Programowanie obiektowe

Podstawowe cechy i możliwości języka Scala

mgr inż. Krzysztof Szwarc

krzysztof@szwarc.net.pl

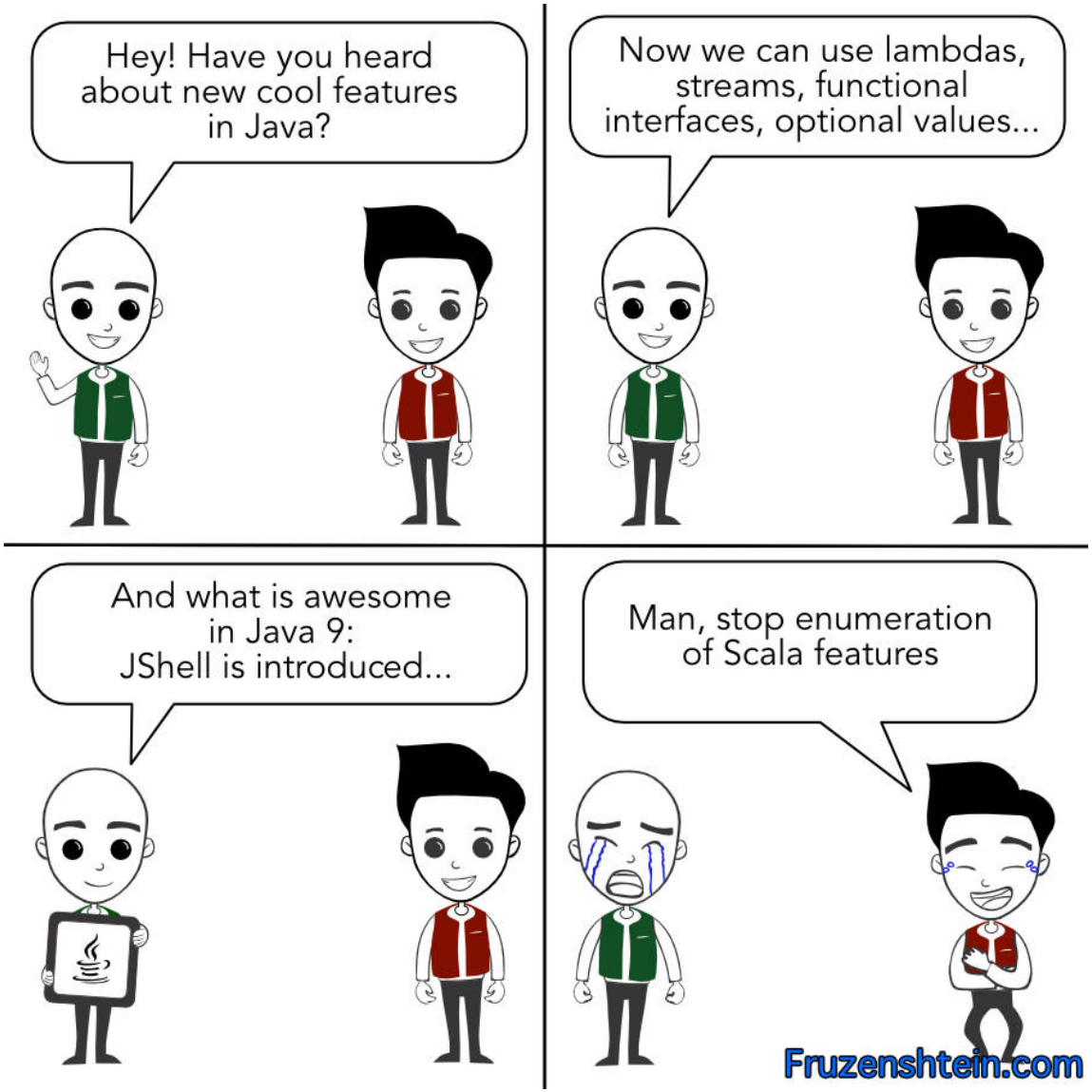
Sosnowiec, 2017

Informacje wstępne

Opis

Scala jest językiem programowania łączącym cechy języków funkcyjnych i obiektowych. Martin Odersky rozpoczął prace nad jego stworzeniem w 2001 roku, by dwa lata później wydać go do wewnętrznych zastosowań. Język upubliczniony został w styczniu 2004 roku na platformie Javy oraz .NET (wsparcie dla niego zostało zakończone w 2012 roku). Druga wersja Scali pojawiła się w marcu 2006 roku (najnowsza wersja: 2.12.2).

