Programowanie wieloplatformowe w Java

dr Krzysztof Podlaski

Kwestie organizacyjne

Wykład: 15 godzin

- ~~Zaliczenie test~~

Ćwiczenia 30 godzin (dr K.Podlaski)

- Zaliczenie projekt

Konsultacje: do ustalenia,

Kontakt:

pokój 331 (III poziom)
email: podlaski (at) uni.lodz.pl
github: http://github.com/kpodlaski
tel: (42) 665 56 52

Treść wykładów

1. Krótki wstęp, dotyczący historii języka i zastosowań
2. Java - język obiektowy.
3. Tworzenie programów, kompilowanie ich i wykonywanie, tworzenie dokumentacji.
4. Podstawowe konstrukcje i instrukcje sterujące.
5. Dziedziczenie i Polimorfizm
6. Inicjalizacja klas, czyszczenie po wykorzystanych obiektach.
7. Wyjątki w Języku Java i ich obsługa.
8. Wykrywanie typów RTTI (Metoda refleksji)
9. Kolekcje obiektów
10. Systemy wejścia-wyjścia
11. Wielowątkowość
12. Tworzenie Appletów, oprogramowanie GUI

Dlaczego Java

„Modny” Język

- Wieloplatformowość
  - » Java Virtual Machine
- Era internetu
- Łatwość programowania ?
- Programowanie dużych projektów
  - » Just In Time
- Wiele specjalistycznych bibliotek
- Prostota tworzenia dokumentacji

Dlaczego Java cd.

- Zastosowania
  - Aplikacje wieloplatformowe
  - Applety dla stron www
  - JSP, Servlety (nowoczesne cgi), Portlety
  - Serwisy Webowe
  - Telefony komórkowe - w wersji **Android**
  - Programowanie sieciowe (czat, wspomaganie systemów CMS)

  Wykorzystanie zasobów jak Bazy Danych, pliki XML, innych technologii informatycznych

