

Stefan Sokołowski

PROGRAMOWANIE OBIEKTOWE II

wykład 2

Inst. Informatyki Stosowanej, ANS
Elbląg, 2024/2025

Wykład 2, str. 1

Jak cofnąć licznik

... albo jeszcze — metody w klasie **mogą** zmieniać własności prywatne:

```
class Pojazd {  
    ...  
    void przejazd(double czas, double predkosc) {  
        if (czas < 0) błąd else {  
            czas_jazdy += czas; przebieg += czas*predkosc;  
        }  
    }  
    ...  
}
```

```
System.out.println(samochod.przebieg());  
samochod.przejazd(2,81);  
System.out.println(samochod.przebieg());  
samochod.przejazd(0.5,50);  
System.out.println(samochod.przebieg());
```

← drukuje 0

← drukuje 162

← drukuje 187

Jeśli to są jedyne nieprywatne metody w klasie **Pojazd**, to zewnętrzny użytkownik nie ma możliwości zmiany stanu licznika innej niż przez *przejazd*.

I BARDZO DOBRZE!

Podsumowanie o klasach i ich polach



Program w Javie jest podzielony na *klasy*. Klasa składa się z *pól*. Polami mogą być deklaracje

- *własności*,
- *metod*,
- *podklas* (czyli klas wewnętrznych).

W programie nie może być „gołych” własności, ani metod. Każda musi być ubrana w jakąś klasę.

Własności *w zasadzie* powinny być prywatne. Specjalne metody do operowania na nich z zewnątrz klasy:

- *getter* — do odczytywania własności
- *setter* — do zapisywania własności

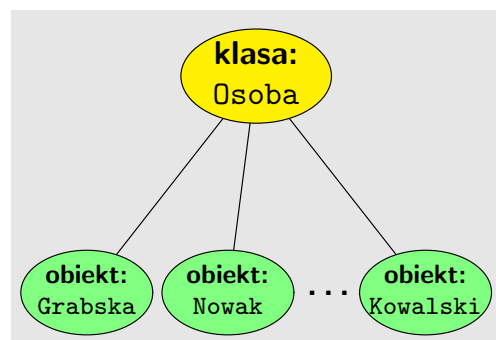


Podsumowanie o klasach i ich polach

Klasa — *w zasadzie* wzorec budowy *obiektów*.

Pola obiektu — wartości lub metody o nazwie i typie takim, jak zadeklarowane w klasie.

Ich wartości — *dowolne* w ramach typu. Klasa opisuje wiele różniących się od siebie obiektów.

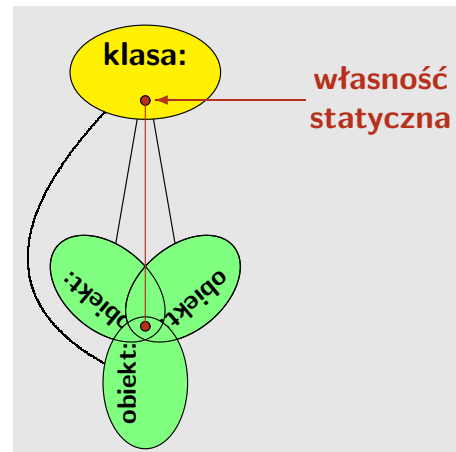
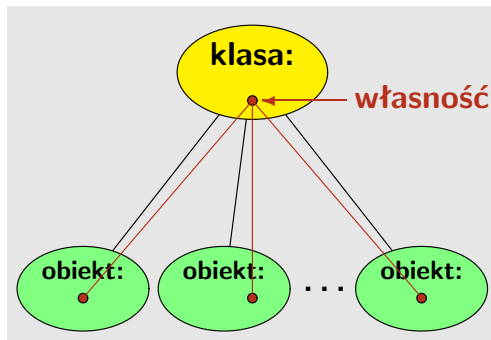


Możemy ograniczyć tę dowolność i ustalić, że jakieś pole ma mieć tę samą wartość we wszystkich obiektach klasy. Deklaracja:

`static nazwa_typu nazwa_własności_lub_metody`

Pole statyczne — takie samo we wszystkich obiektach klasy.