Scala vs Java

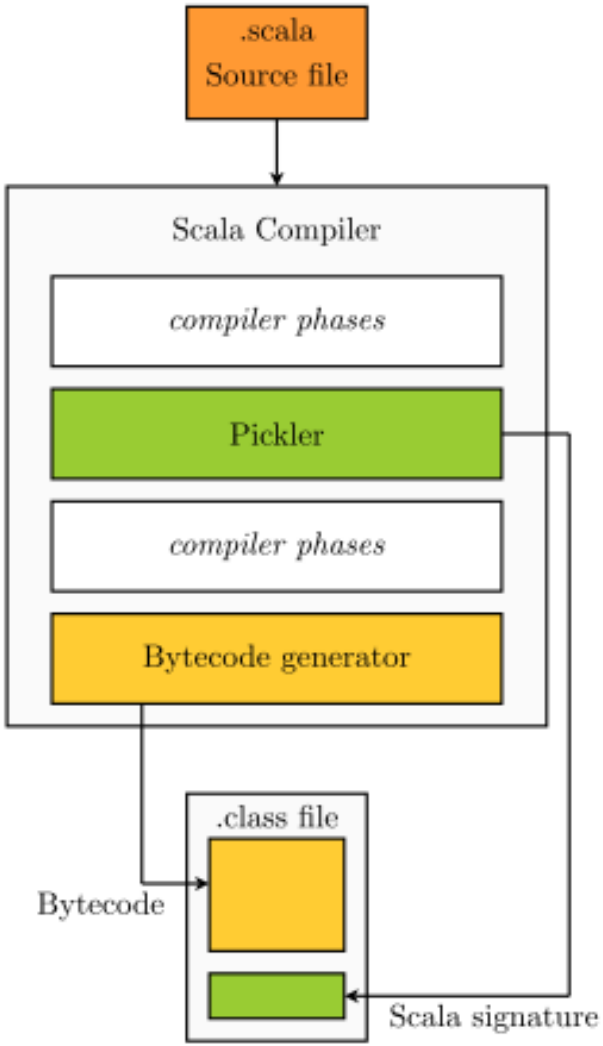


Charakterystyka Scali

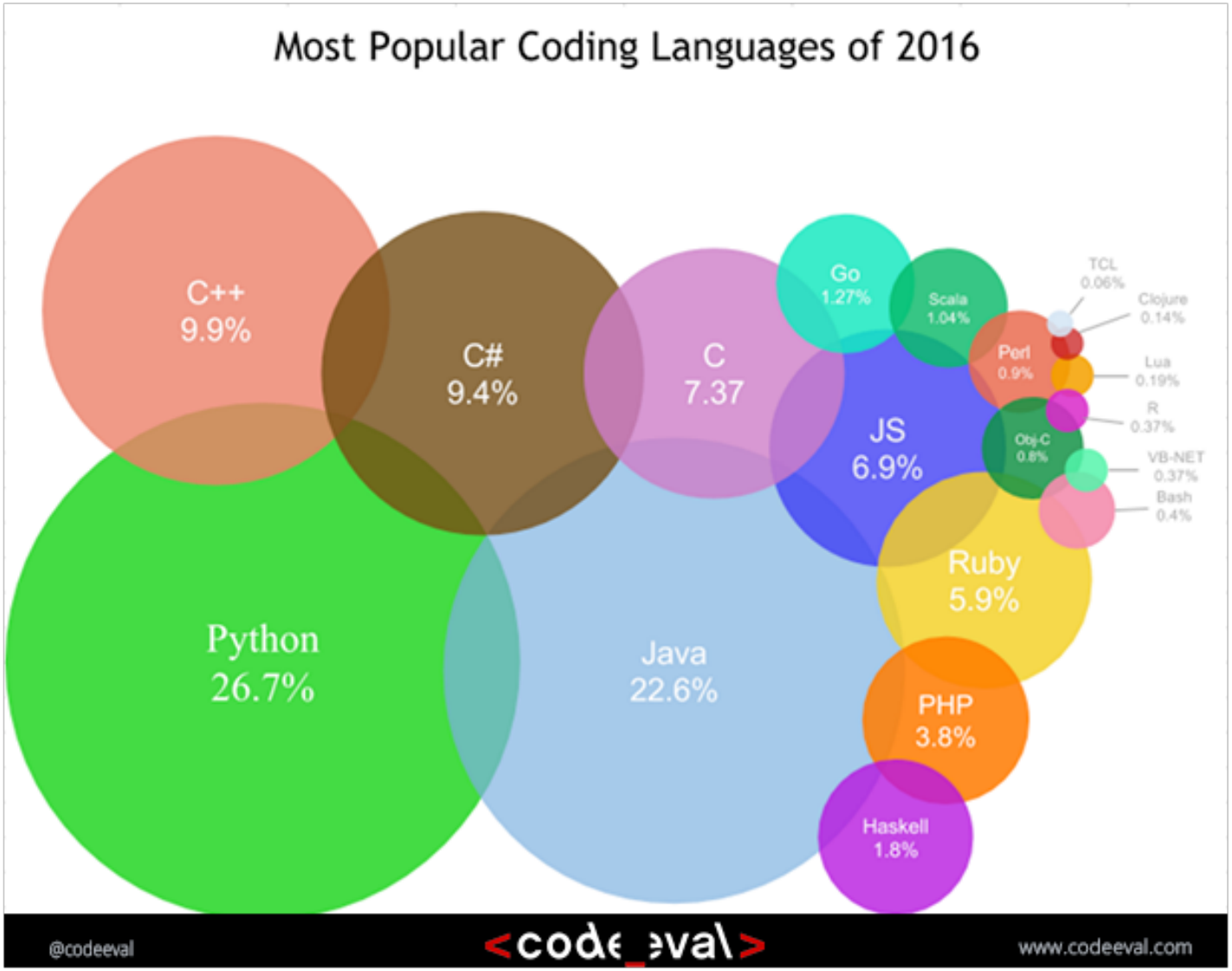
1. Łączy cechy języków funkcyjnych i obiektowych.
2. Jest silnie i statycznie typowany.
3. Ma mechanizm inferencji typów.
4. Działa na JVM.
5. Może wykonywać kod Javy.
6. Wszystkie typy są obiektami.
7. Umożliwia tworzenie funkcji zagnieżdżonych.
8. Funkcje są obiektami.

⋮

Kompilacja kodu



Popularność Scali



Wyrażenia

Słowa kluczowe

W Scali występują trzy słowa kluczowe umożliwiają definiowanie wyrażenia - def, var oraz val. Słowo def pozwala na zdefiniowanie wyrażenia, którego wartość będzie wyliczana zawsze w chwili odwołania się do tego wyrażenia. Przy pomocy słowa var można zdefiniować wyrażenie, którego wartość może podlegać zmianom (ang. mutable), a słowo kluczowe val pozwala zdefiniować wartości, które nie podlegają zmianom (ang. immutable).

Deklaracja stałej i zmiennej

```
[lazy] val Nazwa = wartość
[lazy] val Nazwa:typ = wartość
[lazy] val Nazwa = wartość:typ

var nazwa = wartość
var nazwa:typ = wartość
var nazwa = wartość:typ
```

lazy val, a val

```
val X = { println("x"); 15 } // x
lazy val Y = { println("y"); X+1 }
println("x to: " + X); // x to: 15
println("y to: " + Y) // y y to: 16
```

Definicja klasycznej funkcji

```
def nazwa(par:typ,...) : typ = {
    ciało }
```

Przykład

```
def inkrementuj(x:Int) : Int = {
    x + 1 }
val wynik = inkrementujWartosc(2)
```

W przypadku nieumieszczenia słowa kluczowego return zwrócona zostanie wartość ostatniego wyrażenia (dla przykładu $x + 1$). Podanie zwracanego typu jest opcjonalne. Odpowiednikiem Javowego void jest Unit.

