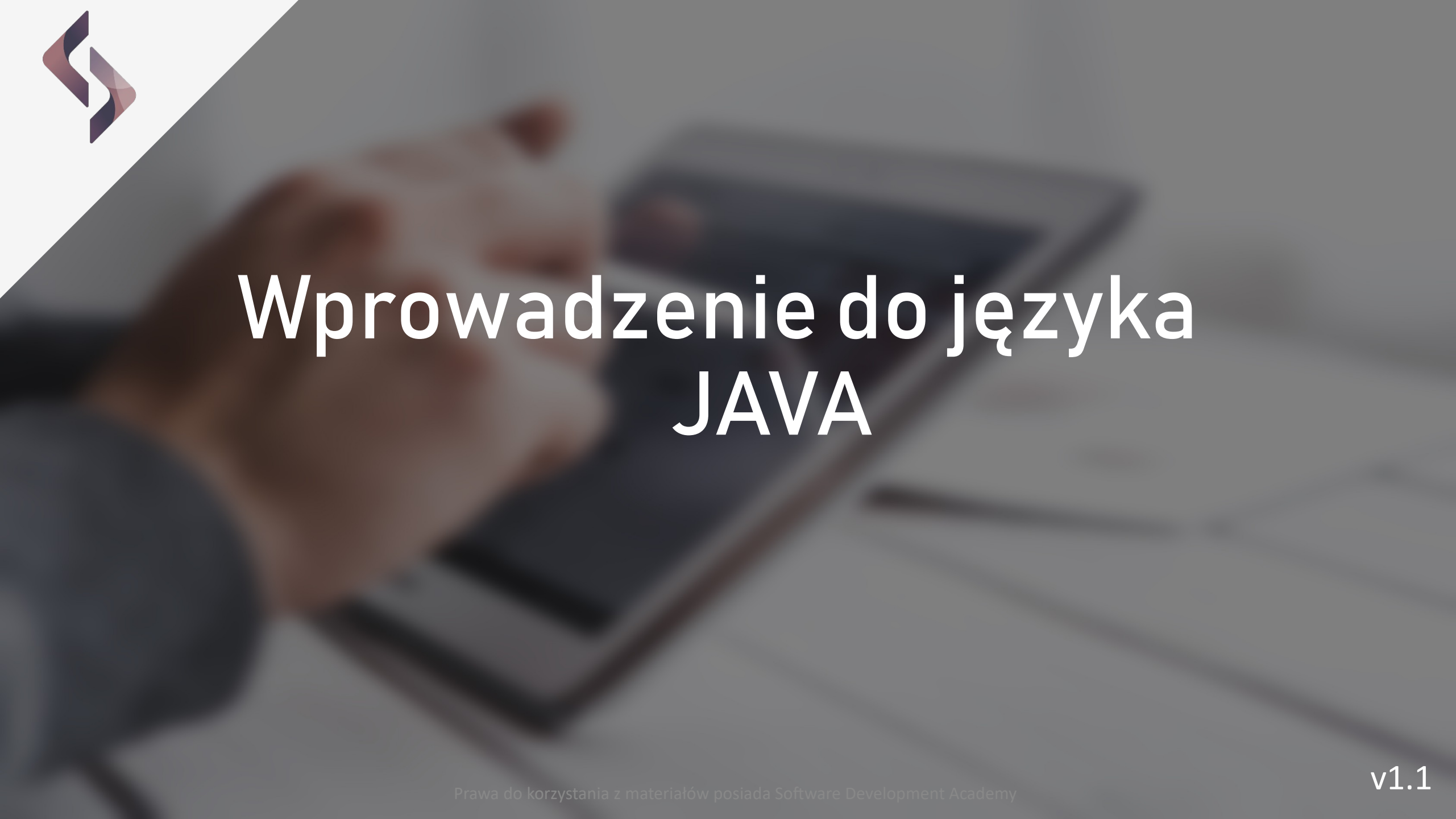


Wprowadzenie do języka JAVA

v1.1

Cel szkolenia

- Przekazanie praktycznej wiedzy związanej z metodami wytwarzania oprogramowania oraz pracą programistów w projektach informatycznych
- Przeprowadzenie ćwiczeń pozwalających doświadczyć i zrozumieć jak działa JAVA
- Wskazanie wyzwań i dobrych praktyk związanych z omawianymi zagadnieniami

Poznajmy się

Czego Ty oczekujesz od tego spotkania?

- Nazywam się?
- Dlaczego biorę udział w szkoleniu?
- Czego oczekuję – indywidualne cele uczestników?
- Czym się zajmuję?

Co to jest programowanie?

Programowanie komputerów – proces projektowania, tworzenia, testowania i utrzymywania kodu źródłowego programów komputerowych lub urządzeń mikroprocesorowych (mikrokontrolery).

Kod źródłowy jest napisany w języku programowania, z użyciem określonych reguł, może on być modyfikacją istniejącego programu lub czymś zupełnie nowym. Programowanie wymaga dużej wiedzy i doświadczenia w wielu różnych dziedzinach, jak projektowanie aplikacji, algorytmika, struktury danych, języki programowania
i narzędzia programistyczne, kompilatory, czy sposób działania podzespołów komputera.

W inżynierii oprogramowania programowanie (implementacja) jest tylko jednym z etapów powstawania programu.

Dlaczego Java?

- Język obiektowy
- Niezależność od platformy (*Write Once, Run Anywhere)*
- Automatyczne zarządzanie pamięcią
- Prostota
- Popularność
- Duża społeczność
- Olbrzymia ilość literatury
- Duże zapotrzebowanie na rynku pracy

Historia języka Java

Początek języka Java możemy określić jako rok 1991. Wtedy firma Sun z Patrickiem Naughtonem oraz Jamesem Goslingiem na czele postanowili stworzyć prosty i niewielki język, który mógłby być uruchamiany na wielu platformach z różnymi parametrami. Projekt zatytułowano Green.

Historia wersji języka JAVA

Java od początku jest językiem w pełni obiektowym, to co nie jest w niej obiektowe to w zasadzie tylko typy proste jak int, czy char, jednak nawet one posiadają typy osłonowe zwiększające ich użyteczność.

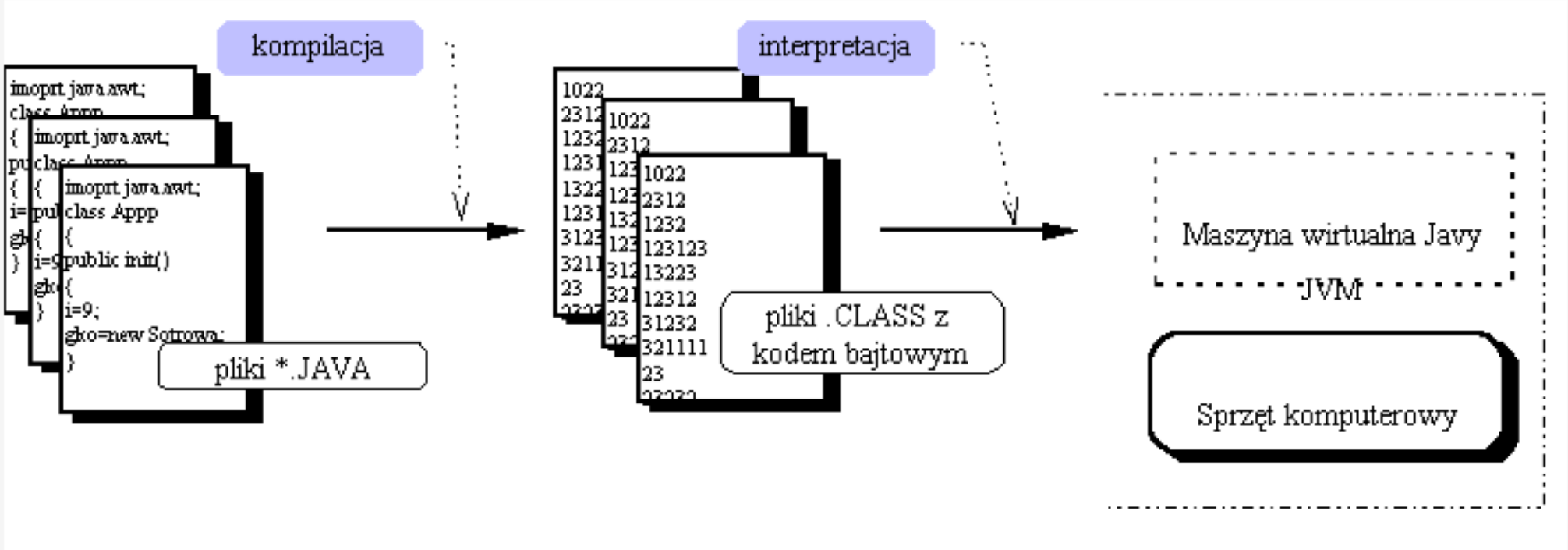
Pierwsza wersja Javy ukazała się w 1996 roku w wersji 1.0. Niestety nie osiągnęła ona wielkiego rozgłosu z czego inżynierowie firmy Sun dokładnie zdawali sobie sprawę. Na szczęście dosyć szybko poprawiono błędy i uzupełniono ją o nowe biblioteki, model zdarzeń GUI, a także poprawiono mechanizm **refleksji**. Była to Java 1.1.

Historia wersji języka JAVA

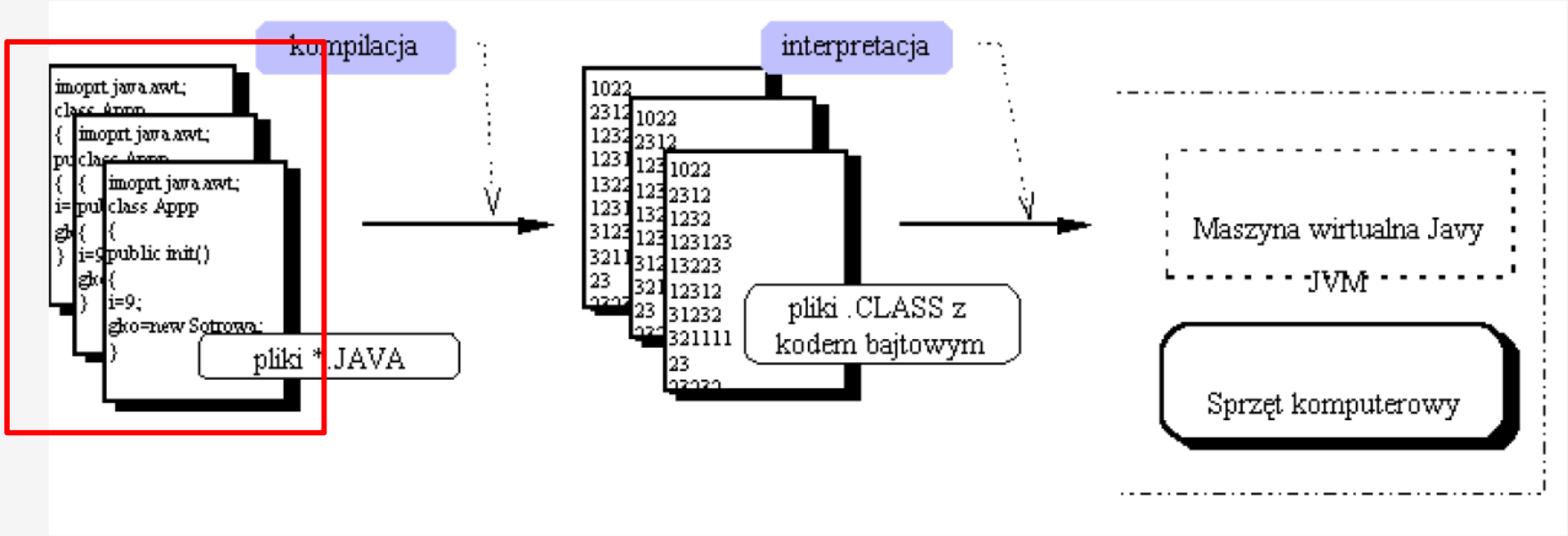
W Java 1.2, opublikowanej w 1998 roku największą zmianą była zmiana nazwy na dumnie brzmiący: **Java 2 Standard Edition Development Kit version 1.2**. Opracowano wtedy także dwa inne wydania Javy, doskonale znana prawie każdemu z nas wersja Micro Edition, przeznaczona na urządzenia mobilne, oraz Enterprise Edition wykorzystywana przede wszystkim w programowaniu aplikacji klient-serwer.

