Programowanie komputerów I

Programowanie obiektowe II

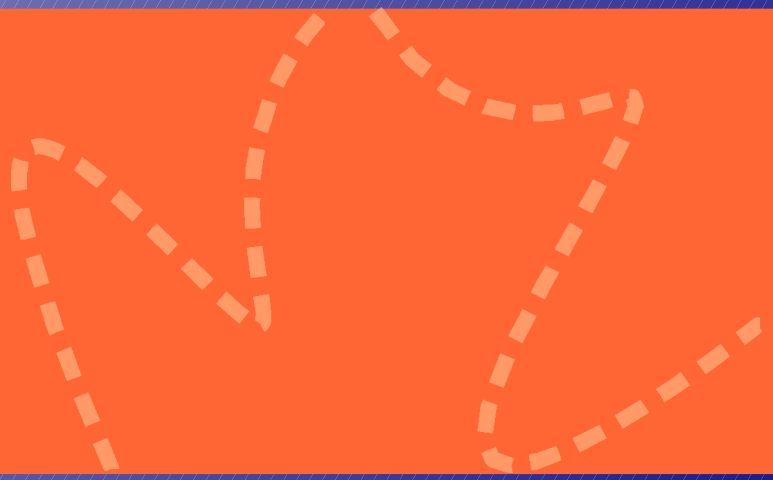
Powrót

Willy Picard
Katedra Technologii Informacyjnych
Akademia Ekonomiczna w Poznaniu
<*picard@kti.ae.poznan.pl*>

Agenda

- Cel(e) wykładu
- Od Włoch do Indonezji
- Interfejsy, Klasy i Obiekty
- Atrybuty i metody
- Kapsułkowanie
- Dziedziczenie i polimorfizm
- Podsumowanie

Cel(e) wykładu



Przegląd wykładu

Podstawowe pojęcia

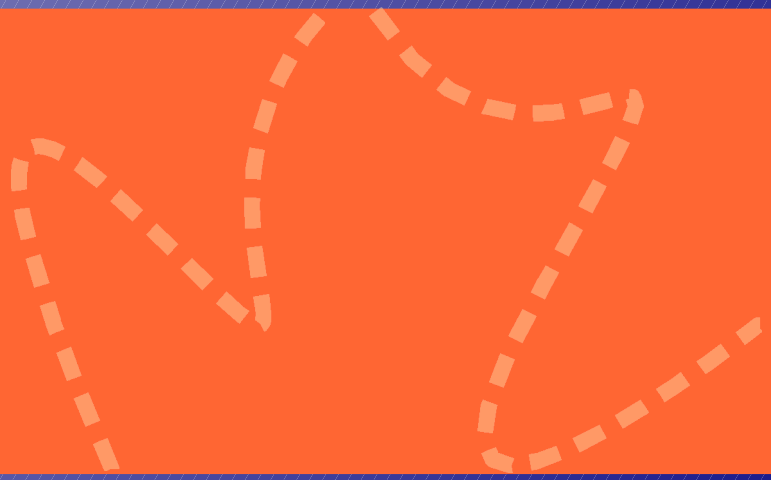
- 1: Wprowadzenie
- 2: Podstawowe struktury danych & instrukcje
- 3: Programowanie obiektowe I
- 4: Programowanie obiektowe II
- 5: Programowanie obiektowe III
- 6: Zaawansowane struktury danych
- 7: Wątki & Wyjątki

CELE WYKŁADU

Cel na dziś

Wprowadzić
programowanie obiektowe
(*object-oriented programming*)