Funkcje wyższego rzędu

Opis

Funkcja wyższego rzędu to funkcja, która zwraca lub przyjmuje jako argument inne funkcje.

```
def moneyTransfer(amount: Double,
    provider: Double => Double):Double={
      amount + 10 + provider(amount)
}

def prov(money: Double) = money * 0.05
println(moneyTransfer(100, prov))
```

Funkcje zagnieżdżone

Opis

Funkcja zagnieżdżona (w Scali nazywana lokalną) to funkcja, która znajduje się w innej funkcji.

```
def factorial(i: Int): Int = {
    def fact(i: Int, acc: Int): Int = {
        if (i <= 1)
            acc
        else
            fact(i - 1, i * acc)
    }
    fact(i, 1)
}
```

Rozwijanie funkcji

Opis

Rozwijanie funkcji to operacja występująca w funkcyjnych językach programowania polegająca na przekształceniu funkcji, która pobiera parę argumentów i zwraca wynik w funkcję, która po pobraniu argumentu zwraca funkcję, która pobiera argument i zwraca wynik.

```
def add(x:Int, y:Int) = x + y
def addCurried(x:Int) = (y:Int) => x + y
add(1, 2)    // 3
addCurried(1)(2)    // 3
```

Klasy w Scali - pełny zapis

```
class Nazwa(parametr: typ,...) {
    var nazwaPola: typ = parametr

    def metoda(nowaWartosc: typ) {
        nazwaPola = nowaWartosc;
    }
}
```

Klasy w Scali - skrócony zapis

```
class Nazwa(var/val nazwaPola: typ,...) {
    def metoda(nowaWartosc: typ) {
        nazwaPola = nowaWartosc;
    }
}
```

Klasy w Scali - przykład dla pełnego zapisu

```
class Punkt(xs : Int, ys : Int) {
    var x = xs
    var y = ys

    def ustawWartosci(xs: Int, ys: Int) {
        x = xs
        y = ys
    }
}
val punkt = new Punkt(5,3)
punkt.ustawWartosci(4,3)
println(punkt.x)
```

Klasy w Scali - przykład dla krótkiego zapisu

```
class Punkt(var x : Int, var y : Int) {
    def ustawWartosci(xs: Int, ys: Int) {
        x = xs
        y = ys
    }
}
val punkt = new Punkt(5,3)
punkt.ustawWartosci(4,3)
println(punkt.x)
```

