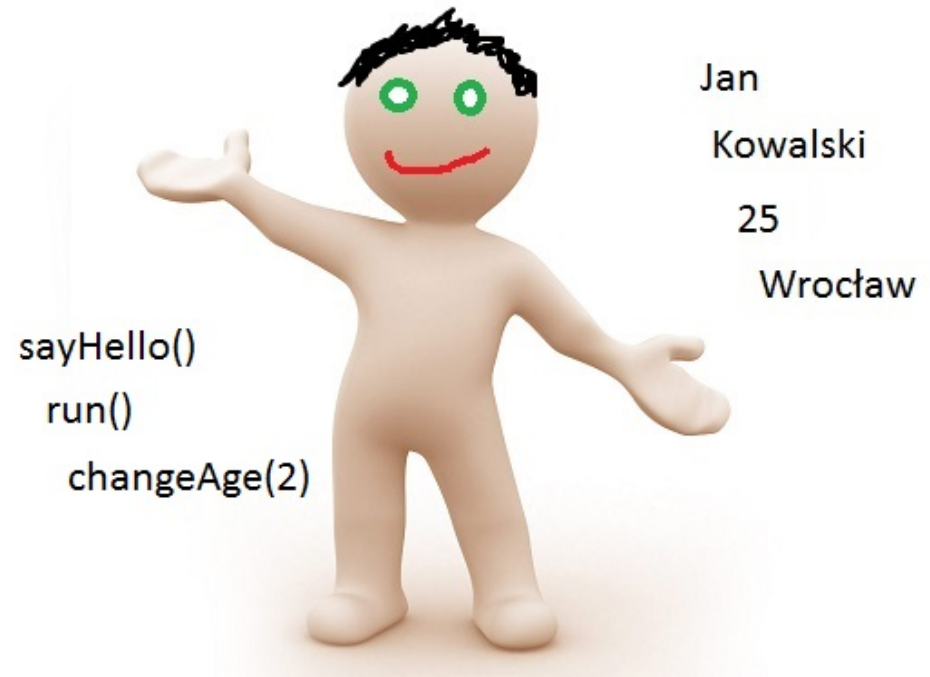


Metody

- Metody pozwalają zdefiniować funkcjonalności klas/obiektów
- Metody wywołujemy (najczęściej) na obiektach
- Mogą zwracać wynik (np. dodawania), ale nie muszą...
- Mogą przyjmować parametry dowolnych typów, ale nie muszą...
- Konstruktory to specjalne metody służące do tworzenia obiektów



Metody

```
typ_zwracany nazwaMetody([parametry]){  
    ciało metody  
    [return]  
}
```

Typy zwracane:

- dowolny typ prosty lub obiektowy (np. int, String, Person) – wymagany **return** jako ostatnia instrukcja. Jeśli metoda coś zwraca to wynik ten można później wykorzystać w innym miejscu
- **void** – gdy metoda nic nie zwraca, wyniku metody nie możemy później wykorzystać, bo tego wyniku po prostu nie ma.

Nazwa:

- lowerCamelCase – tak samo jak zmienne

Parametry:

- są opcjonalne,
- wymieniamy je po przecinku jako listę w postaci: (typ nazwa, typ nazwa, typ nazwa) - tak jak parametry w konstruktorach
- typem może być dowolny typ prosty lub obiektowy

Ciało metody wyznaczają nawiasy klamrowe { }

Metody zwracające wynik

- Jako typ zwracany zapisujemy dowolny typ prosty lub obiektowy
- Ostatnią instrukcją w ciele metody musi być **return**, po której zwrócimy wartość zgodną co do typu z zadeklarowanym typem metody
- Skoro metoda zwraca wynik, to można ten wynik wykorzystać w dalszej części programu
- zwracanie to nie wyświetlanie

```
class Calculator {  
    double sum(double a, double b) {  
        return a + b;  
    }  
}
```

```
public class SalaryCalculator {  
  
    double getEmployeeCost(double nettoSalary) {  
        double bruttoSalary = nettoSalary + nettoSalary * 0.2;  
        double zus = 1300;  
        double result = bruttoSalary + zus;  
        return bruttoSalary + zus;  
    }  
  
}
```

Metody nie zwracające wyniku

- Metody, które nie zwracają wyniku jako typ zwracany mają wpisane słowo kluczowe **void**
- Skoro nie zwracają wyniku, to tego co w nich obliczamy nie możemy później użyć w dalszej części programu
- **void** nie oznacza, że metoda musi coś wyświetlać

```
class Calculator {  
    void sumAndPrint(double a, double b) {  
        double sum = a + b;  
        System.out.println(sum);  
    }  
}
```

```
class SmartPrinter {  
    void print3Times(String text) {  
        System.out.println(text);  
        System.out.println(text);  
        System.out.println(text);  
    }  
}
```

