



*Programowanie komputerów I*

# *Programowanie obiektowe III*

*Zemsta*

**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
- ▶ Odświeżenie i przekąski
- ▶ Dziedziczenie
- ▶ Polimorfizm
- ▶ Kapsułkowanie
- ▶ Podsumowanie

# Cel(e) wykładu



# Przegląd wykładu

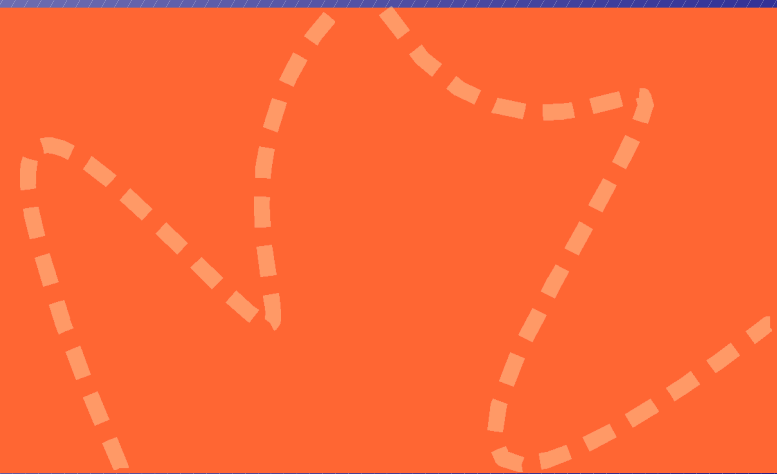
## Podstawowe pojęcia

- ▶ 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

# Cel na dziś

Wprowadzić  
programowanie obiektowe  
(*object-oriented  
programming*)

# Odświeżenie i przekąski



# Przykład interfejsu

```
class IZwierze{  
    int zwróćWagę();  
    String zwróćNazwę();  
    void wydajGłos();  
    void jedz();  
    void jedz(int ilość);  
}
```

# Przykład klasy

```
class Kot implements IZwierze{
    int _waga;
    String _nazwa;

    Kot(int waga, String nazwa){
        _waga = waga;
        _nazwa = nazwa;
    }
    ...
}

Kot mójKot = new Kot(1200, "Felix");
```



# Przykład klasy

```
class Kot implements IZwierze{  
    int _waga;  
    String _nazwa;  
    int zwróćWagę() {  
        return _waga;  
    }  
    String zwróćNazwę() {  
        return _nazwa;  
    }  
    ...  
}
```

# Przykład klasy

```
class Kot implements IZwierze{
    ...
    void wydajGłos(){
        System.out.println("Miau");
    }
    void jedz(){
        _waga += 200;
    }
    void jedz(int ilość){
        _waga += ilość;
    }
}

System.out.println(mójKot.zwróćNazwę() + " mówił:");
mójKot.wydajGłos();
```

# String w Javie

- ▶ Klasa *String*

- ▶ Łańcuch znaków

- ▶ Przykład

```
String nazwaKota = new String("Felix");  
String nazwaKota = "Felix";
```

- ▶ Przydatne “sztuczki”

- ▶ Łączenie:            “Nazwa kota to ” + nazwaKota;
  - ▶ Porównanie:       nazwaKota.equals("Felix");

# Metoda main()

- ▶ 

```
public class Kot{  
    public static void main(String[] args){...}  
}
```
- ▶ **Uruchomiona przez** `java Kot`
- ▶ **Argumenty**
  - ▶ `String[] args`
  - ▶ **Liczba argumentów:** `args.length`
- ▶ **Przykład**
  - ▶ `java Kot "Felix" "1200"`
  - ▶ `args[0] = "Felix", args[1] = "1200"`

# Dziedziczenie



# Klasy i interfejsy

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

# Klasy i interfejsy w Javie

## ► Składnia

```
class <nazwaKlasy> implements <nazwaInterfejsu>{  
    ...  
}
```

## ► Przykład

```
class Kot implements IZwierze{  
    ...  
}
```

# Nadklasy i podklasy

Podklasa (klasa pochodna),  
która rozszerza (*extends*)  
klasę i dziedziczy (*inherits*)  
atrybuty i metody tej  
nadklasy i jej przodków