Statyka



Niestatycznej własności klasy odpowiadają **odrębne** wartości w każdym obiekcie tej klasy.

Statycznej własności klasy odpowiada **ta sama** wartość w każdym obiekcie tej klasy.

Statyka

```
class Punkt {
    double wsp_x, wsp_y;
    static int ile;
    Punkt (double x, double y) {
        ile++; wsp_x = x;  wsp_y = y;
    }
}

class Punkty {
    public static void main(String[] args) {
        Punkt pt0 = new Punkt(0,0),
        pt1 = new Punkt(1,0),
        pt2 = new Punkt(0,1);
        System.out.println("Ile punktow: " + pt0.ile);
    }
}
```

← drukuje: 3

Każde wywołanie konstruktora `Punkt` zwiększa o 1 **wspólną** dla całej klasy zmienną `ile`.

Statyka

```
class Punkt {
    double wsp_x, wsp_y;
    static int ile;
    Punkt (double x, double y) {
        ile++; wsp_x = x; wsp_y = y;
    }
}

class Punkty {
    public static void main(String[] args) {
        Punkt pt0 = new Punkt(0,0),
        pt1 = new Punkt(1,0),
        pt2 = new Punkt(0,1);
        System.out.println("Ile punktow: " + Punkt.ile);
    }
}
```

← drukuje: 3

Zamiast pytać o statyczną własność jakiegoś obiektu możemy pytać o wspólną własność klasy.

Statyka

Dostęp do własności statycznej w ogóle nie wymaga konstruowania obiektu:

```
class Punkt {
    double wsp_x, wsp_y;
    static int ile;
    Punkt (double x, double y) {
        ile++; wsp_x = x; wsp_y = y;
    }
}

class Punkty {
    public static void main(String[] args) {
        System.out.println("Ile punktow: " + Punkt.ile);
    }
}
```

← drukuje: 0

```
double sinus_45_stopni = Math.sin(45*Math.PI/180);
```

- `Math.sin` — statyczna metoda `sin` (czyli *sinus*) w klasie `Math`
- `Math.PI` — statyczna własność `PI` (czyli π) w klasie `Math`

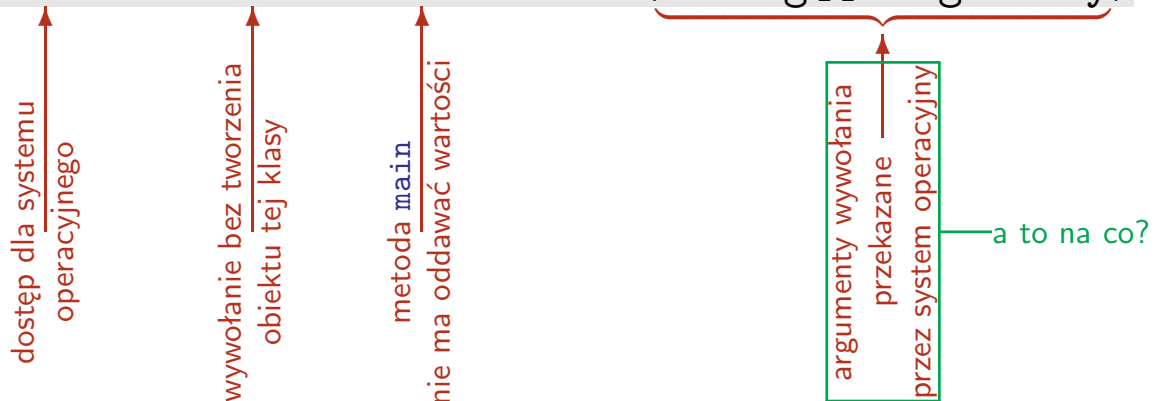
Wykonanie programu

Użytkownik: `java nazwa_głównej_klasy`

System operacyjny wywołuje metodę: `nazwa_głównej_klasy.main`

Wobec tego główna klasa programu **musi** zawierać deklarację metody o takim nagłówku:

```
public static void main (String[] argumenty)
```



