Języki i Techniki Programowania

dr hab. Piotr Kosiuczenko
profesor WAT

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).

2

Literatura

- Cay Horstmann, Java: Podstawy, Helion, 2008 i następne wydania.
- 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.

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]

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-dotyczce-jzyka-java
- Bardzo szczegółowe pytania z interview: https://www.javatpoint.com/corejava-interview-questions

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=PL9HfA4ZKbzin-tNeJi09vJxll_RiOn9eB
- W bibliotece WAT są odpowiednie książki.
- Trzeba jednak uważać na alternatywną terminologię.

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

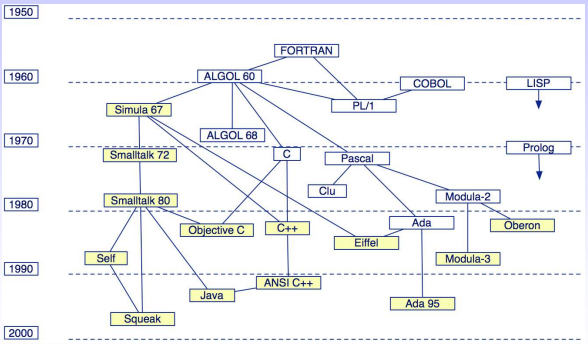
7

Jezyki Obiektowo Zorientowane

- Simula rok 1967;
- Języki oparte na Lisp koniec lat 70ych;
- Smalltalk rok 1980;
- C++, Eiffel połowa 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]

Historia Jezyków Programowania



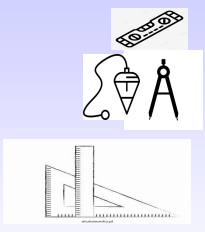
Porownanie: Javy, C++ i Smalltalka

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

Odpowiedz

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


JTP: Podstawy języka Java



dr hab. Piotr Kosiuczenko
prof. WAT

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++.

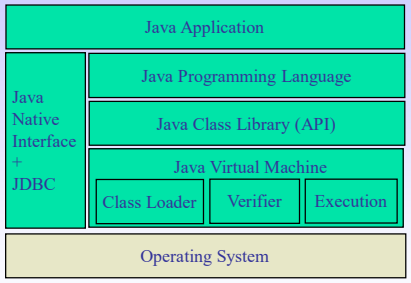
Pryncypia języka Java

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

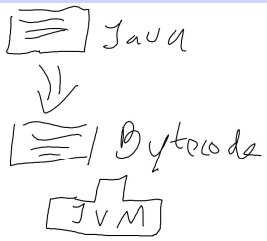
Pryncypia języka Java: JVM

- 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/klasy i piaskownice (sandboxing).

Java: Struktura Systemu



Java: Bytecode and JVM



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.

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.

