

# Programowanie obiektowe

## Wykład 5.

Marcin Młotkowski

22 marca 2024

# Plan wykładu

- 1 Krótka historia Javy
- 2 Model obiektowy
- 3 Wyjątki
- 4 Pakiety
- 5 Zasada otwarte–zamknięte

# Plan wykładu

- 1 Krótka historia Javy
- 2 Model obiektowy
- 3 Wyjątki
- 4 Pakiety
- 5 Zasada otwarte–zamknięte

# Historia

## Początkowe założenia

- Projekt OAK
- Sterowanie urządzeniami domowymi
- Dodanie „życia” do internetu

# Logo



## Mały przykład

```
import java.io.*;
class Osoba
{
    private String Nazwisko;
    private int wzrost;
    Osoba (String Nazwisko, int wzrost) { ... }
    public void drukuj() { ... }
}
```

# Implementacja konstruktora

```
class Osoba {  
  
    Osoba (String Nazwisko, int wzrost)  
    {  
        this.Nazwisko = Nazwisko;  
        this.wzrost = wzrost;  
    }  
}
```

# Implementacja metody

```
class Osoba {  
  
    public void drukuj()  
    {  
        System.out.println("Nazwisko: " + this.Nazwisko);  
        System.out.println("Wzrost: " + this.wzrost);  
    }  
}
```



## Deklaracja podklasy

```
public class Student extends Osoba
{
    String kierunek;
    Student (String Nazwisko, int wzrost,
            String kierunek) {...}
    public void drukuj() { ... }
}
```

# Implementacja konstruktora

```
Student (String Nazwisko, int wzrost, String kierunek)
{
    super(Nazwisko, wzrost);
    this.kierunek = kierunek;
}
```

# Implementacja metody

```
public void drukuj()  
{  
    super.drukuj();  
    System.out.println("Kierunek: " + this.kierunek);  
}
```

## Początek programu

```
public static void main(String[] args)
{
    Osoba obj = new Student("Kubuś Puchatek", 35, "inf");
    obj.drukuj();
}
} // koniec klasy Student
```

# Schemat programu

Plik Student.java:

```
class Osoba { ... }

public class Student extends Osoba
{
    public static main(String[] args) { ... }
}
```

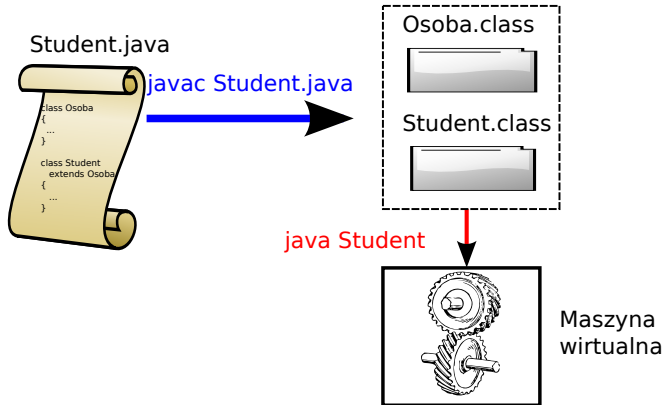
# Kompilacja i uruchomienie

```
$ javac Student.java
```

```
$ java Student
```

```
$ java -cp . Student
```

# Schemat



# Plan wykładu

- 1 Krótka historia Javy
- 2 Model obiektowy
- 3 Wyjątki
- 4 Pakiety
- 5 Zasada otwarte–zamknięte



# Model obiektowy Javy

## Elementy języka

- Klasy i obiekty
- Klasa Object, wszystkie klasy po niej dziedziczą
- Dziedziczenie pojedyncze
- Wszystkie metody są wirtualne
- Interfejsy
- Przestrzenie nazw
- Klasy i interfejsy generyczne

# Klasa Object

```
class Object
{
    Object Clone() { ... }
    bool equals(Object obj) { ... }
    String toString() { ... }
    Class getClass() { ... }
}
```

# Dynamiczna kontrola typów

```
if (obj instanceof Klasa)  
    Klasa zmienna = (Klasa)obj;
```

# Typy proste i złożone

- Typy proste: `int`, `float`, `boolean`
- Typy referencyjne: `Integer`, `Float`, `Boolean`, interfejsy, tablice

## Typy proste i złożone

- Typy proste: `int`, `float`, `boolean`
- Typy referencyjne: `Integer`, `Float`, `Boolean`, interfejsy, tablice

Autoboxing: automatyczna konwersja między typami prostymi i referencyjnymi.