Metody – przykład (demo)

```
class Calculator {  
    void sumAndShow(int a, int b) {  
        int sum = a+b;  
        System.out.println(sum);  
    }  
  
    int sumAndReturn(int a, int b) {  
        int sum = a + b;  
        return sum;  
    }  
}
```

Metody – przykład (demo)

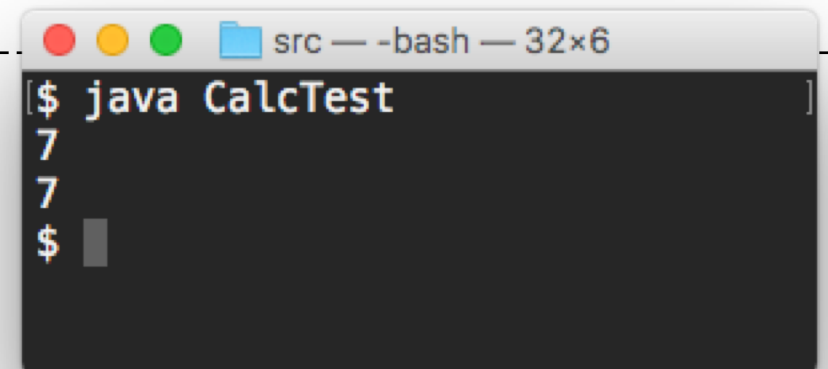
```
class Calculator {  
    void sumAndShow(int a, int b) {  
        int sum = a+b;  
        System.out.println(sum);  
    }  
  
    int sumAndReturn(int a, int b) {  
        int sum = a + b;  
        return sum;  
    }  
}
```

```
class CalcTest {  
    public static void main(String[] args) {  
        Calculator calculator = new Calculator();  
        calculator.sumAndShow(2, 5); // wyświetla 7  
  
        int suma = calculator.sumAndReturn(2, 5);  
        //oblicza, nie wyświetla  
        System.out.println(suma);  
    }  
}
```

Metody – przykład (demo)

```
class Calculator {  
    void sumAndShow(int a, int b) {  
        int sum = a+b;  
        System.out.println(sum);  
    }  
  
    int sumAndReturn(int a, int b) {  
        int sum = a + b;  
        return sum;  
    }  
}
```

```
class CalcTest {  
    public static void main(String[] args) {  
        Calculator calculator = new Calculator();  
        calculator.sumAndShow(2, 5); // wyświetla 7,  
        nie można wykorzystać tego wyniku w dalszej części  
        programu  
  
        int suma = calculator.sumAndReturn(2, 5);  
        //oblicza, nie wyświetla  
        System.out.println(suma);  
    }  
}
```



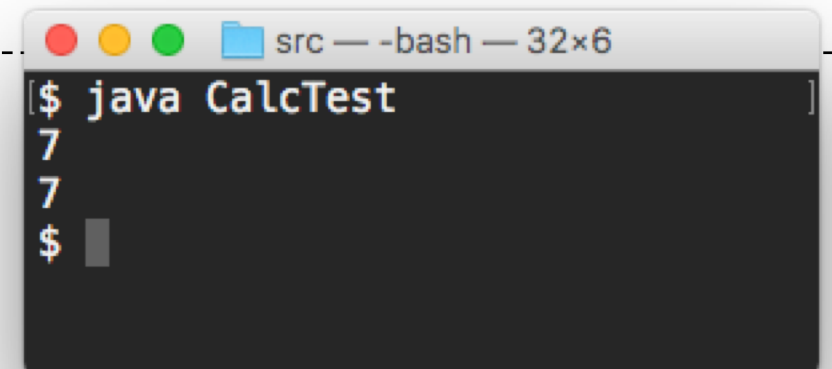
A terminal window titled 'src — -bash — 32x6' showing the execution of the Java program. The command '\$ java CalcTest' is entered, and the output is '7' followed by a blank line and a prompt '\$'.

```
src — -bash — 32x6  
[$ java CalcTest  
7  
7  
$
```

