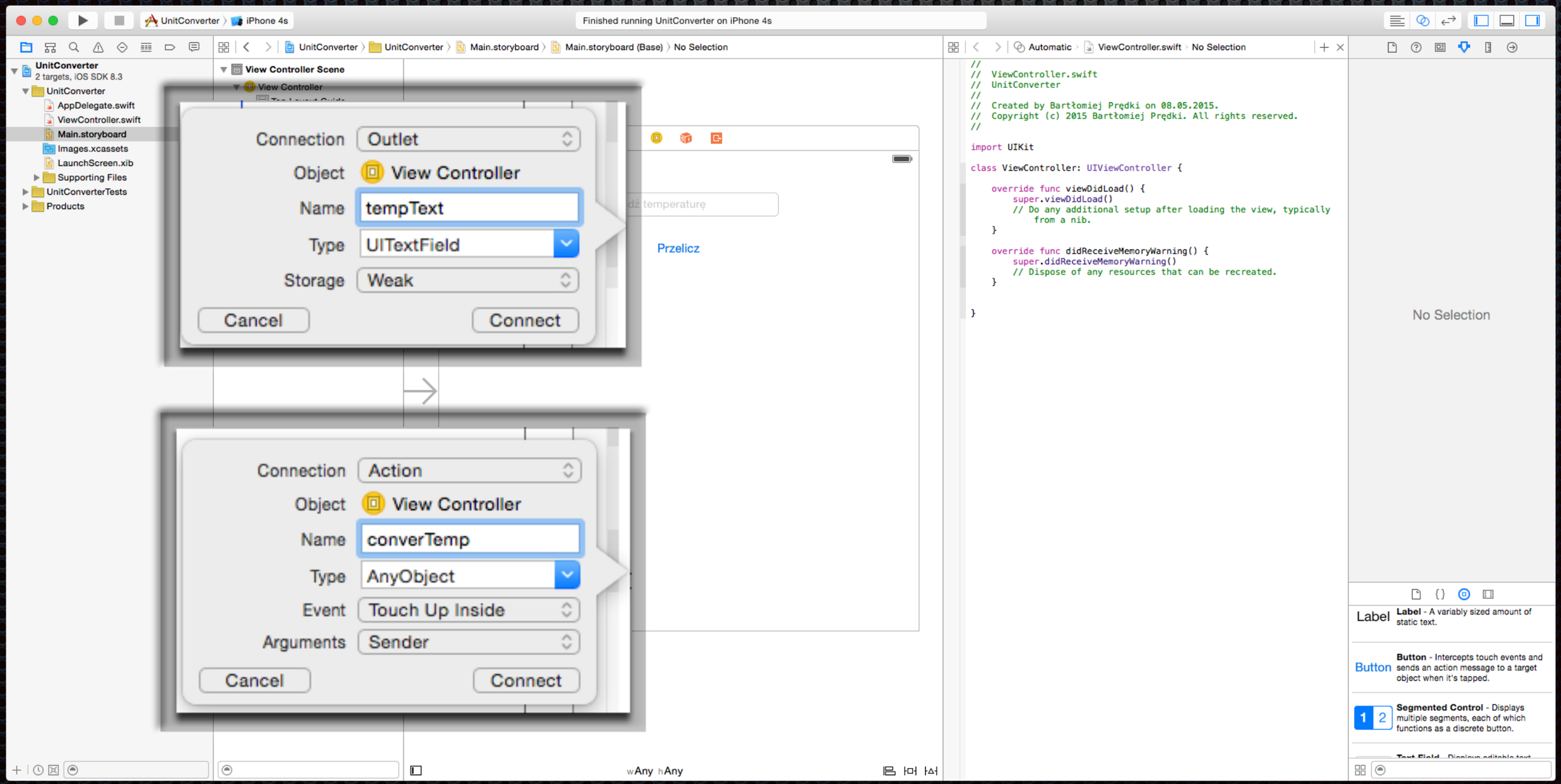
Konwerter

- Tworzymy gniazdka



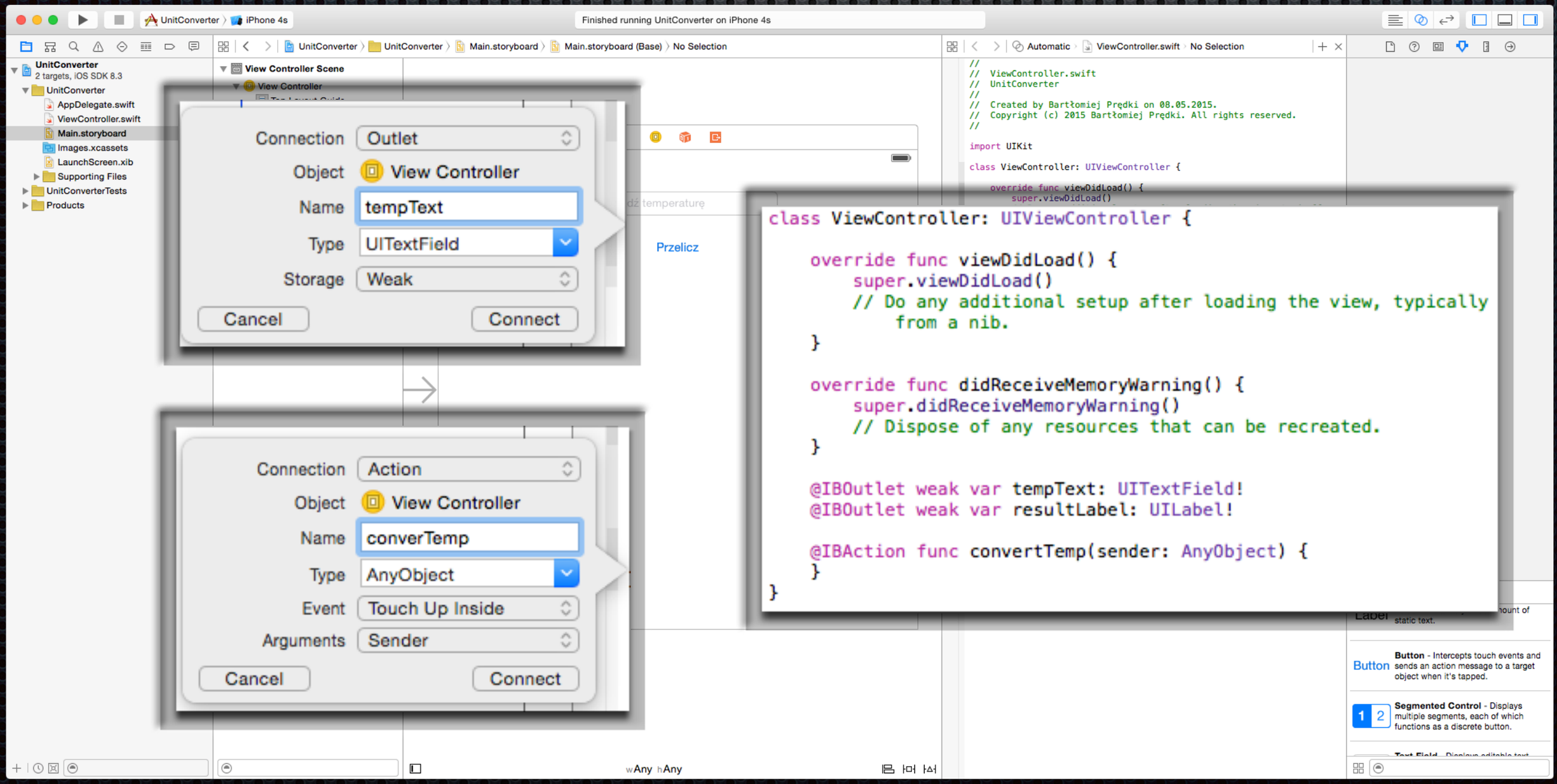
Konwerter

- Tworzymy gniazdka



Konwerter

- Tworzymy gniazdka



Konwerter

Konwerter

- Uzupełniamy metodę

```
@IBAction func convertTemp(sender: AnyObject) {
    let celsius = (tempText.text as NSString).doubleValue
    let fahrenheit = celsius * 1.8 + 32
    let resultText = "Fahrenheit \(fahrenheit)"
    resultLabel.text = resultText
}
```

Konwerter

- Uzupełniamy metodę

```
@IBAction func convertTemp(sender: AnyObject) {
    let celsius = (tempText.text as NSString).doubleValue
    let fahrenheit = celsius * 1.8 + 32
    let resultText = "Fahrenheit \(fahrenheit)"
    resultLabel.text = resultText
}
```

- Dodajemy chowanie klawiatury

```
override func touchesBegan(touches: Set<NSObject>, withEvent event: UIEvent) {
    tempText.endEditing(true)
}
```

Konwerter


iOS Simulator - iPhone 4s - iPhone 4s / iOS 8.3 (12F69)
Carrier
1:04 PM
Game Center
Passbook
Extras
Apple Watch
UnitConverter
Safari

Konwerter


iOS Simulator - iPhone 4s - iPhone 4s / iOS 8.3 (12F69)
Carrier
1:04 PM
Game Center
Passbook
Extras
Apple Watch
UnitConverter
Safari

Do zobaczenia

速