Java: Hello World

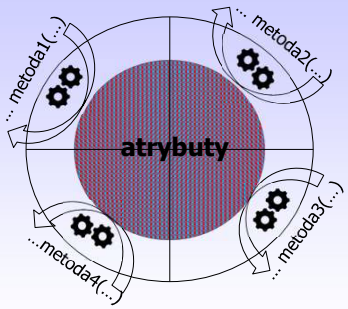
```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

deklaracja klasy; sygnatura metody `main`; metoda statyczna – działa na klasie; parametr domyślny/implicite; korpus/ciało metody; parametr explicite

Statyczna metoda działa na klasie, a nie na obiektach.

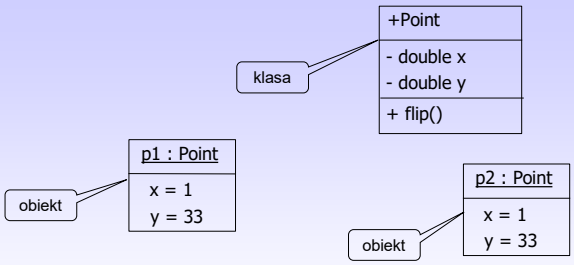
metoda obiektowa – działa na obiekcie będącym wartością parametru implicite

Konstytutywne 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.

Java: Klasy i Obiekty



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

Java: Przekazywanie Wartości Parametrów

- W Jawie 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 wywoływaną metodę nie zmienia wartości zmiennych w metodzie wywołującej
- W konsekwencji, metoda wywołana nie zmienia wartości zmiennych, które zostały przekazane do wywołanej metody jako parametry.

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

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

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]

Java: Czytanie Danych z Konsoli

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

importowanie odpowiednich klas

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

utwórz obiekt skanujący

obiekt czyta dane z konsoli

parsowanie danych

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).

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.

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.

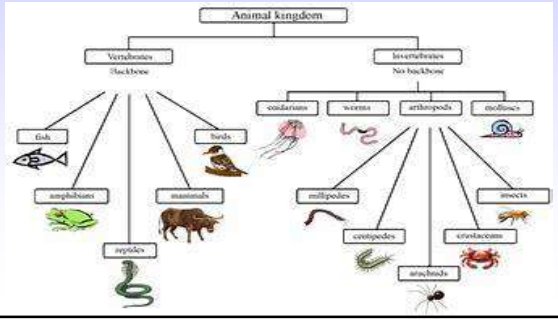
Java: Klasy Konkretne

Podstawową jednostką programistyczną jest **klasa** (słowo kluczowe `class`). Zawiera ona atrybuty klasowe (class attributes) i obiektowe (object attributes) oraz metody klasowe i obiektowe.

- Klasy deklarują atrybuty klasowe i obiektowe, oraz metody klasowe i obiektowe (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 obiektowe, 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).

Java: Klasy Abstrakcyjne

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



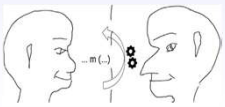
Java: Klasy Abstrakcyjne cd.

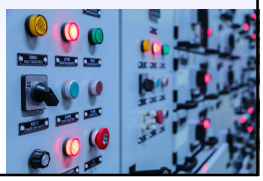
- Może implementować wiele interfejsów ale dziedziczy własności co najwyżej jednej klasy (także abstrakcyjnej).
- Posiada atrybuty i metody.
- Metody nie musza być zaimplementowane (słowo kluczowe **`abstract`**).
- Metoda abstrakcyjna musi być zaimplementowana przez dziedziczącą klasę konkretną.

Java: Interfejsy

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

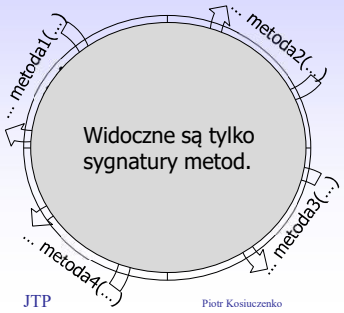
- uniezależnia klientów systemu od jego implementacji.
- zawierają zasadniczo sygnatury, ewentualnie implementacje metod oraz tzw. stałe (niezmienne atrybuty statyczne definiowane z poziomu interfejsu), ale nie atrybuty obiektowe.
- może rozszerzać tylko inne interfejsy (może być ich wiele), ale nie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





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.

Java: Klasy Wewnętrzne

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

Dziedziczenie: zasada Liskov

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

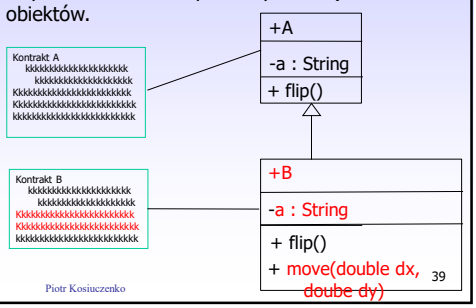
Obiekty podklasy muszą dać się użyć wszędzie tam, gdzie mogą być użyte obiekty nadklasy.

To znaczy w szczególności:

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

Zasada Liskov – dokładniejsze sformułowanie

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



39

Dziedziczenie: mechanizmy wiązania

- W Javie podklasa może na nowo zdefiniować atrybuty i metody nadklasy, choć te dwa przypadki odpowiadają dwu różnym mechanizmom wiązania.
- Istnieją zasadniczo dwa mechanizmy wiązania:
  - statyczne - 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.

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.

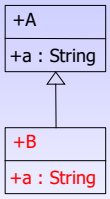
Wynikiem będzie: "Attribute of A";

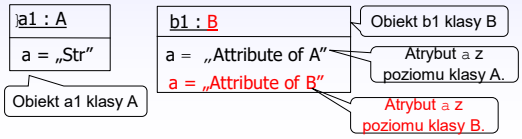
Dlaczego?

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;
  . . .
```





Obiekt a1 klasy A

Obiekt b1 klasy B

Atrybut a z poziomu klasy A.