# Nadklasy i podklasy w Javie

## ► Składnia

```
class <nazwaPodklasa> extends <nazwaNadklasy>{  
    ...  
}
```

## ► Przykład

```
class KotPerski extends Kot{  
    ...  
}
```

# Nadpisanie metody

- ▶ Zmiana definicji metody w podklasie
  - ▶ Specjalizacja klasy
  - ▶ Ponowne wykorzystanie kodu
- ▶ Identyczne podpisy metody
- ▶ Konstruktory

# Nadpisanie metody w Javie

```
class Kot {  
    void jedz() {  
        _waga += 200;  
    }  
}  
  
class KotPerski extends Kot {  
    boolean _czyŚpi = false;  
    void jedz() {  
        super.jedz();  
        zróbSiestę();  
    }  
    void zróbSiestę() { _czyŚpi = true; }  
}
```

# “this.” w Javie

```
class KotPerski extends Kot {  
    int ilość;  
  
    void jedz(int ilość) {  
        this.ilość += ilość;  
        super.jedz(ilość);  
        zróbSiestę();  
    }  
  
    void jedz() {  
        jedz(200);  
    }  
}
```

# Nadpisanie konstruktorów w Javie

```
class Kot {  
    Kot(String nazwa, int waga) {  
        _nazwa = nazwa;  
        _waga = waga;  
    }  
}
```

```
class KotPerski {  
    KotPerski(String nazwa, int waga) {  
        super(nazwa, waga);  
        _ilość = 0;  
        _czyŚpi = false;  
    }  
}
```

# “this()” w Javie

```
class KotPerski {  
  
    KotPerski(String nazwa,  
                int waga,  
                int ilość,  
                boolean czySpi){  
        super(nazwa, waga);  
        _ilość = ilość;  
        _czySpi = czySpi;  
    }  
  
    KotPerski(String nazwa, int waga){  
        this(nazwa, waga, 0, false);  
    }  
}
```

