



Języki i Techniki Programowania

dr hab. Piotr Kosiuczenko
profesor WAT

JTP

1

1

Przegląd Tematów

- Wstęp do języka Java i programowania obiektowego:
 - charakterystyka języka,
 - porównanie z innymi językami.
- Dziedziczenie.
- Maszyna wirtualna.
- Interfejs do programowania aplikacji (Java API).
- Klonowanie i identyczność obiektów.
- Wyjątki.
- Testowanie, JUnit.
- Polimorfizm.
- Rodzaje (Java Generics).

JTP

Piotr Kosiuczenko

2

2

Literatura

- Bruce Eckel, Thinking in Java, PDF:
www.dblab.ntua.gr/~gtsat/collection/Java%20books/Bruce.Eckel.Thinking.In.Java.4th.Edition.Dec.2007.eBook-BBL.pdf
- Ken Arnold, James Gosling: Java; z ang. przeł. Grzegorz Grudziński Warszawa, WNT, 1999, ISBN 83-204-2313-9.
- Cay Horstmann, Java: Podstawy, Helion, 2008 i następne wydania.

The best book on programming for the layman is "Alice in Wonderland"; but that's because it's the best book on anything for the layman. [Perlis, A.: Epigrams on Programming]

JTP

Piotr Kosiuczenko

4

4

Literatura w sieci

- Tutorial: <https://docs.oracle.com/javase/tutorial/index.html/>
- https://www.w3schools.com/java/java_oop.asp
- Dokumentacja
 - Java Standard Edition (tego będziemy używać):
<http://www.oracle.com/technetwork/java/javase/documentation/>
 - Całościowa: <http://www.oracle.com/technetwork/java/i>
- W czasie wykładu i laboratoriów będą podawane dodatkowe odnośniki do literatury materiałów dostępnych w sieci
- Pytania zadawane w czasie interview:
<https://www.edureka.co/blog/interview-questions/java-interview-questions>
- <https://codegym.cc/pl/groups/posts/pytania-rekrutacyjne-dotycze-jzyka-java>
- Bardzo szczegółowe pytania z interview:
<https://www.javatpoint.com/corejava-interview-questions>

JTP

Piotr Kosiuczenko

5

5

Projekty szkoleniowe GreenFoot oraz BlueJ

- GreenFoot to użyteczne środowisko do nauki podstaw programowania obiektowego : <https://www.greenfoot.org/home>
- Kanał video: <https://www.youtube.com/@18km>
- BlueJ to podobny projekt, choć bardziej zaawansowany: <https://www.bluej.org/>
- Opis <http://www.cs.kent.ac.uk/pubs/2008/2697/content.pdf>
- Przykładowy kanał video:
https://www.youtube.com/playlist?list=PL9HfA4ZKbzintNeJi09vJxII_RiOn9eB
- W bibliotece WAT są odpowiednie książki.
- Trzeba jednak uważać na alternatywną terminologię.

JTP

Piotr Kosiuczenko

6

6

Organizacja

- 10 wykładów i 20 laboratoriów st. stac./6 + 12 st. niestacjonarne.
- Zaliczenie laboratorium jest na podstawie
 - pracy w ciągu semestru,
 - kolokwium (czas 45 min., planowany termin: koniec).
- Kolokwium zaliczające wykład - czas trwania około 25ciu minut
- Dopuszczenie do zaliczenia wykładu będzie na podstawie zaliczenia laboratorium
- Konsultacje będą po zajęciach, także wedle kalendarza w e-dziekanacie, jak i na prośbę mailową

JTP

Piotr Kosiuczenko

7

7

Jezyki Obiektowo Zorientowane

- Simula rok 1967;
- Jezyki oparte na Lisp koniec lat 70ych;
- Smalltalk rok 1980;
- C++, Eiffel polowa lat 80ych;
- Java rok 1995;
- C# rok 2005.

