

# Programowanie w Ruby

## Wykład 6

Marcin Młotkowski

14 listopada 2012

# Plan wykładu

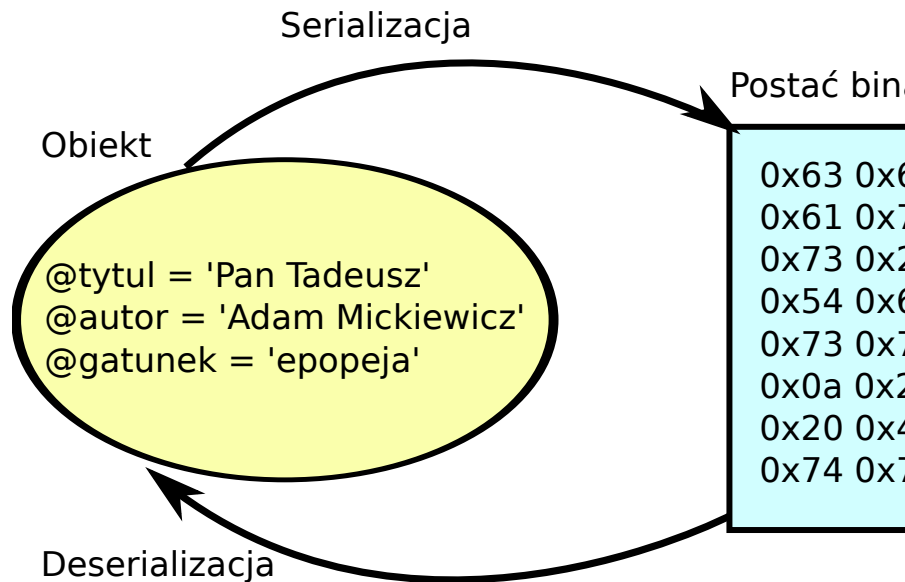
Trwałość obiektów

Bazy danych DBM

Bazy danych SQL

Active records

## Szeregowanie obiektów



# Wybrane formaty zapisu

## YAML: YAML Ain't Markup Language

- ▶ format niezależny od języka
- ▶ format: sformatowany tekst

## Marshal

- ▶ format binarny
- ▶ szybki

# Stosowanie

Import odpowiedniego modułu

```
require 'yaml'
```

# Stosowanie

Import odpowiedniego modułu

```
require 'yaml'
```

Przykład 1.

```
str = YAML.dump([1, 'dwa', 3.0])
```

```
lista = YAML.load(str)
```

# Stosowanie

Import odpowiedniego modułu

```
require 'yaml'
```

Przykład 1.

```
str = YAML.dump([1, 'dwa', 3.0])  
lista = YAML.load(str)
```

Przykład 2.

```
open(plik, 'w') { | f | YAML.dump(obiekt, f) }  
obiekt = open(plik, 'r') { | f | YAML.load(f) }
```

## Metoda to\_yaml

```
[1, "dwa", 3.0].to_yaml
```

---



## Metoda to\_yaml

```
[1, "dwa", 3.0].to_yaml
```

```
---
```

```
- 1
```

```
- dwa
```

```
- 3.0
```

# Marshal

Marshal.dump

Marshal.load

---

# Plan wykładu

Trwałość obiektów

Bazy danych DBM

Bazy danych SQL

Active records

## Co to takiego

- ▶ DBM: database manager
- ▶ Do przechowywania prostych danych
- ▶ Każda baza danych to po prostu tablica haszująca
- ▶ Przykład: Berkeley DB

# Schemat przetwarzania

Import modułu

```
require 'dbm'
```

# Schemat przetwarzania

Import modułu

```
require 'dbm'
```

```
DBM.open(@dbm_name) do | db |
```

```
  db[klucz] = wartosc
```

```
  puts db[klucz]
```

```
end
```

## Porada

DBM-y można traktować jak kolekcje

## Porada

DBM-y można traktować jak kolekcje

```
DBM.open(@dbm_name) do | db |  
  db.each { |k, v| puts "#{k} => #{v}" }  
end
```



## Zapis i odczyt obiektów

```
db[klucz] = Marshal.dump(obj)
obj = Marshal.load(db[klucz])
```

# Plan wykładu

Trwałość obiektów

Bazy danych DBM

**Bazy danych SQL**

Active records

# Mechanizm dostępu do bazy danych

## DataBase Interface — DBI

definiuje zbiór funkcji i wyjątków jakie powinny być implementowane przez moduły do obsługi różnych baz danych

# Przykład

## Sqlite

- ▶ Implementuje SQL
- ▶ Nie wymaga zewnętrznego serwera (osadzona baza danych)
- ▶ Baza jest pamiętana w jednym pliku
- ▶ Implementuje DBI

