



Uniwersytet Ekonomiczny  
we Wrocławiu



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



# Programowanie w języku Java

w ramach projektu

„Trzecia Misja Uniwersytetu Ekonomicznego we Wrocławiu dla dzieci i młodzieży”

Część IV

Rok szkolny 2021/22

Prowadzący: dr inż. Piotr Tutak

# Programowanie obiektowe

Programowanie obiektowe  
OOP - object-oriented programming

Obiekt = stan + zachowanie

- Stan = zmienne = pola / właściwości
- Zachowanie = funkcje = metody

# Programowanie obiektowe

## Reprezentacja obiektu - problem

### Obiekt - Film

film1\_nazwa = „Avatar”  
film1\_rok = 2009

film2\_nazwa = „Titanic”  
film2\_rok = 1997

### Funkcja

informacje(nazwa, rok) {...}

# Programowanie obiektowe

## Reprezentacja obiektu - OOP

### Obiekt - Film

```
class Film {  
    String nazwa;  
    int rok;  
    void informacje(){...}  
}
```

# Programowanie obiektowe

## Reprezentacja obiektu - OOP

### Obiekt - Film

```
class Film {  
    String nazwa;  
    int rok;  
    void informacje(){...}  
}
```

Operator new

Obiekt avatar

```
Film avatar = new Film();  
avatar.nazwa = „Avatar”;  
avatar.rok = 2009;
```

Obiekt avatar to instancja klasy Film

# Programowanie obiektowe

## Reprezentacja obiektu - OOP

### Obiekt - Film

```
class Film {  
    String nazwa;  
    int rok;  
    void informacje(){...}  
}
```

```
Film avatar = new Film();  
avatar.nazwa = „Avatar”;  
avatar.rok = 2009;
```

```
Film titanic = new Film();  
titanic.nazwa = „Titanic”  
titanic.rok = 1997;  
titanic.informacje();
```



# Programowanie obiektowe

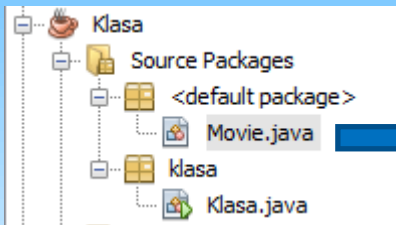
## Programowanie obiektowe - pojęcia

- Klasa - szablon obiektu, wzór, „przepis”
- Obiekt - instancja klasy, „wystąpienie” klasy