Kolejne edycje Javy aż do aktualnej 6 (nie 1.6), to przede wszystkim dodawanie nowych funkcjonalności oraz prace nad wydajnością bibliotek standardowych. Największe zmiany zaszły chyba w wersji 5.0, gdzie wprowadzono **Klasy generyczne (Generic Classes)** oraz między innymi statyczny import.

Jak działa programowanie w Javie?

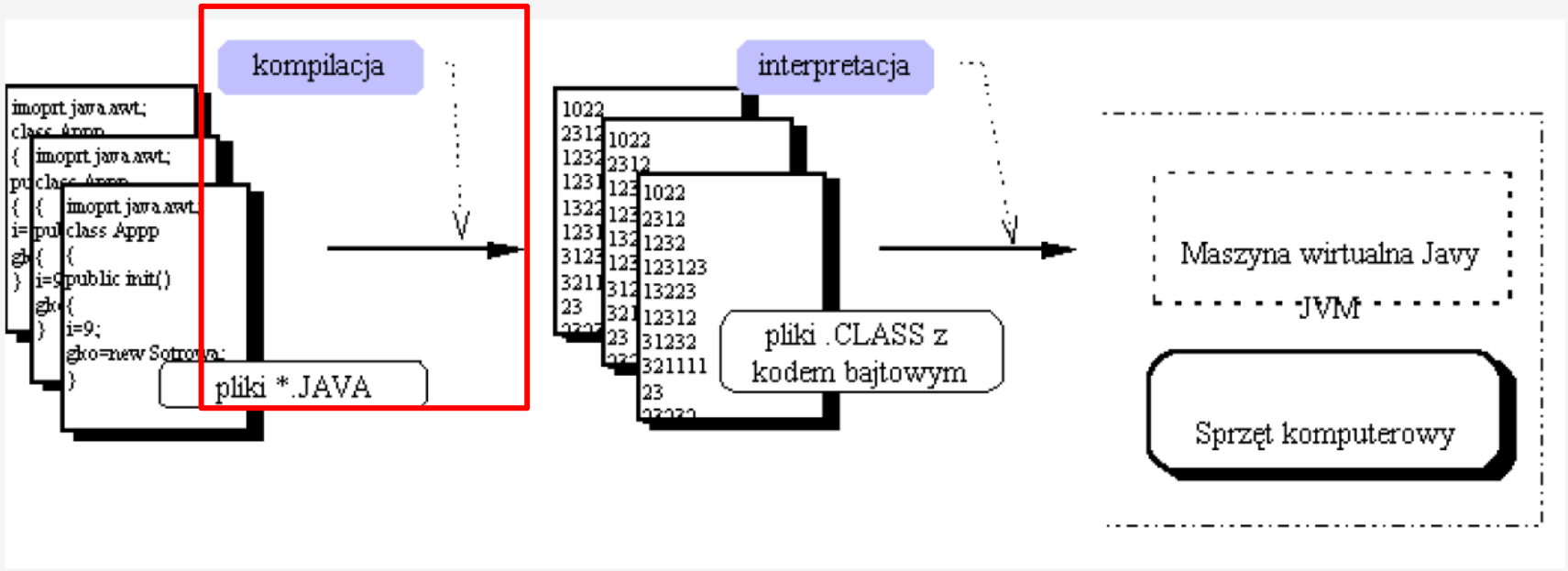


Jak działa programowanie w Javie?



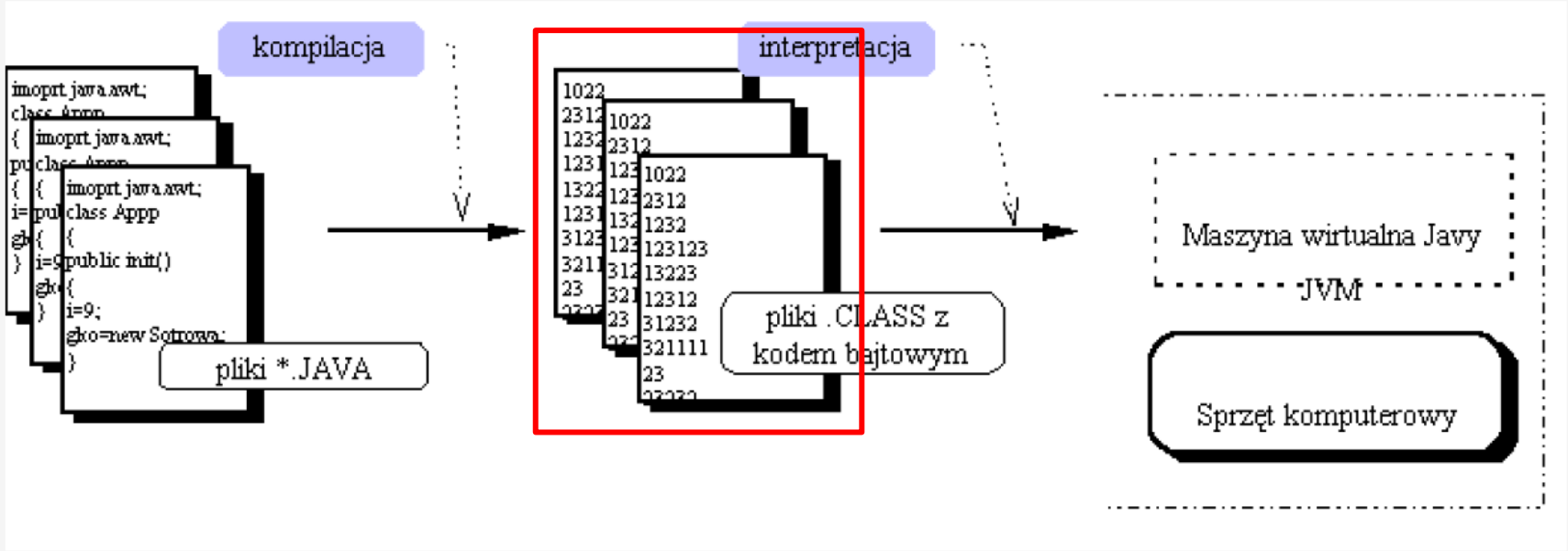
Kod źródłowy programu - zapis programu komputerowego przy pomocy określonego języka programowania, opisujący operacje jakie powinien wykonać komputer na zgromadzonych lub otrzymanych danych. Kod źródłowy jest wynikiem pracy programisty i pozwala wyrazić w czytelnej dla człowieka formie strukturę oraz działanie programu komputerowego.

Jak działa programowanie w Javie?



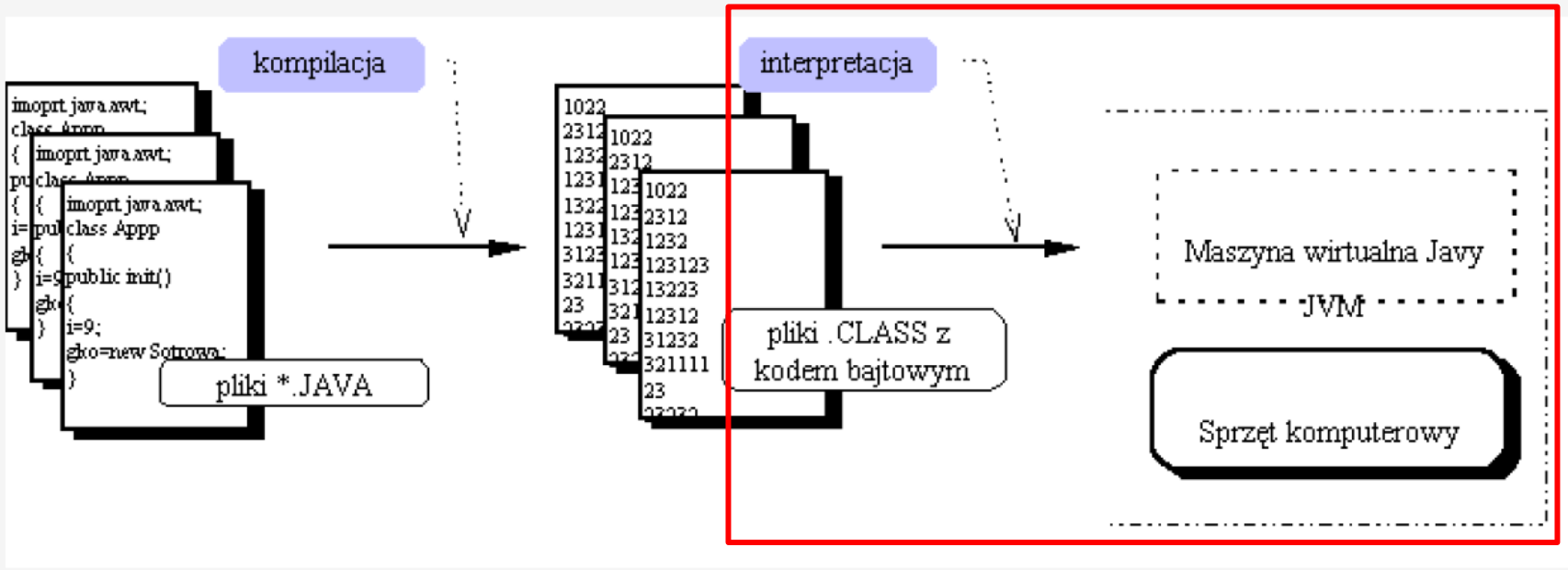
Kompilator - program, który kod źródłowy zamienia w kod napisany w innym języku. Oprócz tego kompilator ma za zadanie odnaleźć błędy leksykalne i semantyczne oraz dokonać optymalizacji kodu.

