

# Język Java i technologie WEB



dr inż. Andrzej Czajkowski  
Instytut Sterowania i Systemów Informatycznych  
Wydział Informatyki, Elektrotechniki i Automatyki

# Plan Wykładu

- 1 Wprowadzenie
- 2 Java – wprowadzenie
- 3 Składnia i struktura programów
- 4 Typy danych

dr inż. Andrzej Czajkowski

e-mail: [a.czajkowski@issi.uz.zgora.pl](mailto:a.czajkowski@issi.uz.zgora.pl)

pokój: 325 A-2

tel: +68 328 2276

WWW: <http://staff.uz.zgora.pl/aczajkow/>



# Warunki zaliczenia

- **Wykład** - warunkiem zaliczenia jest uzyskanie pozytywnej oceny z egzaminu przeprowadzonego w formie pisemnej.
- **Laboratorium** - warunkiem zaliczenia jest uzyskanie pozytywnych ocen ze wszystkich ćwiczeń laboratoryjnych, przewidzianych do realizacji w ramach programu laboratorium.
- **Metody weryfikacji** - wykład: egzamin w formie pisemnej - laboratorium: sprawdzian praktyczny.
- **Składowe oceny końcowej** = wykład: 50% + laboratorium: 50%
- **Udogodnienia**: Termin zerowy na ostatnim wykładzie, zwolnienie na podstawie oceny z laboratorium (ocena **bdb**).

# Literatura

## Literatura obowiązkowa:

- 1 Java Platform, Standard Edition, API Documentation  
<https://docs.oracle.com/javase>
- 2 Bruce Eckel, Thinking in Java, Wydanie 4, Helion, 2011.  
<http://mindview.net/Books/TIJ4>
- 3 Benjamin J Evans, David Flanaga, Java w pigułce. Wydanie VI

## Literatura dodatkowa:

- 1 J . Gosling, B. Joy, G. Steele, G. Bracha, Java Language Specification, Addison-Wesley Professional.
- 2 Andrzej Marciniak, JavaServer Faces i Eclipse Galileo. Tworzenie aplikacji WWW, Helion 2010.
- 3 Cay S. Horstmann, Java 8. Przewodnik doświadczonego programisty.

# Najważniejsze skróty

- J2SE
- J2EE
- JDK
- JRE
- JVM
- JIT
- JAR
- AWT

# Java - historia

- Java jest wysokopoziomowym, kompilowanym, obiektowym językiem programowania z **silną kontrolą typów**.
- Stworzony przez grupę roboczą pod kierunkiem **Jamesa Goslinga** z firmy Sun Microsystems (obecnie przejęty przez Oracle).
- **Główne założenie**: "write once, run anywhere" (WORA)
- **Podstawowe koncepcje** zostały przejęte z języka Smalltalk (maszyna wirtualna, zarządzanie pamięcią) oraz z języka C++ (duża część składni i słów kluczowych).

# Wersje

- JDK 1.0 (January 21, 1996)
- JDK 1.1 (February 19, 1997)
- J2SE 1.2 (December 8, 1998)
- J2SE 1.3 (May 8, 2000)
- J2SE 1.4 (February 6, 2002)
- J2SE 5.0 (September 30, 2004)
- Java SE 6 (December 11, 2006)
- Java SE 7 (July 28, 2011)
- Java SE 8 LTS (March 18, 2014)
- Java SE 9 (September, 2017)
- Java SE 10 (March, 2018)
- Java SE 11 LTS (September, 2018)
- Java SE 12 (March, 2019)



# Mój pierwszy program

```
package wyklad1;

public class WitajSwiecie {
    public static void main(String[] args)
    {
        System.out.println("Witaj Swiecie");
    }
}
```

- nazwa pakietu
- nazwa klasy
- metoda statyczna
- wyświetlanie tekstu na konsoli

Zadanie domowe:

Zainstalowanie JDK i IDE (Eclipse, NetBeans lub IntelliJ) i uruchomienie powyższego programu.

# Założenia języka

- 1 It must be "simple, object-oriented, and familiar".

**Prosty** (ang. simple) – łatwy do nauczenia w bardzo krótkim czasie, zwłaszcza dla programistów znających C++ – **znajomy**, (ang. familiar).

**Zorientowany obiektowo** (ang. object-oriented) – projektowanie obiektowe jest techniką, która kładzie nacisk na dane (obiekty) oraz interfejsy do nich.

# Założenia języka

- 1 It must be "simple, object-oriented, and familiar".
- 2 It must be "robust and secure".

**Odporny** (ang. robust) – poświęcony jest duży nacisk na wczesne sprawdzanie możliwych błędów (podobnie jak w C++), sprawdzanie dynamiczne w trakcie pracy oraz eliminowanie sytuacji które mogą skłaniać do wystąpienia błędu.

**Bezpieczny** (ang. secure) – Java została zaprojektowana jako język dla systemów sieciowych/rozproszonych, co spowodowało duży nacisk na problem bezpieczeństwa

# Założenia języka

- 1 It must be "simple, object-oriented, and familiar".
- 2 It must be "robust and secure".
- 3 It must be "architecture-neutral and portable".

**Neutralny** (ang. architecture neutral) – kompilator generuje instrukcje w kodzie bajtowym, który jest wykonywalny na wielu maszynach pod warunkiem obecności maszyny wirtualnej Javy. Instrukcje w kodzie bajtowym nie mają nic wspólnego ze szczególną architekturą komputera – **przenośny** (ang. portable).

# Założenia języka

- 1 It must be "simple, object-oriented, and familiar".
- 2 It must be "robust and secure".
- 3 It must be "architecture-neutral and portable".
- 4 It must execute with "high performance".

**Wysoce wydajny** (ang. high performance) Proces właściwego generowania kodu maszynowego z kodu bajtowego jest bardzo prosty, gdyż format kodu bajtowego został zaprojektowany z myślą o optymalizacji kodu maszynowego. Najbardziej kontrowersyjny aspekt platformy Java, w nowszych wersjach platformy założenie osiągnięte poprzez takie zaawansowane mechanizmy jak JIT czy współbieżne odśmiecanie pamięci.

# Założenia języka

- 1 It must be "simple, object-oriented, and familiar".
- 2 It must be "robust and secure".
- 3 It must be "architecture-neutral and portable".
- 4 It must execute with "high performance".
- 5 It must be "interpreted, threaded, and dynamic".