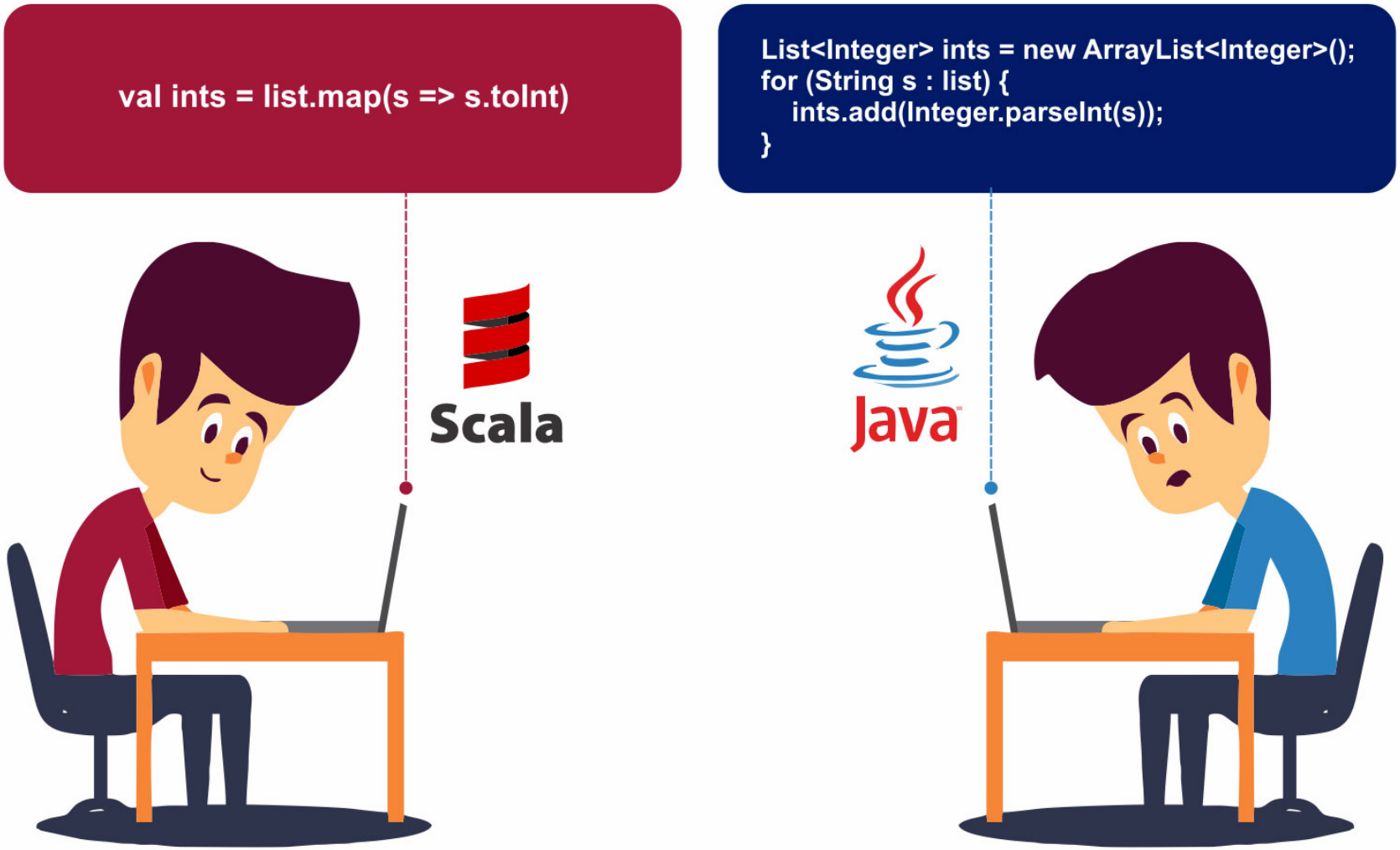
Efekt kompilacji

```
class Cat(val catId:Int, var name:String)
```

```
C:\Users\Kamill\Desktop>scalac app.scala

C:\Users\Kamill\Desktop>javap -p Cat.class
Compiled from "app.scala"
public class Cat {
  private final int catId;
  private java.lang.String name;
  public int catId();
  public java.lang.String name();
  public void name_$eq(java.lang.String);
  public Cat(int, java.lang.String);
}
```

Wniosek


val ints = list.map(s => s.toInt)
Scala
List<Integer> ints = new ArrayList<Integer>();
for (String s : list) {
ints.add(Integer.parseInt(s));
}
Java

