meetup

ERLANG, NIE AŻ TAKI FUNKCYJNY

Prowadzący: Michał Herda

07/05/2019, 18:00
MEBLOTEKA YELLOW

Telco Camp

by ERICSSON

tl;dl

- 0: wstęp – trochę o wszystkim
  - trochę o mnie - prelegencie
  - trochę o Javie - języku obiektowym
  - trochę o Erlangu – języku funkcyjnym
    - ...ale też obiektowym, to tylko kwestia skali (;
  - trochę o faunie i florze - naszym modelu obiektowym
- 1: pojęcia obiektowe
- 2: konstrukcja “obiektu” w Erlangu
- 3: dziedziczenie ze zmianą danych
- 4: dziedziczenie ze zmianą zachowania

Trochę o mnie

https://phoe.github.io

- W godzinach pracy:
  - w Ericssonie od 2016 roku
  - Java, XML/XSLT, JS, Erlang
  - odrobina doświadczenia jako PO i SM
- Po godzinach pracy:
  - programista Common Lispu
  - uczę się SQLa (w wariancie PostgreSQL)
  - miłośnik wolnego oprogramowania
  - czasem muzykuję (:

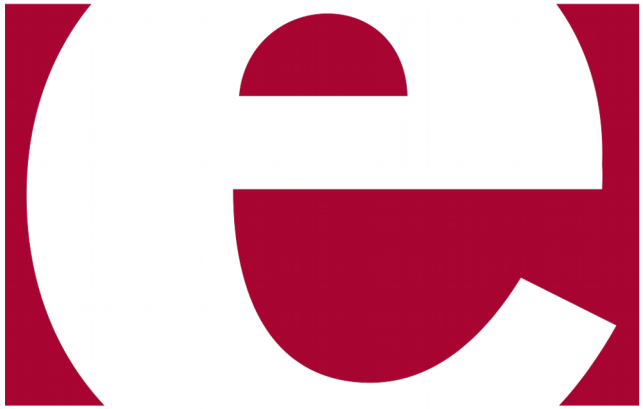
Trochę o Javie

- Wydany w 1995 roku
- Uwolniony w 2007 roku
- Maszyna wirtualna: JVM
- Podstawową jednostką jest klasa
- Składniowo przypomina C++
- Obiektowy

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Trochę o Erlangu

- Wydany w 1986 roku
- Uwolniony w 1998 roku
- Maszyna wirtualna: BEAM
- Podstawową jednostką jest moduł
- Składniowo przypomina Prolog
- ~~Funkcyjny~~ Aktorowy

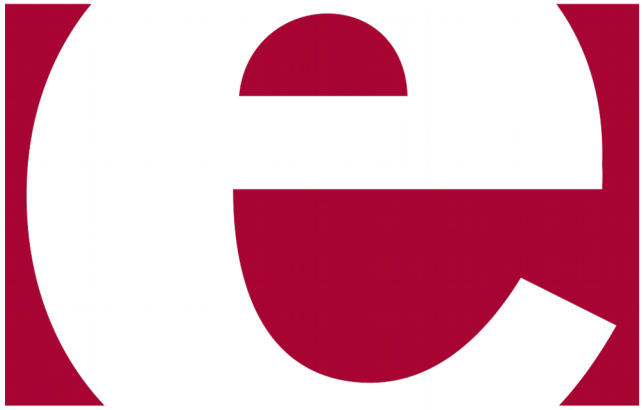


```
-module(hello_world).
-export([hello/0])

hello() -> io:fwrite("Hello, World\n").
```

Trochę o Erlangu

- Telekomunikacja
  - **Ericsson**, Motorola, Nokia, Cisco, ...
- Operatorzy telekomunikacyjni
  - EE, T-Mobile, 2600Hz, Telia, ...
- Komunikacja internetowa
  - ProcessOne, WhatsApp, Yahoo, ~~Facebook Chat,~~ ...
- Kolejki wiadomości i bazy danych
  - RabbitMQ, CouchDB, Riak, Amazon SimpleDB, CouchBase, ...
- Serwery gier
  - Electronic Arts, Riot Games, Wooga, ~~Battlestar Galactica Online~~, ...
- ...