Jak działa programowanie w Javie?



Kod bajtowy - bytecode, wynik kompilacji programu napisanego w Javie, kod ten jest zrozumiały dla środowiska uruchomieniowego Java (JVM).

Jak działa programowanie w Javie?



Interpreter - program komputerowy, który analizuje kod źródłowy programu, a przeanalizowane fragmenty wykonuje.

Dlaczego Java?

- Język obiektowy
- Niezależność od platformy
- Prostota?
- Czy Java jest powolna?
- Java jest „duża”
- Podobieństwo do C#

Instalacja JDK

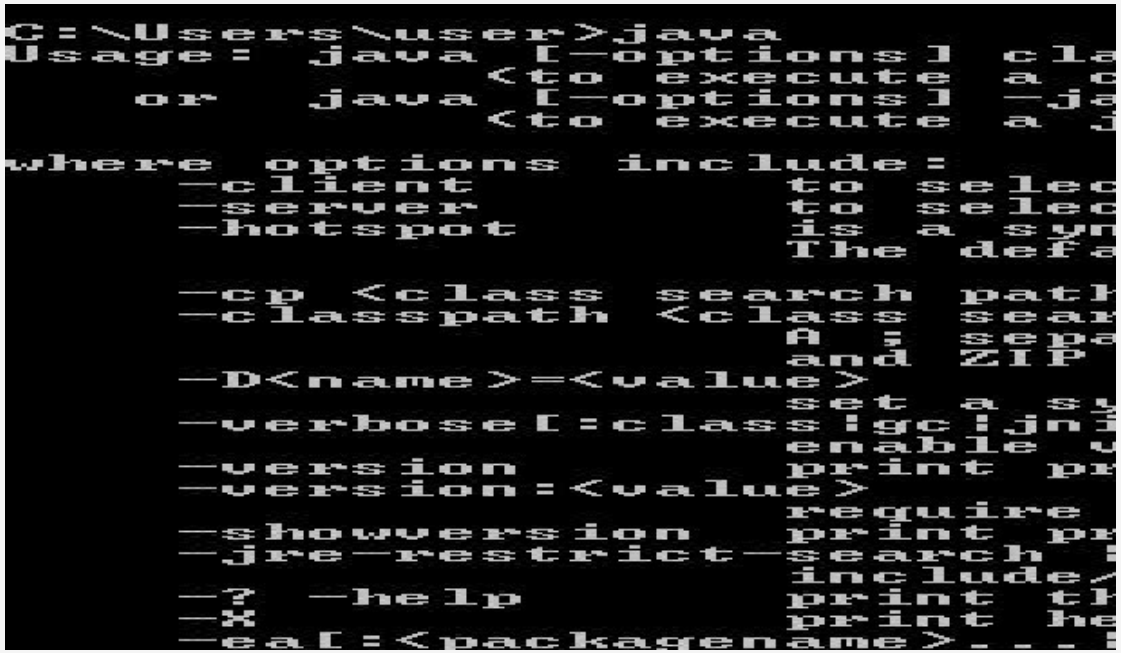
No więc na początek musimy pobrać środowisko JDK ze strony Oracle (zajmuje ono do 200MB pamięci). Najlepiej pobrać najnowszą wersję.

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Instalujemy standardowo, dalej dalej dalej ...

Instalacja JDK

Po zakończonej instalacji wypadałoby sprawdzić, czy wszystko działa jak należy. W tym celu otwieramy konsolę (Start -> Uruchom; wpisujemy cmd i enter). W konsoli wpisujemy najpierw polecenie ***java***, po czym powinno się nam ukazać mniej, a raczej więcej coś takiego jak poniżej:



Dystrybucja języka Java

- Java Platform, Standard Edition (Java SE/J2SE)
- Java Platform, Enterprise Edition (Java EE/J2EE)
- Java Platform, Micro Edition (Java ME/J2ME)

Dystrybucja języka Java

Java Platform, Standard Edition (Java SE/J2SE)

Java Platform, Standard Edition znane wcześniej jako J2SE - opracowana przez firmę Sun Microsystems opisująca podstawową wersję platformy Java. Stanowi podstawę dla Java EE (Java Platform, Enterprise Edition).

Dystrybucja języka Java

Java Platform, Enterprise Edition (Java EE/J2EE)

Java Enterprise, **J2EE** oraz **Java EE** czasami tłumaczona jako Java Korporacyjna - standard tworzenia aplikacji w języku programowania Java opartych o wielowarstwową architekturę komponentową.

Dystrybucja języka Java

Java Platform, Micro Edition (Java ME/J2ME)

Wcześniej jako **Java 2 Platform**, **Micro Edition** lub **J2ME**. Zaprojektowana z myślą o tworzeniu aplikacji mobilnych dla urządzeń o bardzo ograniczonych zasobach.

IntelliJ IDEA

Intellij IDEA jest to środowisko do tworzenia aplikacji w Javie. Jest ono rozwijane przez czeską firmę JetBrains, która jest także autorem narzędzie dedykowanych do innych języków jak PhpStorm dla języka PHP, PyCharm dla Pythona.

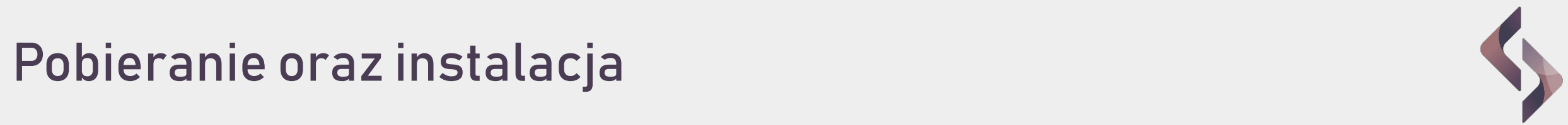
Dzięki temu, że jest to produkt po części komercyjny, to przez wielu uważane jest za najlepsze środowisko dedykowane do Javy. Posiada bardzo dobrą integrację z innymi narzędziami i technologiami dedykowanymi dla Javy (Maven, Spring, JEE, Android).

Community edition a Ultimate

- **Community** przeznaczonej przede wszystkim do aplikacji pisanych w czystej Javie. Nie ma niestety wsparcia dla technologii, w których Java wykorzystywana jest najczęściej, czyli JEE oraz frameworka Spring. Bez problemu jednak stworzymy w nim aplikacje na system Android.

- **Ultimate** dedykowanej dla programistów poważniejszych projektów biznesowych. Licencja jest wykupywana w systemie abonamentowym i kosztuje ok 600zł rocznie, a w przypadku licencji firmowych już ponad 1500zł.

Pobieranie oraz instalacja

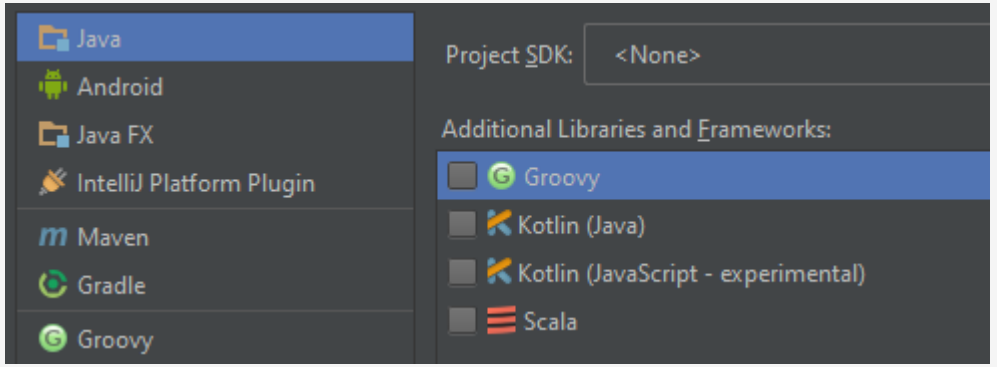


Środowisko można pobrać z oficjalnej strony JetBrains:

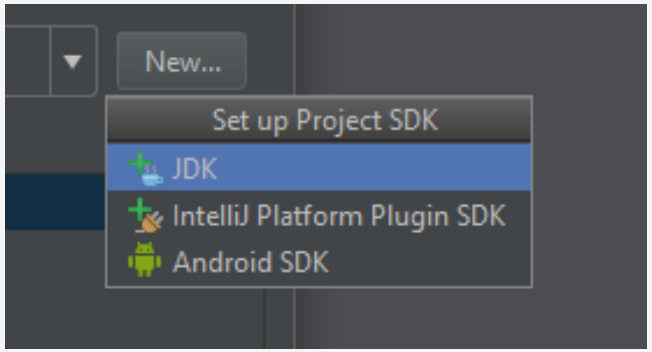
https://www.jetbrains.com/idea/download/

Pierwszy projekt

Rodzaj projektu: wybieramy opcję **Java Project**

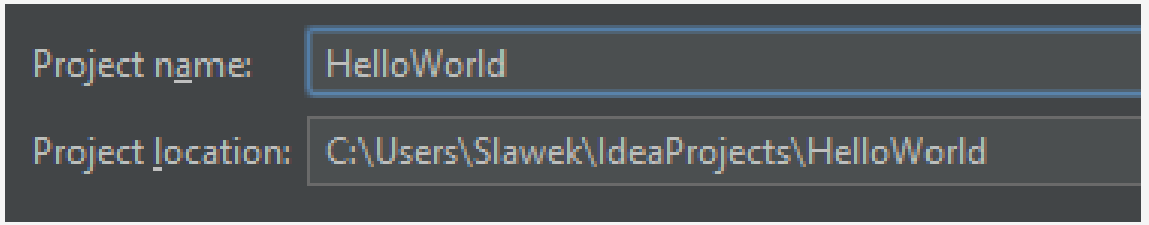


Konfiguracja JDK: wskazujemy lokalizację instalacji

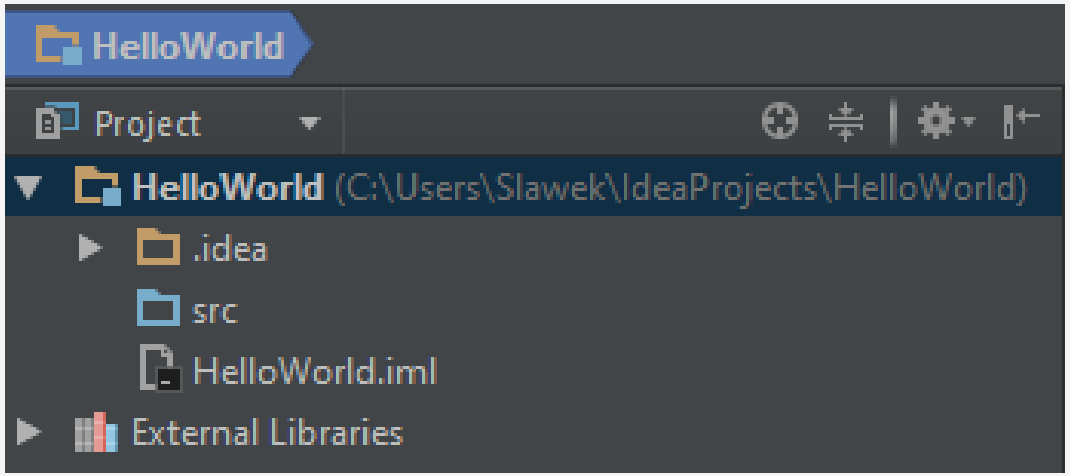


Pierwszy projekt

Nazwa projektu:



gotowe ☺



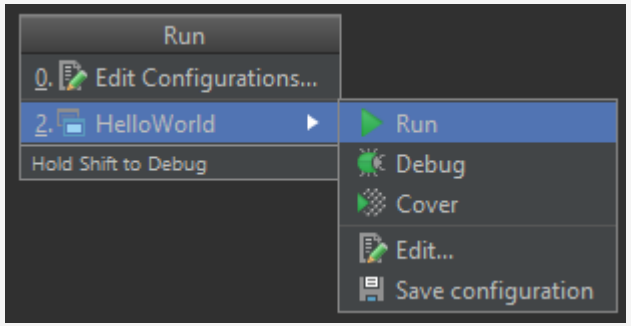
Pierwszy projekt

Dodanie nowej klasy: klikamy prawym przyciskiem myszy i wybieramy opcję *New > Java Class*.