# Różnice między językami

C#

Int32.MaxValue

Java

Integer.MAX\_VALUE

# Interfejsy

```
public class Application implements Runnable  
{  
    ...  
}
```

# Programowanie rodzajowe

```
public interface List<E>
{
    void add(E x);
    Iterator<E> iterator();
}
```



# Klasy

Klasy to też obiekty, należące do klasy `Class`

```
class Class
```

```
{  
    String getName() { ... }  
    Constructor[] getConstructors() { ... }  
    Field[] getFields() { ... }  
}
```

# Refleksje (introspekcje)

Mechanizm umożliwiający zbadanie obiektu: jego klasy, metod i pól.

# Refleksje (introspekcje)

Mechanizm umożliwiający zbadanie obiektu: jego klasy, metod i pól.

```
Class cl = obj.getClass()
```

# Plan wykładu

- 1 Krótka historia Javy
- 2 Model obiektowy
- 3 Wyjątki**
- 4 Pakiety
- 5 Zasada otwarte–zamknięte

# Motywacje

## Reakcja na błędy

- Błąd dzielenia przez zero
- Błąd operacji I/O
- Błąd przepelnienia

# Obsługa wyjątków

## Turbo Pascalu

```
write(fh, 'abcdXYZ');  
if IOResult <> 0 then  
    begin  
        ...  
    end
```

# Obsługa wyjątków

## Turbo Pascalu

```
write(fh, 'abcdXYZ');  
if IOResult <> 0 then  
    begin  
        ...  
    end
```

```
int podziel(int arg1, int arg2, ref int wynik);
```

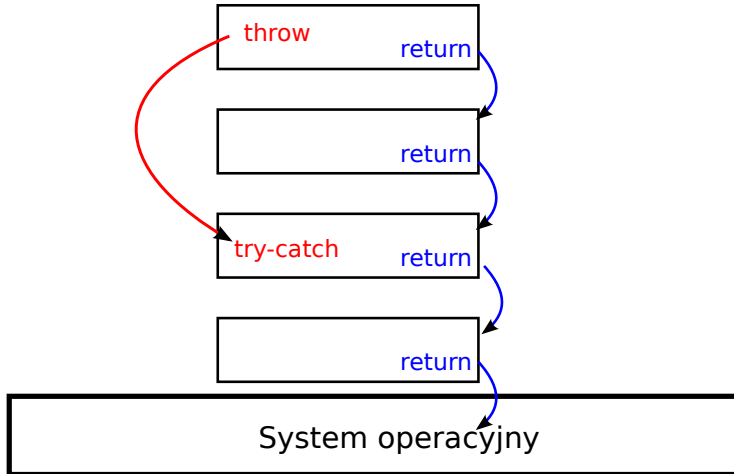
# Wyjątki

- Wyjątki to są obiekty klasy `Exception`;
- z wyjątkami skojarzony jest mechanizm zgłaszania i obsługi wyjątków.



## Przykład

```
public int podziel(int dzielna, int dzielnik)
{
    if (dzielnik == 0)
        throw new Exception();
    return dzielna/dzielnik;
}
```



# Instrukcja **try**

```
try  
{  
    ... // krytyczna instrukcja  
}  
catch (Exception e) { ... }  
finally { ... }
```

# Większy przykład

## Klasa implementująca stos

### Implementowane metody

- `void push(int elem)`  
może zgłosić wyjątek `StackOverflowException`
- `int pop()`  
może zgłosić wyjątek `EmptyStackException`

# Deklaracja wyjątku przepełnienia

```
class StackOverflowException extends Exception
{
    StackOverflowException()
    {
        super();
    }
}
```

## Wyjątek pustego stosu

```
class EmptyStackException extends Exception
{
    EmptyStackException()
    {
        super();
    }
    public void info()
    {
        printStackTrace();
    }
}
```

# Implementacja Stos

```
class Stos
{
    private int stos[];
    private int top;

    Stos(int rozm)
    {
        stos = new int[rozm];
        this.top = 0;
    }
}
```

# Implementacja metod

```
public void push(int elem) throws StackOverflowException
{
    if (top == stos.length)
        throw new StackOverflowException();
    stos[top] = elem;
    top++;
}
```



## Implementacja, cd

```
public int pop() throws EmptyStackException
{
    if (top == 0)
        throw new EmptyStackException();
    top--;
    return stos[top];
}
} // class Stos
```

# Jak korzystać

```
Stos s = new Stos(2);  
try { ... }  
catch (EmptyStackException e) { ... }  
catch (StackOverflowException e) { ... }  
finally { ... }
```

# Jak korzystać

```
Stos s = new Stos(2);  
  
try {  
    s.push(4);  
    s.push(2);  
    s.push(7);  
    System.out.println(s.pop());  
}  
  
catch (EmptyStackException e) { ... }  
  
catch (StackOverflowException e) { ... }  
  
finally { ... }
```