# Wielokrotne dziedziczenie

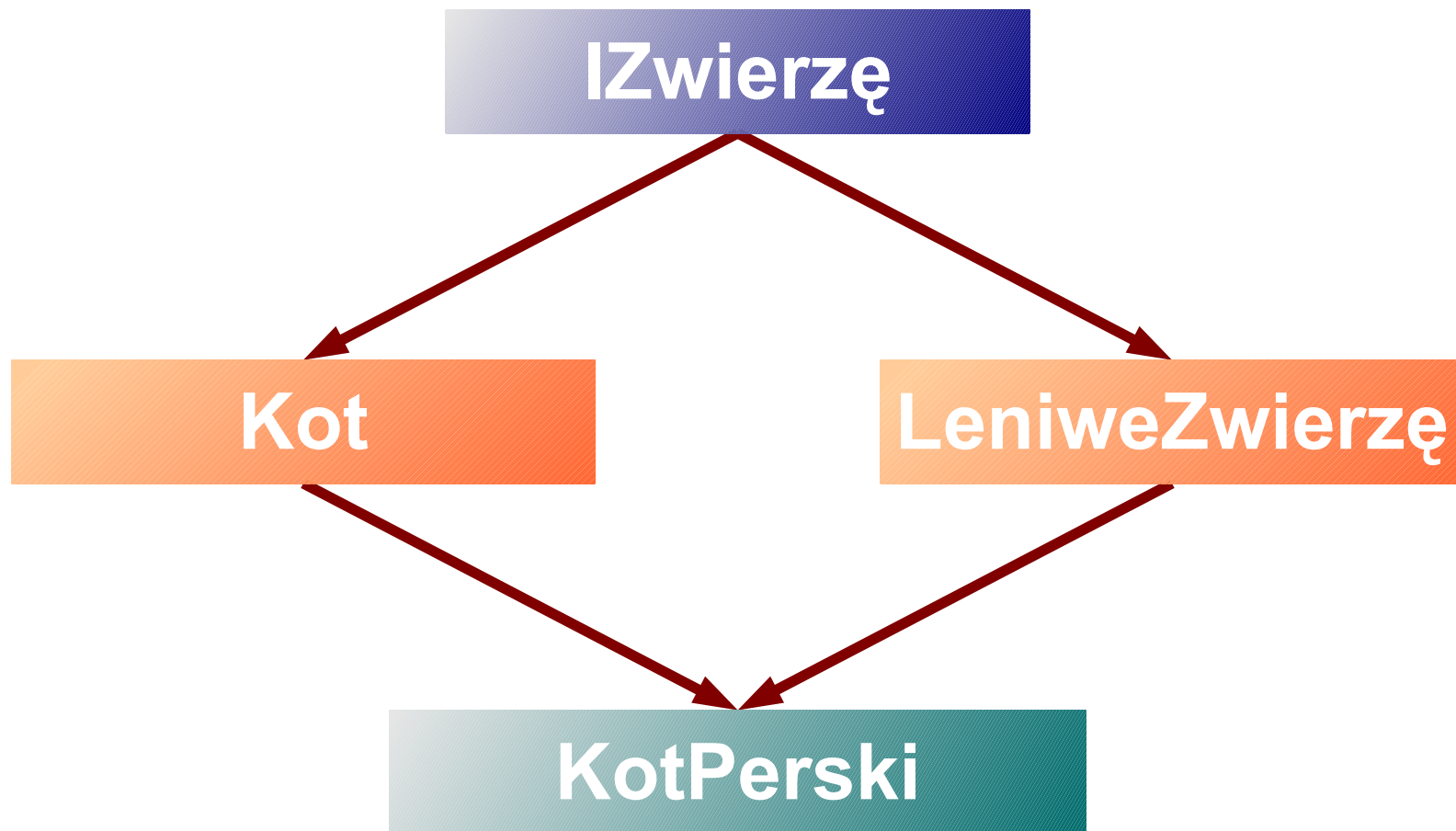
## ► Interfejsy

- Mogą rozszerzyć 0, 1, lub wiele interfejsów
- Brak implementacji → jednoznaczność

## ► Klasy

- Mogą implementować 0, 1, lub wiele interfejsów
- Brak implementacji → jednoznaczność
- Mogą rozszerzyć 0, 1, lub wiele klas
- Niedopuszczalne w Javie