Atrybut a z poziomu klasy B.

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.

43

Dziedziczenie: Co będzie wynikiem kompilacji?

```
class A {
  public A a;
}

class B extends A {
   private A a;
}
```

Wynikiem będzie powstanie Bytecodu.

Dlaczego?

```
class A1 {
  public void m(){
  }
}

class B1 extends A1 {
   private void m(){
  }
}
```

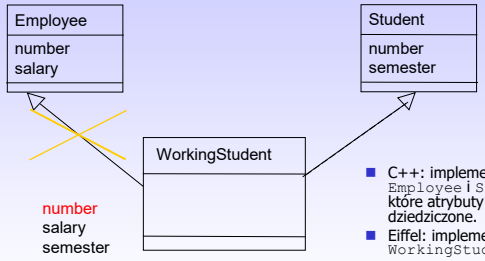
Wynikiem będzie komunikat o błędzie kompilacji.

Dlaczego?

Bo wiązanie metody m z klasy B1 ogranicza jej dostępność.

44

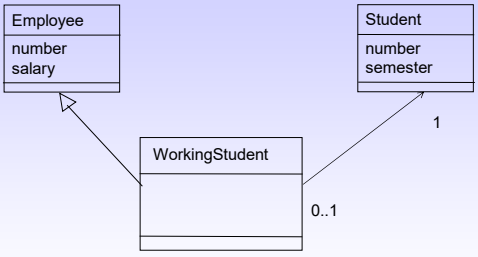
Wielokrotne Dziedziczenie



- C++: implementatorzy klas `Employee` i `Student` decydują, które atrybuty i metody będą dziedziczone.
- Eiffel: implementator `WorkingStudent` o tym decyduje.
- Java: nie ma wielokrotnego dziedziczenia klas;
- jest za to wielokrotne dziedziczenie interfejsów.

45

Dziedziczenie Wielokrotne: idea implementacji przez adapter



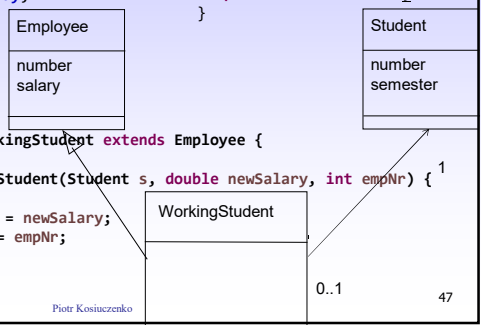
- Implementacja przez adaptację – wzorzec Adapter

46

Dziedziczenie Wielokrotne: implementacja

```
public class Employee {
   int number;
   double salaray;
}
```

```
public class Student {
   private int number;
   private int semester;
}
```




```
public class WorkingStudent extends Employee {
  Student st;
  public WorkingStudent(Student s, double newSalary, int empNr) {
    this.st = s;
    this.salaray = newSalary;
    this.number = empNr;
  }
}
```


47

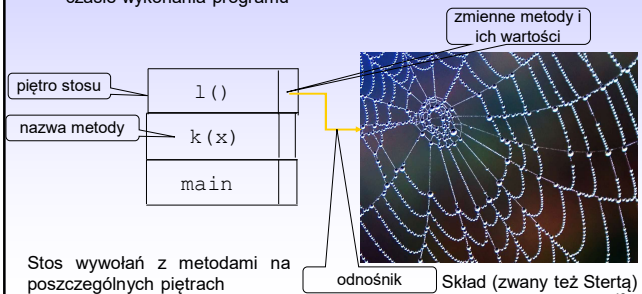
Odnośniki

- W języku Java używa się odnośników (ang. reference) zamiast wskaźników, czyli adresów w pamięci. Odnośniki nie są wskaźnikami (lub adresami, ang. pointer) ale ich abstrakcjami, coś na podobieństwo numeru telefonu komórkowego, bo konkretne adresy mogą się zmieniać np. w skutek zbiorki śmieci i następującej po niej optymalizacji pamięci.