**Interpretowany** (ang. interpreted) – kod bajtowy Javy (Java bytecode) jest tłumaczony na bieżąco na kod maszynowy danego komputera (interpretowany). Kompilacja kodu występuje tylko raz, natomiast interpretacja zawsze gdy tylko program jest uruchamiany.

**Wielowątkowy** (ang. multithreaded) – możliwość wykorzystania mechanizmu implementacji wątków i synchronizacji zasobów aby program mógł być wykonywany w tym samym czasie na różnych rdzeniach i procesorach.

**Dynamiczny** (ang. dynamic) – język Java został zaprojektowany tak, aby adaptować się do rozwijających się środowisk.

|   | Java   | SmallTalk | ICL   | Perl    | C      | C++    |
|---|--------|-----------|-------|---------|--------|--------|
| prosty ( simple)  | ■      | ■         | ■     | ■       | ■      | □      |
| zorientowany obiektowo<br>(object oriented)             | ■      | ■         | □     | ■       | □      | ■      |
| solidny ( robust)                                       | ■      | ■         | ■     | ■       | □      | □      |
| bezpieczny (secure)                                     | ■      | ■         | ■     | ■       | □      | □      |
| interpretowany ( interpreted)                           | ■      | ■         | ■     | ■       | □      | □      |
| dynamiczny ( dynamic)                                   | ■      | ■         | ■     | ■       | □      | □      |
| przenośny ( portable)                                   | ■      | ■         | ■     | ■       | ■      | ■      |
| neutralny ( neutral)                                    | ■      | ■         | ■     | ■       | □      | □      |
| wątki ( threads)  | ■      | □         | □     | ■       | □      | □      |
| automatyczne zwalnianie<br>pamięci (garbage collection) | ■      | ■         | □     | □       | □      | □      |
| wyjątki ( exceptions)                                   | ■      | ■         | □     | ■       | □      | ■      |
| wydajność ( performance)                                | wysoka | średnia   | niska | średnia | wysoka | wysoka |

Realizacja atrybutu:

pełna



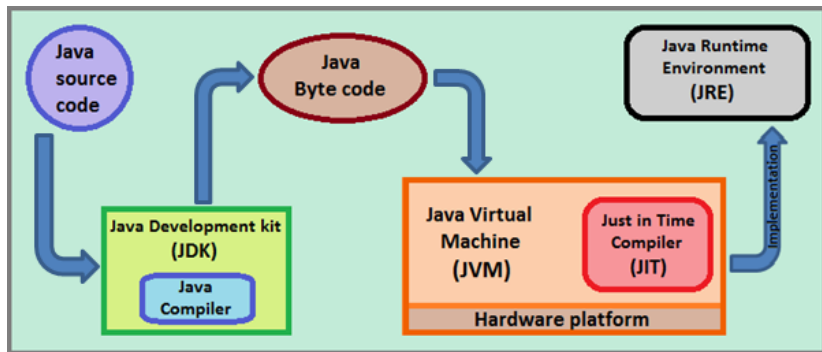
częściowa



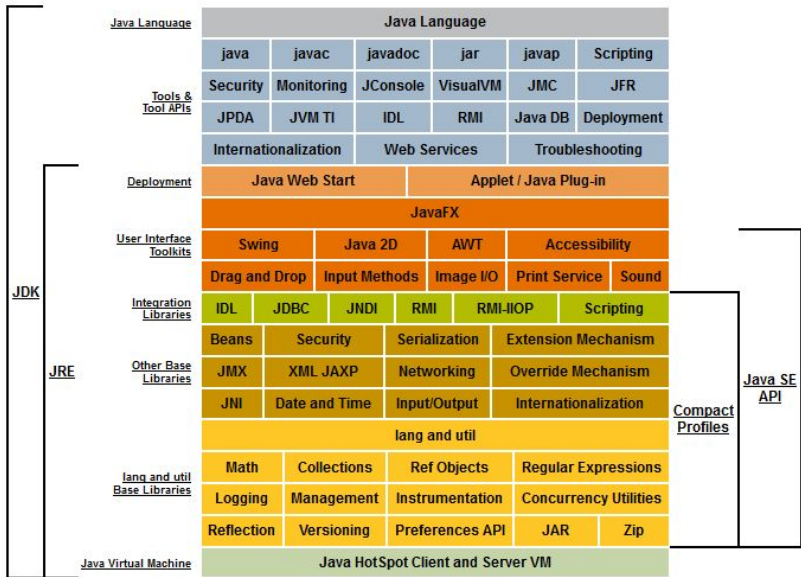
brak



# JRE, JDK i JVM



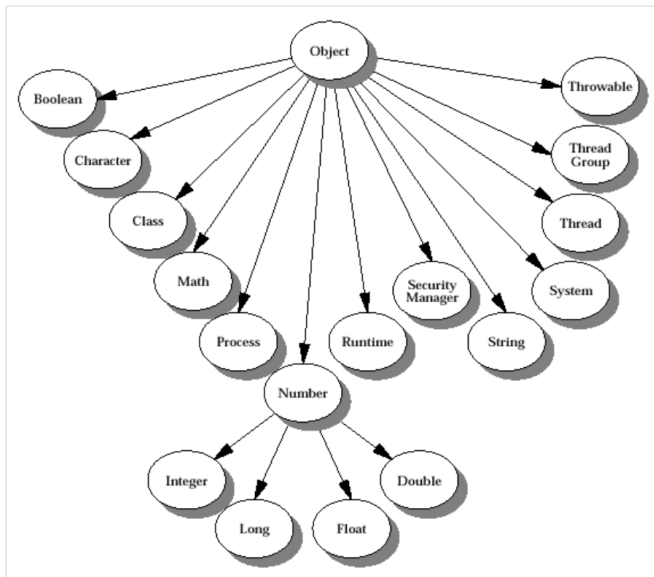




# Biblioteki Javy

- Podstawowa dystrybucja Javy (J2SE) to prawie cztery tysiące klas zgrupowanych w około dwustu pakietach.
- Najistotniejsze pakiety:
  - ❶ `java.lang`  
Pakiet zawiera zbiór typów bazowych (typów języka) które są zawsze importowane do dowolnego kompilowanego kodu. Tam można znaleźć deklarację `Object` (korzeń hierarchii klas) i `Class`, oraz wątków, wyjątków, podstawowych typów danych oraz fundamentalnych klas.