Metody – przykład (demo)

```
class Calculator {  
    void sumAndShow(int a, int b) {  
        int sum = a+b;  
        System.out.println(sum);  
    }  
  
    int sumAndReturn(int a, int b) {  
        int sum = a + b;  
        return sum;  
    }  
}
```

```
class CalcTest {  
    public static void main(String[] args) {  
        Calculator calculator = new Calculator();  
        calculator.sumAndShow(2, 5); // wyświetla 7  
  
        int suma = calculator.sumAndReturn(2, 5); //oblicz  
        i zapamiętaj wynik w zmiennej suma. Możemy wykorzystać  
        wynik w dalszej części programu  
  
        System.out.println(suma); //wyświetl  
    }  
}
```




A terminal window titled 'src — -bash — 32x6' showing the execution of the Java program. The command '\$ java CalcTest' is entered, and the output '7' is displayed twice, followed by a prompt '\$'.

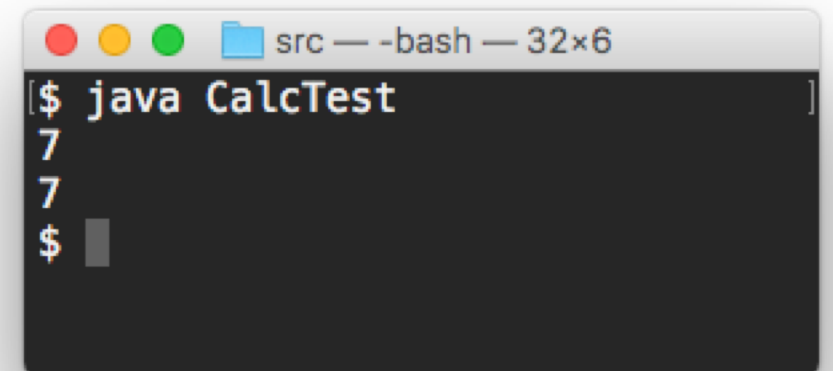
```
src — -bash — 32x6  
[$ java CalcTest  
7  
7  
$
```


Metody – wywołanie w ramach tej samej klasy

```
class Calculator {  
  
    void sumAndShow(int a, int b) {  
        int sum = sumAndReturn(a, b);  
        System.out.println(sum);  
    }  
  
    int sumAndReturn(int a, int b) {  
        int sum = a + b;  
        return sum;  
    }  
}
```



```
class CalcTest {  
    public static void main(String[] args) {  
        Calculator calculator = new Calculator();  
        calculator.sumAndShow(2, 5); // wyświetla 7  
  
        int suma = calculator.sumAndReturn(2, 5); // oblicz  
        System.out.println(suma); // wyświetl  
    }  
}
```



```
src — -bash — 32x6  
$ java CalcTest  
7  
7  
$
```

Metody w klasach z danymi

```
class Person {
    String firstName;
    String lastName;
    int age;
    String city;

    Person(String fn, String ln, int a, String c) {
        firstName = fn;
        lastName = ln;
        age = a;
        city = c;
    }

    void increaseAge() {
        age++;
    }

    void changeAge(int change) {
        age = age + change;
    }
}
```

```
class PersonTest3 {
    public static void main(String[] args) {
        Person person =
            new Person("Jan", "Kowalski", 25, "Wrocław");
        System.out.println(person.age); //25

        person.increaseAge(); //age++
        System.out.println(person.age); //26

        person.changeAge(-6); //age = age + (-6)
        System.out.println(person.age); //20
    }
}
```

firstName = Jan
lastName = Kowalski
age = 25
city = Wrocław

Metody w klasach z danymi

```
class Person {  
    String firstName;  
    String lastName;  
    int age;  
    String city;  
  
    Person(String fn, String ln, int a, String c) {  
        firstName = fn;  
        lastName = ln;  
        age = a;  
        city = c;  
    }  
  
    void increaseAge() {  
        age++;  
    }  
  
    void changeAge(int change) {  
        age = age + change;  
    }  
}
```

```
class PersonTest3 {  
    public static void main(String[] args) {  
        Person person =  
            new Person("Jan", "Kowalski", 25, "Wrocław");  
        System.out.println(person.age); //25  
  
        person.increaseAge(); //age++  
        System.out.println(person.age); //26  
  
        person.changeAge(-6); //age = age + (-6)  
        System.out.println(person.age); //20  
    }  
}
```