What is the difference between a Turing machine and the modern computer? It's the same as that between Hillary's ascent of Everest and the establishment of a Hilton hotel on its peak. [Perlis, A.: Epigrams on Programming]

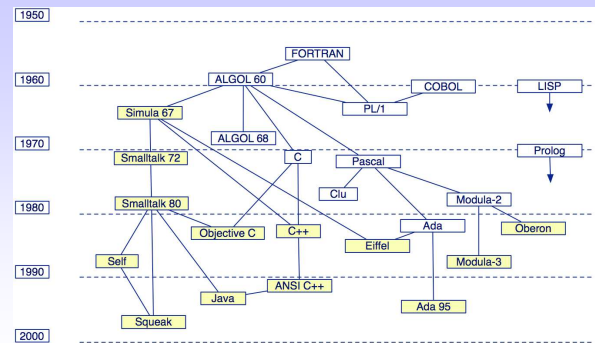
JTP

Piotr Kosiuczenko

8

8

Historia Jezyków Programowania



JTP

Piotr Kosiuczenko

9

9

Porownanie: Javy, C++ i Smalltalka

	<i>Smalltalk</i>	<i>C++</i>	<i>Java</i>
<i>model obiektów</i>	czysto obiektowy	hybrydalny	hybrydalny
<i>zbiórka śmieci</i>	automatyczna	'ręczna' + automatyczna	automatyczna
<i>dziedziczenie</i>	pojedyncze	wielokrotne	pojedyncze
<i>typy</i>	dynamiczne	statyczne	statyczne
<i>refleksyjność</i>	pełna refleksyjność	introspekcja	introspekcja
<i>współbieżność</i>	semafory, monitory	biblioteki	monitory
<i>wykonanie</i>	kodu maszynowego	kodu maszynowego	kodu dla maszyny wirtualnej

JTP

Piotr Kosiuczenko

10

10

Odpowiedz

```
n calls m.
m throws runtime exception e.
m handles e.
m executes its finally-part.
n handles e.
n executes its finally-part.
```

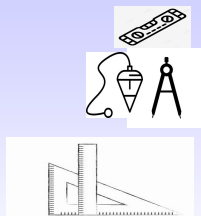
JTP

Piotr Kosiuczenko

11

11

JTP: Podstawy języka Java



dr hab. Piotr Kosiuczenko
prof. WAT

JTP

Piotr Kosiuczenko

12

12

Java: Charakterystyka Języka

- Wszystko jest albo prymitywną wartością, albo obiektem, albo klasą.
- Typami referencyjnymi w jawie są klasy.
- Nie ma zmiennych globalnych.
- Pojedyncze dziedziczenie klas (ang. single inheritance).
- Obiekty są składowane na tzw. składzie (ang. heap) i dzielone przez różne wątki (threads).
- Automatyczne zarządzanie pamięcią - zbiórka śmieci (ang. garbage collection).
- Nie ma metody `free()` znanej z C++.

JTP

Piotr Kosiuczenko

13

13

Principia języka Java

- Abstrakcja
 - Obiekty
 - Asocjacja
 - Zakapslowanie
 - Interfejsy
- Polimorfizm
 - Dziedziczenie
 - Przeciążenie
 - Typy rodzajowe (Java generics)
- Compile once run everywhere - kompilowalność na JVM
- Automatyczna zbiórka śmieci

JTP

Piotr Kosciuzenko

14

14

Principia języka Java: JVM

- Maszyna wirtualna (Java Virtual Machine – JVM) wykonuje tzw. kod bajtowy (bytecode):
 - dynamiczne ładowanie klas (ładowanie klasy następuje tylko, gdy jej kod ma być wykonany),
 - weryfikacja kodu pośredniego (bezpieczeństwo),
 - Interpretacja/wykonanie kodu pośredniego.
- Niezależność od sprzętu: compile once, run everywhere – nie do końca udało się zrealizować.
- Bezpieczeństwo wykonania programu poprzez statyczne typy/klassy i piaskownice (sandboxing).