Dane z liniiki rozkazowej

Przykład: (konwerter dziesiętno-szesnastkowy)

```
class P0_DecHex {
    static int maxDlHexa = 20;
    public static void main (String[] args) {
        if (args.length != 1) System.out.println(
            "\n Argumentem ma byc jedna liczba dziesiętna!\n"
        );
        else {
            long n = Long.parseLong(args[0]);
            int dl = 0;
            char[] hex = new char[maxDlHexa];
            while (dl < maxDlHexa && n > 0) {
                int reszta = (int)(n%16); n = n/16;
                if (reszta < 10) hex[dl] = (char)(reszta+'0');
                else hex[dl] = (char)(reszta-10+'A');
                dl++;
            }
            if (n > 0) System.out.println("\n Za duza liczba!\n");
            else {
                System.out.print(" ");
                for (int i=dl-1; i>=0; i--)
                    System.out.print(hex[i]);
                System.out.println();
            }
        }
    }
}
```

*Pobieranie danych
bez wczytywania:*

Kompilacja:

`javac P0_DecHex.java`

Wywołanie:

`java P0_DecHex liczba`

A jeśli opuścimy któreś zakłęcie...

- `public static void main (String[] argumenty)`

Błąd wykonania:

Error: Main method not found in class ...

please define the main method as:

```
public static void main(String[] args)
```

System operacyjny nie widzi metody `main`, bo to jest „inny” `main`.

- ~~public~~ `static void main (String[] argumenty)`

Błąd wykonania:

Error: Main method not found in class ...

System operacyjny nie widzi metody `main`. SO nie należy do tego samego *pakietu*.

- `public static void main (String[] argumenty)`

Błąd wykonania:

Error: Main method is not static in class ...

`main` nie jest statyczny. SO nie może wykonać `main` — bez wskazania obiektu można wykonać tylko metodę statyczną.

Dlaczego?!

Dlaczego bez wskazania obiektu można wywołać *tylko* metodę *statyczną*?

```
class Zle {
    int x;
    void qq( ) { ← metoda niestaticzna
        System.out.println(x);
    }
}
```

```
class P1_Test {
    public static void main(String[ ] param) {
        Zle zle1 = new Zle( ), zle2 = new Zle( );
        zle1.x = 1; zle2.x = 2;
        Zle.qq( ); ← z którego obiektu pochodzi qq()?
    }                                     drukować 1 czy 2?
}
```

Błąd kompilacji:

error: non-static method qq() cannot be referenced from a static context

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog02/P1-Statyka/P1.Test.java

Dlaczego?!

Dlaczego z metody statycznej można sięgać *tylko* do zmiennej *statycznej*?

```
class Zle {
    int x;
    static void qq( ) {
        System.out.println(x);
    }
}

class P1_Test {
    public static void main(String[ ] param) {
        Zle zle1 = new Zle( ), zle2 = new Zle( );
        zle1.x = 1;  zle2.x = 2;
        Zle.qq( );
    }
}
```

własność niestatyczna (pointing to `int x;`)

drukować x z którego obiektu? (pointing to `System.out.println(x);`)

Błąd kompilacji:

error: non-static variable x cannot be referenced from a static context

iis.ans-elblog.pl/~stefan/Dydaktyka/Prog0biIII/Prog02/P1-Statyka/P2-Test.java

Statyczny świat

```
class Dobrze {
    static int x;
    static void qq( ) {
        System.out.println(x);
    }
}

class P1_Test {
    public static void main(String[ ] param) {
        Dobrze dobrze1 = new Dobrze( ),
        dobrze2 = new Dobrze( );
        dobrze1.x = 1;  dobrze2.x = 2;
        Dobrze.qq( );
    }
}
```

Ten program jest OK, drukuje 2.

iis.ans-elblog.pl/~stefan/Dydaktyka/Prog0biIII/Prog02/P1-Statyka/P3-Test.java

Dziedziczenie (*inheritance*)

X jest obiektem klasy A

oznacza

$X \in A$

????????????????????