firstName = Jan
lastName = Kowalski
age = 26
city = Wrocław

Metody w klasach z danymi (demo)

```
class Person {  
    String firstName;  
    String lastName;  
    int age;  
    String city;  
  
    Person(String fn, String ln, int a, String c) {  
        firstName = fn;  
        lastName = ln;  
        age = a;  
        city = c;  
    }  
  
    void increaseAge() {  
        age++;  
    }  
  
    void changeAge(int change) {  
        age = age + change;  
    }  
}
```

```
class PersonTest3 {  
    public static void main(String[] args) {  
        Person person =  
            new Person("Jan", "Kowalski", 25, "Wrocław");  
        System.out.println(person.age); //25  
  
        person.increaseAge(); //age++  
        System.out.println(person.age); //26  
  
        person.changeAge(-6); //age = age + (-6)  
        System.out.println(person.age); //20  
    }  
}
```

firstName = Jan
lastName = Kowalski
age = 20
city = Wrocław

Metody i typy proste

- Argumenty typów prostych są w Javie przekazywane przez wartość

```
Calculator calculator = new Calculator();  
int x = 5;  
int y = 7;  
double multiply = calculator.multiply(x, y);
```

=

```
Calculator calculator = new Calculator();  
double multiply = calculator.multiply(5, 7);
```

Metody i referencje

- Argumenty typów obiektowych przekazywane są przez wartość referencji.
Oznacza to, że argument metody będzie dodatkową referencją na przekazany obiekt

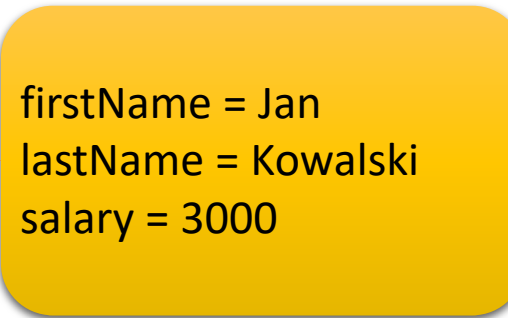
```
class Employee {  
    String firstName;  
    String lastName;  
    double salary;  
  
    Employee(String fn, String ln, double sal) {  
        firstName = fn;  
        lastName = ln;  
        salary = sal;  
    }  
}
```

```
class Company {  
  
    void increaseSalary(Employee emp) {  
        emp.salary = emp.salary + 0.1*emp.salary;  
    }  
}
```

Metody i referencje

```
Employee employee = new Employee("Jan", "Kowalski", 3000);  
Company company = new Company();  
company.increaseSalary(employee);
```

employee



Metody i referencje

```
Employee employee = new Employee("Jan", "Kowalski", 3000);  
Company company = new Company();  
company.increaseSalary(employee);
```

employee



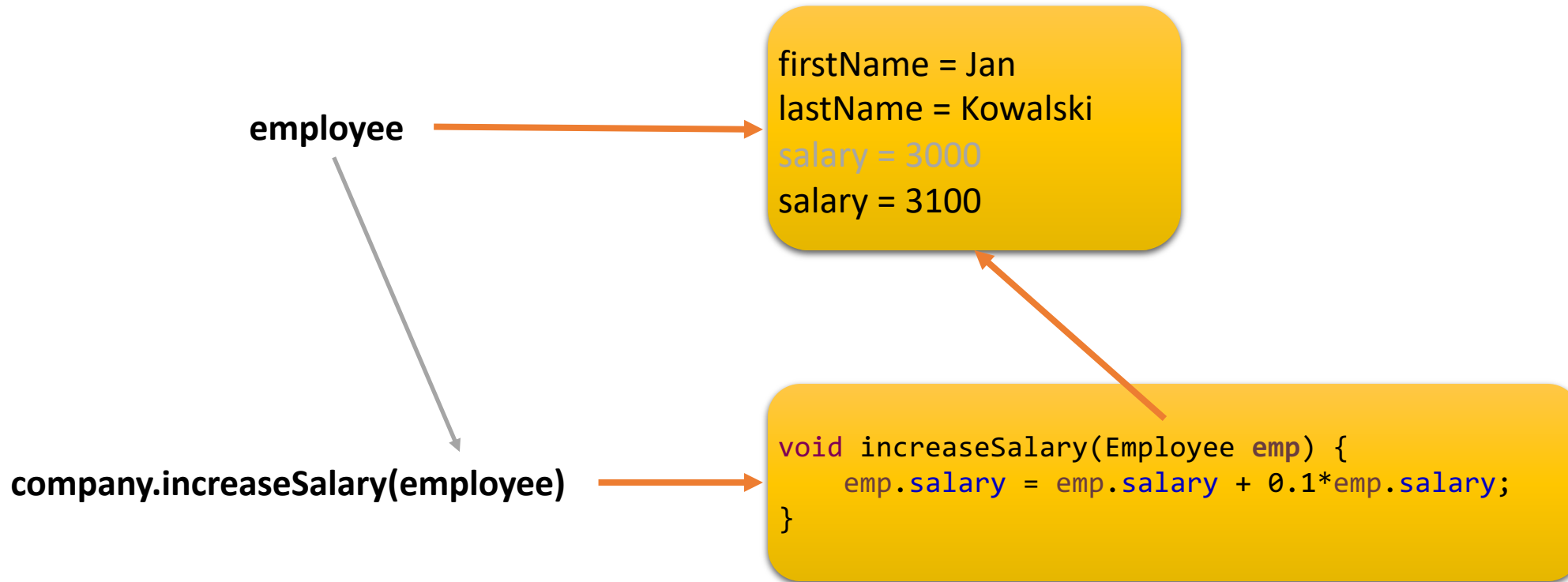
firstName = Jan
lastName = Kowalski
salary = 3000