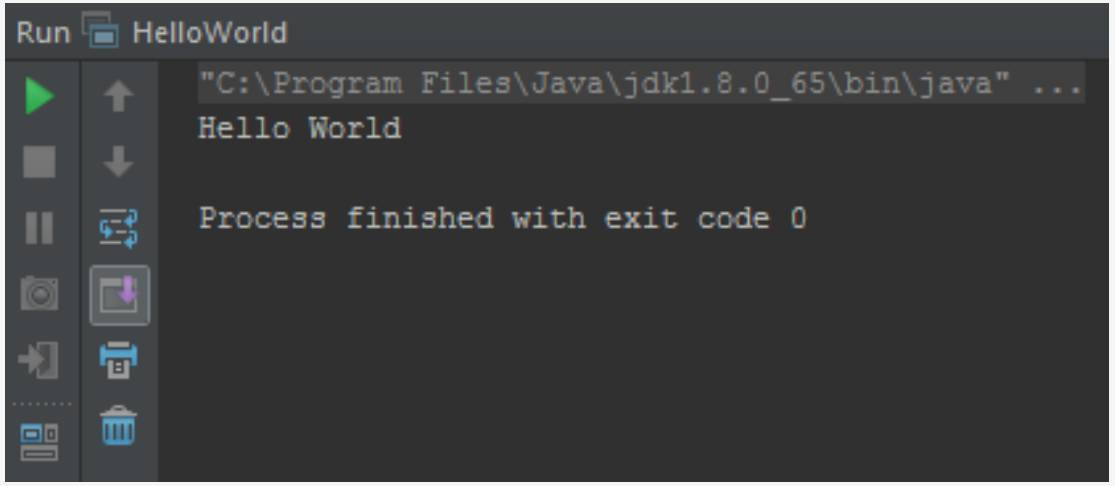
```
1  public class HelloWorld {
2      public static void main(String[] args) {
3          System.out.println("Hello World");
4      }
5  }
```

Uruchomienie programu: klikamy klasę z metodą main (HelloWorld) i wybieramy opcję *Run > Run*

lub wciskamy kombinację *Alt+Shift+F10* i z listy wybieramy opcję Run.



Uruchomienie aplikacji



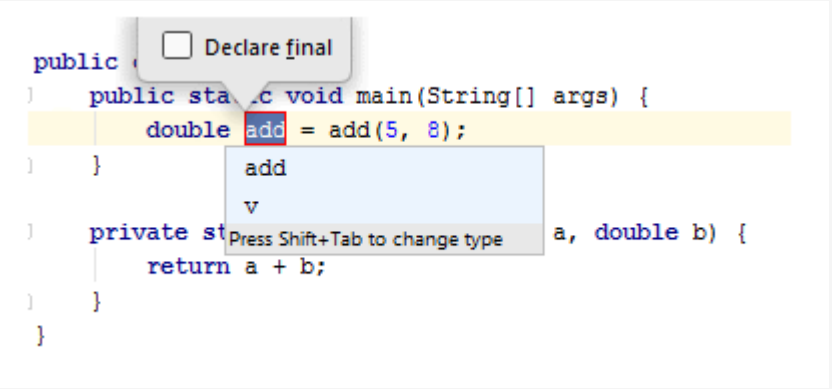
Podczas kolejnych uruchomień wystarczy posługiwać się skrótem *Shift+F10*.

IntelliJ IDEA - skróty klawiaturowe

Uzupełnianie kodu: *Ctrl+Spacja*

```
public class Example {
    public static void main(String[] args) {
        final String veryLongName = "Hello";
        System.out.println(veryLongName);
    }
}
```

Podpowiedzi kontekstowe: *Alt+Enter* - w zależności od kontekstu pokazuje różne możliwości



IntelliJ IDEA - skróty klawiaturowe

Poruszanie się między zakładkami:

- *Alt + Strzałka w lewo/prawo* - zmiana aktywnej zakładki na poprzednią / następną
- Ctrl + Tab - wybór okna
- Dwa razy Shift – „szukaj wszędzie”

Formatowanie kodu: *Ctrl + Alt + L*

```
public class Example {
public static void main(String[] args) {
double asdf = add(5, 8);
}
private static double add(double a, double b) {
            return a + b;
    }
}
```

Pierwszy program

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

1. *public class Hello* - jest to nic innego jak publiczna klasa o nazwie Hello.

2. *public static void main(String[] args)* - jest to metoda main, to od niej rozpoczyna się działanie.

3. *System.out.println("Hello World");* - wyświetl(print) napis podany jako argument("Hello World") przy użyciu strumienia wyjścia w biblioteki System, która dodaje na końcu drukowanego tekstu znak nowej linii "\n.

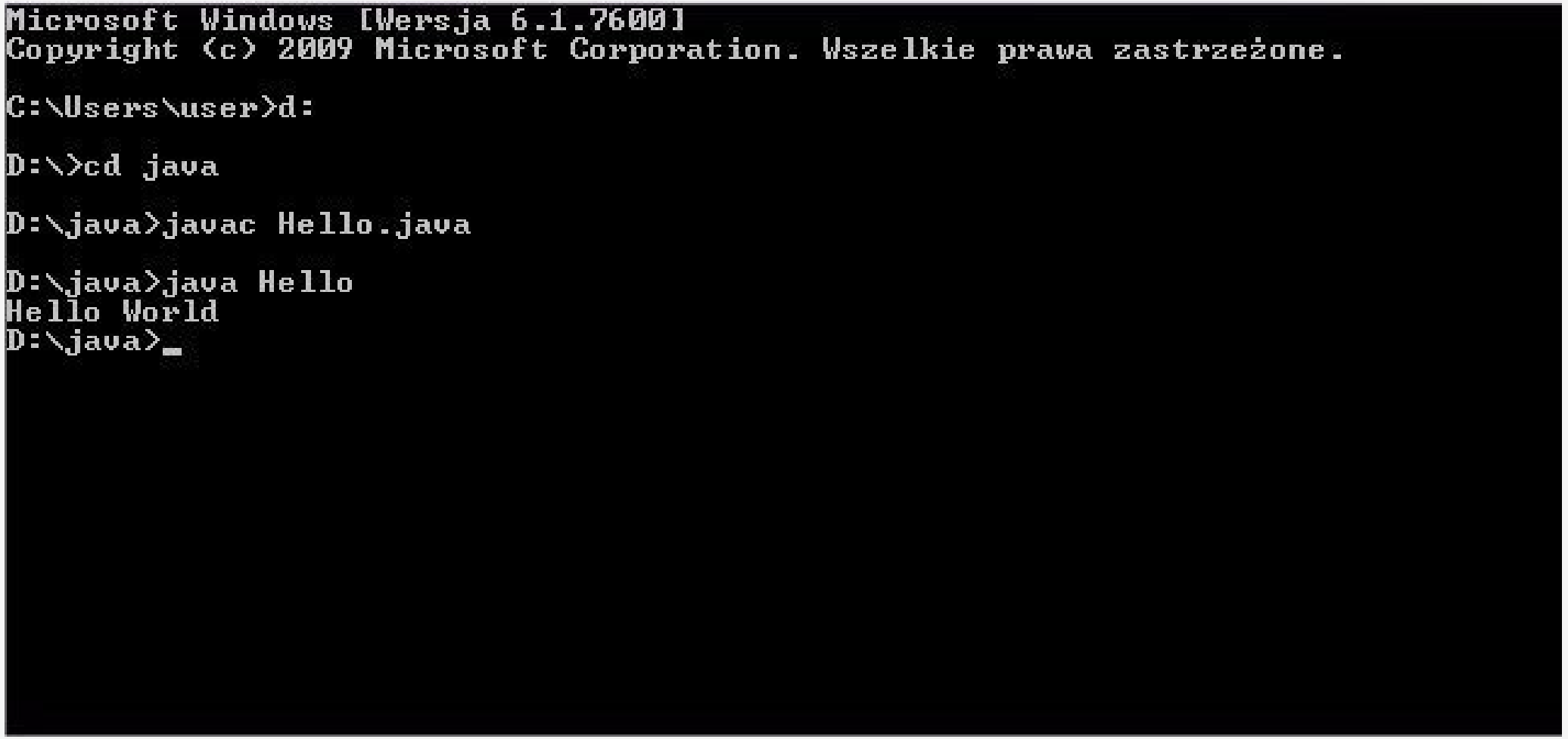
Ćwiczenie

Uruchomienie programu w eksploratorze systemowym

Kompilacja: *javac nazwa_klasy.java*

Uruchomienie: *java nazwa_klasy*

Pierwszy program



Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, który wyświetli w 3 kolejnych liniach trzy imiona: Ania, Bartek, Kasia.

Komentarze

W Javie istnieją dwa rodzaje komentarzy:

- *//text* - tekst umieszczony za podwojonym znaku slash jest uznawany za komentarz aż do końca linii
- */* text *\/* - tekst umieszczony w takich znacznikach jest traktowany jako komentarz przez wiele linii kodu

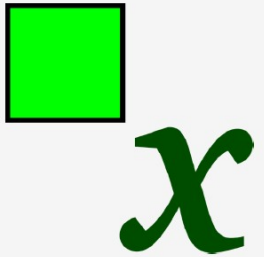
```
public class Komentarze{
  //poniżej rozpoczyna się działanie programu
  public static void main(String[] args){
    /*Ten tekst
      nie ma wpływu
      na program */
    System.out.print("Hello World");
  }
}
```

Ćwiczenie

Zadania do samodzielnego wykonania

Skompiluj powyższy przykład i zobacz jego działanie. Następnie umieść instrukcję wyświetlającą tekst w znaczniku komentarza i ponownie sprawdź jego działanie.