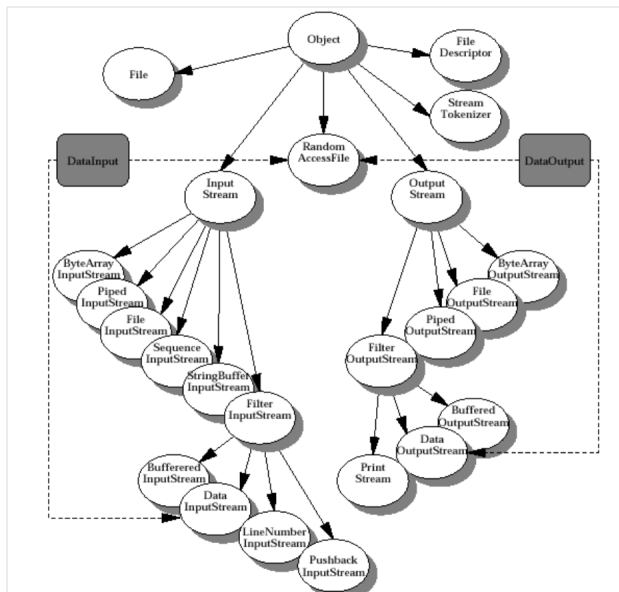
# Biblioteki Javy



# Biblioteki Javy

- Podstawowa dystrybucja Javy (J2SE) to prawie cztery tysiące klas zgrupowanych w około dwustu pakietach.
- Najistotniejsze pakiety:
  - 1 `java.lang`
  - 2 `java.io` i `java.nio`

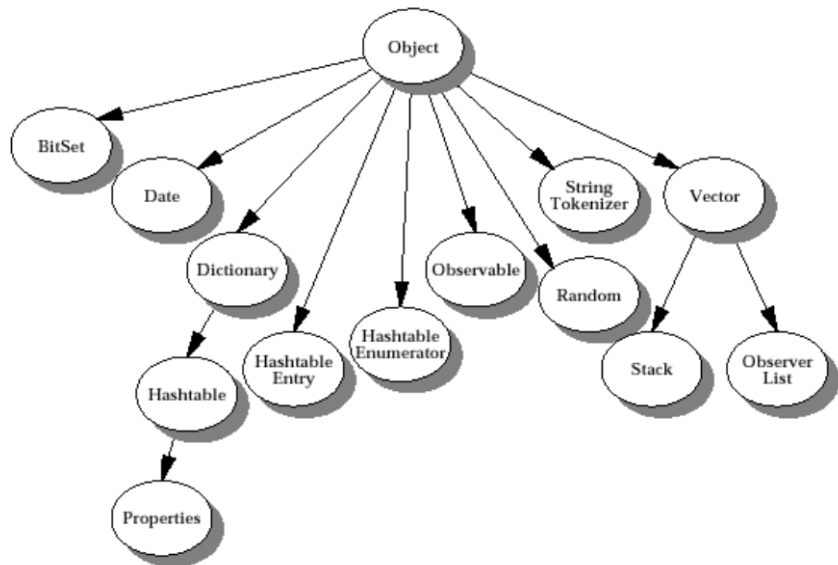
# Biblioteki Javy



# Biblioteki Javy

- Podstawowa dystrybucja Javy (J2SE) to prawie cztery tysiące klas zgrupowanych w około dwustu pakietach.
- Najistotniejsze pakiety:
  - 1 `java.lang`
  - 2 `java.io` i `java.nio`
  - 3 `java.util`

# Biblioteki Javy

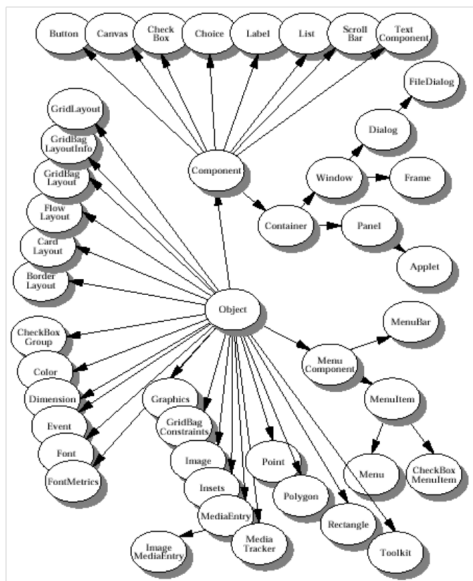


# Biblioteki Javy

- Podstawowa dystrybucja Javy (J2SE) to prawie cztery tysiące klas zgrupowanych w około dwustu pakietach.
- Najistotniejsze pakiety:
  - 1 `java.lang`
  - 2 `java.io` i `java.nio`
  - 3 `java.util`
  - 4 `java.awt`, `javax.swing`



# Biblioteki Javy



# Rodzaje programów w Javie

**Aplikacje** – samodzielne programy uruchamiane na platformie Javy (w tym serwery, tj. aplikacje służące klientom w sieci np. Web serwery, proxy, serwery pocztowe, serwery drukarek itp.)

**Applety** – programy „doklejane” do stron internetowych i odczytywane za pomocą przeglądarek z zainstalowanymi interpreterami Javy,

**Servlety** – programy które podobnie jak applety są stworzone dla potrzeb Internetu, ale uruchamiane są po stronie serwera. Mogą być wykorzystane do budowy interaktywnych aplikacji internetowych, zastępując skrypty CGI.

**Midlety** – programy uruchamiane w urządzeniach PDA, tel. itp.

# Słowa kluczowe

abstract

```
abstract class{  
    abstract metod();  
}
```

# Słowa kluczowe

abstract

continue

for

if

```
for (int i=0;i<5;i++){  
    if(i%2==0){  
        continue;  
    }  
}
```

# Słowa kluczowe

abstract

continue

for

if

assert

```
assert ref != null : "ref is null";  
if (ref == null)  
    throw new AssertionError();
```