Klasa to np. meble

Obiekt to np. krzesło

# Programowanie obiektowe - przykład

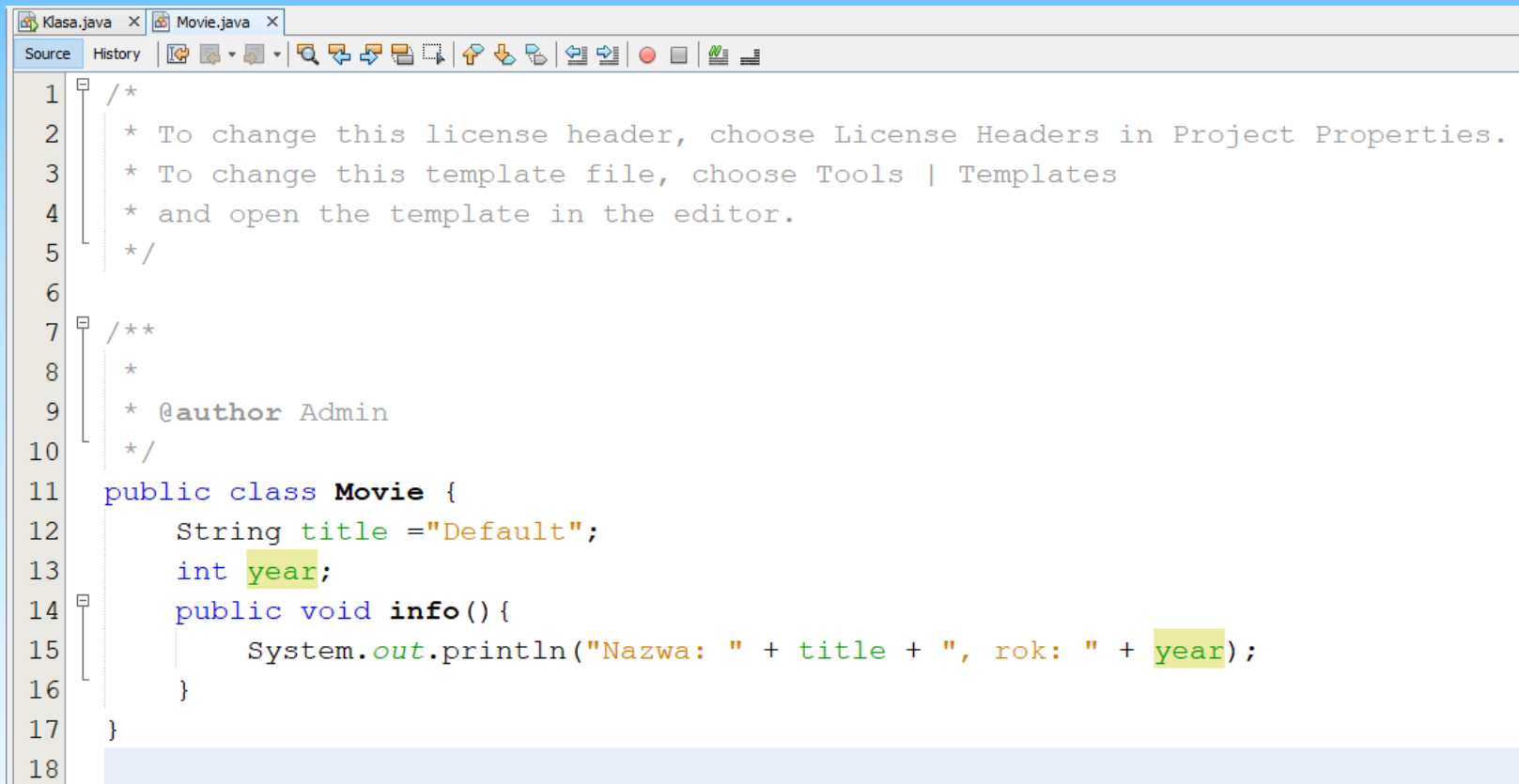


```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7  /**
8   *
9   * @author Admin
10  */
11  public class Movie {
12
13  }
14
```

```
public class Movie {
    String title = "Default";
    int year;
    public void info() {
        System.out.println("Nazwa: " + title + ", rok: |");
    }
}
```



# Programowanie obiektowe - przykład



```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7  /**
8   *
9   * @author Admin
10  */
11  public class Movie {
12      String title = "Default";
13      int year;
14      public void info() {
15          System.out.println("Nazwa: " + title + ", rok: " + year);
16      }
17  }
18
```

# Programowanie obiektowe - przykład

```
public class Main {  
  
    public static void main(String[] args) {  
        Movie avatar = new Movie();  
        avatar.title = "Avatar";  
        avatar.year = 2009;  
        avatar.info();  
    }  
}
```

# Programowanie obiektowe - przykład

```
public class Main {  
  
    public static void main(String[] args) {  
        Movie avatar = new Movie();  
        avatar.title = "Avatar";  
        avatar.year = 2009;  
  
        Movie titanic = new Movie();  
        titanic.title = "Titanic";  
        titanic.year = 1997;  
  
        titanic.info();  
        avatar.info();  
    }  
}
```

# Programowanie obiektowe - przykład

```
Movie movie1 = avatar;  
movie1.info();
```

# Programowanie obiektowe - przykład

```
Movie movie1 = avatar;  
movie1.info();  
movie1.title = "Random"  
avatar.info();
```

# Dziedziczenie

Dziedziczenie występuje zawsze pomiędzy 2 klasami

Animal  
Cat  
Main

```
public class Animal {  
    String name;  
    int age;  
  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

```
public class Cat extends Animal {  
    String color;  
  
    public void getVoice() {  
        System.out.println("Meow");  
    }  
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        Cat cat = new Cat();  
  
