Programowanie obiektowe

Wykład 3: Klasy i obiekty

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.2
2 marca 2021

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

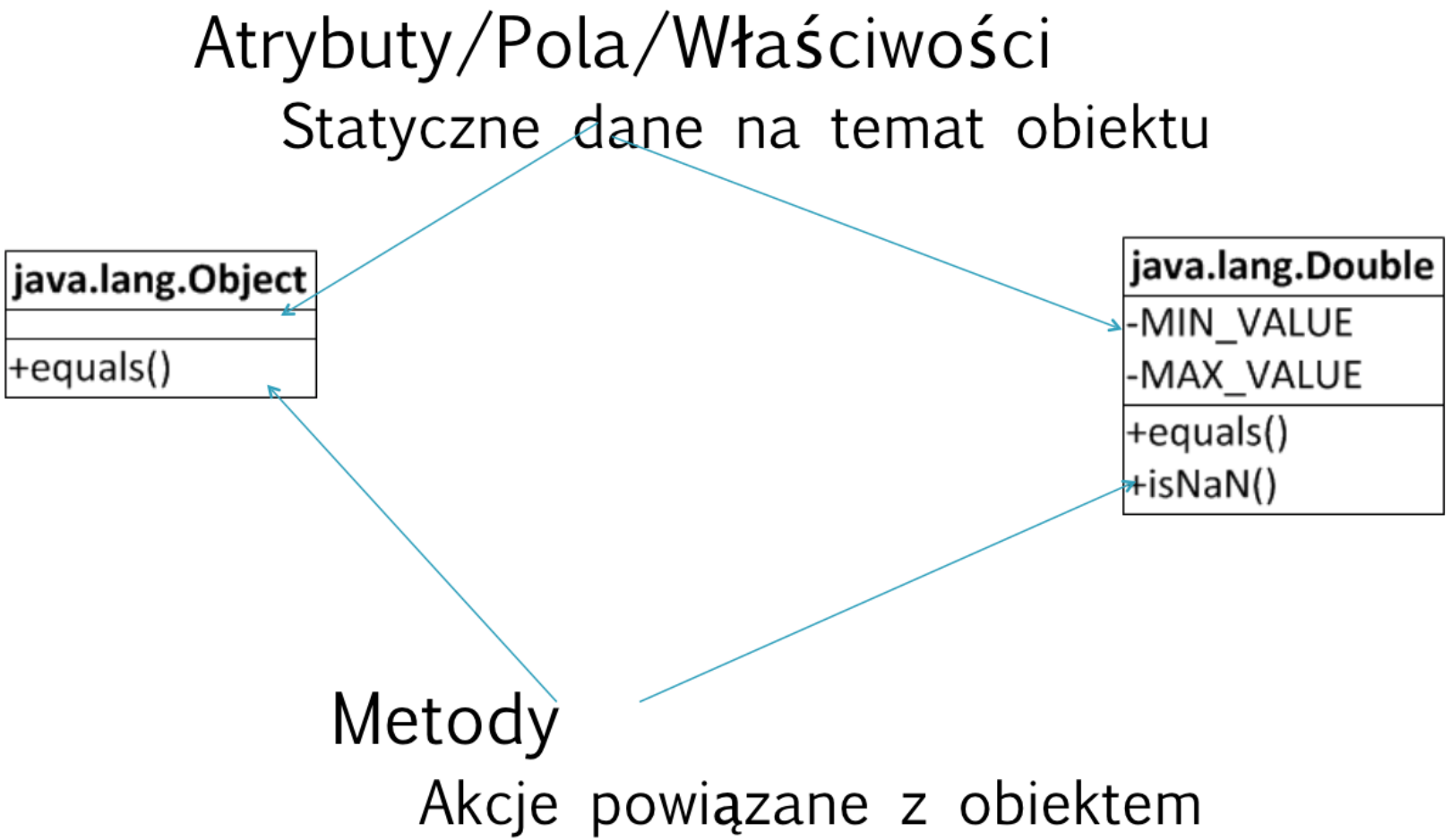
Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Klasa

- Klasa
  - jest szablonem, z którego tworzymy obiekty.
  - pozwala współdzielić swoim obiektom dane, które są ukryte dla obiektów z innych klas.
  - definiuje interfejs (zbiór metod) który umożliwia komunikację między obiektami.
- Klasy można budować poprzez rozszerzanie innych klas.
- Wszystkie klasy w Javie rozszerzają klasę `Object`

Metody i atrybuty



Rysunek 1: Klasa definiuje metody i atrybuty obiektów

Definiowanie własnej klasy

Klasa Person

```
1  import java.time.LocalDate;
2
3  public class Person {
4  //Atrybuty
5      String name;
6      LocalDate birthday;
7  //Metody
8  //Konstruktor
9      public Person(String name, LocalDate birthday) {
10         this.name = name;
11         this.birthday = birthday;
12     }
13
14     @Override //Nadpisanie - na nastepnym wykladzie
15     public String toString() {
16         return "Person{" +
17                 "name='" + name + '\'' +
18                 ", birthday=" + birthday +
19                 '}';
20     }
21 }
```

Słowo kluczowe `this`

- Tworząc implementację klasy możemy używać słowa kluczowego `this`.
- Jest ono referencją obiektu do samego siebie.
- Przy tworzeniu konstruktorów pozwala na rozróżnienie argumentów konstruktora i atrybutów obiektu.

Konstruktor

```
1    public Person(String name, LocalDate birthday) {
2           this.name = name;
3           this.birthday = birthday;
4       }
```

- Służy jako metoda do wywołania innego konstruktora.