Odświeżenie i przekąski



Definicja interfejsu

Interfejs jest definicją zachowania jako zbioru funkcji

ODŚWIŻENIE

Definicja klasy

Klasa jest definicją implementacji zachowania

ODŚWIŻENIE

Definicja obiektu

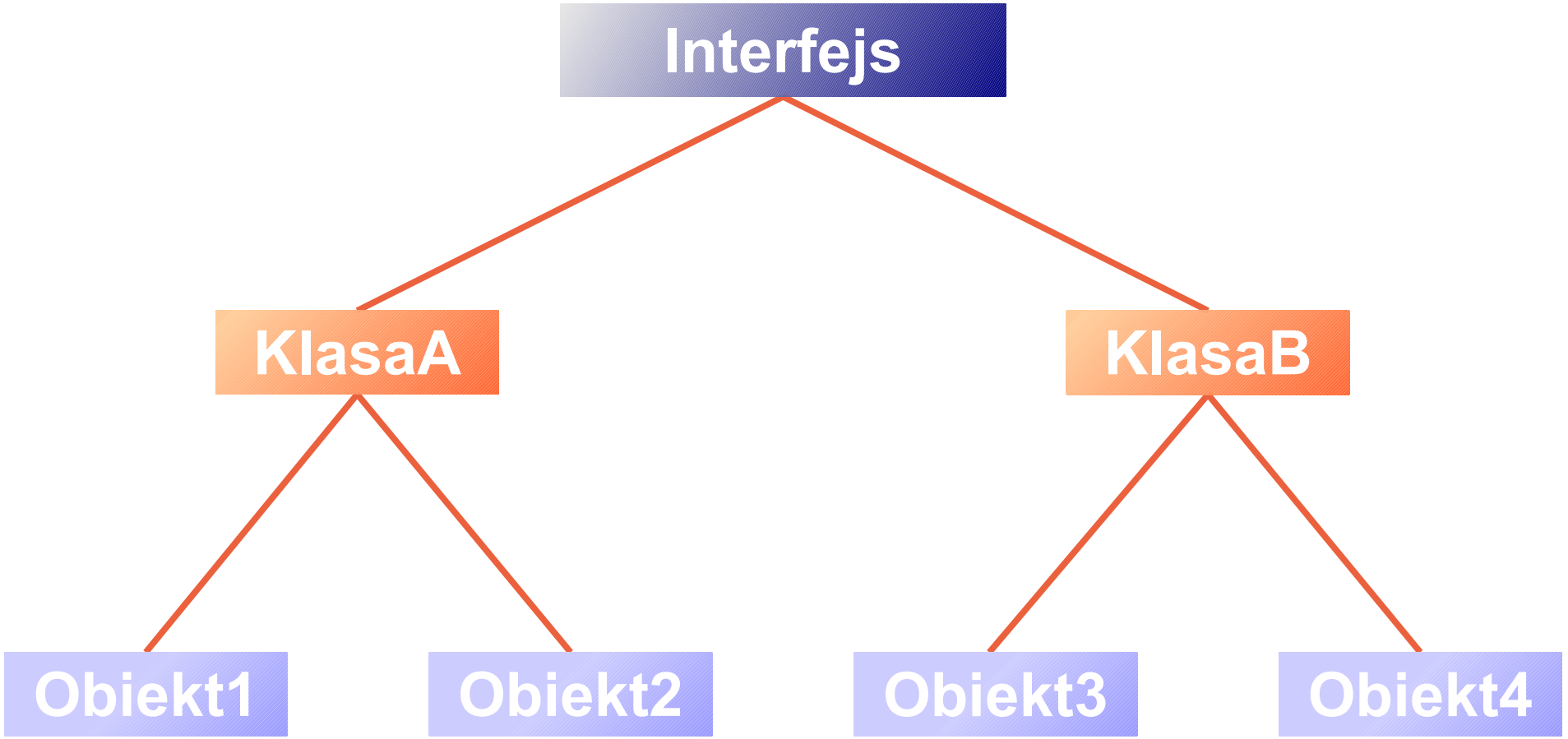
Obiekt jest
instancją klasy

ODŚWIŻENIE

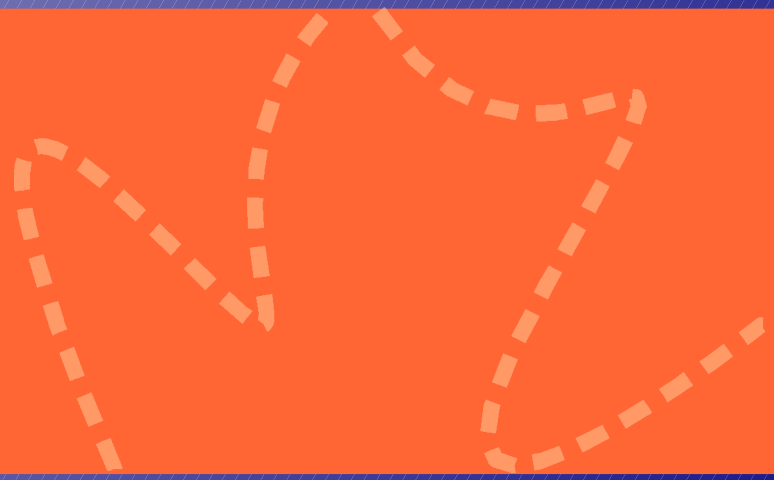
Klasy i interfejsy

Klasa, która implementuje (*implements*) interfejs musi zdefiniować wszystkie metody zadeklarowane w interfejsie