        cat.getVoice();  
        cat.eat();  
    }  
}
```

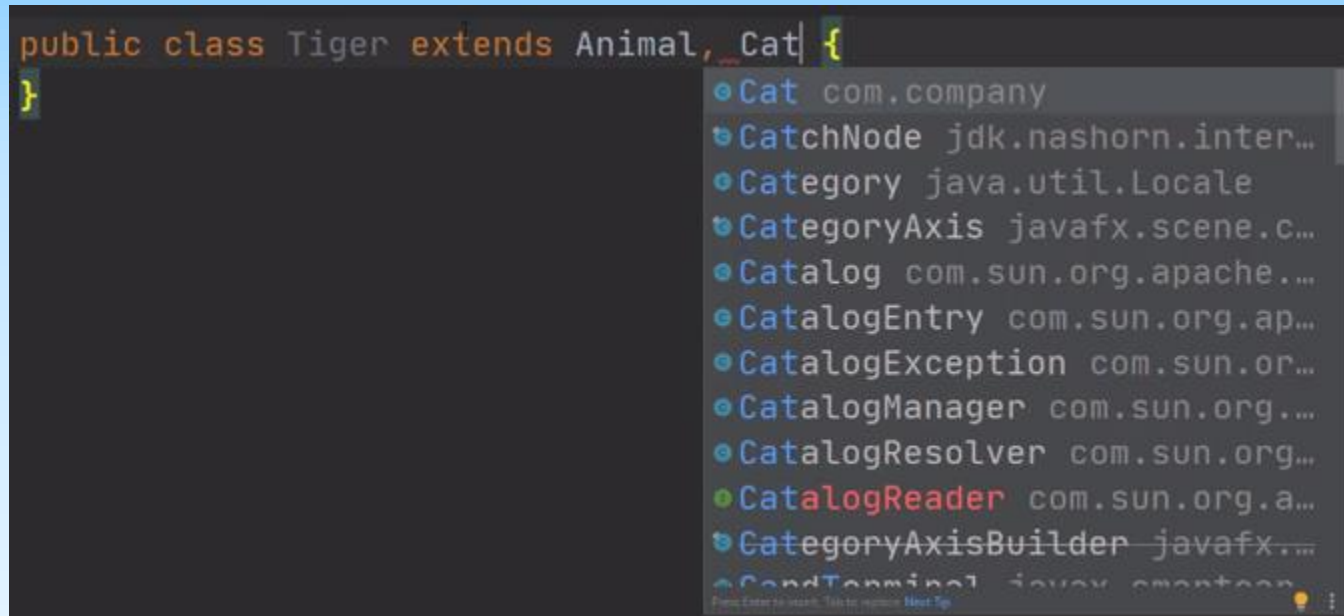
# Dziedziczenie – private

```
package com.company;  
  
public class Animal {  
    String name;  
    private int age;  
  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```



# Dziedziczenie – private

```
public class Tiger extends Animal, Cat {  
}
```



Jedna klasa może dziedziczyć tylko po jednej klasie, nie po dwóch!!!

# Dziedziczenie – private

```
final public class Cat extends Animal {  
    String color;  
  
    public void getVoice() {  
        System.out.println("Meow");  
    }  
}
```

Nie można dziedziczyć po dodaniu słówka final

# Zadanie 1

Utwórz klasę reprezentującą prostokąt, musi posiadać atrybuty długość i szerokość. Klasa powinna posiadać metody obliczające pole, obwód i długość przekątnej.

# Zadanie 1 – rozwiązanie

```
package rectangle;

import static java.lang.Math.*;

public class Prostokat {

    double a;
    double b;
    double pole;
    double obwod;
    double przekatna;

    public void pole() {
        pole=a*b;
        System.out.println("Pole prostokąta wynosi " + pole);
    }

    public void obwod() {
        obwod=a+a+b+b;
        System.out.println("Obwód prostokąta wynosi " + obwod);
    }

    public void przekatna() {
        przekatna=pow(a,2)+pow(b,2);
        przekatna=sqrt(przekatna);
        System.out.println("Przekatna prostokąta wynosi " + przekatna);
    }
}
```

```
package rectangle;

/**
 *
 * @author Admin
 */
public class Rectangle {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Prostokat pl = new Prostokat();
        pl.a=3;
        pl.b=3;
        pl.pole();
        pl.obwod();
        pl.przekatna();
    }
}
```

# Zadanie 2

Utwórz klasę Human reprezentującą człowieka, musi posiadać atrybuty takie jak wiek, waga, wzrost, imię i płeć.