# Problem “Rombu”





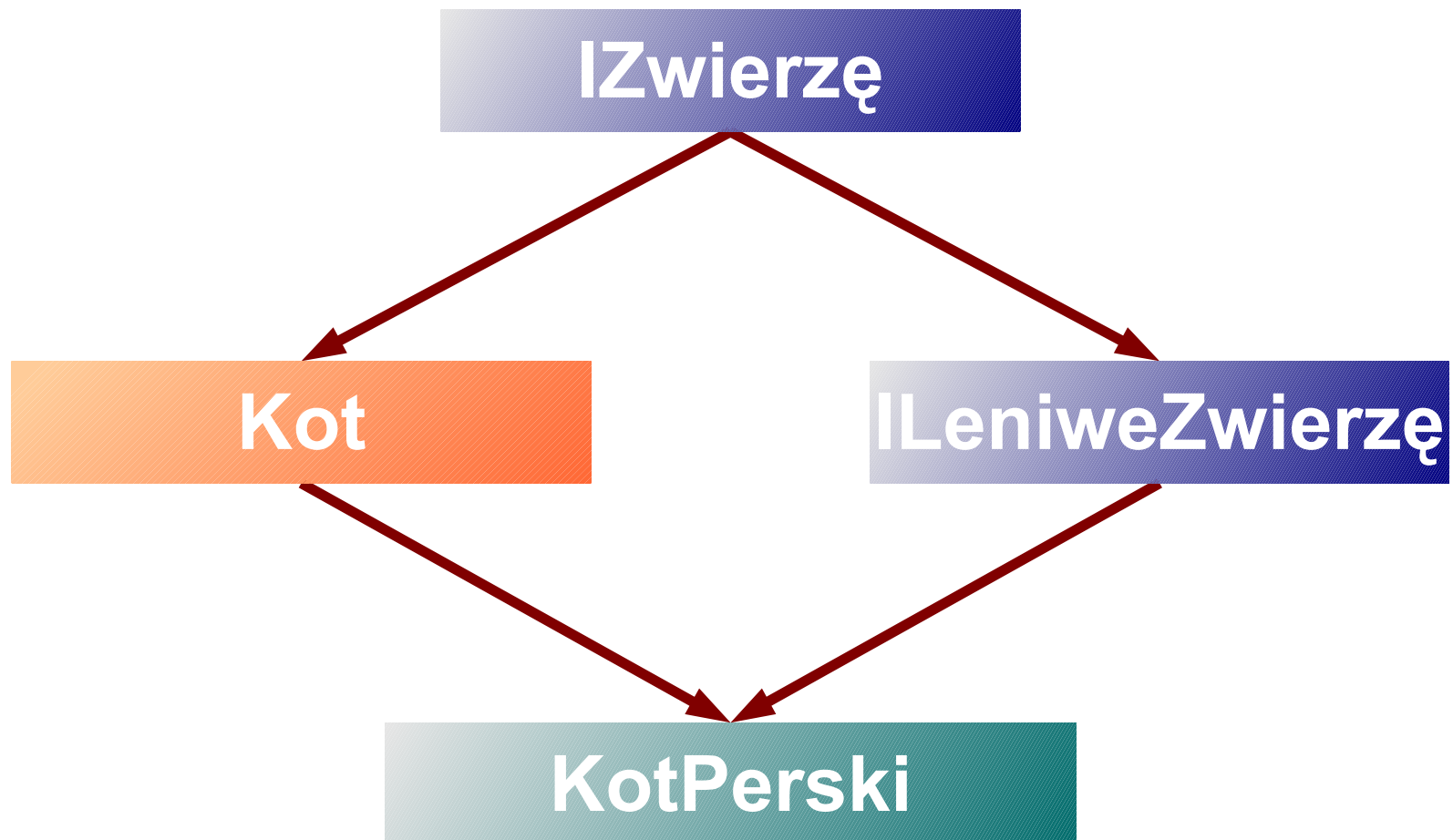
# Polimorfizm



# Definicja polimorfizmu

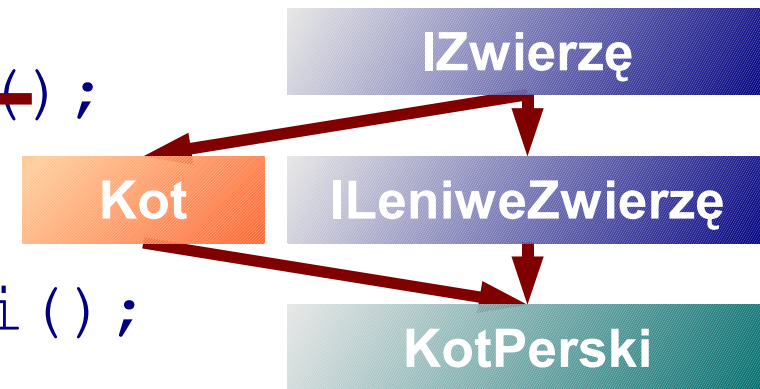
Polimorfizm jest zdolnością  
niektórych obiektów do  
przybrania różnych postaci

# Jeden obiekt, wiele postaci



# Jeden obiekt, wiele postaci w Javie

- ▶ ~~IZwierzę kot = new IZwierzę();~~
- ▶ IZwierzę kot = new Kot();
- ▶ IZwierzę kot = new KotPerski();
- ▶ ~~IZwierzę kot = new ILeniweZwierzę();~~
- ▶ Kot kot = new Kot();
- ▶ Kot kot = new KotPerski();
- ▶ ILeniweZwierzę kot = new KotPerski();
- ▶ KotPerski kot = new KotPerski();



# Definicja rzutowania

Rzutowanie (*casting*) jest  
konwersją danego obiektu  
do innej postaci

# Rzutowanie w Javie


- ▶ `Kot kot1 = new KotPerski();`
- ▶ `ILeniweZwierze kot2 = new KotPerski();`
- ▶ `kot1.zróbSiestę()` **jest niepoprawne**
- ▶ `kot2.zróbSiestę()` **jest poprawne**
- ▶ `ILeniweZwierze kot3 =  
    (ILeniweZwierze) kot1;`
- ▶ `kot3.zróbSiestę()` **jest poprawne**

# Obsługa polimorficznych obiektów

- ▶ Przeciążanie metod
  - ▶ W jednej klasie, w jednym interfejsie
  - ▶ Wiele metod
  - ▶ Jedna nazwa metody
  - ▶ Różne parametry
- ▶ Nadpisanie metod
  - ▶ W wielu interfejsach, klasach i podklasach
  - ▶ Identyczny podpis metody
  - ▶ Różne implementacje

# Przeciążanie

- ▶ Wiele metod z różnymi argumentami
- ▶ Sprawdzenie w czasie kompilacji
- ▶ Przykład



```
class WłaścicielZwierząt {  
    void karm(Kot kot) {...}  
    void karm(KotPerski kot) {...}  
}  
Kot kot;  
KotPerski kotPerski;  
...  
właściciel.karm(kot);  
właściciel.karm(kotPerski);
```