Trochę historii

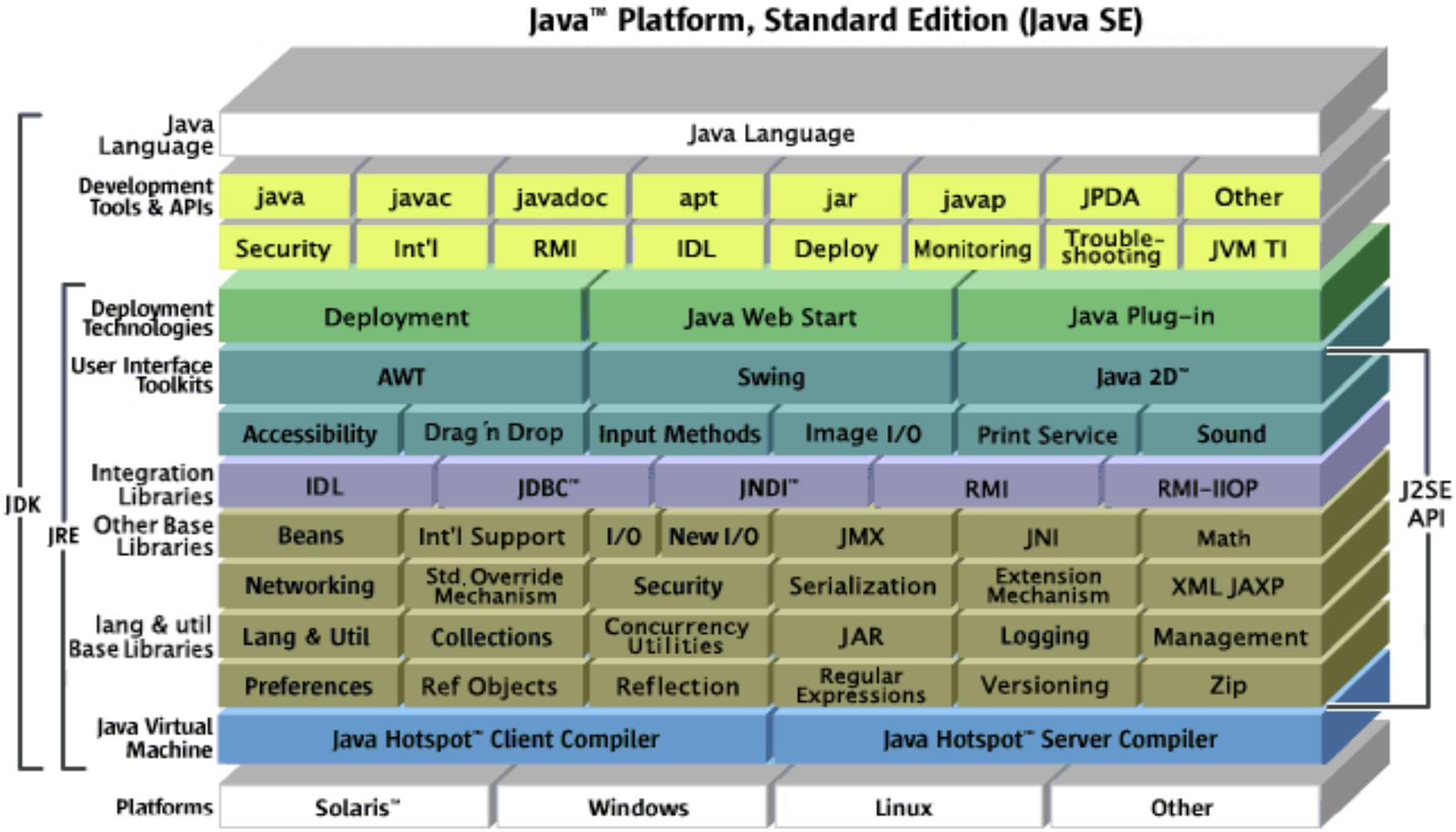
James Gosling, Sun Microsystems

- Projekt „Oak” czerwiec 1991
  - WORA - Write Once, Run Everywhere
  - Język obiektowy prostszy niż c++
- Pierwsza implementacja Java 1.0, 1995
- Nowa Java 2 (Java 1.2, 1.3, 1.4, 1.5)
- Aktualnie Java 1.9 (Java 2 SE 9.0)

Język Java Dzisiaj

- J8SE (Java 8/9 Standard Edition)
  - JDK (Java Developement Kit)
    - J8RE (Java 8/9 Runtime Environment)
      - Plug-in dla przeglądarek
- J7EE (Java 7 Enterprise Edition)
- JME (Java 2 Micro Edition)
    - Zastąpiona przez Davlik == Android

JDK


Java™ Platform, Standard Edition (Java SE)
Java Language
Java Language
Development Tools & APIs
java
javac
javadoc
apt
jar
javap
JPDA
Other
Security
Int'l
RMI
IDL
Deploy
Monitoring
Trouble-shooting
JVM TI
Deployment Technologies
Deployment
Java Web Start
Java Plug-in
User Interface Toolkits
AWT
Swing
Java 2D™
Accessibility
Drag 'n Drop
Input Methods
Image I/O
Print Service
Sound
Integration Libraries
IDL
JDBC™
JNDI™
RMI
RMI-IIOP
Other Base Libraries
Beans
Int'l Support
I/O
New I/O
JMX
JNI
Math
Networking
Std. Override Mechanism
Security
Serialization
Extension Mechanism
XML JAXP
lang & util Base Libraries
Lang & Util
Collections
Concurrency Utilities
JAR
Logging
Management
Preferences
Ref Objects
Reflection
Regular Expressions
Versioning
Zip
Java Virtual Machine
Java Hotspot™ Client Compiler
Java Hotspot™ Server Compiler
Platforms
Solaris™
Windows
Linux
Other
JDK
JRE
J2SE API

JDK - narzędzia

- javac – kompilowanie plików źródłowych
- java – JVM
- appletviewer – przeglądarka appletów
- jdb – debuger javy
- javadoc – generator dokumentacji w formacie HTML
- jar – archiwizator wieloplatformowy

Po Pomoc

- Strony Oracle
  - www.oracle.com, java.oracle.com, www.java.com
- Strony niezależne
  - www.google.pl
  - www.onjava.com, today.java.net, www.javalobby.org
- Alternatywne wersje Javy
  - www-128.ibm.com/developerworks/java/jdk/
  - Android - Dalvik

Literatura

- **Paul Dietel, Harvey Dietel, Java How to Program, Early Objects, 10th edition, Pearson 2014**
  - **Jest już wersja 11 z 2017 roku**
- **Bruce Eckel, Thinking in Java, wyd. 4, Helion 2006 (wyd. 3, 2003)**
    - **Dostępne w wersji angielskiej (wyd. 3) www.mindview.net/Books/TIJ/**
- **Boone Barry, Java dla programistów C i C++, WNT, 1998.**