Klasa powinna także zawierać metody:

- `getAge`
- `getWeight`
- `getHeight`
- `getName`
- `isMale`



Uniwersytet  
Ekonomiczny  
we Wrocławiu



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



# Konstruktor (1/4)

- Konstruktor to taki odpowiednik metody który jest wywoływany podczas tworzenia nowego obiektu czyli instancjonowania obiektu.
- Nazwa konstruktora odpowiada nazwie klasy
- Używamy ich do tworzenia obiektów które już na wstępie przyjmują jakieś wartości

Przykład:

```
Person(){  
    System.out.println ("Konstruktor klasy");  
}
```

→ jest to konstruktor domyślny, który nie przyjmuje argumentów

```
package com.company;  
  
public class Person {  
    Person() {  
        System.out.println("Konstruktor klasy");  
    }  
    String name;  
    int age;  
}
```

```
package com.company;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
    }  
}
```

# Konstruktor (2/4)

Konstruktor domyślny nie przyjmuje argumentów, jeżeli go usuniemy nie może stworzyć obiektu, który nie przyjmuje żadnych argumentów.

```
package com.company;

public class Person {
    Person() {
        System.out.println("Konstruktor klasy");
    }
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    String name;
    int age;
}
```

```
package com.company;

public class Main {

    public static void main(String[] args) {
        Person p1 = new Person("Adrian", 18);

        System.out.println(p1.name + ", " + p1.age);
    }
}
```



# Konstruktor (3/4)

```
package com.company;

public class Main {

    public static void main(String[] args) {
        Person p1 = new Person("Adrian",18);
        Person p2 = new Person("Bartek");

        System.out.println(p1.name + ", " + p1.age);
        System.out.println(p2.name + ", " + p2.age);
    }
}
```

```
package com.company;

public class Person {
    Person(String name) {
        this.name = name;
        age = -1;
    }
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    String name;
    int age;
}
```

Konstruktorów używamy do tworzenia obiektów które już na wstępie przyjmują jakieś wartości

# Konstruktor (4/4)

```
package konstruktor;
```

```
/**
```

```
 *
```

```
 * @author Admin
```

```
 */
```

```
public class Person {
```

```
    Person() {
```

```
        System.out.println("Konstruktor klasy");
```

```
    }
```

```
    Person(String name, int age) {
```

```
        this.name=name;
```

```
        this.age=age;
```

```
    }
```

```
    String name;
```

```
    int age;
```

```
}
```

```
package konstruktor;
```

```
/**
```

```
 *
```

```
 * @author Admin
```

```
 */
```

```
public class Konstruktor {
```

```
    /**
```

```
     * @param args the command line arguments
```

```
     */
```

```
    public static void main(String[] args) {
```

```
        Person p1=new Person("Adrian",18);
```

```
        Person p2=new Person("Ola",19);
```

```
        Person p3=new Person("Tomek",17);
```

```
        System.out.println(p1.name + " , " + p1.age);
```

```
        System.out.println(p2.name + " , " + p2.age);
```

```
        System.out.println(p3.name + " , " + p3.age);
```

```
    }
```

```
}
```

```
Output - Konstruktor (run) x
run:
Adrian , 18
Ola , 19
Tomek , 17
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Static vs non-static (1/2)

```
package com.company;

public class Person {
    Person(String name, int age) {
        this.name = name;
        this.age = age;
        popultion++;
    }

    String name;
    int age;

    int popultion = 0;
}
```

konstruktor Person

```
package com.company;

public class Main {

    public static void main(String[] args) {
        Person p1 = new Person("Adrian", 18);
        Person p2 = new Person("Bartek", 22);

        System.out.println(p1.popultion);
        System.out.println(p2.popultion);
    }
}
```

Sprawdzamy ile jest  
obiektów za pomocą  
zmiennej population



Wynik:

1  
1

Obiekty o sobie nie  
wiedzą

# Static vs non-static (2/2)

```
package com.company;

public class Person {
    Person(String name, int age) {
        this.name = name;
        this.age = age;
        popultion++;
    }

    String name;
    int age;

    static int popultion = 0;
}
```

Zmienna ta występować będzie tylko jedna reprezentacja całej klasy

```
package com.company;

public class Main {

    public static void main(String[] args) {
        Person p1 = new Person("Adrian", 18);
        Person p2 = new Person("Bartek", 22);

        System.out.println(p1.popultion); NOK
        System.out.println(p2.popultion); NOK
        System.out.println(Person.popultion); OK
    }
}
```

NOK – działa, ale jest niezgodne z konwencją, gdyż do zmiennych statycznych nie powinniśmy się odwoływać przez instancję obiektu tzn. przez zmienne p1 i p2 (bo mogą już nie istnieć) a przez klasę gdyż ona będzie zawsze istnieć.

Dostęp do zmiennej statycznej population → bezpośrednio przez nazwę klasę.

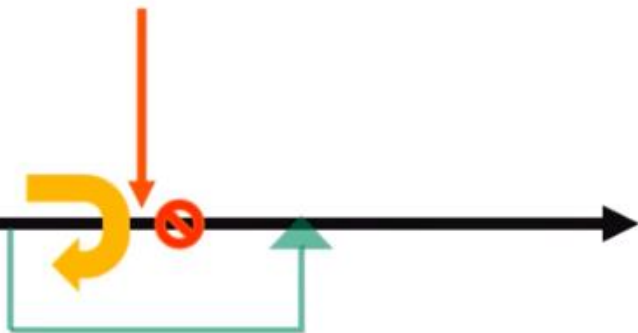
Wynik:

2  
2  
2

Obiekty o sobie wiedzą

# Obsługa wyjątków (1/6)

Error!