Trochę o faunie i florze

- **Owoce - jabłka**, **banany**, **ogórki**
- **Zwierzę** - może **dostawać**, **trzymać** i **jeść** owoce
- Gatunki zwierząt
  - **Koń** - je tylko jabłka
  - **Małpa** - je tylko banany
  - **Człowiek** - je wszystko
- Rodzaje ludzi
  - **Zbieracz** - nigdy nie je niczego
  - **Pożeracz** – je wszystko od razu po dostaniu
  - **Chory** – nie wolno mu jeść niczego

Trochę o faunie i florze

```
enum Fruit {APPLE, BANANA, CUCUMBER}

class Animal {
    listFruit(); // prints all fruit
    give(Fruit); // gives new fruit to animal
    eat();       // eats last fruit from inventory
    eat(Fruit);  // eats the provided fruit from inventory
}

class Horse extends Animal;  // eats apples only
class Monkey extends Animal; // eats bananas only
class Human extends Animal;  // eats all fruit

class Hoarder extends Human;  // eats nothing
class Devourer extends Human; // instantly eats everything
class Sick extends Human;     // dies upon eating anything
```

Implementacja wzorcowa w Javie

<TelcoCamp.java>

Pojęcia obiektowe

Pojęcia

Pojęcia

Wzorzec

Pojęcia

Wzorzec

Instancja

Pojęcia

Wzorzec

Instancja

Zachowanie

Pojęcia

Wzorzec

Instancja

Zachowanie

Stan

Pojęcia

	Java	Erlang
Wzorzec		
Instancja		
Zachowanie		
Stan		

Pojęcia

	Java	**Erlang**
Wzorzec	Klasa	
Instancja	Obiekt	
Zachowanie	Metoda	
Stan	Pole	

Pojęcia

	Java	**Erlang**
Wzorzec	Klasa	Funkcja
Instancja	Obiekt	Proces
Zachowanie	Metoda	Wiadomość
Stan	Pole	Argument

Konstrukcja “obiektu” w Erlangu

Konstrukcja ~~“obiektu”~~ w Erlangu

Konstrukcja procesu w Erlangu

Klasa kontra funkcja

```
class Animal {
    ...
}
```

```
animal () ->
  ....
```

Metoda kontra wiadomość

```
class Animal {
    void listFruit() {...}
    void give(Fruit fruit) {...}
    void eat() {...}
    void eat(Fruit fruit) {...}
}
```

```
animal() ->
    receive
        {list_fruit}  -> ...;
        {give, Fruit} -> ...;
        {eat}         -> ...;
        {eat, Fruit}  -> ...
    end.
```

Metoda kontra wiadomość

```
class Animal {
    void listFruit() {...}
    void give(Fruit fruit) {...}
    void eat() {...}
    void eat(Fruit fruit) {...}
}
```

```
animal() ->
    receive
        {list_fruit}  -> ..., animal();
        {give, Fruit} -> ..., animal();
        {eat}         -> ..., animal();
        {eat, Fruit}  -> ..., animal()
    end.
```

Pole kontra argument

```
class Animal {
    List<Fruit> inventory = ...;

    void listFruit() {...}
    void give(Fruit fruit) {...}
    void eat() {...}
    void eat(Fruit fruit) {...}
}
```

```
animal(Inventory) ->
    receive
        {list_fruit}  -> ..., animal(Inventory);
        {give, Fruit} -> ..., animal(Inventory);
        {eat}         -> ..., animal(Inventory);
        {eat, Fruit}  -> ..., animal(Inventory)
    end.
```