Własności języka Java

Obiektowy język programowania

- Wszystko jest obiektem
  - » Prawie wszystko
- Podobieństwo do języka C++
  - » Podobna składnia
  - » Uproszczona obsługa pamięci
  - » Garbage Collector
  - » Ścisła kontrola błędów (wyjątki)

Proces tworzenia programu

- Pomysł
- Implementacja w języku java
  - Plik .java
- Kompilacja
  - Kod pośredni (java bytecode) .class
- Wykonanie – Wirtualna Maszyna Javy
  - Wykonanie kodu pośredniego
  - Kompilacja Just in Time

Dlaczego kod pośredni

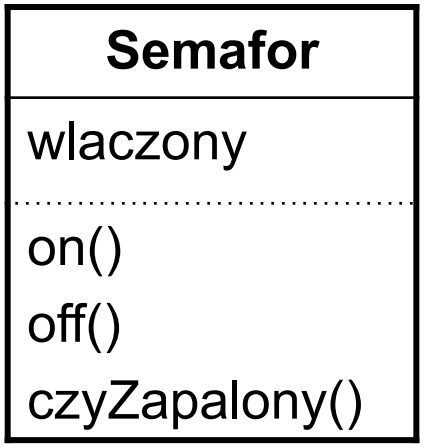
- Wieloplatformowość
  - Odpowiedzialność zrzucona na JVM
- Bezpieczeństwo i ochrona praw autorskich
  - Dystrybucja tylko plików .class
  - Trudność w „odkodowaniu” programu
    - » obfusyfikacja kodu
- Ułatwienie pracy w zespole
  - Kompilacja JiT

Programowanie Obiektowe

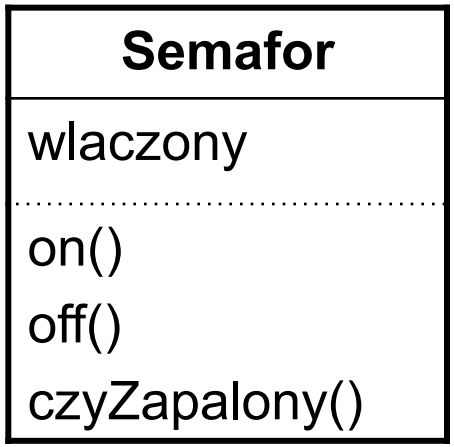
- Dwa światy
  - Problem do rozwiązania (abstrakcja problemu)
  - Specyfika maszyny (abstrakcja implementacji)

Semafor:
własności:
• stan: (zapalony, zgaszony)
czynności:
• włączanie (zapal, zgaś)
• sprawdzanie stanu

Klasa:



- Przykładowa implementacja




```
class Semafor
{
   boolean wlaczony;
   void on() {wlaczony = true;}
   void off() {wlaczony = false;}
   boolean czyZapalony {return wlaczony;}
}
```

Paradygmat obiektowości

- Wszystko jest obiektem
- Program – zbiór obiektów komunikujących się ze sobą
- Każdy obiekt ma określony typ (klasa)
- Obiekty tej samej klasy używają takich samych metod komunikowania się

Wszystko jest obiektem

- Cały program Javy jest obiektem !!
  - Definiujemy obiekt

```
public   class NazwaKlasy
         {
                  .....   Pola i metody
         }
```

- Sterowanie dostępem public, private, protected, package access

Wszystko jest obiektem cd.

- Obiekt posiada:
  - Pola – stan obiektu
    - » Pole może również być obiektem
  - Metody – komunikowanie się ze „światem”
  - Tożsamość

- Każdy obiekt musi zostać utworzony przed pierwszym użyciem

  instrukcja new

Wszystko jest obiektem cd.