# Nadpisanie

- ▶ Jedna metoda
  - ▶ W wielu interfejsach, klasach, podklasach
  - ▶ Różne implementacje
- ▶ Sprawdzenie w czasie wykonania
- ▶ Po angielsku
  - ▶ Overriding
  - ▶ Late binding
  - ▶ Dynamic binding
  - ▶ Run-time binding

# Nadpisanie w Javie

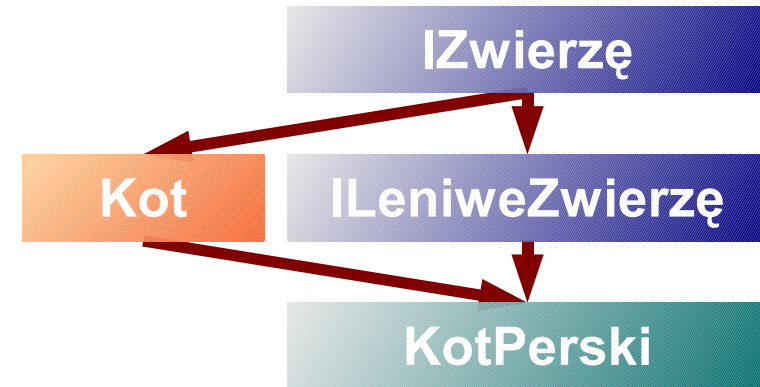
```
class Kot {  
    void jedz() {  
        _waga += 200;  
    }  
}  
  
class KotPerski extends Kot {  
    boolean _czySpi = false;  
    void jedz() {  
        super.jedz();  
        zróbSiestę();  
    }  
    void zróbSiestę() { _czySpi = true; }  
}
```

# Nadpisanie w Javie

```
IZwierzę kot = new Kot();  
IZwierzę kotPerski =  
    new KotPerski();
```

```
kot.jedz() {  
    _waga += 200;  
}
```

```
kotPerski.jedz() {  
    super.jedz();  
    zróbSiestę();  
}
```



# Kapsułkowanie



# Pakiety w Javie

- ▶ Zgrupowanie klas i interfejsów
- ▶ Nazwa pakietu
  - ▶ Pusta: domyślny pakiet
  - ▶ (`<identyfikator>.`)\* `<identyfikator>`
  - ▶ np. `java.lang`
- ▶ Deklaracja
  - ▶ `package <nazwaPakietu>;`
- ▶ Wykorzystanie
  - ▶ `import <nazwaPakietu>.*;`
  - ▶ `import <nazwaPakietu>.<nazwaKlasy>;`

# Ukrywanie detali implementacji

- ▶ Reguła 1
  - ▶ Żaden atrybut nie jest widoczny poza klasą, w której jest zdefiniowany
- ▶ Reguła 2
  - ▶ Metoda jest widoczna jeżeli jest używana przez inną klasę
- ▶ Reguła 3
  - ▶ Używaj interfejsy!!!

# Widoczność w Javie

- ▶ Zbiór słów kluczowych
  - ▶ `private`
  - ▶ `protected`
  - ▶ `public`
- ▶ Składnia
  - ▶ `<modifykator> <pole>`
  - ▶ `np. private int _waga`

# Reguły widoczności w Javie

	Klasa	Podklasa	Pakiet	Inne
<code>private</code>	X			
<code>protected</code>	X	X	X	
<code>public</code>	X	X	X	X
<code>pusty</code>	X		X	



# Podsumowanie



# Język C vs. OOPs

- ▶ Powiązanie między
    - ▶ Procedurami/funkcjami
    - ▶ Strukturami danych
  - ▶ Ponowne wykorzystanie kodu
  - ▶ Kod rozproszony
  - ▶ Deklaracja vs. definicja
- klasy*
- dziedziczenie*
- klasy,  
dziedziczenie*
- interfejsy,  
kapsułkowanie*

# Złote reguły

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

# Złote reguły II

- ▶ Reguła 4
  - ▶ Ukrywaj wszystko, co można ukryć
- ▶ Reguła 5
  - ▶ Oddziel swoje klasy
- ▶ Reguła 6
  - ▶ Daj obiektom odpowiedzialność

# 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**