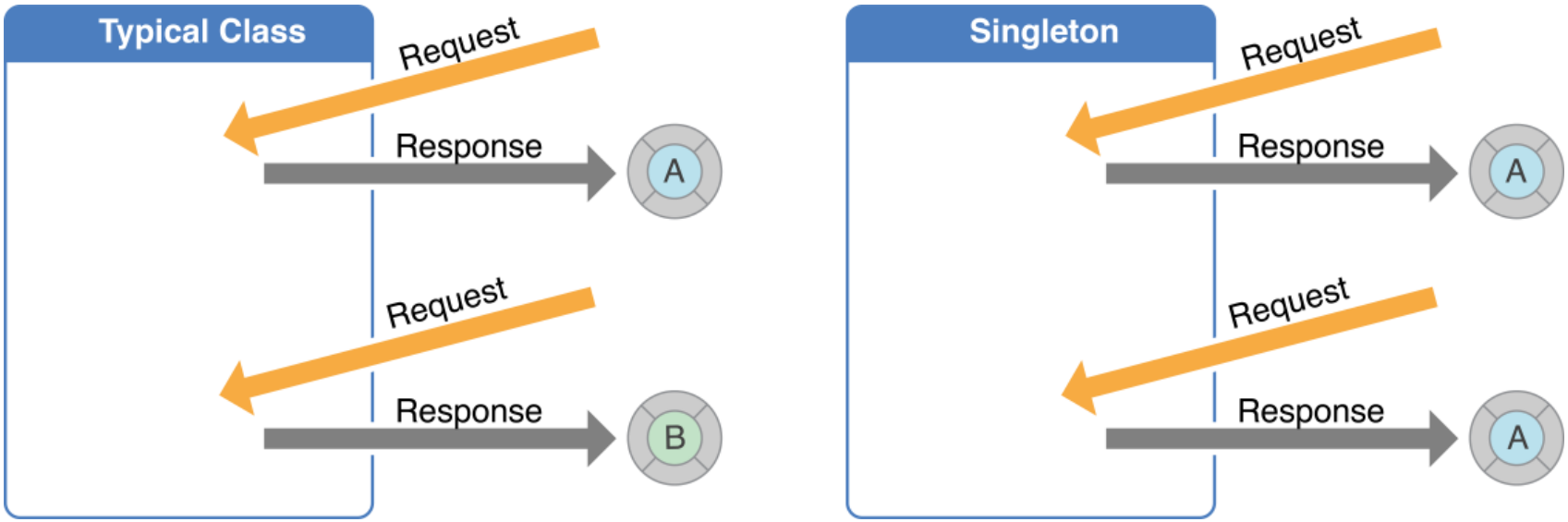
Singleton - (anty)wzorzec

Opis

Singleton zapewnia występowanie maksymalnie jednej instancji danej klasy (tworzenie obiektu następuje przy pierwszej próbie użycia) oraz umożliwia globalny dostęp do stworzonego obiektu. Zazwyczaj wykorzystywany jest do przechowywania konfiguracji aplikacji oraz utrzymywania połączenia z bazą danych. Przy implementacji należy pamiętać o środowisku wielowątkowym i trudnościach z testowaniem.

Singleton - schemat działania

Singleton
- instance : Singleton = null
+ getInstance() : Singleton - Singleton() : void



Singleton w Javie

```
public class Conf {
    private static final Conf INS = new
        Conf();

    private Conf() {
    }

    public static Conf getInstance() {
        return INS;
    }
}
```

Singleton w Javie z lazy loading

```
public class Conf {
    private static Conf ins = null;

    private Conf() {
    }

    public static Conf getInstance() {
       if(ins == null) {
            ins = new Conf();
        }
        return ins;
    }
}
```

Singleton w Javie z lazy loading 2

```
public class Conf {
    private static Conf ins = null;

    private Conf() {
    }

    public static Conf getInstance() {
      if(ins == null) {
          synchronized(Conf.class) {
              ins = new Conf();
          }
       }
       return ins;
    }
}
```

Singleton w Scali

```
object Conf
```

Typizacja

Silna

Silna typizacja to system typów, w którym każde wyrażenie ma ściśle ustalony typ i nie można go używać w kontekście przeznaczonym dla innych typów. Scala się nią cechuje.

Słaba

Słaba typizacja to system typów, w którym typ wyrażenia może być zmieniony, jeśli wymaga tego kontekst.

```
val liczba = 1;
if ("1" == liczba) {} // comparing
   values of types Int and String using
   '==' will always yield false
```

Inferencja

Opis

Inferencja typów to technika używana w językach programowania, która zwalnia programistę z obowiązku pisania typów i przerzuca obowiązek identyfikacji typów na kompilator.

```
val napis = "Ala ma kota w głowie"
print(napis.getClass.getSimpleName)
// String
```

Funkcja anonimowa

Opis

Funkcja anonimowa jest definicją funkcji, która nie jest powiązana z żadnym identyfikatorem. Jeśli funkcja jest używana tylko jeden raz lub ograniczoną liczbę razy, to użycie funkcji anonimowej może być wygodniejsze niż użycie funkcji posiadającej identyfikator.

```
(x:Int) => x + 1
```

Przykład użycia

```
val wynik = (x:Int) => x + 1
print(wynik(5))
```

val, a def

```
class A(a: Int) {
    val x = { println("x"); a }
    def y = { println("y"); a }
}
val a = new A(1) // x
println(a.x) // 1
println(a.x) // 1
println(a.y) // y 1
println(a.y) // y 1
```