- Klasa publiczna -> NazwaKlasy.java
  - Tylko 1 klasa publiczna w pliku o tej samej nazwie
  - Inne klasy – ograniczony dostęp
- Kompilacja -> pliki NazwaKlasy.class
  - 1 klasa jeden kod pośredni NazwaKlasy.class
  - Każda klasa ma swój plik .class
- Uruchomienie programu
  - Klasa publiczna
  - Posiada jedną metodę publiczną main

No prawie wszystko ☺

- Wygoda programistów
  - Typy proste
    - boolean (true, false)
    - char (znak Unicode) 16 bit
    - byte (całkowite [-127, 127]) 8bit
    - short, int, long (całkowite) 16, 32, 64 bit
    - float, double 32, 64 bit
    - void

- Do każdego z typów klasy opakowujące (Wrappers)
  Boolean, Character, Byte, Short, Integer, Long, Float, Double, Void

Podział Pamięci

- Rejestr
  - „Wewnątrz procesora”
  - Brak dostępu i kontroli
- Stos
  - Bezpośredni dostęp dla procesora
  - Mało miejsca
  - Określone przez kompilator czas życia i wielkość
- Sterta
  - Przyznawane dynamicznie
- Obszar statyczny
- Obszar stałych

Obiekt a typ prosty

- Obiekty
  - Tworzone poprzez new
  - Umieszczone na stercie
    - » Dużo miejsca
    - » Wolny dostęp
- Typy podstawowe
  - Tworzone w trakcie deklaracji
  - Umieszczane na stosie
    - » Mało miejsca
    - » Szybki dostęp
    - » Określony czas życia

Zwyczaje w nazewnictwie

- Nazwy Klas zaczynamy z Dużej Litery
  - Hello, HelloJava, Prostokat, ZbiorProstokatow
- Metody i pola z małej litery
  - on(), off(), wlaczony, pokazWartosc()
- Nazwa klasy/pola/metody zaczyna się od litery znaku _ lub $.

Pakiety w Javie

- Unikatowa nazwa
  - Zwyczajowo pochodne nazw domen
    - Programista domena kowalski.net
    - pakiet zawiera Narzędzia
    - nazwa pakietu
      - » net.kowalski.Narzedzia
  - Pakiet w katalogu
    - net/kowalski/Narzedzia/

Pakiety w Javie

- Przykładowa klasa
  - MacierzOdwrotna.class
    - na początku pliku linijka:
      - » package net.kowalski.Narzedzia;
    - net/kowalski/Narzedzia/MacierzOdwrotna.class
  - import net.kowalski.Narzedzia.MacierzOdwrotna;
  - uruchamianie programów
    - java net.kowalski.Narzedzia.NazwaKlasy

Składnia

- Komentarze
  - /* komentarz wielowierszowy
    jak w c++
    */
  - // komentarz jednolinijkowy
  - /** komentarz wykorzystywany przez javadoc
    automatyczne tworzenie dokumentacji
    */

- Zmienne podstawowe:

```
double   epsilon=2.7;
boolean prawda=true, falsz=false;
```

  - Całkowite
    2, -322, 0123, 0x23f, 0X7A3, 15L
  - Logiczne
    true, false
  - Znakowe
    ‘a’, ‘\n’, ‘\u00ff’, ‘\077’
  - Tekstowe
    ”Hello\n”

Operatory

- Przypisania =
- Arytmetyczne

  +,-,*,/,%
- Inkrementacja, dekrementacja

  ++,--
- Porównywania

  >,<,>=,<=,==,!=
- Logiczne

  &&, ||, ^ (xor), ! (not)

  &,|

Operatory cd.

- Przypisanie złożone

  +=, -=

- Łączenie łańcuchów znaków

  +

Brak przeładowywania operatorów znanego z C++

Operacje rzutowania

- Operacja zmiany typu
  - Dla typów podstawowych
    - Automatyczne
      - Bezstratne
    - Wymuszone
  - Dla Obiektów
    - na String (metoda toString())

```
int i=7;
float k=i;
```

```
int i; char c='a';
i=(int) c;
```

Nie ma rzutowania na typ boolean !!

Instrukcje sterujące

- Instrukcja warunkowa:

```
if ( warunek ) { ....}
else {....}
```

```
if ( warunek ) { ....}
else if (warunek 2) {....}
else {....}
```

Pętle

- for

```
for (int i=0; i<10; i++)
{ ....}
```

- while

```
while (warunek)
{ ....}
```

- do while

```
do
....
while (warunek)
```

Przerwanie pętli

- modyfikatory
  - return
  - break
  - continue
    - etykieta

1 - wyjście z pętli for

2 – następna iteracja for

3 – następna iteracja while

4 – koniec obu pętli

```
etykieta:
while(true){
   for(;true;) {
            ....
            break;              //1
            continue;           //2
            continue etykieta;  //3
            break etykieta;     //4
   }.....
}
```

Wybór z wielu opcji

- switch, case

```
switch (zmienna)
 {
        case warość1:  instrukcje; break;
        case wartość2: instrukcje; break;
        .......
        default : instrukcje;
}
```

zmienna - byte, short, char, int, String, Enum

Klasy

```
dostęp class NazwaKlasy
{
        dostęp typ nazwaPola;
        dostęp typ nazwaMetody (typ1 argument1,  typ2 argument2)
        {
                        ciało metody
        }
}
```

dostęp – private, public, protected, nic czyli package access.