Podsumowanie



Atrybuty i metody



Definicja artybutu

Atrybut jest zmienną,
w której jest przechowana
część stanu obiektu

Artybuty w Javie

- Składnia

```
class <nazwaKlasy>{
    ...
    <typ> <nazwaAtrybutu>;
    ...
}
```

- Przykład

```
class CitroenC3 {
    ...
    Pedał _pedałGazu;
    Silnik _silnik;
    Rejestracja _rejestracja;
    ...
}
```

Definicja metody

Metoda jest implementacją części zachowania klasy obiektów

Komunikaty i metody

- Obiekty komunikują się
  - Wymiana komunikatów
  - Podejście „zrób coś”
- Komunikaty
  - Synchroniczne
  - Asynchroniczne
- Wysyłanie komunikatów

=

Odwołanie się do metody

Metody w Javie

- Składnia

```
interface <nazwaInterfejsu>{
    ...
    <typ_wyniku> <nazwaMetody>(<parametry>);
    ...
}
class <nazwaKlasy>{
    ...
    <typ_wyniku> <nazwaMetody>(<parametry>){
        ...
    }
    ...
}
```

- Słowo kluczowe `void`

```
void wyświetlCzas();
```

Metody w Javie

- Przykład

```
class CitroenC3 {
    ...
    int przyspiesz(int moc){
       _pedałGazu.wciśnij(moc);
       return _silnik.zwróćObecnąPrzędkość();
    }
    ...
}
```

Metody abstrakcyjne w Javie

- Niepełna definicja klasy
- Przykład

```
abstract class CitroenC3 {
    ...
    int przyspiesz(int moc){
       _pedałGazu.wciśnij(moc);
       return
         _silnik.zwróćObecnąPrzędkość();
    }
    ...
    abstract hamuj();
}
```

Definicja konstruktora

Konstruktor jest szczególną „metodą”, która tworzy instancje danej klasy

Konstruktory w Javie

- Składnia

```
class <nazwaKlasy>{
    ...
    <nazwaKlasy>(<parametry>){
        ...
    }
    ...
}
```

Konstruktory w Javie

- Przykład

```
class CitroenC3 {
    ...
    CitroenC3(Pedał gaz,
              Silnik silnik,
              Rejestracja rejestracja){
      _pedałGazu = gaz;
      _silnik = silnik;
      _rejestracja = rejestracja;
      ustawPrędkość(0);
    }
    ...
}
```

Tworzenie obiektów w Javie

- Przykład

```
...
samochód = new CitroenC3( mójPedał,
                          slinikHDI,
                          mojaRejestracja);
...
```

- Domyślnie
  - Pusty konstruktor

Definicja destruktora

Destruktor jest szczególną „metodą”, która usuwa instancje danej klasy z pamięci komputera

Destruktory w C++

- Składnia

```
class <nazwaKlasy>{
    ...
    ~<nazwaKlasy>(<parametry>){
        ...
    }
    ...
}
```

Destruktory w Javie

- Brak destruktorów w Javie
- Odśmiecacz (ang. *Garbage collector*)
  - Usuwa nieużywane obiekty
- Zarządzanie pamięcią
  - Odpowiedzialność JVM
- `null`
  - np. `_silnik = null;`

Pola obiektu *vs* Pola klasy

- Pola
  - atrybuty
  - metody
- Pola obiektu (ang. *Instance Field*)
  - Powiązane do danego obiektu
- Pola klasy (ang. *Class Fields)*
  - Wspólne dla wszystkich instancji danej klasy

Pola w Javie

- Domyślnie
  - Pola obiektu
- Pola klasy
  - Słowo kluczowe `static`
- Przykład
  - `static Data _dzisiejszaData;`
  - `static Data zwróćDzisiejszaDatę();`