*przełącznik przy uruchomieniu -enableassertions lub -ea*

# Słowa kluczowe

abstract  
switch

continue  
default

for  
case

if  
break

assert

# Słowa kluczowe

abstract  
switch

continue  
default

for  
case

if  
break

assert

# Słowa kluczowe

abstract  
switch

continue  
default

for  
case

if  
break

assert



# Słowa kluczowe

abstract  
switch

continue  
default

for  
case

if  
break

assert

# Słowa kluczowe

abstract  
switch

continue  
default

for  
case

if  
break

assert

# Słowa kluczowe

abstract  
switch

continue  
default

for  
case

if  
break

assert

# Słowa kluczowe

abstract  
switch

continue  
default

for  
case

if  
break

assert  
synchronized

# Słowa kluczowe

abstract  
switch  
boolean

continue  
default

for  
case

if  
break

assert  
synchronized

# Słowa kluczowe

abstract  
switch  
boolean

continue  
default  
do

for  
case

if  
break

assert  
synchronized

# Słowa kluczowe

abstract  
switch  
boolean

continue  
default  
do

for  
case  
goto

if  
break

assert  
synchronized

# Słowa kluczowe

abstract  
switch  
boolean

continue  
default  
do

for  
case  
goto

if  
break  
private

assert  
synchronized



# Słowa kluczowe

abstract  
switch  
boolean

continue  
default  
do

for  
case  
goto

if  
break  
private

assert  
synchronized  
this

# Słowa kluczowe

abstract  
switch  
boolean  
package

continue  
default  
do

for  
case  
goto

if  
break  
private

assert  
synchronized  
this

# Słowa kluczowe

abstract  
switch  
boolean  
package

continue  
default  
do  
double

for  
case  
goto

if  
break  
private

assert  
synchronized  
this

# Słowa kluczowe

|          |          |            |         |              |
|----------|----------|------------|---------|--------------|
| abstract | continue | for        | if      | assert       |
| switch   | default  | case       | break   | synchronized |
| boolean  | do       | goto       | private | this         |
| package  | double   | implements |         |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     |          |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     |            |           |              |



# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      |          |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |



# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    |          |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     |          |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  |           |              |



# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    |          |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       | strictfp  |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       | strictfp  | volatile     |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       | strictfp  | volatile     |
| const    |          |            |           |              |



# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       | strictfp  | volatile     |
| const    | float    |            |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       | strictfp  | volatile     |
| const    | float    | native     |           |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       | strictfp  | volatile     |
| const    | float    | native     | super     |              |

# Słowa kluczowe

|          |          |            |           |              |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for        | if        | assert       |
| switch   | default  | case       | break     | synchronized |
| boolean  | do       | goto       | private   | this         |
| package  | double   | implements | protected | throw        |
| byte     | else     | import     | public    | throws       |
| new      | enum     | instanceof | return    | transient    |
| catch    | extends  | int        | short     | try          |
| char     | final    | interface  | static    | void         |
| class    | finally  | long       | strictfp  | volatile     |
| const    | float    | native     | super     | while        |

# Operatory

|                        |    |    |     |     |      |    |   |
|------------------------|----|----|-----|-----|------|----|---|
| operatory arytm.:      | +  | ++ | -   | --  | *    | /  | % |
| operatory relacyjne:   | >  | <  | =>  | =<  | ==   | != |   |
| operatory logiczne:    | !  | && |     | &   |      | ^  |   |
| operatory przypisania: | =  | += | -=  | *=  | /=   | &= |   |
|                        | =  | ^= | <<= | >>= | >>>= | %= |   |
| operatory bitowe:      | << | >> | >>> | ~   | ^    |    |   |
| operatory pozostałe:   | ?: | .  | []  | ()  |      |    |   |

# Priorytet Operatorów

| Priorytet | Operatory                   | Priorytet | Operatory |
|-----------|-----------------------------|-----------|-----------|
| 1         | . [] ()                     | 9         | ^         |
| 2         | ++ -- ! ~ <i>instanceof</i> | 10        |           |
| 3         | * / %                       | 11        | &&        |
| 4         | + -                         | 12        |           |
| 5         | << >> >>>                   | 13        | ? :       |
| 6         | < > <= >=                   | 14        | = op=     |
| 7         | == !=                       | 15        | ,         |
| 8         | &                           |           |           |

# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne

# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne
- typy referencyjne



# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne
- typy referencyjne

**Typy pierwotne** to grupa ośmiu typów zawierających wartości proste. Tymi typami są:

- typ wartości logicznych: boolean,

# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne
- typy referencyjne

**Typy pierwotne** to grupa ośmiu typów zawierających wartości proste.

Tymi typami są:

- typ wartości logicznych: boolean,
- typy całkowitoliczbowe: byte, short, int, long, char,

# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne
- typy referencyjne

**Typy pierwotne** to grupa ośmiu typów zawierających wartości proste.

Tymi typami są:

- typ wartości logicznych: boolean,
- typy całkowitoliczbowe: byte, short, int, long, char,
- typy zmiennopozycyjne: float, double

| typ     | rozmiar (bity) | przedział zmienności                         |
|---------|----------------|--|
| boolean | 8              | true - false                                 |
| byte    | 8              | -128 - 127                                   |
| char    | 16             | Unicode 0-65536                              |
| short   | 16             | -32 768 - 32 767                             |
| int     | 32             | -2 147 483 648 - 2 147 483 647               |
| long    | 64             | $-9,2 \cdot 10^{18}$ - $9,2 \cdot 10^{18}$   |
| float   | 32 IEEE 754    | $3,4 \cdot 10^{-38}$ - $3,4 \cdot 10^{38}$   |
| double  | 64 IEEE 754    | $1,7 \cdot 10^{-308}$ - $1,7 \cdot 10^{308}$ |

# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne
- typy referencyjne

**Typy pierwotne** to grupa ośmiu typów zawierających wartości proste.

Tymi typami są:

- typ wartości logicznych: boolean,
- typy całkowitoliczbowe: byte, short, int, long, char,
- typy zmiennopozycyjne: float, double

**Typy referencyjne** dzielą się z kolei na następujące kategorie:

- typy klas,

# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne
- typy referencyjne

**Typy pierwotne** to grupa ośmiu typów zawierających wartości proste.

Tymi typami są:

- typ wartości logicznych: boolean,
- typy całkowitoliczbowe: byte, short, int, long, char,
- typy zmiennopozycyjne: float, double

**Typy referencyjne** dzielą się z kolei na następujące kategorie:

- typy klas,
- typy interfejsów,

# Typy danych

Specyficzną cechą Javy jest to, że typy w tym języku są podzielone na dwie kategorie:

- typy pierwotne
- typy referencyjne

**Typy pierwotne** to grupa ośmiu typów zawierających wartości proste.

Tymi typami są:

- typ wartości logicznych: boolean,
- typy całkowitoliczbowe: byte, short, int, long, char,
- typy zmiennopozycyjne: float, double

**Typy referencyjne** dzielą się z kolei na następujące kategorie:

- typy klas,
- typy interfejsów,
- typy tablic.