## Połączenie z serwerem bazy danych

```
require 'dbi'
```

```
db = DBI.connect("DBI:SQLite3:test.db")
```

Inny przykład: MySQL

```
db = DBI.connect("DBI:Mysql:test:localhost", "testuser",  
"testpass")
```

## Połączenie z serwerem bazy danych

```
require 'dbi'
```

```
db = DBI.connect("DBI:SQLite3:test.db")
```

Inny przykład: MySQL

```
db = DBI.connect("DBI:Mysql:test:localhost", "testuser",  
"testpass")
```

Zamknięcie połączenia

```
db.disconnect
```

# Operacje na otwartej bazie danych

- ▶ Operacje niezwracające żadnych danych
- ▶ Operacje zwracające listę wierszy
- ▶ Operacje zwracające liczbę

## Operacje bez wyniku

Utworzenie nowej tabeli

```
db.do(" CREATE TABLE Wyklady\  
( Nr INT NOT NULL,\  
Tresc VARCHAR(40) )")
```



## Operacje zwracające liczbę

### Wstawienie wiersza do tabeli

```
row = @db.do("INSERT INTO Wyklady\  
(Nr, Tresc) VALUES (#{id}, '#{t}')")  
puts "Wstawiono #{row}"
```

# Wyszukiwanie

## Przykład 1.

```
rows = db.execute("SELECT * FROM Wyklady")
rows.each do | row |
  puts "#{row[0]}, #{row[1]}"
end
rows.finish
```

## Przykład 2.

```
rows = db.execute("SELECT * FROM Wyklady")
rows.each do | row |
  puts "#{row['Nr']}, #{row['Tresc']}"
end
rows.finish
```

## Kilka drobnych przykładów

```
db.select_all("SELECT * FROM Wyklady") do | row |  
  puts "#{row[0]}, #{row[1]}"  
end
```

```
db.select_one("SELECT ... WHERE NR=1")
```

# Plan wykładu

Trwałość obiektów

Bazy danych DBM

Bazy danych SQL

Active records

# Dygresja: symbole

## Symbol

identyfikator odpowiadający łańcuchowi znaków

## Tworzenie identyfikatorów

:obiekt

:**'obiekt'**

# Odwzorowania obiektowo-relacyjne

```
class Student
  @imie
  @nazw
  ...
end
```

```
CREATE TABLE stud (
  imie VARCHAR(30),
  nazw VARCHAR(40),
  ...);
```

# Zarządzanie odwzorowaniem O-R w Rubym

## ActiveRecord

- ▶ Zapewnia automatyczne odwzorowanie między obiektami a bazą danych
- ▶ Ukrywa wykorzystanie SQL
- ▶ Implementuje związki między tabelami
- ▶ **Wymaga przestrzegania wielu konwencji**

# Użycie ActiveRecord

## Zadanie

Zaprogramować bazę danych zawierającą dane o

- ▶ studentach
- ▶ zadaniach oddanych przez studentów



# Rozwiązanie w SQL

## Deklaracja tabeli studs

```
'CREATE TABLE studs (  
    id INT(11) NOT NULL AUTO_INCREMENT,  
    index VARCHAR (20),  
    punkty INT,  
    PRIMARY KEY(id))'
```

## Deklaracja tabeli stud\_exercises

```
'CREATE TABLE stud_exercises (  
    id INT NOT NULL AUTO_INCREMENT,  
    stud_id INT(11),  
    zad INT,  
    tresc TEXT,  
    PRIMARY KEY(id))'
```

# Rozwiązanie obiektowe

## Deklaracja klas

```
require 'activerecord'
```

```
class Stud < ActiveRecord::Base  
  has_many :stud_exercises  
end
```

```
class StudExercise < ActiveRecord::Base  
  belongs_to :stud  
end
```

## Active record

|              |               |
|--------------|---------------|
| Klasa        | tabela        |
| Obiekt       | wiersz tabeli |
| Pole obiektu | kolumna       |

# Konwencje

`class stud`

`CREATE TABLE studs`

`class StudExercise`

`CREATE TABLE stud_exercise`

## Tworzenie obiektów

```
madry = Stud.create(:index => '10011101', :punkty => 0)
zadanie = StudExercise.create(:stud_int => madry,
  :zad => 2, tresc => 'puts 0')
```

# Odwołania do elementów bazy danych

## Pobieranie danych

```
studs = Stud.find (:all,  
                  :conditions => "index > '100011' ",  
                  :order => 'index DESC')
```

# Odwołania do elementów bazy danych

## Pobieranie danych

```
studs = Stud.find (:all,  
                  :conditions => "index > '100011' ",  
                  :order => 'index DESC')
```

## Modyfikacja danych

```
studs.each do |s|  
  s.punkty = s.punkty + 10  
  s.save  
end
```