Przykład *static* w Javie

- Lampki choinkowe

```
class ILampka{
   void zakręć();
   void odkręć();
}
class Lampka{
   boolean _zakręcona;
   void zakręć(){...}
   void odkręć(){...}
   boolean jestZakręcona(){
      return _zakręcona;
   }
}
```

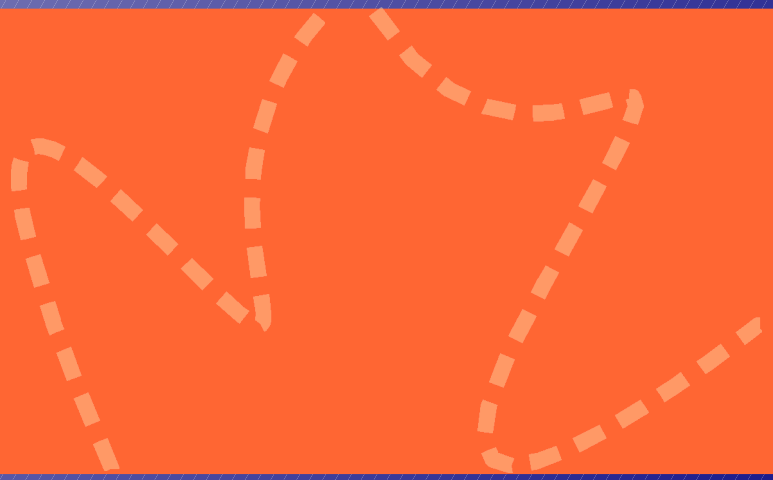
Przykład *static* w Javie

```
class Lampka{
    boolean _zakręcona;
    static int _licznikOdkręconych;
    void zakręć(){
        _licznikOdkręconych --;
        _zakręcona = true;
    }
    void odkręć(){
        _licznikOdkręconych ++;
        _zakręcona = false;
    }
}
```

Przykład *static* w Javie

```
class Lampka{
    static int _licznikOdkręconych;
    ...
    static boolean świeciSię(){
        return _licznikOdkręconych == 0;
    }
}
```

Podsumowanie



Podsumowanie

- Reguła 1
  - Używaj interfejsy
- Reguła 2
  - Używaj interfejsy
- Reguła 3
  - Używaj interfejsy

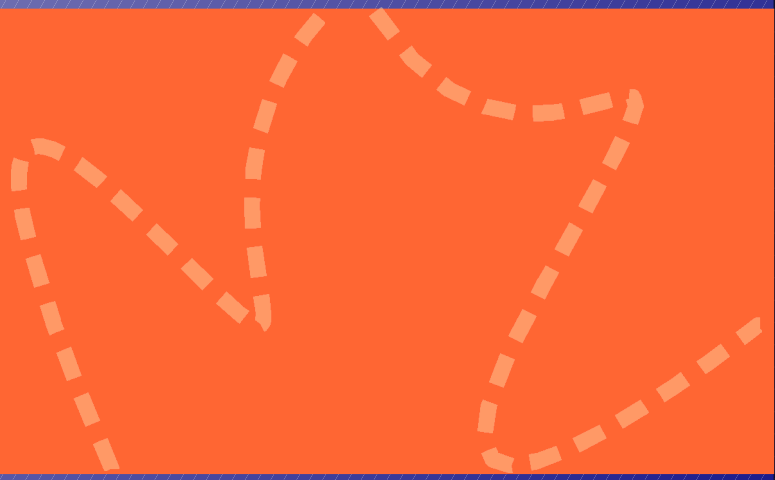
Przykład

```
package pl.poznan.ae.compProg;
import java.util.*;

public class Sorter {
  private List _words;

  public void sort(String[] words){
    _words = Arrays.asList(words);
    Collections.sort(_words);
  }
  public String getSortedWords(){
    String sortedString = "";
    for (int i = 0; i< _words.size(); i++){
      sortedString += _words.get(i);
    }
    return sortedString;
  }
 public static void main(String[] args){
    Sorter sorter = new Sorter();
    sorter.sort(args);
    System.out.println(sorter.getSortedWords());
 }
}
```

Do zobaczenia za tydzień



Programowanie obiektowe III

Zemsta