- Konstruktor - dla każdej klasy istnieje specjalna metoda

  dostęp NazwaKlasy (argumenty) {...}
- W Javie brak destruktora
  - Garbage Collector
  - Metoda finalize()
- Metody statyczne
  - public void static main(String[] args){...}

Używanie metod obiektów

```
class Test
{
    private int wynik=0;
    public int jakiWynik { return wynik;}
    public void ustawWynik(int w)
            { wynik = w; }
}
```

Zastosujmy tę klasę jako pole innej

```
class Student
{
    Test fizyka=new Test();
    public Student(int _wynik)
            {
             fizyka.ustawWynik(_wynik);
            }
}
```

Proste przykłady

- Przykład z kilkoma klasami – Przykład 1_1
- Kompilowanie pliku z wieloma klasami
    - Dla każdej klasy plik .class
    - Klasy wewnętrzne

Atrybuty pól i metod

- Atrybuty
  - static
    - Z metod statycznych dostęp TYLKO do pól statycznych
  - final
    - typy podstawowe ←→ stała
    - obiekty ←→ stała referencja

Przeładowanie metod

```
class Student
{
        private byte rok;
        private String nazwisko;
        private String pesel;
        public Student(String nazw, String npesl)
        { rok =1; nazwisko=nazw; pesel=npesl;}
        public Student(String nazw, String npesl, int _rok)
        {nazwisko=nazw; pesel=npesl; rok =_rok;}
}
```

Inicjalizacja obiektów

- Kolejność inicjalizacji elementów obiektu

  Przykład 1_2

  – Początek programu
    – Inicjalizacja elementów statycznych
  – Deklaracja zmiennych
  – Inicjalizacja zmiennych
    – Inicjalizacja pól
    – Uruchomienie konstruktora
- Typy podstawowe inicjowane przy deklaracji.

Operacje na obiektach

- Przyrównywanie obiektów

  Przykład 1_3

  - operator ==
  - metoda equals()

- Rzutowanie obiektu na String

  Przykład 1_4

  - metoda toString()

Tablice

- Typy proste vs Obiekty klas

```
int[ ] tab = {1, 2, 4, 6};
```

```
int tab[ ] = {1, 2, 4, 6};
```

```
int[ ] tab = new int[3];
....
i[0]=1; i[1]=2; i[2]=13;
```

```
class Element {...};
```

```
Element[ ] tab = new Element[3];
....
tab[0]=new Element();
tab[1]=new Element();
```

Tablice cd.

- Własności
  - Stała wielkość
  - Dane jednego typu
    - » Można to obejść
  - Czas dostępu
    - » Jednakowy czas dostępu

Projektowanie

- Projektowanie
  - Wymyślanie struktury klas
  - Programowanie klas, metod
  - Testowanie
- Nowy projekt
  - Zaczynamy od zera ?
  - NIE
- Ponowne wykorzystanie Klas
  - API (***A****pplication* ***P****rogramming* ***I****nterface*)

Powtórne użycie klasy

- Klasa bazowa

Semafor
wlaczony
on() off() czyZapalony()

- Klasa pochodna

Trasa
Semafor[ ]
stanSemafora(int) wlaczSemafor(int) wylaczSemafor(int) polozenieSemafora(int) ktorySemafor(int)

Powtórne użycie klasy cd.