oznacza

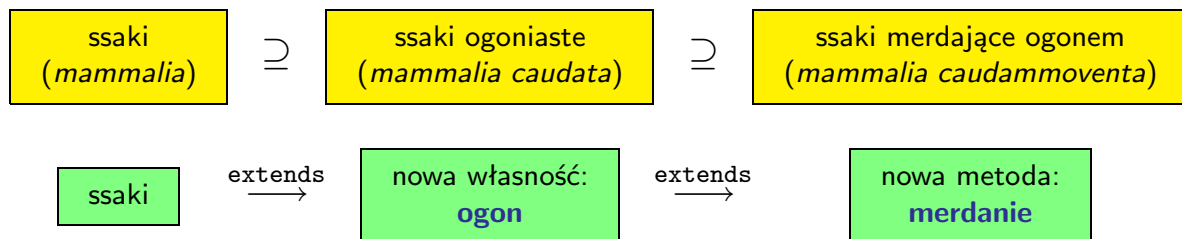
$A \subseteq B$

W C typy są rozłączne, więc **nie ma** zawierania typów.

W **Jawie** istnieje odpowiednik *zawierania* zbiorów.

np. 12
to nie to samo
co 12.0

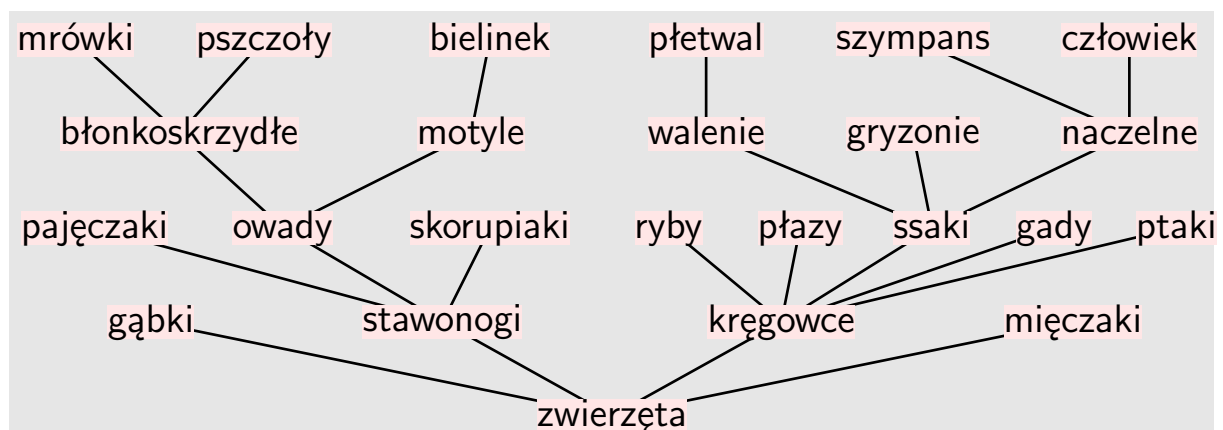
Im więcej wymagań tym mniejszy zbiór:



Każdy *ssak ogoniasty* jest *ssakiem*.

Każdy *ssak merdający* jest *ssakiem ogoniastym*.

Systematyka zwierząt

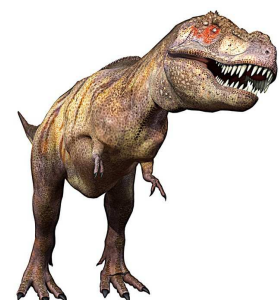


Im wyżej w drzewie,

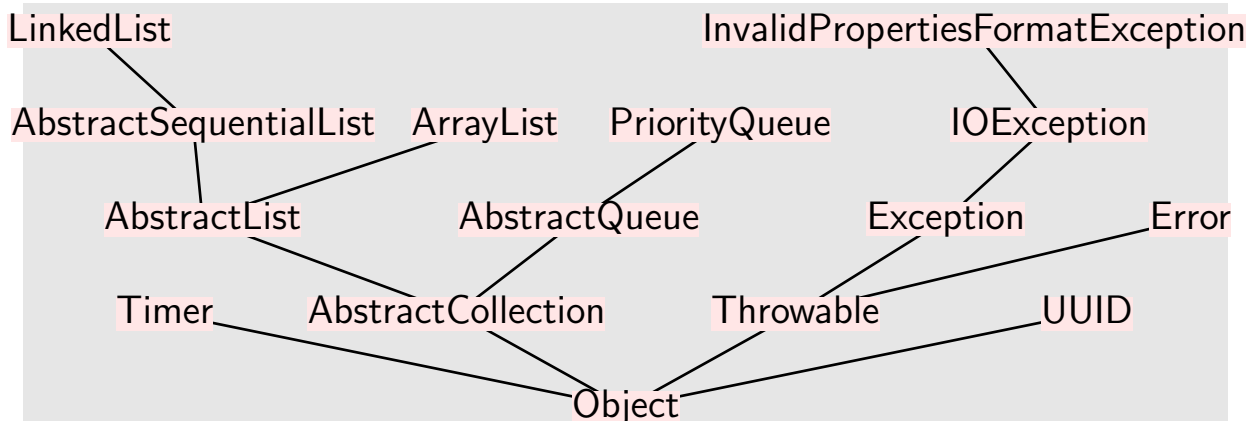
- tym mniejszy zbiór,
- tym więcej cech szczególnych.