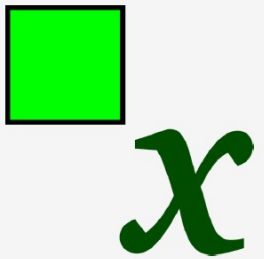
Co to jest zmienna?



Zmienna - konstrukcja programistyczna posiadająca **trzy** podstawowe **atrybuty**: symboliczną nazwę, miejsce przechowywania i wartość; pozwalająca w kodzie źródłowym odwoływać się przy pomocy nazwy do wartości lub miejsca przechowywania.

Nazwa służy do identyfikowania zmiennej w związku z tym często **nazywana jest identyfikatorem**. Miejsce przechowywania przeważnie znajduje się w pamięci komputera i określane jest przez adres i długość danych. Wartość to zawartość miejsca przechowywania.

Jak wygląda zmienna w Javie?

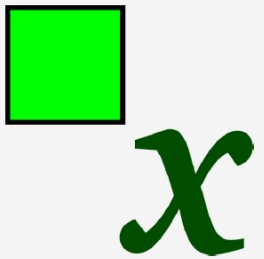


Zmienna w Javie składa się z następujących pól:

{akcesor*} {typ zmiennej} {identyfikator zmiennej}

```
private int liczba;
```

Jak wygląda zmienna w Javie?



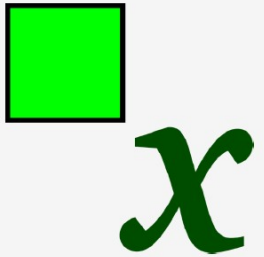
Wyróżniamy dwie fazy tworzenia zmiennej:

- **Deklaracja** - tutaj określamy typ i nazwę zmiennej
- **Inicjalizacja** - nadanie wartości zmiennej

```
public class Zmienne{
  public static void main(String[] args){

    int liczba; // Deklaracja

    liczba = 5; // Inicjalizacja
  }
}
```

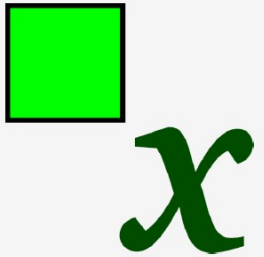
Zakres widoczności zmiennych

Zmienne "globalne" w obrębie klasy



```
class JakasKlasa{
 int zmienna_globalna = 5;

 void metoda1(){
    tutaj możesz użyć zmiennej globalnej
 }
}
```

Zakres widoczności zmiennych

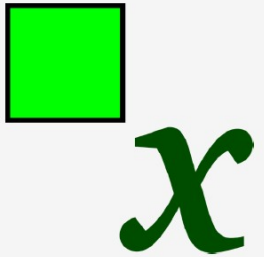


Zmienne „lokalne" w obrębie klasy

```
1  class JakasKlasa2{
2    void metoda(){
3        int zmienna_lokalna = 3;
4    }
5
6    int zmienna_globalna = zmienna_lokalna; //BŁĄD!
7  }
```

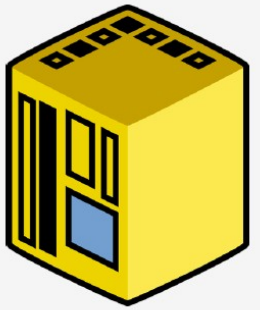
Zakres widoczności zmiennych

Deklaracja wewnątrz instrukcji sterujących lub jako licznik w pętli



```
1  class JakasKlasa3{
2    void metoda(){
3        if(warunek){
4           int a=5;
5        }
6     //tutaj zmienna "a" jest niewidoczna
7
8    for(int i=0; i<10; i++){  //zmienna obowiązuje tylko w pętli
9      System.out.println("Java");
10   }
11 }
```

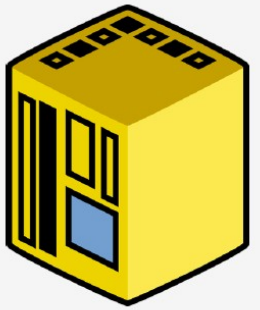
Typy danych



Co to jest typ danych?

Typ danych – opis rodzaju, struktury i zakresu wartości, jakie może przyjmować dany literał, zmienna, stała, argument, wynik funkcji lub wartość.

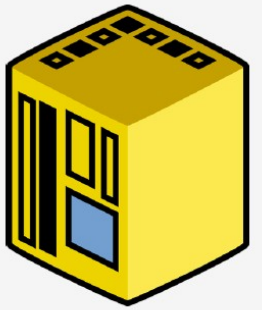
Rodzaje typów danych w Javie



W Javie wyróżniamy następujące typy danych:

- typ liczbowy
  - ✓ całkowity
  - ✓ zmiennoprzecinkowy
- typ logiczny
  - ✓ prawda
  - ✓ fałsz
- typ znakowy
- typ tekstowy

Rodzaje typów danych w Javie

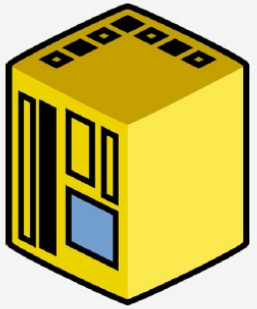


Java jest językiem programowania typowanym statycznie

- typy zmiennych nadawane są w czasie kompilacji programu
- łatwość wykrycia błędów w czasie kompilacji
- konieczność deklaracji typów zmiennych przed ich inicjalizacją

```
String liczba;

incompatible types: int cannot be converted to String
----
(Alt-Enter shows hints)

liczba = 2;
```

Podstawowe typy danych – liczby całkowite

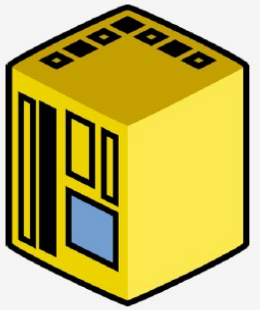


Byte - 1 bajt

- 8 bitów pojemności
- 2^8 = 256
- przechowuje liczby od -128 do 127

```
byte liczba = -128;
```

Podstawowe typy danych – liczby całkowite

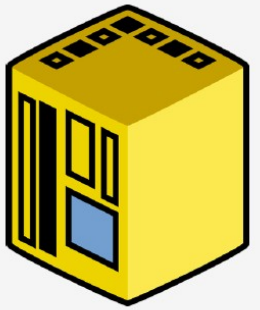


Short - 2 bajty

- 16 bitów pojemności
- 2^16 = 65535
- przechowuje liczby od -32768 do 32767

```
short liczba = 10000;
```

Podstawowe typy danych – liczby całkowite

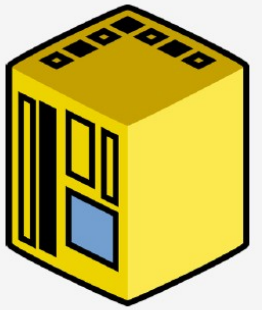


Int - 4 bajty

- 32 bitów pojemności
- 2^32 = 2147483468
- przechowuje liczby od -2 147 483 648 do 2 147 483 647
- najczęściej stosowany typ zmiennej liczbowej

```
int liczba = 100000000;
```

Podstawowe typy danych – liczby całkowite

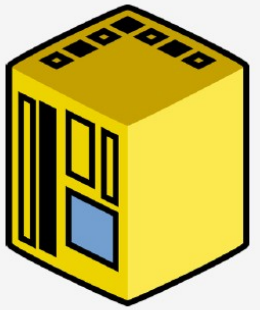


Long - 8 bajtów

- 64 bitów pojemności
- 2^64 = dużo ☺
- w sytuacji gdy musimy przechowywać naprawdę duże liczby np. do przechowywania identyfikatorów encji w bazie danych

```
long liczba = 25454554544554545 4L;
```

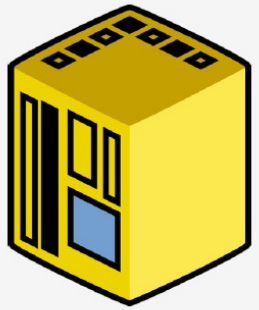
Podstawowe typy danych – liczby całkowite



Dodatkowo istnieją klasy osłonowe, które są obiektowymi odpowiednikami typów prostych. Udostępniają one metody, dzięki którym wiele rutynowych czynności mamy zawsze pod ręką.

Java **nie posiada** też typu **Unsigned** (bez znaku), czego konsekwencją jest to, że przekraczając zakres danego typu przejdziemy na zakres ujemny.

Podstawowe typy danych – liczby zmiennoprzecinkowe

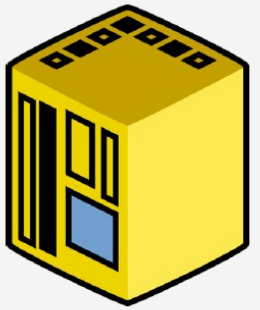


Float - 4 bajty

- 32 bity pojemości
- maksymalnie około 6-7 liczb po przecinku (posiadają przyrostek **F**, lub **f**)

```
float liczba = 1.0005f;
```

Podstawowe typy danych – liczby zmiennoprzecinkowe



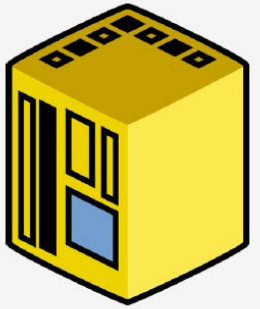
Double - 8 bajtów

- 64 bity pojemości
- maksymalnie około 15 cyfr po przecinku (posiadają przyrostek D, lub d)

```
double liczba = 1.0000000000000005d;
```


Prawa do korzystania z materiałów posiada Software Development Academy

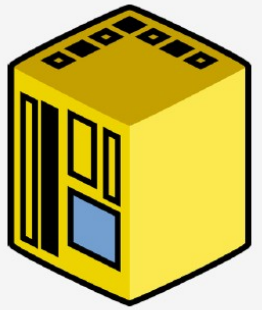
Podstawowe typy danych – liczby zmiennoprzecinkowe



Należy też pamiętać, że liczby zmiennoprzecinkowe nie nadają się do **obliczeń finansowych!**, w których liczy się dokładność. Spowodowane jest to tym, że w systemie dwójkowym nie da się przedstawić wszystkich liczb.

Z pomocą przychodzi tutaj specjalna klasa **BigDecimal**. Istnieje również klasa **BigInteger**, która jest odpowiednikiem dla liczb całkowitych i może reprezentować w zasadzie nieograniczone co do wielkości liczby.