48

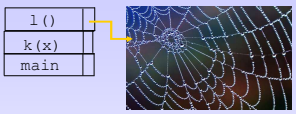
Maszyna wirtualna i jej działanie

- Stos na piętrach zawiera wywołane, ale jeszcze nie zakończone metody
- Na piętrach stosu umieszczane są pary zmienna wartość
- Skład to miejsce w pamięci, gdzie są umieszczane obiekty utworzone w czasie wykonania programu



49

Maszyna wirtualna i jej działanie



- Stos wywołań metod zawiera na swoich piętrach metody wywołane, ale jeszcze nie zakończone
- Metoda aktualnie wykonywana jest na szczycie stosu
- Na piętrach stosu odpowiadających metodom są pary zmienna wartość
- Te zmienne odpowiadają parametrom danej metody oraz zmiennym zadeklarowanym w tej metodzie
- Wartościami zmiennych mogą być elementy typu prymitywnego, czyli `int`, `boolean`, ..., oraz odnośniki do obiektów na składzie
- Na składzie są obiekty powiązane odnośnikami tworzące rodzaj sieci

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
  }
}
```


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
  }
}
```

$[(args, ...), (x, 5)]_{main}$	A
$[(args, ...), (x, 5)]_{main}\ [(x, 5), (y, 3)]_k$	K1
$[(args, ...), (x, 5)]_{main}\ [(x, 6), (y, 9)]_k$	K2
$[(args, ...), (x, 5)]_{main}\ [(x, 6)]k$	K3
$[(args, ...), (x, 5)]_{main}\ [(x, 6)]k\ [(w, 2)]_l$	L
$[(args, ...), (x, 5)]_{main}\ [(x, 6)]k$	K4
$[(args, ...), (x, 5)]_{main}$	B

wykonywany kod — kolejne stany stosu wywołań metod

Działanie maszyny wirtualnej : przykład cd.

odnośnik do obiektu

$[(args, ...), (x, 5)]_{main}$	A
$[(args, ...), (x, 5)]_{main}\ [(x, 5), (y, 3)]_k$	K1
$[(args, ...), (x, 5)]_{main}\ [(x, 6), (y, 9)]_k$	K2
$[(args, ...), (x, 5)]_{main}\ [(x, 6)]k$	K3
$[(args, ...), (x, 5)]_{main}\ [(x, 6)]k\ [(w, 2)]_l$	L
$[(args, ...), (x, 5)]_{main}\ [(x, 6)]k$	K4
$[(args, ...), (x, 5)]_{main}$	B

Pary w nawiasach okrągłych zawierają zmienną i jej wartość. Ta wartość jest albo typu prymitywnego, np. 5, albo jest odnośnikiem (zaznaczone przez „…”).

Maszyna wirtualne: **zadania domowe**

1. Zasymuluj działanie maszyny wirtualnej w przypadku metody **move** klasy **Line** (zobacz implementację na laboratoriach).

   Użyj notacji tekstowej do zapisu zawartości stosu (zobacz zadanie powyżej i zadanie z laboratorium).

2. Zastanów się, co to znaczy przejść do następnego polecenia w przypadku poleceń postaci **if else**, oraz **while** i **for**.

JTP: Klonowanie

dr hab. Piotr Kosiuczenko
prof. WAT

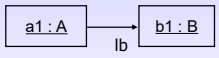
Klonowanie

- Jeśli klasa implementuje interfejs `Cloneable`, to jest możliwe klonowanie za pomocą standardowej metody `clone()`.
- Klasa `Object` zawiera metodę `protected Object clone()`.
- Metoda `clone()` z klasy `object` zwraca wyjątek, jeśli obiekt, na którym została wykonana, należy do klasy, która nie implementuje interfejsu `Cloneable`.
- Jeśli obiekt należy do klasy implementującej `Clonable`, to metoda ta zwraca nowy obiekt, którego atrybuty są identyczne z atrybutami klonowanego obiektu.
- W przypadku tablicy i wektora zwracana/y jest nowa tablica/nowy wektor, których elementami są elementy klonowanej tablicy/wektora.

Klonowanie

- `clone()` może być przedefiniowana w podklasach a jej dostępność rozszerzona do publicznej:

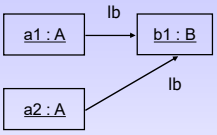
```
protected Object clone() throws CloneNotSupportedException
```



- Co będzie wynikiem klonowania obiektu `a1`?
- W Javie mamy płytkie klonowanie.
- Czasem głębsze klony są potrzebne.