Uwaga:

To jest tylko malutki fragment drzewa;
w dodatku niekonsultowany z biologami.



Systematyka klas w java.util



Im wyżej w drzewie,

- tym mniejszy zbiór,
- tym więcej cech szczególnych (**extends** prowadzi w górę).

Uwaga:

To jest tylko mały fragment drzewa;
w dodatku zestawiony niestarannie.

Dziedziczenie (*inheritance*)

```
class A extends B {
    ...
}
```

- klasa A automatycznie zawiera wszystkie **publiczne** własności i metody klasy B ;
- wewnątrz klamry $\{ \dots \}$ można określić dodatkowe własności i metody, charakterystyczne dla klasy A ;
- w ciele **konstruktora** klasy A na ogół wywołuje się konstruktor klasy B przez słowo kluczowe **super**, potem wykonuje się operacje „przerabiające” obiekt klasy B na obiekt klasy A .

Czy należy do rozszerzenia typu

element instanceof podklasa

```
class Zwierze {
    public String gatunek;
    public Zwierze(String gat) {
        gatunek = gat;
    }
}

class Ssak extends Zwierze {
    public Ssak(String gat) {
        super(gat);
    }
}

class P1_Test {
    public static void main
        (String[ ] args)
        . . .
}
```

```
Zwierze karp = new Zwierze("karp"),
    mucha = new Zwierze("mucha"),
    aligator = new Zwierze("aligator");

Ssak pies = new Ssak("pies"),
    kot = new Ssak("kot"),
    osiol = new Ssak("osiol");

ArrayList<Zwierze> zwierzaki =
    new ArrayList<Zwierze>( );

zwierzaki.add(aligator);
zwierzaki.add(karp);
zwierzaki.add(kot);
zwierzaki.add(mucha);
zwierzaki.add(osiol);
zwierzaki.add(pies);
for (Zwierze z : zwierzaki)
    if (z instanceof Ssak)
        System.out.println(z.gatunek);
```

iis.ans-elblog.pl/~stefan/Dydaktyka/ProgObiII/Prog02/P2_Dziedzicz/P1_Test.java

Dziedziczenie (inheritance)

```
class Pracownik {
    private String nazw; private int pensja;
    public Pracownik(String nazw, int pensja) {
        this.nazw = nazw; this.pensja = pensja;
    }
    public String nazw( ) { return this.nazw; }
    public int pensja( ) { return this.pensja; }
    public void nowaPensja(int pensja) { this.pensja = pensja; }
}

class Szef extends Pracownik {
    private ArrayList<Pracownik> podwladni;
    public Szef(Pracownik prac) {
        super(prac.nazw( ), prac.pensja( ));
        podwladni = new ArrayList<Pracownik>( );
    }
    public ArrayList<Pracownik> podwladni( ) {
        return podwladni;
    }
    public void nowyPodwladny(Pracownik prac) { podwladni.add(prac); }
}
```

← własności prywatne
 ← konstruktor
 ← selektory
 ← metoda
 ← dodatkowa własność prywatna
 ← nowy konstruktor
 ← dodatkowy selektor
 ← dodatkowa metoda

iis.ans-elblog.pl/~stefan/Dydaktyka/ProgObiII/Prog02/P2_Dziedzicz/P2_Kadry.java