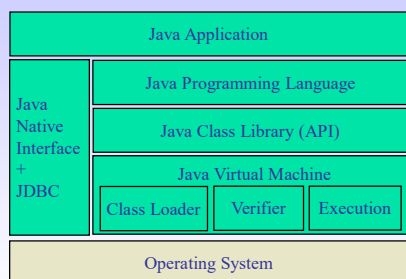
JTP

Piotr Kosciuzenko

15

15

Java: Struktura Systemu



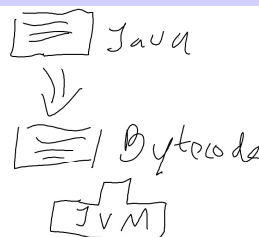
JTP

Piotr Kosciuzenko

16

16

Java: Bytecode and JVM



JTP

Piotr Kosciuzenko

17

17

Java: JNI I JRE

- JNI (Java Native Interface) pozwala na bezpośredni dostęp do instrukcji systemu operacyjnego i użycie innych programów.
- JNI jest używany gdy:
 - standardowa biblioteka Javy nie zawiera specyficznych funkcji lub programów;
 - jest potrzeba używania baz danych – np. sterowniki JDBC używają JNI do wywołania Oracle Call Interface
- Steruje przepływem danych, np. czytanie i zapis danych, dźwięk i obraz.

Java Runtime Environment (JRE) implementuje JNI i JVM.

JTP

Piotr Kosciuzenko

18

18

Java: API

- Interfejs do Programowania Aplikacji (ang. Application Programming Interface - API) jest biblioteką interfejsów i klas standardowo należących do języka Java.
- W czasie tego wykładu omówimy min. kolekcje, wyjątki, klasy do programowania równoległego oraz Aplety.
- Wspomnimy także o prymitywnych typach danych.

JTP

Piotr Kosciuzenko

19

19

Java: Hello World

```

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}

```

deklaracja klasy sygnatura metody main

metoda statyczna – działa na klasie

parametr domyślny/implicit

korpus/ciało metody

parametr explicite

Statyczna metoda działa na klasie, a nie na obiektach.

metoda obiektowa – działa na obiekcie będącym wartością parametru implicit

JTP Piotr Kosieczko 20

20

Konstitutywne składowe obiektu

- Obiekt jest jednostką funkcjonalną zawierającą **dane i funkcje** na nich operujące.
- Właściwości/dane obiektu są charakteryzowane przez atrybuty.
- Atrybuty są modyfikowane przez metody (funkcje).
- Zakapslowanie znaczy, że własności obiektu modyfikujemy tylko za pomocą wywoływanych metod, a nie bezpośredniego dostępu do atrybutów.

JTP Piotr Kosieczko 21

21

Java: Klasy i Obiekty

klasa

```

+Point
- double x
- double y
+ flip()

```

obiekty

p1 : Point

```

x = 1
y = 33

```

p2 : Point

```

x = 1
y = 33

```

- Klasy pozwalają na tworzenie obiektów do nich należących.
- Klasa może posiadać własne (statyczne, tj. nieobiektywne) atrybuty oraz własne (statyczne, tj. nieobiektywne) metody.

JTP Piotr Kosieczko 22

22

Java: Przekazywanie Wartości Parametrów

- W Javie wartości parametrów przekazywane są zgodnie z zasadą wywołania przez wartość (ang. call by value).
- Znaczy to, że przekazywane parametry są:
 - liczone
 - kopiowane do nowych zmiennych w wywołanej metodzie
 - zmiana wartości tych nowych zmiennych przez wywołaną metodę nie zmienia wartości zmiennych w metodzie wywołującej
- W konsekwencji, metoda wywołana nie zmienia wartości zmiennych, które zostały przekazane do wywołanej metody jako parametry.

