

Programowanie w Java Klasy i obiekty

Łukasz Rybka · Gdańsk 2016/17

Podstawowe pytania

Podstawowe pytania

**CO JEST FUNDAMENTALNYM
POJĘCIEM Z ZAKRESU
TECHNOLOGII OBIEKTOWEJ?:**

Podstawowe pytania

CO JEST FUNDAMENTALNYM
POJĘCIEM Z ZAKRESU
TECHNOLOGII OBIEKTOWEJ?:
?

Podstawowe pytania

**CO JEST FUNDAMENTALNYM
POJĘCIEM Z ZAKRESU
TECHNOLOGII OBIEKTOWEJ?:**
?

CO TO JEST KLASA?:

Podstawowe pytania

CO JEST FUNDAMENTALNYM
POJĘCIEM Z ZAKRESU
TECHNOLOGII OBIEKTOWEJ?:
?

CO TO JEST KLASA?:
?

Podstawowe pytania

CO JEST FUNDAMENTALNYM
POJĘCIEM Z ZAKRESU
TECHNOLOGII OBIEKTOWEJ?:
?

CO TO JEST KLASA?:
?

CO TO JEST OBIEKT?:

Podstawowe pytania

CO JEST FUNDAMENTALNYM
POJĘCIEM Z ZAKRESU
TECHNOLOGII OBIEKTOWEJ?:
?

CO TO JEST KLASA?:
?

CO TO JEST OBIEKT?:
?

Co to jest klasa?

“ *Klasa to abstrakcyjny typ danych wraz z całkowitą lub częściową implementacją.*

— Bertrand Meyer

Przykład prostej klasy

```
package pl.org.dragonia.oopl;

public class Human {

    public String name;
    public int age;

    public void walk() {
        System.out.println("Human with name " + name + " is walking...");
    }
}
```

Przykład prostej klasy

```
package pl.org.dragonia.oopl;

public class Human {

    public String name;
    public int age;

    public void walk() {
        System.out.println("Human with name " + name + " is walking...");
    }
}
```

```
Human human1 = new Human();
```

```
human1.name = "Jacek";
human1.age = 11;
human1.walk();
```

Java Bean

Java Bean

Java Bean - konwencja reużywalnych komponentów w Javie

Java Bean

Java Bean - konwencja reużywalnych komponentów w Javie

Jest serializowalna

Java Bean

Java Bean - konwencja reużywalnych komponentów w Javie

Jest serializowalna

Posiada bezargumentowy konstruktor

Java Bean

Java Bean - konwencja reużywalnych komponentów w Javie

Jest serializowalna

Posiada bezargumentowy konstruktor

Dostęp do (prywatnych) zmiennych przez settery i gettery

Poprawny przykład prostej klasy

```
package pl.org.dragonia.oopl;

public class Human {

    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void walk() {
        System.out.println("Human with name " + name + " is walking...");
    }
}
```

Wykorzystanie klasy Human

```
Human human1 = new Human();
```

```
human1.setName("Jacek");
```

```
human1.walk();
```

Modyfikatory dostępu

Modyfikatory dostępu

PUBLIC:

Modyfikatory dostępu

PUBLIC:

pozwala na dostęp wszystkim klasom z dowolnego pakietu

Modyfikatory dostępu

PUBLIC:

pozwala na dostęp wszystkim klasom z dowolnego pakietu

PACKAGE (DOMYŚLNY):

Modyfikatory dostępu

PUBLIC:

pozwała na dostęp wszystkich klasom z dowolnego pakietu

PACKAGE (DOMYŚLNY):

dostęp do danej klasy/metody/pola mają jedynie klasy z tego samego pakietu

Modyfikatory dostępu

PUBLIC:

pozwała na dostęp wszystkich klasom z dowolnego pakietu

PACKAGE (DOMYŚLNY):

dostęp do danej klasy/metody/pola mają jedynie klasy z tego samego pakietu

PRIVATE:

Modyfikatory dostępu

PUBLIC:

pozwała na dostęp wszystkim klasom z dowolnego pakietu

PACKAGE (DOMYŚLNY):

dostęp do danej klasy/metody/pola mają jedynie klasy z tego samego pakietu

PRIVATE:

nikt poza samą klasą nie ma dostępu do danej klasy/pola

Constructors

Constructors

Służą do tworzenia obiektów - wywoływane przy słowie kluczowym **new**

Constructors

Służą do tworzenia obiektów - wywoływane przy słowie kluczowym **new**

Jeżeli nie zdefiniujemy żadnego konstruktora - kompilator zrobi to za nas!

Constructors

Służą do tworzenia obiektów - wywoływane przy słowie kluczowym **new**

Jeżeli nie zdefiniujemy żadnego konstruktora - kompilator zrobi to za nas!

Istnieje możliwość przeciążania konstruktorów przez parametry (overloading)

Constructors

Służą do tworzenia obiektów - wywoływane przy słowie kluczowym **new**

Jeżeli nie zdefiniujemy żadnego konstruktora - kompilator zrobi to za nas!