Wartościami typów referencyjnych są referencje (w pewnym uproszczeniu można o nich myśleć jako o wskaźnikach) do obiektów lub wartość **null**.

# Literały

W Javie mamy 6 rodzajów literałów:

- Liczby całkowite (np. 13 czy -2627). Format dziesiętny, szesnastkowy (0xC) lub ósemkowe (np. 015). Typ literałów całkowitych to `byte`, `short`, `int`, `long`.
- Liczby rzeczywiste (np. 1.0 czy -4.9e12), mogą być zapisane w systemie dziesiętnym lub szesnastkowym.
- Literały logiczne `false` i `true`.
- Literały znakowe (np. `'a'`).
- Literały napisowe (np. `"Ala ma kota"`). Na uwagę zasługuje fakt, że napisy nie są w Javie wartościami typu pierwotnego, lecz obiektami klasy `String`.
- Literał `null`.

# Zmienne

Zmienne są (zwykle) nazwanymi pojemnikami na pojedyncze wartości, typu z jakim zostały zadeklarowane. Zmienne typów pierwotnych przechowują wartości dokładnie tych typów, zmienne typów referencyjnych przechowują wartość null albo wartość referencji do obiektu.

Najczęściej wykorzystywane typy zmiennych:

- zmienne klasowe (statyczne - należące do klasy),
- zmienne egzemplarzowe (należące do obiektu),
- zmienne lokalne (iteratory pętli, zmienne deklarowane w metodach),
- zmienne tablicowe (mogą być anonimowe: `new int[] { 1, 2, 3 }`),
- parametry metod i konstruktorów,
- parametry obsługi wyjątków.

Każda zmienna musi być zadeklarowana. Z każdą zmienną związany jest jej typ podawany przy deklaracji zmiennej. **Typ ten jest używany przez kompilator do sprawdzania poprawności operacji wykonywanych na zmiennych.**



# Tablice

Tablica jest typem umożliwiającym grupowanie zmiennych tego samego typu i odwoływanie się do nich za pomocą wspólnej nazwy.

Tablice można deklarować z podaniem rozmiaru lub bez, np.

```
int month[], week[7];  
month = new int[12];
```

Tablice można inicjalizować automatycznie np.

```
int month[]={1,2,3}
```

# Tablice

W Javie podobnie jak w C/C++ nie ma tablic wielowymiarowych, a jedynie tablice tablic, np.

```
double macierz[] []= new double [3] [3];
```

Tę samą deklarację można przedstawić również na inny sposób, np.

```
double macierz[] []= new double [3] [];  
macierz[0]=new double [3];  
macierz[1]=new double [3];  
macierz[2]=new double [3];
```

# Podstawowa obsługa wyjątków

Do obsługi wyjątków służy instrukcja **try-catch**. Po **try** podaje się blok instrukcji, których wyjątki chcemy obsługiwać. Następnie podawana jest lista bloków **catch**, które przypominają deklaracje metod.

```
try {  
    //kod ktory moze zglosic wyjatki  
} catch (Typ1 w) {  
    //obsługa wyjatkow Typ1  
} catch (Typ2 w) {  
    //obsługa wyjatkow Typ2  
} catch (Typ3 w) {  
    //obsługa wyjatkow Typ3  
}
```

# Plan Wykładu

- 5 Koncepcja programowania obiektowego
- 6 Enkapsulacja
- 7 Definiowanie klas, tworzenie obiektów
- 8 Dziedziczenie
- 9 Polimorfizm
- 10 Modyfikatory klas, metod i pól
- 11 Klasy zagnieżdżone, anonimowe i lokalne

# Koncepcja programowania obiektowego

**Obiekt** jest programowym zestawem powiązanych **zmiennych** i **metod**. Zmienne przechowują informacje dotyczące stanu modelowanych obiektów lub procesów świata rzeczywistego, a metody definiują ich zachowanie.

# Programowanie Strukturalne a Obiektowe

## Programowanie strukturalne

- Najistotniejszy element to realizowany proces.
- Top-down approach– podejście do problemu poprzez zdemontowanie go na poszczególne funkcje.
- Realizacja poprzez funkcje.
- Mniej bezpieczne – brak możliwości ukrycia danych.
- Umożliwia realizację średnio skomplikowanych problemów.
- Mniejsze możliwości ponownego użycia kodu, mniejszy poziom abstrakcji i elastyczności.

## Programowanie obiektowe

- Najistotniejszy element to dane.
- Bottom-up approach– podejście do problemu poprzez realizację poszczególnych funkcjonalności i połączenie ich rezultatów.
- Realizacja poprzez obiekty.
- Bardziej bezpieczne – pola prywatne.
- Umożliwia realizację dowolnie skomplikowanych problemów.
- Większe możliwości ponownego użycia kodu, większy poziom abstrakcji i elastyczności.

# Główne mechanizmy programowania obiektowego

Języki zorientowane obiektowo zawierają następujące mechanizmy wymuszające stosowanie obiektów:

- enkapsulacja (ang. encapsulation),
- dziedziczenie (ang. inheritance),
- polimorfizm (ang. polymorphism).

Mechanizmy te funkcjonują w Javie bardzo podobnie jak w C++.

# Enkapsulacja

**Enkapsulacja** polega na łączeniu danych i instrukcji, które wykonują na nich działania, przez umieszczanie ich we wspólnych obiektach. Środkiem do osiągnięcia enkapsulacji w Javie są klasy (ang. *classes*).

**Klasa** stanowi model abstrakcyjny pewnej grupy obiektów wyróżniających się tą samą strukturą i zachowaniem, jest modułem posiadającym nazwę i atrybuty w postaci pól danych i metod. Zatem **obiekt** (ang. *object*) to pojedynczy egzemplarz klasy.

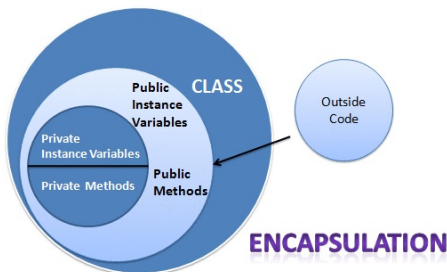
**Obiekty** nazywa się nieraz **egzemplarzami klas** (ang. *instances of classes*). Dane należące do klasy i przechowujące informacje o stanie każdego obiektu określa się **zmiennymi egzemplarzowymi** (ang. *instance variables*). Wykonywane zadania i sposób dostępu do danej klasy określają jej **metody**.