JTP Piotr Kosieczko 23

23

Java: Przekazywanie Wartości Parametrów

```

public class Point {
    private double x, y;
    public double getX() {
        return this.x;
    }
    public double getY() {
        return this.y;
    }
    public Point(double nx, double ny) {
        this.x = nx;
        this.y = ny;
    }
}

```

atrybuty

typ zwracanej wartości

metoda obiektowa

konstruktor

JTP Piotr Kosieczko 24

24

Java: Przekazywanie Wartości Parametrów

```

public double sumUpAndIncreaseBy(double dx) {
    return ++this.x + ++this.y + ++dx;
}
public static void main(String[] args) {
    Point p1 = new Point(3, 3);
    double r = 5;
    System.out.println("p1.x = " + p1.getX() + ", p1.y = " + p1.getY() + "; " + "p1.sumupANDIncreaseBy(r) results in " + p1.sumupAndIncreaseBy(r) + ", p1.x = " + p1.getX() + ", p1.y = " + p1.getY() + ", r = " + r);
}

```

zwracana 4

zwracana 4

JTP Piotr Kosieczko 25

25

Programowanie

It goes against the grain of modern education to teach children to program. What fun is there in making plans, acquiring discipline in organizing thoughts, devoting attention to detail and learning to be self-critical?

[Perlis, A.: Epigrams on Programming]

JTP

Piotr Kosciuzenko

26

26

Java: Czytanie Danych z Konsoli

- W Javie jest możliwe czytanie danych z konsoli i dialog z użytkownikami.
- Jest wiele możliwości – prezentujemy jedną z nich:

```
import java.util.Scanner;

public class InputFloatFromConsole {
    public static void main(String[] args) {
        Scanner myScanner = new Scanner(System.in);
        System.out.println("Enter float");

        String input = myScanner.nextLine();
        float x = Float.parseFloat(input);
        System.out.println("Float is: " + input); // Output user input
    }
}
```

importowanie odpowiednich klas

utwórz obiekt skanujący

obъект czyta dane z konsoli

parsowanie danych

JTP

Piotr Kosciuzenko

27

27

Java: Typy prymitywne oraz typy referencyjne

- Typy prymitywne w Javie (mają swoje stowarzyszone klasy)
- W Javie istnieją cztery podstawowe typy referencyjne, będące zresztą jednostkami programistycznymi:
 - klasy konkretne (ang. concrete class),
 - klasy abstrakcyjne (ang. abstract class),
 - klasy wewnętrzne (ang. inner class),
 - interfejsy (ang. interface).

JTP

Piotr Kosciuzenko

29

29

Java: typy prymitywne

boolean	1-bit albo true albo false
byte	8-bit integer (ze znakiem, od -2^7 do 2^7-1)
char	16-bit unicode character
short	16-bit integer (ze znakiem, od -2^{15} do $2^{15}-1$)
int	32-bit integer (ze znakiem)
long	64-bit integer (ze znakiem)
float	32-bit floating-point
double	64-bit floating-point

Integer, Boolean i Character są klasami, a nie prymitywnymi typami.

JTP

Piotr Kosciuzenko

30

30

Java: Typy Referencyjne

- Klasy są albo zaimplementowane przez użytkownika, albo są zawarte w bibliotece Application Programming Interface (API).
- Java API zawiera wiele użytecznych klas:
C:\Program Files (x86)\Java\docs\api\index.html
- Klasa `Object` obejmuje wszystkie inne klasy i zawiera podstawowe metody dostępne poprzez dziedziczenie wszystkim innym klasom.