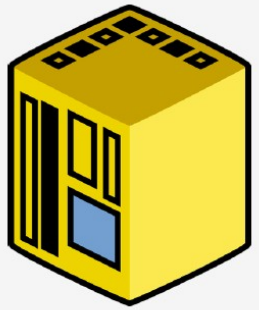
Wielkie liczby



Aby utworzyć nową zmienną typu **BigInteger** lub **BigDecimal** musimy najpierw utworzyć **obiekt**. W nagłówku trzeba oczywiście też zaimportować używaną klasę.

```java
import java.math.BigInteger;

public class WielkaLiczba{
  public static void main(String[] args){
    BigInteger wielkaLiczba = new BigInteger("123123123123123123");
    System.out.println(wielkaLiczba.toString());
  }
}
```

Funkcje matematyczne

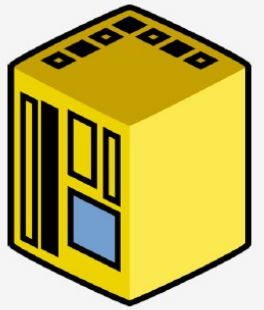


Kilka najważniejszych funkcji biblioteki **Math**

- *sqrt*(double liczba) - zwraca pierwiastek z liczby double. Jako parametr możemy również podać dowolny typ liczbowy, wtedy nastąpi jego automatyczna konwersja na double
- *pow*(double a, double b) - zwraca liczbę a podniesioną do potęgi b
- *abs*(liczba) - parametrem może być dowolna liczba, metoda zwraca wartość bezwzględną z argumentu

Funkcje matematyczne

Wszystkie metody są statyczne, aby je wywołać należy użyć konstrukcji

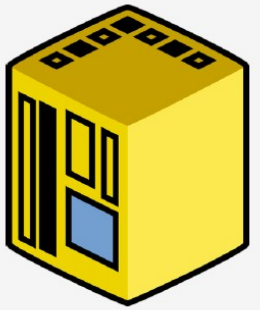


```
Math.nazwa_metody(argumenty)
```

Praktyczny przykład programu, który obliczy pierwiastek z liczby, a następnie podnosi ją do 3 potęgi.

```
public class Funkcje{
  public static void main(String[] args){
    double liczba = 9.0;
    int b = 3;
    double pierwiastek = Math.sqrt(liczba);
    double potega = Math.pow(liczba, b);

    System.out.println("Pierwiastek z "+liczba+" wynosi: "+pierwiastek);
    System.out.println("Liczba "+liczba+" podniesiona do potegi "+b+" to "+potega);
  }
}
```

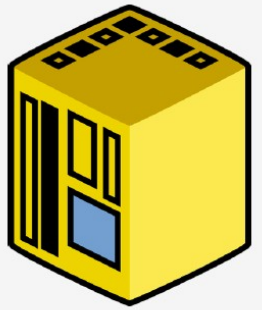
Funkcje matematyczne – import statyczny



Import statyczny - dzięki jego zastosowaniu będziemy mogli pomijać przedrostki Math przed nazwami funkcji. Poniżej powyższy przykład z jego zastosowaniem.

```
1  import static java.lang.Math.*;
2
3  public class Funkcje{
4    public static void main(String[] args){
5      double liczba = 9.0;
6      int b = 3;
7      double pierwiastek = sqrt(liczba);
8      double potega = pow(liczba, b);
9
10     System.out.println("Pierwiastek z "+liczba+" wynosi: "+pierwiastek);
11     System.out.println("Liczba "+liczba+" podniesiona do potegi "+b+" to "+potega);
12   }
13 }
```

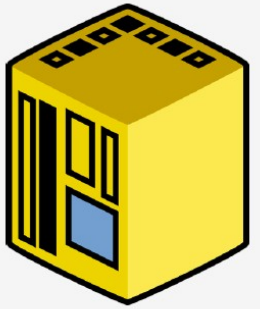
Funkcje matematyczne



W klasie Math występują także dwie **stałe PI** oraz **E**, dzięki nim nie musimy deklarować własnych liczb pi oraz e. Wywołujemy je podobnie jak funkcje.

```
1  Math.PI
2  Math.E
```

Podstawowe typy danych – typ logiczny



Boolean - reprezentuje on tylko dwie wartości

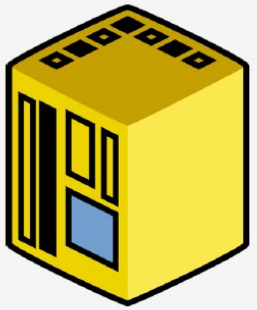
- **true** - prawda
- **false** – fałsz

Stosuje się jako wynik funkcji lub flagę, zwiększa czytelność kodu.

```
boolean logiczna = true;
```

```
boolean prawda = true;
boolean falsz = false;

boolean prawdaIFalsz = prawda && falsz;
boolean prawdaLubFalsz = prawda || falsz;
```

Podstawowe typy danych – typ znakowy

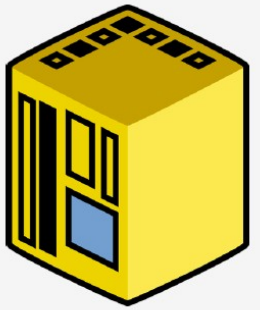


Char

- ujęty w pojedynczym cudzysłowie ,’
- służy do reprezentacji pojedynczych znaków kodu Unicode

```
char znak = 'a';
```

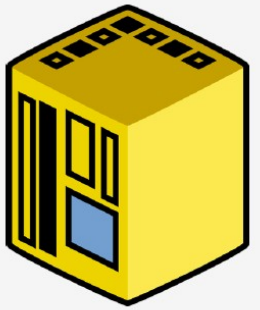
Podstawowe typy danych – typ znakowy



Znaki specjalne, które muszą być poprzedzone znakiem backslash **** :

- \t - tab
- \n - nowa linia
- \r - powrót karetki
- \" - cudzysłów
- \' - apostrof
- \\ - backslash

Podstawowe typy danych – typ tekstowy

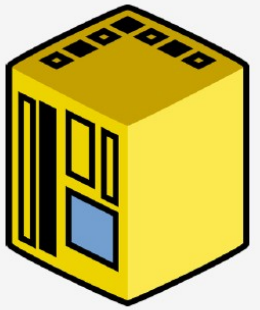


String

- służy do reprezentacji ciągu znaków kodu Unicode
- niezmienny stan obiektu (*immutable*)
- ujęty w podwójnym cudzysłowiu „”

```
String ciagZnakow = "ala ma kota";
```

Podstawowe typy danych – typ tekstowy



Konkantacja

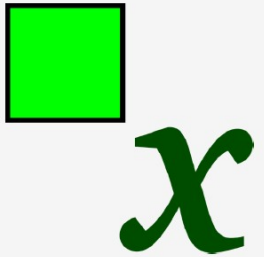
- dodawanie łańcuchów znaków do siebie

```
String rzeczownik = "Samolot";
String przymiotnik = "pasażerski";
String spacja =" ";

String opis = rzeczownik + spacja + przymiotnik;
```



```
Samolot pasażerski
```

Jak wygląda zmienna w Javie?



Program dodający 2 liczby całkowite:

```
public class Kalkulator{
  public static void main(String[] args){
    int a=5;
    int b=3;
    System.out.println("a+b = "+(a+b));
  }
}
```

Ćwiczenie

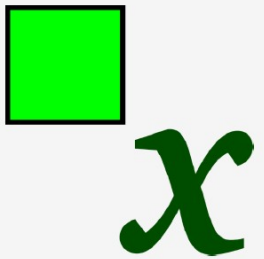
Zadania do samodzielnego wykonania

Przypisz do zmiennej c wynik dodawania a i b.

Postępując analogicznie utwórz klasy: *„Odejmowanie”, „Mnożenie” i „Dzielenie”*. Co się stanie jeśli podzielisz przez 0?

Jak wygląda zmienna w Javie?

Zmienne finalne - słowo kluczowe **final** umieszczane przed typem zmiennej



```
public class Stale{
  public static void main(String[] args){
    final double liczbaPi=3.14;
    final int finalna2;
    liczbaPi=8;     //błąd, zmienna była zainicjowana
    finalna2=3;    // tak można zrobić - pierwsze przypisanie
  }
}
```

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, w którym zadeklarujesz kilka zmiennych finalnych, lub zmiennych różnych typów o dowolnych nazwach, a następnie wyświetlisz je w

raz drugi wartości jakiejś zmiennej finalnej.

Ćwiczenie

Zadania do samodzielnego wykonania

W tym samym programie zadeklaruj cztery zmienne typu String. Trzy z nich zainicjuj jakimiś wyrazami a czwartemu przypisz ciąg znaków utworzony z trzech wcześniejszych zmiennych. Następnie wyświetl czwartą zmienną na ekranie.

Zapis i konwencja nazewnictwa

Nazwy klas rozpoczynamy Wielką literą.

```
1  class Klasa{ ... }
```

Nazwy metod rozpoczynamy z małej litery.

```
1  void metoda(){ ... }
```

Zapis i konwencja nazewnictwa

Nazwy zmiennych rozpoczynamy z małej litery.

```
1 | int zmienna;
```

Jeśli nazwa składa się z kilku wyrazów, to kolejne człony rozpoczynamy także z Wielkiej litery, co nazywamy notacją camelCase (rozpoczynamy z małej litery) oraz PascalCase (rozpoczynamy z wielkiej litery).

```
1 | class NazwaMojejKlasy{ ... }
2 | void toMojaMetoda(){ ... }
3 | int jakasZmienna;
```

Zapis i konwencja nazewnictwa

Nazwy stałych piszemy WIELKIMI literami, a kolejne człony oddzielamy znakiem podkreślenia.

```
1  static final int TO_JEST_STALA;
```

W Javie wielkość liter ma znaczenie!

```
1  int liczba;
2  int Liczba;
```

To dwie różne zmienne!

Pakiety

Co to jest pakiet w Javie?

Pakiet - Java nie jest monolitem, lecz składa się z szeregu klas definiujących obiekty różnego typu. Dla przejrzystości klasy te pogrupowane są w hierarchicznie ułożone pakiety.

Pakiety

Cechy pakietów:

- Przyjętą praktyką jest aby nazwy pakietu stanowiły odwrócone znaki domenowe np. *com.example.main*
- Każda klasa może być elementem pakietu oraz może należeć tylko do jednego pakietu
- Pakiety mogą zawierać podpakiety
- Pakiety mają na celu grupowanie klas
  - ✓ o podobnych funkcjach
  - ✓ współpracujących ze sobą
  - ✓ stanowiących samodzielny moduł (pod)systemu

package nazwaPakietu[.nazwaPodpakietu]*;

Ćwiczenie

Zadania do samodzielnego wykonania

Utwórz pakiet dla programu Dodawanie, Odejmowanie z poprzedniego zadania.
Nazwij go stosując odwrócona nazwę domenową.
Nadaj znaczącą nazwę końcówce pakietu, które określi jego przeznaczenie.
Przenieś tam utworzone klasy.