Celem enkapsulacji jest zmniejszenie stopnia złożoności programu poprzez możliwość ukrycia szczegółów dotyczących funkcjonowania klasy przez zadeklarowanie ich jako **prywatne**. Natomiast elementy tworzące interfejs klasy deklaruje się jako **publiczne**.

Definicja klasy jest jedynym sposobem zdefiniowania nowego typu danych w Javie. Posługując się pojęciami klasy, programista może w wygodny i elegancki sposób definiować różnorodne typy danych wykorzystując:

- strukturę hierarchiczną deklaracji klas (kompozycja),
- prefiksowanie klas tzw. "dziedziczenie", umożliwiające tworzenie hierarchii typu: ogólny - bardziej szczegółowy.



# Definicja Klasy

Definicja klasy przyjmuje następującą formę (modyfikatory zostaną omówione później):

```
[modyfikatory] class NazwaKlasy [ extends NazwaNadklasy ]
[implements NazwyInterfejsow] {
Cialo klasy:
Tutaj znajdują się definicje pól danych,
metod i klas wewnętrznych klasy
}
```

Elementy deklaracji pomiędzy nawiasami [ i ] są opcjonalne. Stąd najprostsza postać definicji to:

```
class A{
}
```

# Operator new i konstruktory

Deklaracja zmiennej nie powoduje utworzenia obiektu - jest on tworzony dopiero za pomocą operatora `new` i konstruktora. Do tego czasu referencja jest pusta (ang. *null reference*).

Operator `new` tworzy pojedynczy egzemplarz danej klasy i zwraca wartość odwołania do niego.

```
Punkt p=new Punkt();  
Punkt p2=p;
```

Zapisanie wartości zmiennej `p` w `p2` nie powoduje zarezerwowania dodatkowej pamięci lub przekopiowania jakiegokolwiek części obiektu wskazywanego przez `p`. Istnieje tylko jeden obiekt i dwa odwołania.

# Konstruktory

Konstruktor zostaje wywołany podczas tworzenia nowego obiektu klasy. Każda klasa może posiadać wiele konstruktorów, różniących się listą argumentów. Ponieważ każda klasa w Javie dziedziczy z klasy `Object`, posiada też konstruktor bezparametrowy odziedziczony z tej klasy.

```
class MojaKlasa {  
    MojaKlasa(){ //pierwszy konstruktor  
        System.out.print("Konstruktor");  
    }  
    MojaKlasa(int a){...} //drugi konstruktor  
}
```

...

```
MojaKlasa zmienna=new MojaKlasa();
```

# Konstruktor kopiujący

W Javie nie używa się konstruktora kopiującego niejawnie i nie jest on tak często wykorzystywany jak w C++.

```
class MojaKlasa {  
    String dane;  
    MojaKlasa(){//...jakies cialo  
    }  
    MojaKlasa(MojaKlasa x){  
        dane=new String(x.dane);  
    }  
}
```

...

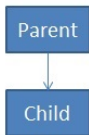
```
MojaKlasa a=new MojaKlasa();  
MojaKlasa zmienna=new MojaKlasa(a); //wywołanie konstruktora  
                                     //kopiującego
```

# Dziedziczenie

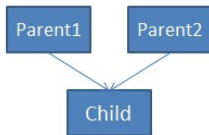
Większość ludzi czuje potrzebę uporządkowania obiektów świata rzeczywistego poprzez tworzenie złożonych taksonomii (klasyfikacji, kategoryzacji). W przypadku gdy pewne klasy tworzą abstrakcyjne, wspólne modele opisu (np. klasa Zwierzęta), wówczas na ich podstawie można utworzyć podklasy (np. klasa Ssaki) będące w relacji zawierania się w klasach nadrzędnych. Opis ssaków zawierać może dodatkowe informacje dotyczące ich charakterystycznych cech.

Ponieważ ssaki są dokładniej opisanymi zwierzętami to dziedziczą wszystkie ich atrybuty. Klasę zwierząt nazwiemy nadklasa (ang. *superclass*), natomiast klasę ssaków podklasa (ang. *subclass*).

## Types of Inheritance



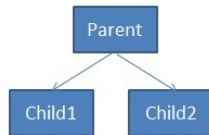
1. Single Inheritance



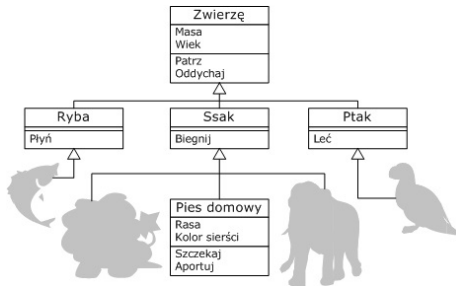
2. Multiple Inheritance



3. Multi-Level Inheritance



4. Hierarchical Inheritance



# Polimorfizm

Aby wykonać dwa różne zadania w większości języków programowania trzeba utworzyć dwie funkcje o różnych nazwach. **Polimorfizm** to możliwość tworzenia wielu metod o takiej samej nazwie, zgodnie z zasadą: *jeden przedmiot, wiele kształtów*.

Polimorfizm pozwala stworzyć wiele implementacji tej samej metody, co w informatyce nazywa się jej przeładowywaniem (ang. *overloading*). Wybór implementacji metody jest zależny od przekazywanych jej parametrów.



```
public class StatycznyPolimorfizm {  
    void metoda(){  
  
    }  
    void metoda(double y){  
  
    }  
    int metoda(int x){  
        return x;  
    }  
    /*  
    int metoda(){ //blad!!!  
  
    }  
  
    */  
}
```

## Polimorfizm czasu przebiegu

Przeładowywanie metod umożliwia ujednolicenie działań wykonywanych na danych różnych typów i jest najbardziej przydatne podczas tworzenia małych klas, gdyż ma charakter statyczny (trzeba przewidzieć wszystkie typy danych na których trzeba będzie wykonywać działania).