JTP

Piotr Kosciuzenko

31

31

Java: Klasy Konkretnie

- Podstawową jednostką programistyczną jest **klasa** (słowo kluczowe `class`). Zawiera ona atrybuty klasowe (class attributes) i obiektywne (object attributes) oraz metody klasowe i obiektywne.
- Klasy deklarują atrybuty klasowe i obiektywne, oraz metody klasowe i obiektywne (działają na parametrze domyślnym `this`).
 - Obiekty zawsze tworzone są w odniesieniu do pewnej konkretnej klasy; używa się wtedy słowa kluczowego `new` (zobacz przykład klasy `Point`).
 - Obiekty posiadają atrybuty obiektywne, ale nie klasowe, oraz metody.
 - Klasy implementują interfejsy (może ich być wiele) oraz dziedziczą (inherit/extend) własności innych klas (w tym abstrakcyjnych).
 - Dziedziczenie jest jednokrotne, tzn. klasa może dziedziczyć własności co najwyżej jednej innej klasy (abstrakcyjnej bądź nie).

JTP

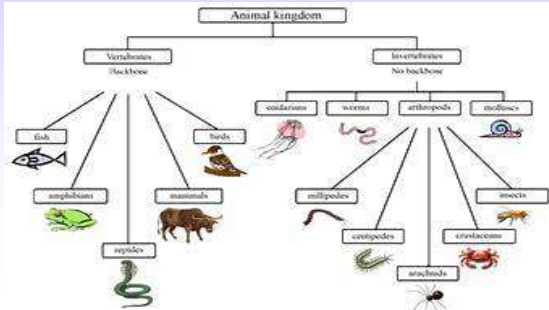
Piotr Kosciuzenko

32

32

Java: Klasy Abstrakcyjne

Klasa abstrakcyjna (słowo kluczowe **abstract class**) nie pozwala na generowanie jej instancji, tj. mniej można tworzyć jej obiektów.



33

33

Java: Klasy Abstrakcyjne cd.

- Może implementować wiele interfejsów ale dziedziczy własności co najwyżej jednej klasy (także abstrakcyjnej).
- Posiada atrybuty i metody.
- Metody nie muszą być zaimplementowane (słowo kluczowe **abstract**).
- Metoda abstrakcyjna musi być zaimplementowana przez dziedziczącą klasę konkretną.

JTP

Piotr Kosiuczenko

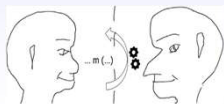
34

34

Java: Interfejsy

Interfejs (słowo kluczowe **interface**) jest listą dostępnych metod i stałych. Interface:

- niezależna klientów systemu od jego implementacji.
- zawierają zasadniczo sygnatury, ewentualnie implementacje metod oraz tzw. stałe (niezmienne atrybuty statyczne definiowane z poziomu interfejsu), ale nie atrybuty obiektowe.
- może rozszerzać tylko inne interfejsy (może być ich wiele), ale nie implementuje klas.
- jest implementowany przez klasy, używane jest wtedy słowo kluczowe **implements**.



JTP

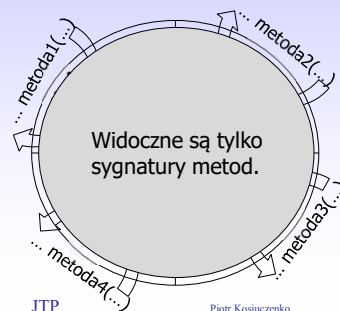
Piotr Kosiuczenko



35

Widok obiektu poprzez interfejs

```
public interface Figure {
    public void move(double dx, . . .);
    public void flip();
}
```



- Interfejs to lista sygnatur metod.
- Metodę wywołujemy podając nazwę metody oraz wartości parametrów.
- Metoda zwraca wartość określonego typu.
- Nie mamy tu dostępu do atrybutów ani do implementacji metod.
- Możliwe są więc różne reprezentacje własności i różne implementacje metod.