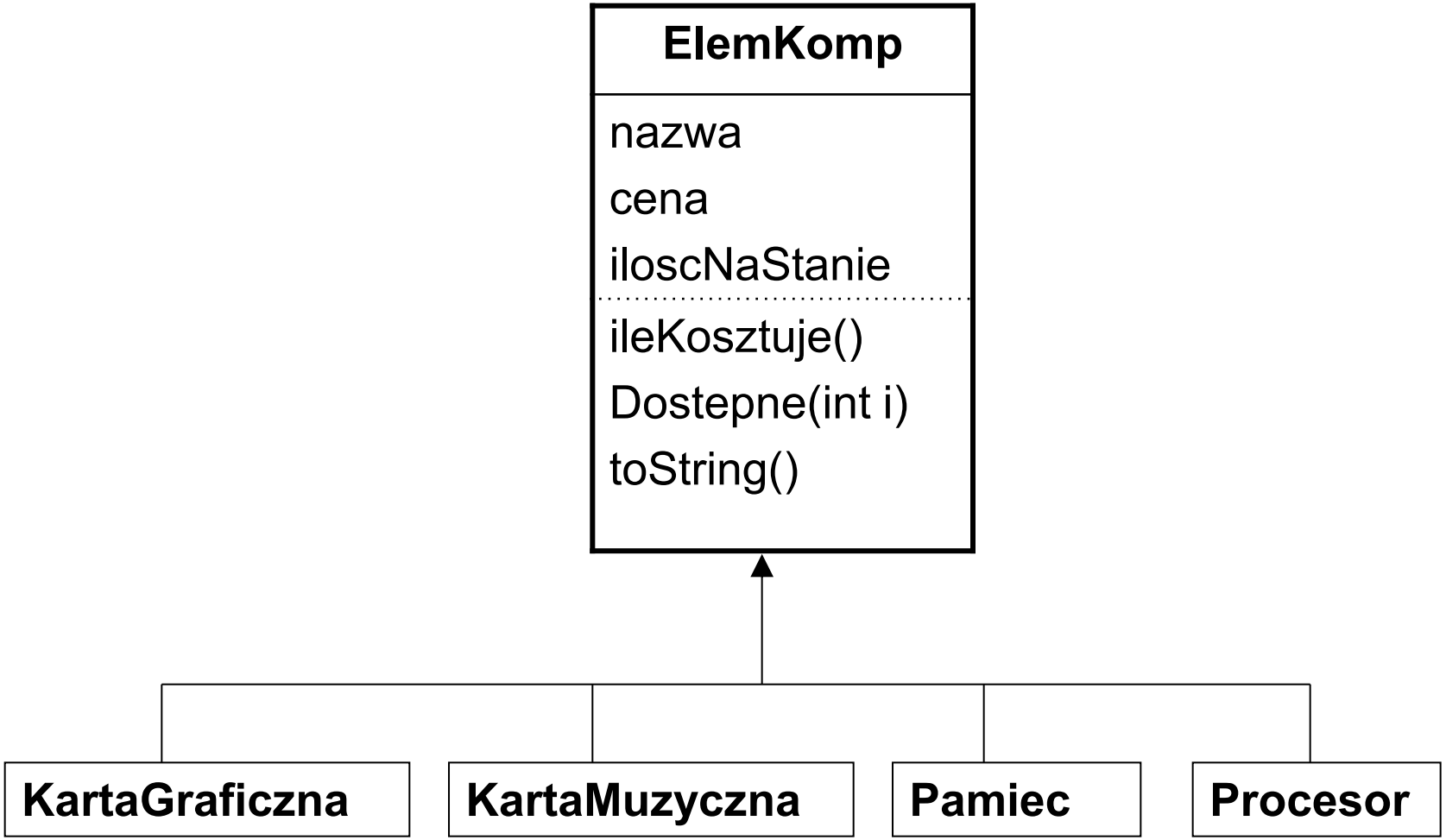
- Klasa pierwotna

ElemKomp
nazwa cena iloscNaStanie
ileKosztuje() Dostepne(int i) toString()

•Klasa pochodna

KartaGraficzna
nazwa cena iloscNaStanie rodzajZlacza iloscPamieci
ileKosztuje() Dostepne(int i) toString()

Dziedziczenie



Dziedziczenie cd.

- Dziedziczenie
  - Zachowanie struktury
  - Przeładowanie metod
    - » ostrożnie z metodami prywatnymi
  - Klasa ma TYLKO jedną Klasę przodka

Przykład

```
class Ksztalt
{
    public rysuj(int x, int y){...}
    public skasuj(int x, int y){...}
}
```

```
class Kwadrat extends Ksztalt
{
    public Kwadrat(int, int, int);
    public rysuj() {...}
    public skasuj() {...}
}
```

```
class Trojkat extends Ksztalt
{
    public Trojkat(int, int, int);
    public rysuj() {...}
    public skasuj() {...}
}
```

```
class Kolo extends Ksztalt
{
    public Kolo(int, int, int);
    public rysuj() {...}
    public skasuj() {...}
}
```