Istnieje możliwość przeciążania konstruktorów przez parametry (overloading)

Kiedy zdefiniujemy konstruktor z parametrami - domyślny nie będzie istnieć!

Klasa z wieloma konstruktorami

```
package pl.org.dragonia.oopl;

public class Human {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public Human() {
    }

    public Human(String name) {
        this.name = name;
    }
}
```

Słowo kluczowe **this**

Słowo kluczowe **this**

Może być stosowane wyłącznie w metodach niestatycznych (!), także konstruktorach

Słowo kluczowe **this**

Może być stosowane wyłącznie w metodach niestatycznych (!), także konstruktorach

Przechowuje referencję do bieżącego obiektu

Słowo kluczowe **this**

Może być stosowane wyłącznie w metodach niestatycznych (!), także konstruktorach

Przechowuje referencję do bieżącego obiektu

Może być pominięte jeśli w metodzie nie zdefiniujemy zmiennej lokalnej o tej samej nazwie co pole obiektu

Słowo kluczowe **this**

Może być stosowane wyłącznie w metodach niestatycznych (!), także konstruktorach

Przechowuje referencję do bieżącego obiektu

Może być pominięte jeśli w metodzie nie zdefiniujemy zmiennej lokalnej o tej samej nazwie co pole obiektu

Przekazywane niejawnie

Statyczne elementy klasy

Statyczne elementy klasy

Statyczne mogą być zarówno pola jak i metody

Statyczne elementy klasy

Statyczne mogą być zarówno pola jak i metody

Konstruktor nie może być statyczny

Statyczne elementy klasy

Statyczne mogą być zarówno pola jak i metody

Konstruktor nie może być statyczny

Dostęp do statycznego pola/metody nie wymaga inicjalizacji nowego obiektu

Statyczne elementy klasy

Statyczne mogą być zarówno pola jak i metody

Konstruktor nie może być statyczny

Dostęp do statycznego pola/metody nie wymaga inicjalizacji nowego obiektu

Kwestie wydajności, poprawności i unikania błędów

Klasa z elementami statycznymi

```
package java.lang;

public final class Math {
    public static final double PI = 3.14159265358979323846;
    public static final double E = 2.7182818284590452354;

    public static double sqrt(double a) {
        // ...
    }

    public static double log(double a) {
        // ...
    }

    public static double log10(double a) {
        // ...
    }
}
```

Słowo kluczowe final

Słowo kluczowe final

"To coś nie może być zmienione"

Słowo kluczowe **final**

"To coś nie może być zmienione"

Różne znaczenie w zależności od kontekstu

Słowo kluczowe **final**

"To coś nie może być zmienione"

Różne znaczenie w zależności od kontekstu

Może być zastosowane dla pól, metod oraz klas

Słowo kluczowe **final**

"To coś nie może być zmienione"

Różne znaczenie w zależności od kontekstu

Może być zastosowane dla pól, metod oraz klas

Może poprawić wydajność - z tym należy uważać !

Finalne pole klasy

Finalne pole klasy

Podstawowym zastosowaniem jest tworzenie "stałych czasu kompilacji"

Finalne pole klasy

Podstawowym zastosowaniem jest tworzenie "stałych czasu kompilacji"

Pole finalne przechowujące referencję do obiektu nie może być zmienione, ale sam obiekt już tak

Finalne pole klasy

Podstawowym zastosowaniem jest tworzenie "stałych czasu kompilacji"

Pole finalne przechowujące referencję do obiektu nie może być zmienione, ale **sam obiekt już tak**

Pole finalne nie musi być zainicjalizowane w czasie deklaracji

Finalne pole klasy

Podstawowym zastosowaniem jest tworzenie "stałych czasu kompilacji"

Pole finalne przechowujące referencję do obiektu nie może być zmienione, ale **sam obiekt już tak**

Pole finalne nie musi być zainicjalizowane w czasie deklaracji

Argumenty metod również mogą być finalne

Finalne metody

Finalne metody

Klasy pochodne nie mogą zmieniać metod finalnych

Finalne metody

Klasy pochodne nie mogą zmieniać metod finalnych

Wywołanie metody finalnej może zostać przez kompilator na tzw. **wywołanie w miejscu** (ang. **inline**) co skutkuje wyższą wydajnością

Finalne metody

Klasy pochodne nie mogą zmieniać metod finalnych

Wywołanie metody finalnej może zostać przez kompilator na tzw. **wywołanie w miejscu** (ang. **inline**) co skutkuje wyższą wydajnością

Każda metoda prywatna jest **de facto** finalna

Finalne klasy

Finalne klasy

Klasy finalnej nie można dziedziczyć

Finalne klasy

Klasy finalnej nie można dziedziczyć

Dopisywanie słowa `final` do metod klasy finalnej jest nadmiarowe i może wprowadzać w błąd!

ANY
QUESTIONS
?