company



Metody i referencje (demo)

```
Employee employee = new Employee("Jan", "Kowalski", 3000);  
Company company = new Company();  
company.increaseSalary(employee);
```



Przeciążanie metod (demo)

- Przeciążanie metod polega na definiowaniu kilku metod o identycznej nazwie, ale różnej liczbie lub różnych typach przyjmowanych parametrów
- Przeciążać można również konstruktory, definiując ich dowolną ilość

```
class Calculator {  
  
    double add(double a, double b) {  
        return a + b;  
    }  
  
    double add(double a, double b, double c) {  
        return a + b + c;  
    }  
}
```

Przeciążanie konstruktorów (demo)

```
class Person {  
    String firstName;  
    String lastName;  
    int age;  
    String city;  
  
    Person(String fn, String ln) {  
        firstName = fn;  
        lastName = ln;  
    }  
  
    Person(String fn, String ln, int a, String c) {  
        firstName = fn;  
        lastName = ln;  
        age = a;  
        city = c;  
    }  
}
```

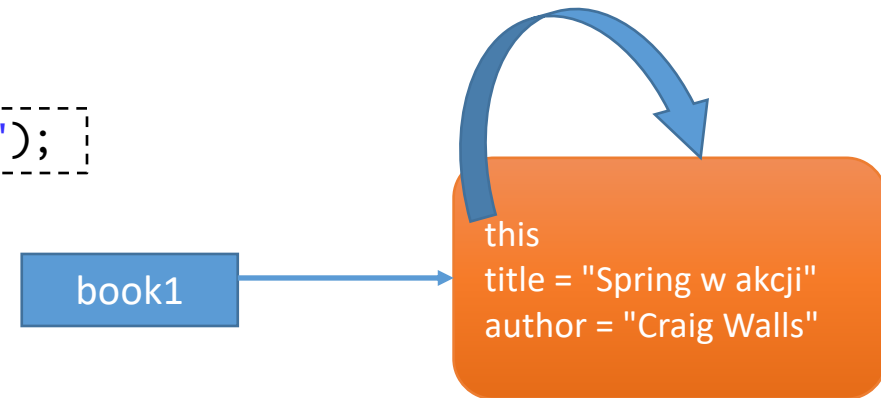
```
class PersonTest5 {  
    public static void main(String[] args) {  
        Person person1 = new Person("Jan", "Kowalski");  
        Person person2 =  
            new Person("Andrzej", "Zawada", 22, "Kraków");  
    }  
}
```

Słowo kluczowe **this**

- Słowo kluczowe **this** pozwala rozróżnić pola klasy od parametrów metod i konstruktorów o pokrywających się nazwach
- **this** jest w rzeczywistości referencją obiektu na samego siebie

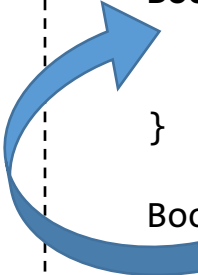
```
class Book {  
    String title;  
    String author;  
  
    Book(String title, String author) {  
        this.title = title;  
        this.author = author;  
    }  
}
```

```
Book book1 = new Book("Spring w akcji", "Craig Walls");
```



this w konstruktorach (demo)

- this pozwala także wywołać inną wersję konstruktora w ramach tej samej klasy
- Często pozwala to ograniczyć ilość powtarzanego kodu



```
class Book {  
    String title;  
    String author;  
    double price;  
  
    Book(String title, String author) {  
        this.title = title;  
        this.author = author;  
    }  
  
    Book(String title, String author, double price) {  
        this(title, author);  
        this.price = price;  
    }  
}
```