Konstruktor bezparametrowy

```
1    public Person() {
2           this("Joe Doe",null);
3       }
```

Obiekt

Obiekt ma stan, zachowanie i tożsamość

Grady Booch

stan	wartości atrybutów obiektu
zachowanie	możliwe działanie (metody)
tożsamość	identyfikacja obiektu

Tworzenie obiektu

- Obiekt *musi* być stworzony.

Tworzenie obiektu

```
1  String s; //referencja
2  String s1 = new String("Stworzony tekst"); //obiekt
3  String s2 = "String jest specjalnym przypadkiem";
       //Inicjacja bez operatora new.
```

- Utworzony obiekt jest *instancją* Klasy.
- Obiekt tworzymy przy pomocy specjalnej metody *konstruktora*.

Życie obiektu

- Obiekty nie muszą być usuwane przez programistę.
- Java posiada *garbage collector*, który automatycznie usuwa obiekty bez referencji.
- Działanie mechanizmów czyszczących jest powiązane z ilością dostępnej pamięci. Uruchamiając aplikację decydujemy jaka jest maksymalna ilość pamięci, którą może ona wykorzystać.
  - `java -jar -Xmx128m -Xms64m MojaAplikacja.jar`

Zachowanie

- Zachowanie to zbiór metod (interfejs), które można wywołać dla obiektu lub klasy.
- Metody są wspólne dla wszystkich obiektów w klasie, ale ich działanie może zależeć od stanu obiektu.
  - Metody oznaczone jako `static` nie odwołują się do stanu obiektu.
  - Metody te reprezentują zachowanie wspólne dla wszystkich obiektów klasy lub zależne tylko od parametrów zewnętrznych
    - `Integer.signum(-5)`
  - Pozostałe metody mogą odwoływać się do stanu obiektu i muszą być wywoływane z instancji klasy.
    - `mojWujek.toString()`
- Specjalne metody, konstruktory, służą do tworzenia obiektów.

Stan

- Stan jest określony poprzez wartości atrybutów

Stan obiektu

```
1  String s1 = new String("Unique text");
2  //Stan obiektu to "Unique text"
```

- Dwa różne obiekty mogą mieć ten sam stan

Obiekty o tym samym stanie

```
1  Person mojWujek
2    = new Person("Zbyszek", LocalDate.of(1969, 8, 1));
3  Person sasiad
4    = new Person("Zbyszek", LocalDate.of(1969, 8, 1));
```

Tożsamość

- Tożsamość pozwala na identyfikację obiektu.
- Pozwala jednoznacznie rozróżniać obiekty.
- Istotnym jest, aby rozróżniać porównywanie tożsamości obiektów i ich stanów.

Narzędzia porównawcze

- Operator ==
  - Porównuje tożsamość obiektów.
  - Jest nim umiejscowienie obiektów w pamięci.
- Metoda `equals(Object o)` klasy `Object`
  - Porównuje stan obiektów
- Metoda `hashCode()` klasy `Object`
  - Wylicza jednowartościowy identyfikator stanu obiektu.
  - Wyliczana wartość nie musi być unikalna.

Dostosowanie narzędzi porównawczych

- Operator ==
  - W Javie nie można zmienić sposobu działania operatora.
- Metody `equals(Object o)` i `hashCode()`
  - Można zdefiniować sposób działania metody dla danej klasy.
  - Domyślnie działa wersja zaimplementowana dla klasy `Object`.

Domyślne działanie metod porównawczych

- Metoda `equals(Object o)`
  - Sprawdza, czy referencje odnoszą się do tego samego obszaru pamięci.
- Metoda `hashCode()`
  - Jest obliczany na podstawie adresu obiektu w pamięci.

Porównanie dla klasy `String`

- Niektóre klasy mają od razu zaimplementowane specyficzne działanie metod porównawczych
- Przykładem jest klasa `String`
  - Metoda `equals(Object o)`
    - Sprawdza czy teksty maję ten sam zestaw znaków umieszczonych w tej same kolejności
  - Metoda `hashCode()`
    - Zwraca kod dla części lub wszystkich znaków w tekście.
    - Dwa obiekty typu `String` mające tę samą kolejność znaków mają też tę samą wartość funkcji haszującej

Działanie porównania dla klasy `String`

Porównywanie

```
1  String body1 = new String("Joe Doe");
2  String body2 = new String("Joe Doe");
3  String body3 = body1;
4
5  body1==body3;
6  body1==body2;
7  body1.equals(body2));
```

Wynik

`true, false, true`

Wyliczanie kodu haszującego

```
1  body1.hashCode(); //227929242
2  body2.hashCode(); //227929242
```

Tworzenie własnych metod porównawczych

Metoda equals(Object o) klasy Person

```
1    @Override
2       public boolean equals(Object o) {
3           if (this == o) return true;
4           if (o == null || getClass() != o.getClass()) return false;
5
6           Person person = (Person) o;
7
8           if (name != null ? !name.equals(person.name) : person.name
                != null) return false;
9           return birthday != null ? birthday.equals(person.birthday)
                : person.birthday == null;
10      }
```

Metoda hashCode() klasy Person

```
1       @Override
2       public int hashCode() {
3           int result = name != null ? name.hashCode() : 0;
4           result = 31 * result + (birthday != null ?
                birthday.hashCode() : 0);
5           return result;
6       }
```

Wymagania co do metody equal

- Metoda `equals` musi być zwrotna `x.equals(x)==true`
- Metoda `equals` musi być symetryczna `x.equals(y)==y.equals(x)`
- Metoda `equals` musi być przechodnia. Jeżeli `x.equals(y)==y.equals(z)` to `x.equals(z)`
- Zatem metoda jest relacją równoważności.
- Dodatkowo `x.equals(null)==false`, a kolejne wywołania metody są nieimiennicze, o ile nie zmieniły się obiekty, których dotyczy.

Bibliografia