Pole kontra argument

```
class Animal {
    List<Fruit> inventory = ...;

    void listFruit() {...}
    void give(Fruit fruit) {...}
    void eat() {...}
    void eat(Fruit fruit) {...}
}
```

```
animal(Inventory) ->
    receive
        {list_fruit}  -> ..., animal(Inventory);
        {give, Fruit} -> ..., animal(NewInventory);
        {eat}         -> ..., animal(NewInventory);
        {eat, Fruit}  -> ..., animal(NewInventory)
    end.
```

Obiekt kontra proces

```
MyClass myClass = new MyClass(Arg1, Arg2, ...);
```

```
Pid = erlang:spawn(Module, Function, Arguments),
```

Obiekt kontra proces

```
MyClass myClass = new MyClass(Arg1, Arg2, ...);
```

```
Pid = erlang:spawn(Module, Function, Arguments),
```

```
Animal animal = new Animal(new Fruit[]{APPLE});
```

```
Pid = erlang:spawn(telcocamp, animal, [[apple]]),
```

Dziedziczenie ze zmianą danych

Trochę o faunie i florze

- **Owoce - jabłka**, **banany**, **ogórki**
- **Zwierzę** - może **dostawać**, **trzymać** i **jeść** owoce
- Gatunki zwierząt
  - **Koń** - je tylko jabłka
  - **Małpa** - je tylko banany
  - **Człowiek** - je wszystko
- Rodzaje ludzi
  - **Zbieracz** - nigdy nie je niczego
  - **Pożeracz** – je wszystko od razu po dostaniu
  - **Chory** – nie wolno mu jeść niczego

Trochę o faunie i florze

- **Owoce** - **jabłka**, **banany**, **ogórki**
- **Zwierzę** - może **dostawać**, **trzymać** i **jeść** owoce
- Gatunki zwierząt
  - **Koń** - je tylko jabłka
  - **Małpa** - je tylko banany
  - **Człowiek** - je wszystko
- Rodzaje ludzi
  - **Zbieracz** - nigdy nie je niczego
  - **Pożeracz** – je wszystko od razu po dostaniu
  - **Chory** – nie wolno mu jeść niczego

Dziedziczenie ze zmianą danych

```
class Horse extends Animal {
    Horse() {
        super(new Fruit[]{APPLE});
    }
}

class Monkey extends Animal {
    Monkey() {
        super(new Fruit[]{BANANA});
    }
}

class Human extends Animal {
    Human() {
        super(new Fruit[]{APPLE, BANANA, CUCUMBER});
    }
}
```

Dziedziczenie ze zmianą danych

```
horse() ->
    animal([apple]).

monkey() ->
    animal([banana]).

human() ->
    animal([apple, banana, cucumber]).
```

Dziedziczenie ze zmianą zachowania

Dziedziczenie ze zmianą danych

```
class Hoarder extends Human {
    @Override
    ...
}

class Devourer extends Human {
    @Override
    ...
}

class Sick extends Human {
    @Override
    ...
}
```

Dziedziczenie ze zmianą danych

```
hoarder() ->
    human(...?).

devourer() ->
    human(...?).

sick() ->
    human(...?).
```

Podsumowanie

Użyteczność

- Jako wyjaśnienie?
  - ...mam nadzieję, że tak ;__;
- Jako ćwiczenie?
  - Do nauki Erlanga – warto
  - Do nauki systemów aktorowych – również warto
- Na produkcji?
  - **hell no**
  - Istnieją dojrzalsze frameworki erlangowe

gen_server / gen_event / gen_statem

gen_server

Module

gen_server

Module Summary

Generic server behavior.

Description

This behavior module provides the server of a client-server relation. A generic server process (`gen_server`) implemented using this module has a standard set of interface functions and includes functionality for tracing and error reporting. It also fits into an OTP supervision tree. For more information, see section **`gen_server Behaviour`** in OTP Design Principles.