61

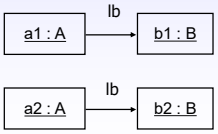
Klonowanie: płytkie i głębokie



Kopiujemy wartości prymitywne i odnośniki

W Javie jest zaimplementowany tylko płytki sposób klonowania

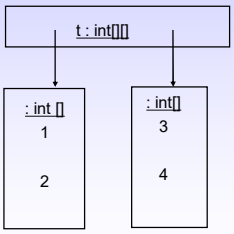
Kopiujemy wartości prymitywne i klonujemy zlinkowane obiekty



62

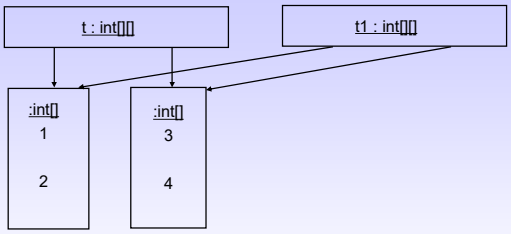
Klonowanie: tablice - problem z klonowaniem płytkim

```
int[][] x =  {{1, 2}, {3, 4}};
```



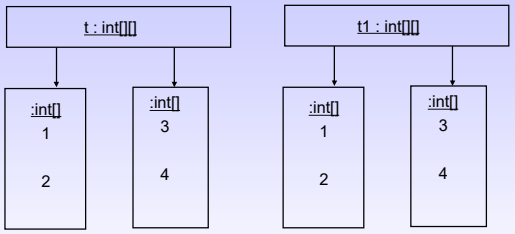
63

Tablice: klonowanie płytkie, problem



64

Klonowanie: tablice – klonowanie głębokie



65

Klonowanie

- Co się stanie gdy metoda `clone()` zostanie wykonana w następującym przypadku?

```
public class A implements Cloneable {
  private A a = new A();
  public Object clone() throws
   CloneNotSupportedException {
    A x = (A)super.clone();
    if(this.a != null)
      x.a = (A)a.clone();
    return x;
  }
}
```


66

Klonowanie

- Co się stanie gdy metoda `clone()` zostanie wykonana w następującym przypadku?

```
public class A implements Cloneable {
  private A a = new A();
  public Object clone() throws
   CloneNotSupportedException {
    A x = (A)super.clone();
    if(this.a != null)
      x.a = (A)a.clone();
    return x;
  }
}
```


Klonowanie

- Co się stanie gdy metoda `clone()` zostanie wykonana w następującym przypadku?

```
public class Child extends Parent {
  public Object clone () throws
        CloneNotSupportedException {
    Child x = (Child)super.clone();
  }
}
```

Rzucony zostanie wyjątek, bo metoda sklonuje this jako obiekt klasy `Parent`.

Klonowanie

- Co się stanie gdy metoda `clone()` zostanie wykonana w następującym przypadku?

```
int[][] x = {{1, 2}, {3, 4}};
int[][]  y = x.clone();
```


API: Klasa Object, Kolekcje

dr hab. Piotr Kosiuczenko
prof. WAT

API: Klasa `Object`

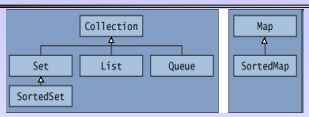
Klasa `Object` jest korzeniem całego systemu klas i zawiera metody używane przez wszystkie inne klasy, min:

- `protected Object clone()` zwraca nową kopię danego obiektu.
- `boolean equals(Object obj)` zwraca `true` jeśli dany obiekt jest równy aktualnemu obiektowi; standardowo działa jako porównanie odnośników. Powinna być przedefiniowana w podklasach.
- `Class<?>getClass()` zwraca klasę obiektu, klasa jest ustalana w czasie wykonania (dokładniej zwracany jest obiekt tę klasę reprezentujący).
- `void notify()` budzi pewien inny czekający wątek.
- `void notifyAll()` budzi wszystkie inne wątki czekające na monitor.
- `String toString()` zwraca napis reprezentujący dany obiekt, powinna być przedefiniowana w podklasie.
- `void wait()` każe wątkowi czekać, aż inny wątek wywoła metodę `notify()` lub `notifyAll()`.
- ...