Operatory

Wyróżniamy w Javie następujące operatory:

- Przypisania
  - ✓ =, +=, -=, *=, /=
- Arytmetyczne
  - ✓ +,-,*,/,%,++,--
- Logiczne
  - ✓ &&, ||, !
- Relacji
  - ✓ <,<=,>,>=,==, !=

Operatory arytmetyczne

- **+** dodaje 2 liczby
- **-** odejmuje dwie liczby
- ***** znak mnożenia
- **/** dzielenie całkowite
- **%** reszta z dzielenia

```
1  public class Operatory{
2    public static void main(String[] args){
3        int a = 17;
4        int b = 4;
5        int c = a+b; //=21
6        c = a-b;     //=13
7        c = a*b;     //=68
8        c = a/b;     //=4 ponieważ 4*4=16 i zostaje reszty 1
9        c = a%b;     //=1 reszta z dzielenia
10   }
11 }
```

Operatory relacji

Operator	Zapis w kodzie
Równy	==
Nierówny	!=
Większy niż	>
Mniejszy niż	<
Większy bądź równy	>=
Mniejszy bądź równy	<=


```java
int a = 5;
int b = 3;
boolean prawda = a>b; //prawda=true
boolean falsz = a<b;  //falsz=false
boolean porownanie = a==b;  //porownanie=false
boolean koniunkcja = (a>b)&&(a!=b); //true prawda i prawda = prawda
```

Operatory logiczne

Operator logiczny	Zapis w kodzie
Negacja (NOT)	!
Koniunkcja (AND)	&&
Alternatywa (OR)	||

```
1  public class Test{
2    public static void main(String[] args){
3        String ja = "Slawek";
4        String ty = "Slawek";
5
6        boolean porownanie1 = ja==ty;
7        boolean porownanie2 = ja.equals(ty);
8        System.out.println(porownanie1);
9        System.out.println(porownanie2);
10   }
11 }
```

Ćwiczenie

Zadania do samodzielnego wykonania

Wypisz na ekran wartości poniższych wyrażeń logicznych

```
3 == 3 && 3 == 4
3 != 5 || 3 == 5
(2+4) > (1+3)
"cos".equals("cos")
"cos" == "cos"
```

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz prosty kalkulator, w którym będziesz przechowywał 3 zmienne typu double o nazwach a,b,c. Wypróbuj wszystkie operatory matematyczne:

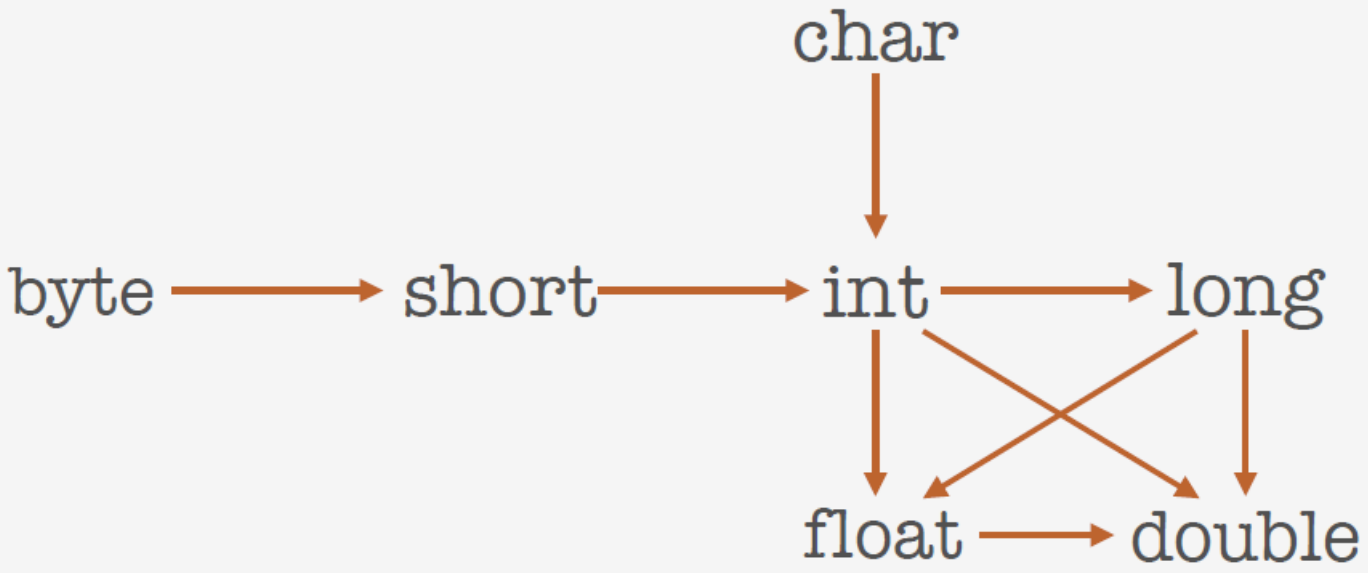
(a+b)*c

a-b/c

użyj operatorów inkrementacji i zwiększ wszystkie zmienne o 1.

Konwersje typów

Możliwe konwersje typów (zmiany typu A na typ B) bez ryzyka utraty danych.



Konwersje typów

Możliwe konwersje typów (zmiany typu A na typ B)

- typ o mniejszej pojemności konwertujemy na typ większej pojemności bitowej
- ogólna formuła:
  - ✓ *(typ_konwertowany) zmienna_mniejszej_pojemności*

```
byte sto = 100; //100
byte dwa = 2; //2
short stoDwa = (short) (sto + dwa); //102
```

Konwersje typów

- Jeśli jeden z operandów jest typu double, to drugi także zostanie zamieniony na double
- Lub, jeśli któryś z operandów jest typu float to drugi także zostanie zamieniony na float
- Lub, jeśli któryś z operandów jest typu long, drugi także zostanie zamieniony na long
- Gdy żadne z wyższych nie występuje oba typy zostaną zamienione na int

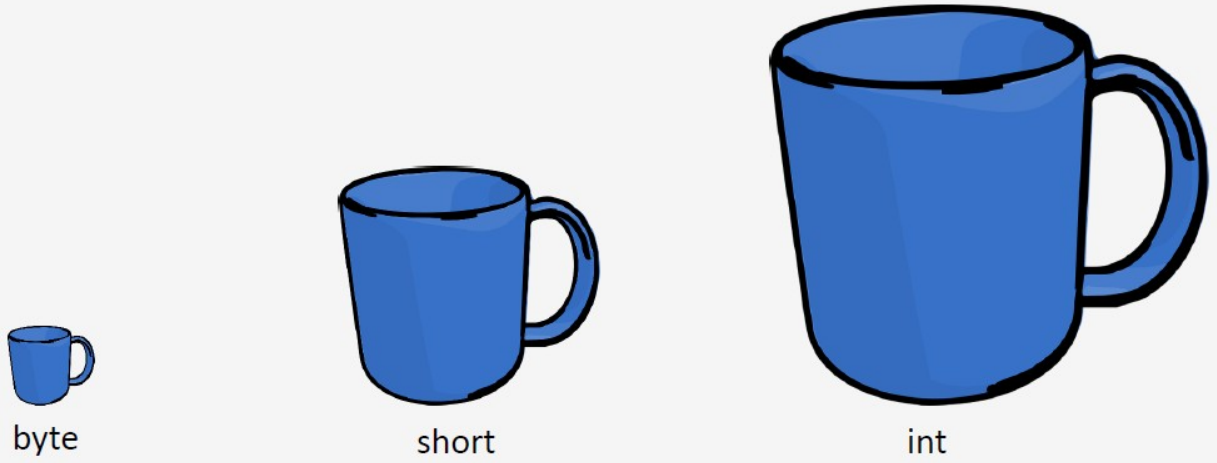
```
1  public class Konwersje{
2    public static void main(String[] args){
3        int a = 5;
4        double b = 13.5;
5        double c = b/a;
6        System.out.println(c);
7    }
8  }
```

```
1  public class Konwersje{
2    public static void main(String[] args){
3        int a = 5;
4        double b = 13.5;
5        int c = b/a;                //TAK NIE MOŻNA ZROBIĆ!
6        System.out.println(c);
7    }
8  }
```

Konwersje typów

Możliwe konwersje typów (zmiany typu A na typ B):

- Analogia do kubków



Ćwiczenie

Zadania do samodzielnego wykonania

Wykorzystaj konwersję i rzutowanie wypróbuj zamiany różnych typów prostych między sobą. Szczególną uwagę zwróć na rzutowanie char na int. Jak myślisz, co w ten sposób otrzymujesz?

Instrukcje sterujące

Co to jest instrukcja sterująca?

Instrukcja sterująca – instrukcja zdefiniowana w **składni** kreślonego języka programowania, umożliwiająca wyznaczenie i zmianę kolejności wykonania instrukcji zawartych w kodzie źródłowym.

Instrukcje sterujące – instrukcja warunkowa if

Wyrażenie if

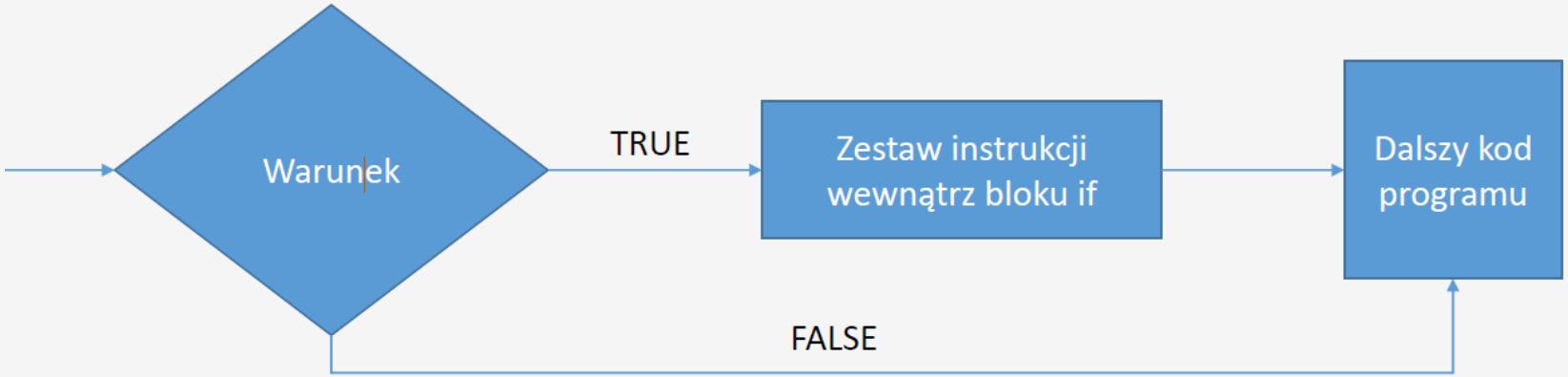
- Przy spełnieniu logicznego warunku wykonuje instrukcje znajdujące się wewnątrz bloku

```
boolean warunekLogiczny = true;

if(warunekLogiczny){
    //Wykonaj kod
}
```