Czasami potrzebne jest rozwiązanie przewidujące możliwość zmiany implementacji danej metody w podklasie. Wówczas jej wersja zdefiniowana w nad klasie może w ogóle nie wykonywać żadnego działania. Tego typu polimorfizm nazywa się polimorfizmem czasu przebiegu (ang. *runtime polymorphism*).

```
class A{
    void metoda(){
        system.out.println(" to _metoda _A" );
    }
}
class B extends A{
    void metoda(){
        system.out.println(" to _metoda _B" );
    }
}
class C extends A{
    void metoda(){
        system.out.println(" to _metoda _C" );
    }
}
class Aplikacja{
    public static void main(String [] args){
        A obj=null;
        double val=Math.random();
        if (val<0.5)    obj=new B();
        else            obj=new C();
        obj.metoda();
    }
}
```

# Odwołania do obiektów

Deklaracja zmiennej typu obiektowego przyjmuje postać podobna do deklaracji zmiennej typu pierwotnego. Aczkolwiek w odróżnieniu od takiej zmiennej jej inicjalizacja musi zostać wykonana z wykorzystaniem operatora **new** i określonego konstruktora lub metod tworzących dany obiekt (wzorzec projektowy fabryka – factory). Utworzenie obiektu może nastąpić równocześnie z deklaracją zmiennej obiektowej:

```
KontoOsobiste konto=new KontoOsobiste();
```

lub w późniejszej części kodu:

```
KontoOsobiste konto;  
...  
konto=new KontoOsobiste();
```

Odwołania do obiektów przypominają wskaźniki z C/C++, z tą różnicą że nie można na nich wykonywać operacji arytmetycznych.

Bezpieczeństwo Javy wynika właśnie z braku możliwości utworzenia odwołania, które by wskazywało na dowolny fragment pamięci.

Odwołania do obiektów danej klasy są kompatybilne z odwołaniami do obiektów wszystkich jej podklas.

Ponadto, wszystkie typy wewnętrzne mają swoje odpowiedniki obiektowe (np.: typ `int` ma odpowiadający mu typ obiektowy `Integer`) i jako obiekty mają wiele konstruktorów i metod.

# Pola danych i metody

- **Pola Danych** - są atrybutami klasy, pełniącymi role podobna do zmiennych lub stałych. Sa one deklarowane na tych samych zasadach, co zmienne lokalne, wg. schematu:

```
modyfikatoryPola TypPola NazwaPola;
```

- **Metody** – są modułami programowymi przypominającymi funkcje z języka C++. Każda funkcja w Javie jest związana z definicją klasy (spełnia role jej metody) i deklarowana jest wg. schematu:

```
modyfikatory TypRezultatu NazwaMetody(ListaParametrów)
{
//tresc metody
}
```

Metody sa modułami programowymi przypominającymi funkcje z języka C++. Każda funkcja w Javie jest związana z definicją klasy (spełnia role jej metody)

# Pola danych i metody – przykład

```
public class Punkt {  
    public int x = 0;  
    public int y = 0;  
  
    public ustaw(int x, int y){  
        this.x = x;  
        this.y = y;  
    }  
}
```

# Modyfikatory klas, metod i pól

W Javie modyfikatory możemy podzielić na dwa rodzaje:

- modyfikatory dostępu wpływające na reguły widoczności i umożliwiające kontrole dostępu do pól danych i metod klasy z innych klas,
- modyfikatory właściwości modyfikowanego elementu.



# Modyfikatory dostępu

Modyfikatory dostępu:

- public - wszystkie klasy mają dostęp do pól danych i metod public,
- private - dostęp do metod i pól danych posiadają jedynie inne metody tej samej klasy,
- protected - metoda lub pole danych protected lub może być używana jedynie przez metody swojej klasy oraz metody wszystkich jej klas pochodnych,
- package - jest to **modyfikator domyślny**, wszystkie metody i pola danych bez modyfikatora dostępu traktowane są jako typu package. Metody (lub pola danych) typu package mogą być używane przez inne klasy danego pakietu.

# Poziomy dostęp

| modyfikator | klasa | podklasa | pakiet | wszędzie |
|-------------|-------|----------|--------|----------|
| private     | X     |          |        |          |
| package     | X     | *        | X      |          |
| protected   | X     | X        | X      |          |
| public      | X     | X        | X      | X        |

\* - dostęp istnieje o ile klasa i podklasa należą do tego samego pakietu.

# Gettery i settery

- Gettery i settery to publiczne metody, które pozwalają odpowiednio pobierać i zapisywać wartości prywatnych pól klasy.
- Można je traktować jak swego rodzaju interfejs do obsługi funkcjonalności jaką oferuje dana klasa.
- Pozwalają na ukrycie operacji, których sposób działania jest nie istotny z perspektywy użycia danej klasy.

# Gettery i settery – przykład

```
public class A {  
    private int x;  
  
    public int getX() {  
        return x;  
    }  
  
    public void setX(int x) {  
        this.x = x;  
    }  
}  
public class B extends A {  
    void metoda () {  
        this.setX(10);  
    }  
}  
public class C {  
    A obj=new A();  
    void metoda() {  
        obj.setX(10);  
    }  
}
```

# Modyfikatory właściwości

- pól danych
  - static,
  - final,
  - transient,
  - volatile.
- klas
  - public,
  - final,
  - abstract,
- metod
  - abstract,
  - static,
  - final,
  - synchronized,
  - native;

# Pola i metody statyczne

Deklaracja ze słowem **static** oznacza, że pole danych lub metoda dotyczy klasy, a nie obiektu, tzn. dla wszystkich obiektów danej klasy pole statyczne ma tę samą wartość.

Metody statyczne podobnie jak statyczne pola danych są przypisane do klasy, a nie konkretnego obiektu i służą do operacji tylko na polach statycznych. Przykład. Mamy klasę opisującą rachunek bankowy. Dla wszystkich rachunków jednego typu oprocentowanie wkładów jest jednakowe, więc oprocentowanie rachunku zadeklarowane zostało jako pole statyczne aby w razie zmiany oprocentowania nie zmieniać jego wartości dla wszystkich obiektów tej klasy.

```
class KontoOsobiste{
    static byte oprocentowanie = 10;
    private String wlasciciel;
    private String saldo;
    static void ZmienOprocentowanie(byte nowyProcent) {
        oprocentowanie = nowyProcent;
    }
    public void doliczProcent{
        saldo+=saldo*oprocentowanie/100;
    }

    public static void main (String[] args){
        KontoOsobiste Kowalski
            =new KontoOsobiste("Grazyna_Kowalska",500);
        KontoOsobiste Nosacz
            =new KontoOsobiste("Janusz_Nosacz",1500);
        Kowalski.doliczProcent();
        KontoOsobiste.ZmienOprocentowanie(15);
        Nosacz.doliczProcent();
    }
}
```

# Inicjowanie pól statycznych

Do inicjalizacji zmiennych statycznych wykorzystuje się zamiast konstruktorów tzw. inicjatory zmiennych statycznych. Inicjacja następuje wtedy, gdy klasa jest pierwszy raz ładowana do pamięci (gdy nie ma jeszcze żadnego obiektu danej klasy).