API: Interfejs `Collection`

- W Javie, kolekcje są obiektami grupującymi inne, zwykle jednorodne, elementy w jedną całość.
- Kolekcje mają trzy zasadnicze elementy:
  - Interfejs, czyli abstrakcyjny typ danych pozwalający na manipulowanie kolekcja niezależnie od implementacji,
  - Implementację, z reguły ponownie używalną, oraz
  - Algorytmy, które operują na nich jak np. wyszukiwanie lub sortowanie.

API: Collection



- `Collection<E>` jest parametrycznym interfejsem i zarazem korzeniem hierarchii interfejsów kolekcji.
- Implementujące podklasy: `LinkedList`, `Stack`, ...
- `boolean add(E e)` dodaje element `e` do aktualnej kolekcji.
- `void clear()` usuwa wszystkie elementy z aktualnej kolekcji.
- `boolean contains(Object o)` sprawdza, czy `o` jest elementem aktualnej kolekcji.
- `int size()` zwraca liczbę będąca długością aktualnej kolekcji.
- ...

74

API: Collection

```
Collection<Employee> employees;
...
for(Employee e : employees) {
  e.salary = e.salary + amount;
}
/*zapis alternatywny*/
for(ListIterator<Employee> it =
  list.iterator(); it.hasNext(); ) {
      Employee employee = it.next();
      ...
  }
```

Forma zwarta, ale z ograniczeniami

To trzeba wyspecyfikowac.

Forma ogólna z użyciem iteratora, pozwalająca min. na wstawianie I usuwanie obiektow.

75

API: SET

- `Interface Set<E>`
- Jest to kolekcja bez powtórzeń – zbiór (ang. set) w sensie matematycznym.
- Type Parameters:
  - E - the type of elements maintained by this set
- All Superinterfaces:
  - Collection<E>, Iterable<E>
- All Known Subinterfaces:
  - NavigableSet<E>, SortedSet<E>
- All Known Implementing Classes:
  - AbstractSet, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet, JobStateReasons, LinkedHashSet, TreeSet

76

API: List

`List<E>` jest parametrycznym interfejsem implementującym interfejs `Collection`.

```
public interface List<E> extends Collection<E> {
    E get(int index); // Positional access
    E set(int index, E element);    //optional
    boolean add(E element);         //optional
    void add(int index, E element); //optional
    E remove(int index);            //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional
    int indexOf(Object o); // Search
    int lastIndexOf(Object o);
    ListIterator<E> listIterator(); // Iteration
    ListIterator<E> listIterator(int index);
    List<E> subList(int from, int to); // Range-view
}
```


77

API: LinkedList

- `LinkedList<E>` jest parametryczna klasa implementująca interfejs `List<E>`.
- Ta klasa implementuje także `Dequeue` interfejs pozwalający na użycie `LinkedList` jako stosu i jako kolejki.

78

Przykład

dr hab. Piotr Kosiuczenko
prof. WAT

79

Przykład: Konto w Banku

Zaimplementować konta bankowe z takimi informacjami jak: nazwisko, numer klienta, stan konta.

80

Przykład: Konto w Banku

```
public class BankAccount {
  private static int numberOfAccounts = 0;
  private String name;
  private final int number;
  private double balance = 0;
  public BankAccount(double amount) {
    number = ++numberOfAccounts;
    balance = amount;
  }
  ...
  public void credit(double amount) {
    balance = balance + amount;
  }
  public void debit(double amount) {
    balance = balance - amount;
  }
```

atrybuty

konstruktor

metody

81

Przykład: Konto w Banku

```
  public void transfer(BankAccount target, double amount) {
    this.debit(amount);
    target.credit(amount);
  }
  public String toString() {
    return "[name = " + name + ", number = " + number + ",
   balance = " + balance + "]";
  }
}
```

Metoda `toString()` jest dziedziczona z klasy `Object`, ale powinna być przedefiniowana.

82

Przykład: Konto w Banku

```
  public void credit(double amount) {
    balance = balance + amount;
  }
  public void debit(double amount) {
    balance = balance - amount;
  }
```

A co będzie gdy `amount < 0`?
To jest sytuacja wymagająca działania.

83