JTP

Piotr Kosiuczenko

36

36

Java: Klasy Wewnętrzne

- Klasa wewnętrzna jest zdefiniowana wewnątrz innej klasy (nie ma specjalnego słowa kluczowego).
- Dostęp do niej i jej obiektów ma tylko kod zawierający ją klasy i podklasy;
- dla innych klas jest niewidoczna.
- Pozwala na uniknięcie zaśmiecenia przestrzeni nazw (ang. name spece).

JTP

Piotr Kosiuczenko

37

37

Dziedziczenie: zasada Liskov

Idea dziedziczenia jest oparta na zasadzie podstawiania sformułowanej przez Barbarę Liskov:

Obiekty podklasy muszą dać się użyć wszędzie tam, gdzie mogą być użyte obiekty nadklasy.

To znaczy w szczególności:

- widoczne atrybuty obiektowe nadklasy są dostępne w podklasie
- metody nadklasy są dostępne lub zdefiniowane w podklasach
- podklasa powinna zachowywać kontrakt nadklasy
- oczywiście podklasa może posiadać dodatkowe atrybuty i metody

JTP

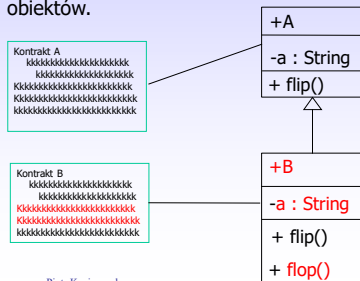
Piotr Kosiuczenko

38

38

Zasada Liskov – dokładniejsze sformułowanie

Kontrakt nadklasy musi być spełniany przez podklasę; w szczególności obiekty z podklasy muszą spełniać kontrakt obiektów z nadklasy. Jednak kontrakt podklasy nie wiąże nadklasy ani jej obiektów.



JTP

Piotr Kosieczko

39

39

Dziedziczenie: mechanizmy wiązania

- W Javie podklasa może na nowo zdefiniować atrybuty i metody nadklasy, choć te dwa przypadki odpowiadają dwóm różnym mechanizmom wiązania.
- Istnieją zasadniczo dwa mechanizmy wiązania:
 - styczne - w czasie kompilacji na podstawie typu wyrażenia
 - dynamiczne - w czasie wykonania na podstawie aktualnej klasy obiektu zapisanego w parametrze domyślnym **this**.
- W Javie *atrybuty, metody statyczne i konstruktory są wiązane statycznie*
- metody obiektowe są wiązane dynamicznie* zależnie od typu aktualnego parametru domyślnego, tj. **this**.
- Wiązaniu metod obiektowych służy algorytm „look up”. Wiązana jest najbardziej aktualna metoda.

JTP

Piotr Kosieczko

40

40

Dziedziczenie: Co będzie wynikiem?

```
class A {
    public String a = "Attribute of A";
}
class B extends A {
    public String a = "Attribute of B";
    public static void main(String[] args) {
        A o = new B();
        String x = o.a;
        System.out.println(x);
    }
}
```

Typem zmiennej o jest klasa A.

Wynikiem będzie: "Attribute of A";

Dlaczego?

JTP

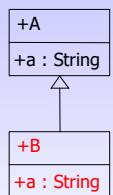
Piotr Kosieczko

41

41

Dziedziczenie: Co będzie wynikiem?

```
class A {
    public String a = "Attribute of A";
}
public class B extends A {
    private String a = "Attribute of B";
    ...
    A o = new B();
    String x = o.a;
    ...
}
```



a1 : A
a = „Str”
Obiekt a1 klasy A

b1 : B
a = „Attribute of A”
a = „Attribute of B”
Obiekt b1 klasy B
Atrybut a z poziomu klasy A.
Atrybut a z poziomu klasy B.

JTP

Piotr Kosieczko

42

42

Dziedziczenie: Co będzie wynikiem?

```
class A {
    public String m() {
        return "A";
    }
}
class B extends A {
    public String m() {
        return "B";
    }
}
public static void main(String[] args) {
    A o = new B();
    System.out.println("Class name of o is " + o.m());
}
```

Wynikiem będzie: "Class name of o is B";

Dlaczego?