# Jak korzystać

```
Stos s = new Stos(2);  
try { ... }  
catch (EmptyStackException e) {  
    e.info();  
}  
catch (StackOverflowException e) { ... }  
finally { ... }
```

# Jak korzystać

```
Stos s = new Stos(2);  
  
try { ... }  
  
catch (EmptyStackException e) { ... }  
  
catch (StackOverflowException e) {  
    e.printStackTrace();  
    throw e;  
}  
  
finally { ... }
```

# Jak korzystać

```
Stos s = new Stos(2);  
  
try { ... }  
  
catch (EmptyStackException e) { ... }  
  
catch (StackOverflowException e) { ... }  
  
finally {  
    System.out.println("Zawsze się wykona");  
}
```

# Wyjątki

Deklaracja wyjątków jest częścią specyfikacji:

```
public void push(int elem) throws StackOverflowException
{
    ...
}
```

## Standardowe interfejsy z wyjątkami

```
interface Serializable
{
    private void writeObject(java.io.ObjectOutputStream out)
        throws IOException
    private void readObject(java.io.ObjectInputStream in)
        throws IOException, ClassNotFoundException;
}
```



# Plan wykładu

- 1 Krótka historia Javy
- 2 Model obiektowy
- 3 Wyjątki
- 4 Pakiety**
- 5 Zasada otwarte–zamknięte

## Przykładowy program

test.java

```
public class test { ... }  
  
class B { ... }
```

Po kompilacji powstają dwa pliki:

- test.class
- B.class

# Deklaracja pakietu

W katalogu wykład\java\:

Test.java

```
package wyklad.java;  
  
public class Test { ... }
```

## Odwołanie do elementów pakietu:

```
import wyklad.java.Test;  
  
wyklad.java.Test t = new wyklad.java.Test();
```

```
import wyklad.java.Test;  
  
Test t = new Test();
```

# Widzialność pól i metod

- **public**
- **protected** (domyślny): widoczny w ramach pakietu
- **private**

# Plan wykładu

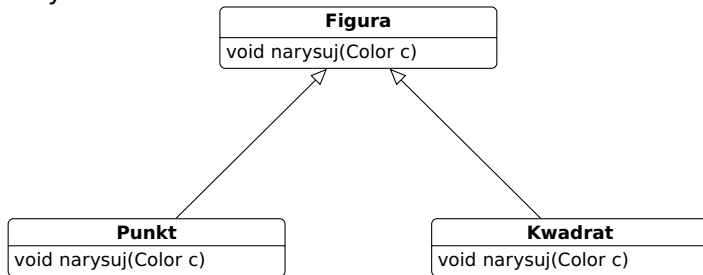
- 1 Krótka historia Javy
- 2 Model obiektowy
- 3 Wyjątki
- 4 Pakiety
- 5 Zasada otwarte–zamknięte

## Zasada otwarte–zamknięte

Klasy i metody powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji.

## Co to znaczy?

Przykład:





## Rysowanie obrazków, wersja 1.

```
Figura[] obrazek;  
for (Figura f: obrazek)  
{  
    f.narysuj();  
}
```

## Rysowanie obrazków, wersja 2.

Najpierw należy narysować obiekty klasy Punkt, potem Kwadrat.

## Rysowanie obrazków, wersja 2.

Najpierw należy narysować obiekty klasy Punkt, potem Kwadrat.

```
Arrays.sort(obrazek);  
for (Figura f: obrazek)  
{  
    f.narysuj();  
}
```

Exception in thread "main" java.lang.ClassCastException:  
Punkt cannot be cast to java.lang.Comparable

# Jak porządkować obiekty

```
public interface Comparable <T>  
{  
    int compareTo(T o);  
}
```

# Implementacja interfejsu

```
public class Figura implements Comparable <Figura>
{
    public int compareTo(Figura o) { ... }
}
```

```
public class Punkt extends Figura
{
    public int compareTo(Figura o)
    {
        if (o instanceof Kwadrat) return 1;
        return -1;
    }
}
```

## Mamy za darmo sortowanie

```
import java.util.Arrays;
```

```
Figura[] obrazek = new Figura[3];  
obrazek[0] = new Punkt();  
obrazek[1] = new Prostokat();  
obrazek[2] = new Figura();  
Arrays.sort(obrazek);
```

```
for (Figura f: obrazek)  
{  
    f.narysuj();  
}
```

# Rozwój oprogramowania

Dodajemy klasę Okrąg

# Rozwój oprogramowania

Dodajemy klasę Okrąg

## Konsekwencje

Musimy zmienić implementację `compareTo()` we wszystkich już zaimplementowanych klasach.



# Wesołych Świąt!

# Wesołych Świąt!