Każda klasa może zawierać dowolną liczbę inicjatorów statycznych. Inicjatory statyczne wykonywane są w kolejności ich wystąpienia w definicji klasy.

```
class MojaKlasa {  
    static int licznik=0;  
    static {  
        System.out.println("inicjalizacja pól statycznych")  
        licznik=0;  
    }  
}
```



## Klasy zagnieżdżone (ang. *nested*)

Klasa zagnieżdżona jest członkiem innej klasy. Klasy zagnieżdżone, mogą mieć modyfikator `static`, wskazujący że mają takie same właściwości jak klasa zewnętrzna. Jeśli nie są statyczne, wówczas określa się je jako wewnętrzne. Oprócz tego, klasy zagnieżdżone mogą być oczywiście opatrzone modyfikatorami: `private`, `protected` i `public` oraz `abstract` i `final` - a znaczenia tych modyfikatorów są takie same jak w przypadku zwykłych klas.

```
class KlasaZewnetrzna{  
    static class KlasaZagnizdzonaStatyczna {  
        . . .  
    }  
    class KlasaWewnetrzna {  
        . . .  
    }  
}
```

# Klasy wewnętrzne a statyczne zagnieżdżone

## Klasy zagnieżdżone statyczne:

- nie mogą bezpośrednio odwoływać się do atrybutów klasy zewnętrznej (muszą używać kwalifikowanego odnośnika)
- obiekty tych klas mogą istnieć nawet gdy nie istnieją obiekty klas zewnętrznych,
- mogą być traktowane jak klasy zewnętrzne ale ze specjalizacją funkcjonalności dla konkretnej klasy w której są zagnieżdżone. Usprawniają proces enkapsulacji poprzez powiązanie metod operujących na danych klasach z tymi klasami.

## Klasy wewnętrzne:

- mają nieograniczony dostęp do atrybutów klasy zewnętrznej,
- obiekty tych klas istnieją tylko gdy istnieją obiekty klas zewnętrznych,
- są często wykorzystywane w mechanizmie obsługi zdarzeń AWT

```
public class OuterClass {  
    private int val;  
    private static int staticVal;  
    public class InnerClass {  
        void method() {  
            val = 2;  
        }  
    }  
    static class StaticClass {  
        void methodInStaticClass() {  
            staticVal = 5;  
        }  
    }  
    public static void main(String[] args) {  
        StaticClass myStaticObj=new StaticClass();  
        myStaticObj.methodInStaticClass();  
        OuterClass myObj=new OuterClass();  
        InnerClass myInnerObj= myObj.new InnerClass();  
        myInnerObj.method();  
        System.out.println(myObj.val);  
    }  
}
```

## Klasy wewnętrzne - przykład uchwytu

Założmy, że do istniejącej klasy kontenerowej `Stack` chcemy dodać nową funkcjonalność – pozwolić innym klasom na zliczanie elementów na stosie wykorzystując interfejs `java.util Enumeration`.

Interfejs zawiera dwie deklaracje metod:

```
public boolean hasMoreElements();
```

```
public Object nextElement();
```

i definiuje interfejs dla pętli po elementach:

```
while (hasMoreElements()) nextElement()
```

Jeżeli `Stack` zaimplementuje `Enumeration` w sobie, nie będzie można zliczyć zawartości stosu więcej niż raz (zostanie wyczyszczony), jak również zastosować dwóch wyliczeń równolegle. Musi istnieć klasa pomocnicza (adapter), w tym przypadku wewnętrzna, która ma dostęp do wszystkich elementów ponieważ klasa `Stack` wspiera tylko kolejki LIFO.

```

public class Stack {
    private Vector<Integer> items = new Vector<Integer>();
    public Stack() {
        items.add(5); items.add(6); items.add(7); items.add(8);
    }
    public Enumeration<Integer> enumerator() {
        return new StackEnum();
    }

    class StackEnum implements Enumeration<Integer> {
        int currentItem = items.size() - 1;

        public boolean hasMoreElements() {
            return (currentItem >= 0);
        }

        public Integer nextElement() {
            if (!hasMoreElements())
                throw new NoSuchElementException();
            else
                return items.elementAt(currentItem--);
        }
    }
}

```

```
public static void main(String[] args) {  
    Stack obj=new Stack();  
    for (Enumeration e=obj.enumerator();  
    e.hasMoreElements();) {  
        int liczba= (int) e.nextElement();  
        System.out.println(liczba);  
        for (Enumeration e2=obj.enumerator();  
        e2.hasMoreElements();) {  
            int liczba2= (int) e2.nextElement();  
            System.out.println(liczba2);  
        }  
    }  
}
```

# Klasy anonimowe

Klasa anonimowa jest to klasa wewnętrzna zadeklarowana bez nazwy i konstruktora (tylko przy użyciu `new`). Ze względu na nieczytelność kodu jest stosowana rzadko.

```
public class AnonKlassExample {  
    int x;  
    void metoda() {  
        x = (new Object() {  
            int obliczInt() {  
                return 5;  
            }  
        }).obliczInt();  
    }  
}
```

Możliwe jest utworzenie klasy anonimowej implementującej określony interfejs lub dziedziczącej po określonej klasie (w obu przypadkach klasa ta nie zostaje nazwana i możliwe jest utworzenie tylko jednej instancji tego typu).

# Klasy lokalne

Klasy lokalne to klasy zdefiniowane w bloku programu Javy. Klasa taka może odwoływać się do wszystkich zmiennych widocznych w miejscu wystąpienia jej definicji. Klasa lokalna jest widoczna, i może zostać użyta, tylko w bloku w którym została zdefiniowana.

```
class Test {  
    void test()  
    {  
        // definicja klasy lokalnej  
        class KlasaLokalna{  
            ...  
        }  
        // deklaracja obiektu typu: KlasaLokalna  
        KlasaLokalna kl = new KlasaLokalna();  
    }  
}
```