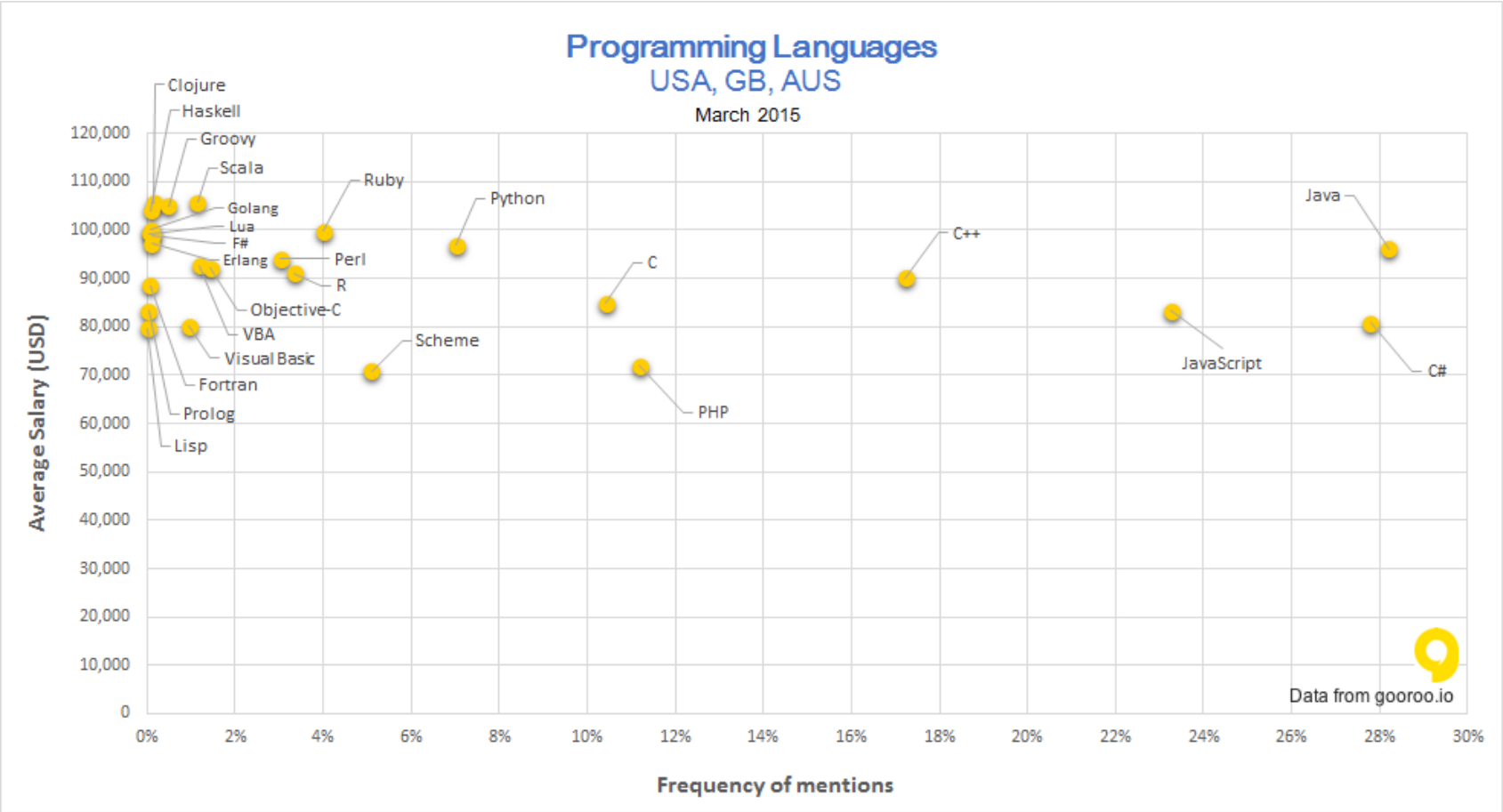
Zapis def $x = e$ nie spowoduje obliczenia e (zostanie ono wyznaczone każdorazowo w momencie użycia x). Natomiast val $x = e$ skutkuje natychmiastowym obliczeniem e i przypisaniem wartości do x.

Obliczenie silni w Scali

```
print("Liczba: ")
val input = scala.io.StdIn.readInt()
print("Silnia z liczby " + input + " to:
    " + factorial(input))

def factorial(n:BigInt):BigInt = {
        if(n == 0 || n == 1) 1
        else n * factorial(n - 1)
}
```

Dla nieprzekonanych



Dziękuję za uwagę