```
metoda( arg1, arg2){
```

```
    instrukcja 1;
```

```
    instrukcja 2;
```

```
    instrukcja 3; // wyjątek lub error
```

```
    instrukcja 4;
```

```
    instrukcja 5;
```

```
}
```



# Obsługa wyjątków (2/6)

```
package wyjątki;

/**
 *
 * @author Admin
 */
public class Wyjątki {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int res = 5/0;

        System.out.println("Dalsze instrukcje");
    }
}
```

Tutaj program się zakończy,  
nie przejdzie do linii  
odpowiedzialnej za wyświetlenie  
komunikatu

Output - Wyjątki (run) ×

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at wyjątki.Wyjątki.main(Wyjątki.java:18)
C:\Users\Admin\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

# Obsługa wyjątków (3/6)

Aby zapobiec takiej sytuacji musimy nasz „niebezpieczny” kod umieścić w specjalnej klauzuli:

Try → spróbuj

Catch → złap



# Obsługa wyjątków (4/6)

```
package wyjatki;

/**
 *
 * @author Admin
 */
public class Wyjatki {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try{
            int res = 5/0;
        }
        catch(ArithmeticException ex){ // łapiemy wyjątek, który zwraca nam blo try.
            // zmienna ex typu wyjątku ArithmeticException
            System.out.println("Dzielenie przez zero!!!!");
            System.out.println(ex.getMessage()); // odwołujemy się do metody który dostarczy nam info. o tym wyjątku
        }

        System.out.println("Dalsze instrukcje");
    }
}
```

Output - Wyjątki (run) X

```
run:
Dzielenie przez zero!!!!
/ by zero
Dalsze instrukcje
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Obsługa wyjątków (5/6)

```
package wyjatki;

/**
 *
 * @author Admin
 */
public class Wyjatki {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try{
            int res = 5/2;
            int []tab=new int[1];
            tab[4]=5;

        }
        catch(ArithmeticException ex){ // łapiemy wyjątek, który zwraca nam blo try.
            // zmienna ex typu wyjątku ArithmeticException
            System.out.println("Dzielenie przez zero!!!!");
            System.out.println(ex.getMessage()); // odwołujemy się do metody który dostarczy nam info. o tym wyjątku
        }

        System.out.println("Dalsze instrukcje");
    }
}
```

Output - Wyjatki (run) ×

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at wyjatki.Wyjatki.main(Wyjatki.java:21)
C:\Users\Admin\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

# Obsługa wyjątków (6/6)

```
package wyjątki;

/**
 *
 * @author Admin
 */
public class Wyjątki {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try{
            int res = 5/2;
            int []tab=new int[1];
            tab[4]=5;
        }
        catch(ArithmeticException ex){ // łapiemy wyjątek, który zwraca nam blo try.
            // zmienna ex typu wyjątku ArithmeticException
            System.out.println("Dzielenie przez zero!!!!");
            System.out.println(ex.getMessage()); // odwołujemy się do metody który dostarczy nam info. o tym wyjątku
        }
        catch(Exception ex){ //klasa ogólna Exception
            System.out.println("Inny błąd");
            System.out.println(ex.toString()); // metody toString dała nam informację o klasie błędu
        }
        System.out.println("Dalsze instrukcje");
    }
}
```

Output - Wyjątki (run) ×

```
run:
Inny błąd
java.lang.ArrayIndexOutOfBoundsException: 4
Dalsze instrukcje
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Linki

Pojęcia klasy, metody i obiektu

<https://www.youtube.com/watch?v=zSATsNzNV0E> 17:24

<https://www.youtube.com/watch?v=y0EvXmqnkM8> 26:29

Dziedziczenie

<https://www.youtube.com/watch?v=9xdzH5GE4bw> 23:37

Wyjątki i interfejsy

<https://www.youtube.com/watch?v=w48Pvz35vg> 27:51

# Dziękuję za uwagę!