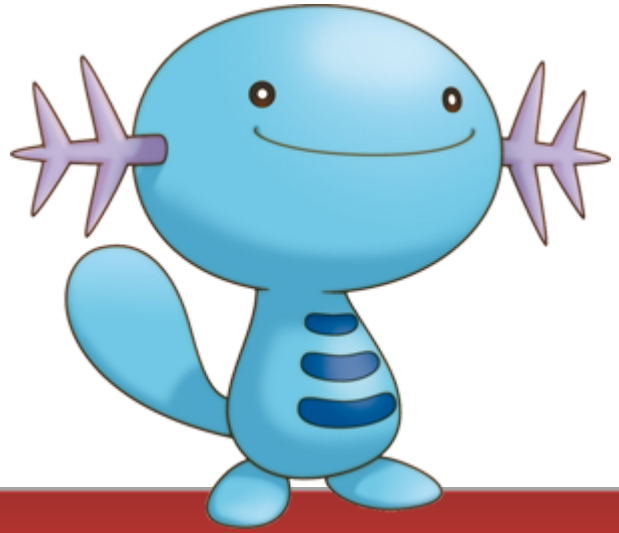
A `gen_server` process assumes all specific parts to be located in a callback module exporting a predefined set of functions. The relationship between the behavior functions and the callback functions is as follows:

```
gen_server module            Callback module
-----------------            ---------------
gen_server:start
gen_server:start_link -----> Module:init/1
```

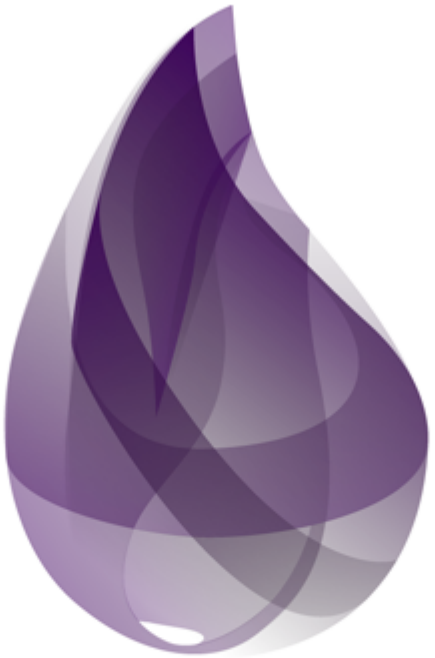
Wrapper for Object-Oriented Programming in Erlang

Wrapper for Object-Oriented Programming in Erlang

WOOPER



Elixir



elixir

tl;dl

- 0: wstęp – trochę o wszystkim
  - trochę o mnie - prelegencie
  - trochę o Javie - języku obiektowym
  - trochę o Erlangu – języku funkcyjnym
    - ...ale też obiektowym, to tylko kwestia skali (;
  - trochę o faunie i florze - naszym modelu obiektowym
- 1: pojęcia obiektowe
- 2: konstrukcja “obiektu” w Erlangu
- 3: dziedziczenie ze zmianą danych
- 4: dziedziczenie ze zmianą zachowania

tl;dl

- 0: wstęp – trochę o wszystkim
  - trochę o mnie - prelegencie
  - trochę o Javie - języku obiektowym
  - trochę o Erlangu – języku funkcyjnym
    - ...ale też obiektowym, to tylko kwestia skali (;
  - trochę o faunie i florze - naszym modelu obiektowym
- 1: pojęcia obiektowe
- 2: konstrukcja “obiektu” w Erlangu
- 3: dziedziczenie ze zmianą danych
- 4: dziedziczenie ze zmianą zachowania

tl;dl

- ...ale też obiektowym, to tylko kwestia skali (;
  - izolacja
  - wymiana wiadomości
  - polimorfizm

Q&A

<3

<3

https://github.com/phoe/telco-camp-erlang