Bo metody są wiązane dynamicznie i zostanie związana najbardziej aktualna metoda.

JTP

Piotr Kosieczko

43

43

Dziedziczenie: Co będzie wynikiem kompilacji?

```
class A {
    public A a;
}
class B extends A {
    private A a;
}
class A1 {
    public void m() {
    }
}
class B1 extends A1 {
    private void m() {
    }
}
```

Wynikiem będzie powstanie Bytecodu.

Dlaczego?

Wynikiem będzie komunikat o błędzie kompilacji.

Dlaczego?

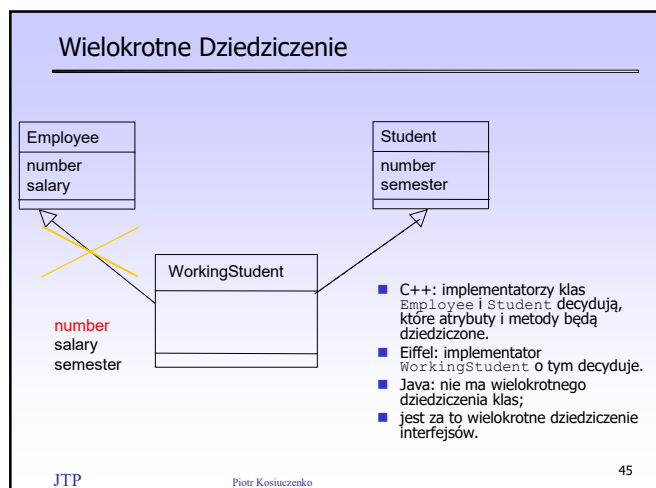
Bo wiązanie metody m z klasy B1 ogranicza jej dostępność.

JTP

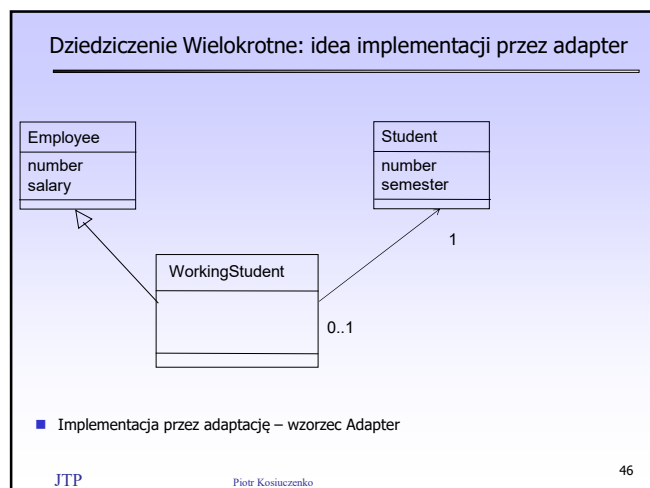
Piotr Kosieczko

44

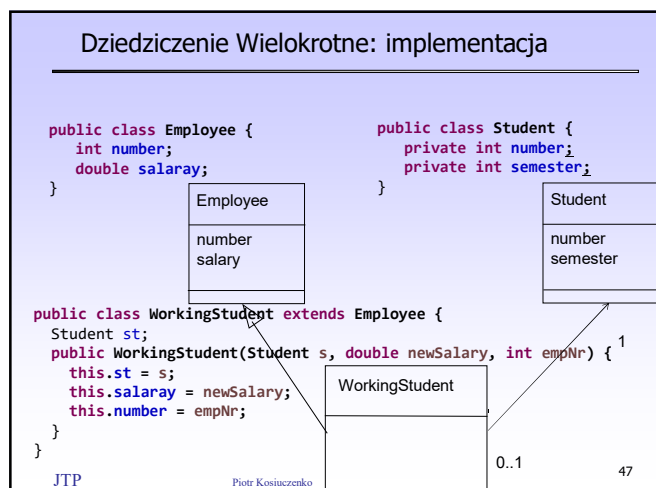
44



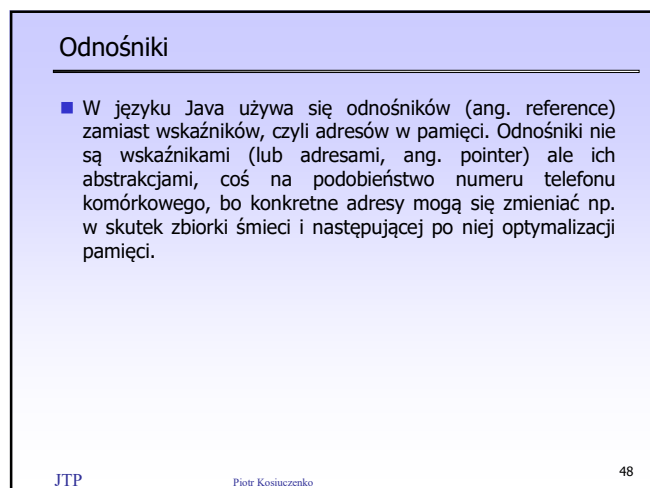
45



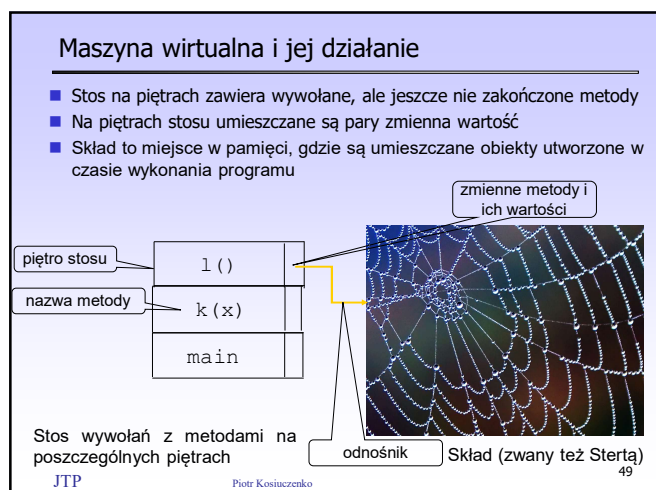
46



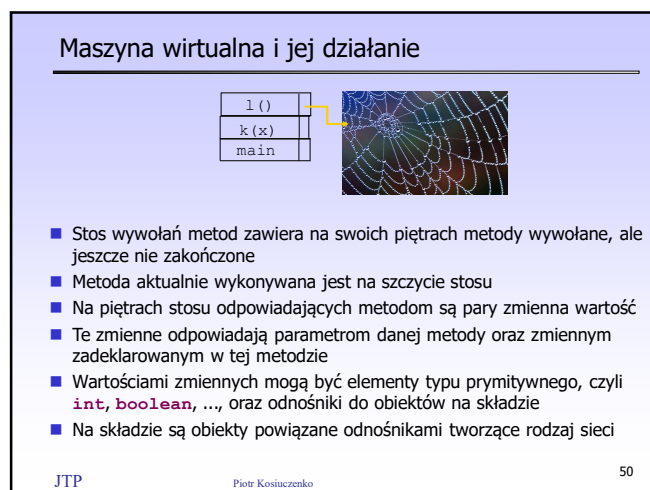
47



48



49



50

Działanie maszyny wirtualnej: przykład

```
public class T {  
    public static void l() {  
        int w = 2;      //L  
    }  
    public static void k(int x) {  
        { int y = 3;      //K1  
          y = y + ++x;    //K2  
        }                //K3  
        l();              //K4  
    }  
    public static void main(String[] args){  
        int x = 5;        //A  
        k(x);              //B  
    }  
}
```