Instrukcje sterujące – instrukcja warunkowa if

Wyrażenie if

- przebieg działania



Ćwiczenie

Zadania do samodzielnego wykonania

Napisać program, który sprawdza poniższe warunki i przy spełnieniu warunku wypisuje „ok”

a. 2 > 3, b. 4 < 5,

c. (2-2) == 0, d. true

e. 9%2 == 0, f. 9%3 == 0

Instrukcje sterujące – instrukcja warunkowa if

Wyrażenie if ... else

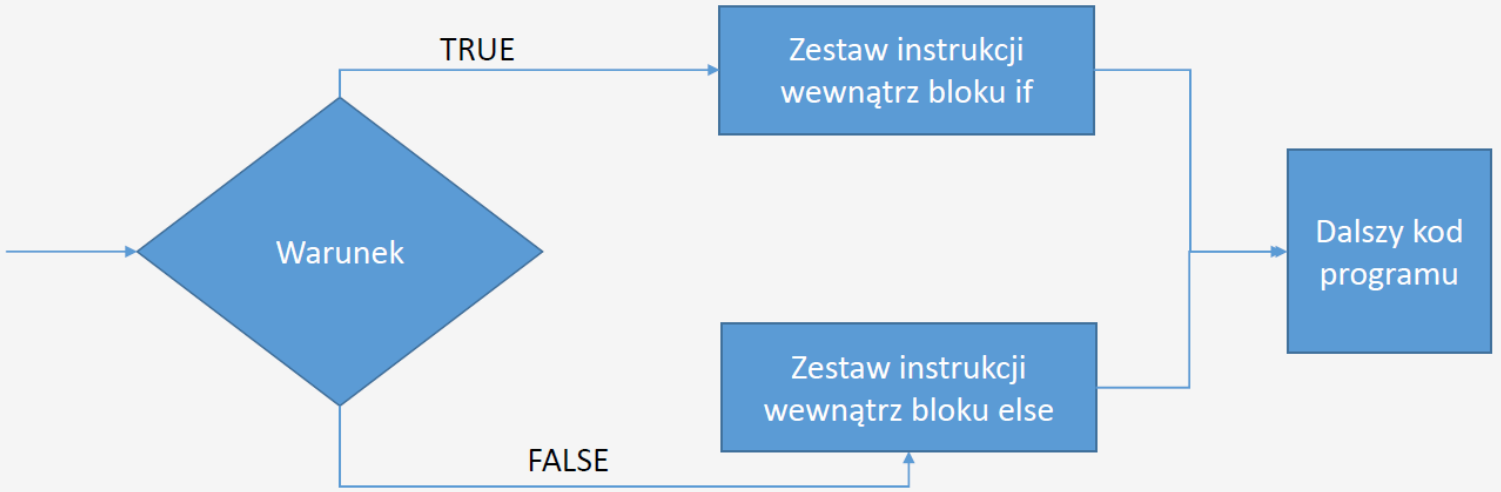
- Kod w bloku else jest wykonywany wtedy i tylko wtedy gdy warunek logiczny zadeklarowany w nawiasie if(..) nie zostaje spełniony

```
if(warunekLogiczny){
    //Wykonaj kod dla spełnionego warunku
}else{
    //Wykonaj kod dla niespełnionego warunku
}
```

Instrukcje sterujące – instrukcja warunkowa if

Wyrażenie if … else

- przebieg działania



Ćwiczenie

Zadania do samodzielnego wykonania

Rozbuduj poprzedni program o wypisywanie „błąd” w przypadku niespełnienia warunku

a.2 > 3, b.4 < 5

c.(2-2) == 0, d.true

e.9%2 == 0, f.9%3 == 0

Instrukcje sterujące – instrukcja switch

Instrukcja switch

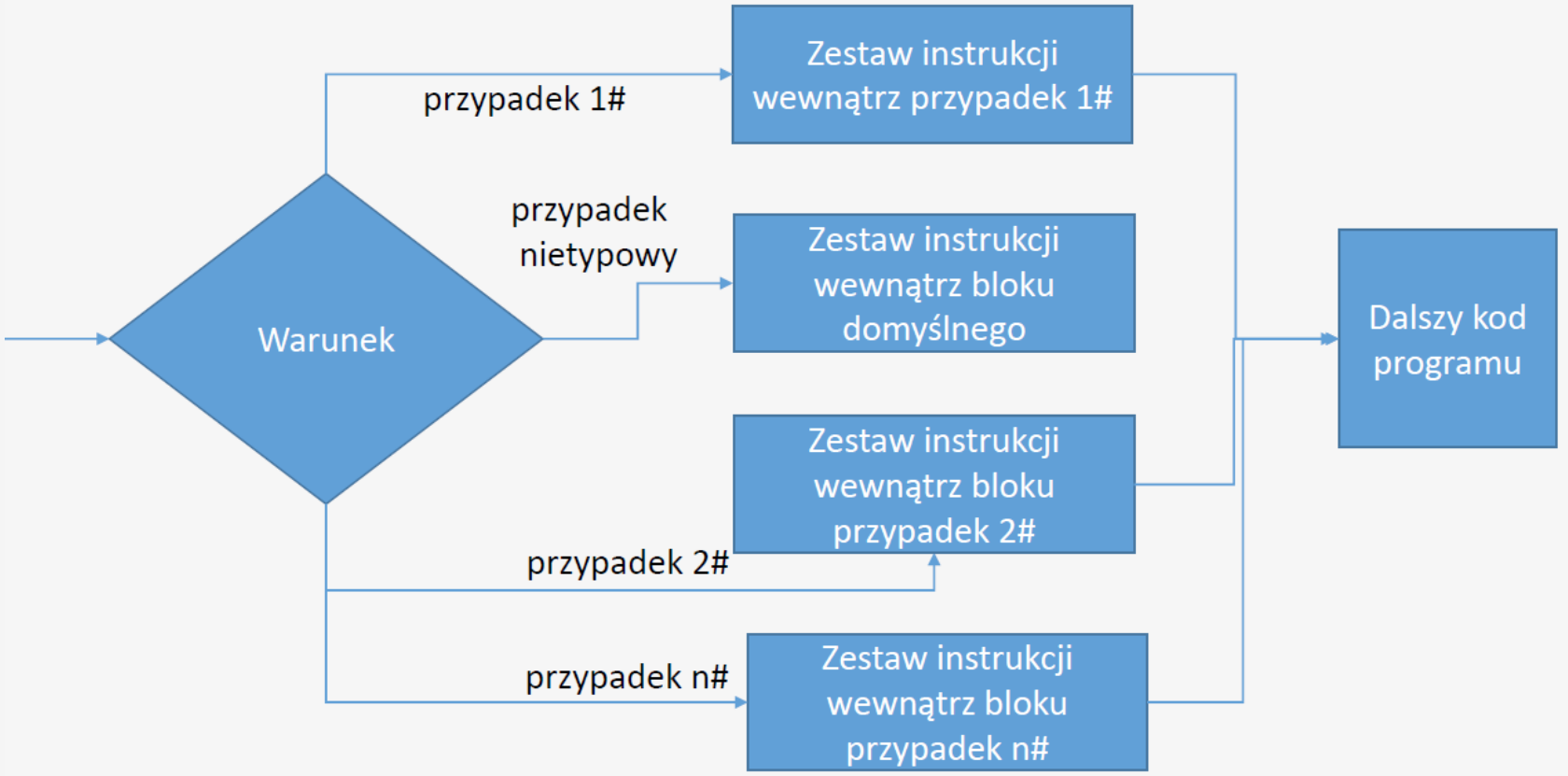
- Pokrywa wielokrotne wyrażenia if-else
- Składa się z wielu warunków
- Domyślny kod przy braku spełnienia pozostałych warunków
- Od Java SE7 możliwe jest użycie zmiennej typu String

```
int zmienna=1;

switch (zmienna){
    case 1:
        //wykonaj akcje dla przypadku 1.
        break;
    case 2:
        //wykonaj akcje dla przypadku 2.
        break;
    //... other cases
    default:
        //wykonaj wtedy i tylko wtedy
        //gdy poprzednie warunki nie zostały spełnione
        break;
}
```

Instrukcje sterujące – instrukcja switch

Wyrażenie switch

- przebieg działania



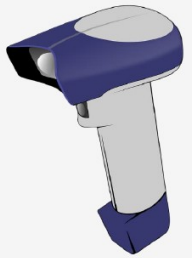
Podstawowe wejście / wyjście

Scanner

- Klasa umożliwiająca interakcję użytkownika z programem
- Korzysta z metod niestatycznych
- Jako argument konstruktora należy podać źródło strumienia danych jak np. klawiatura, czy plik tekstowy

Przydatne metody

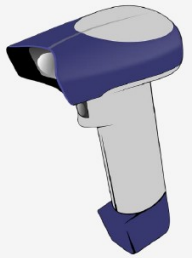
- nextLine():String –zwraca wpisaną linię tekstu (do wciśnięcia entera)
- nextInt():int–zwraca pierwszy napotkany int (jeśli przedzielimy spacją, to następne inty są ignorowane)



Podstawowe wejście / wyjście

Pobieranie danych od użytkownika

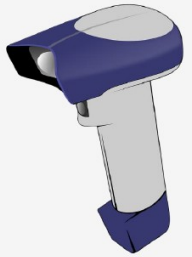
```
1  import java.util.Scanner;
2
3  public class Witaj{
4    public static void main(String[] args){
5        String imie; //w nim zapiszemy swoje imie
6        Scanner odczyt = new Scanner(System.in); //obiekt do odebrania danych od użytkownika
7
8        imie = odczyt.nextLine();
9
10       System.out.println("Witaj "+imie); //wyświetlamy powitanie
11   }
12 }
```



Podstawowe wejście / wyjście

Scanner oferuje również szereg innych metod do odczytu innych typów danych

- nextInt() - odczytuje kolejną liczbę całkowitą
- nextDouble() - czyta kolejną liczbę zmiennoprzecinkową (uwaga, separatorem może być tutaj zarówno kropka jak i przecinek - wszystko zależy od standardu kraju na jaki ustawiona jest maszyna wirtualna - można to ustawić za pomocą metody *useLocale(Locale)*)
- itp. istnieje także szereg przydatnych funkcji, które mogą być użyteczne na przykład przy odczycie plików



Ćwiczenie

Zadania do samodzielnego wykonania

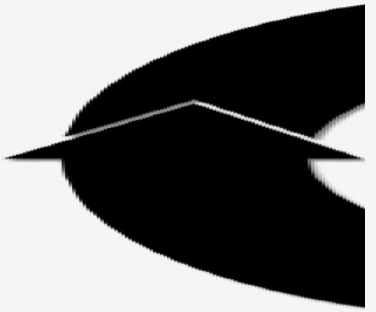
Utwórz dwie zmienne typu double.

Następnie przy użyciu klasy Scanner pobierz od użytkownika dwie liczby i wykonaj na nich dodawanie, odejmowanie, mnożenie i dzielenie, wyświetlając wyniki w kolejnych liniach na konsoli.

Pętle

Co to jest pętla w programie?

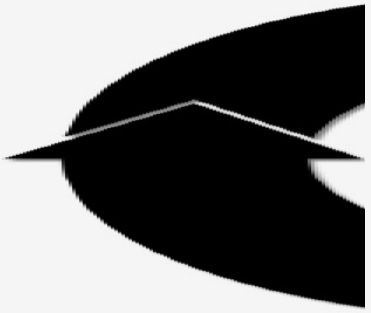
Pętla - umożliwia cykliczne wykonywanie ciągu instrukcji określoną liczbę razy, do momentu zajścia pewnych warunków, dla każdego elementu kolekcji lub w nieskończoność



Pętle

Cechy Pętli

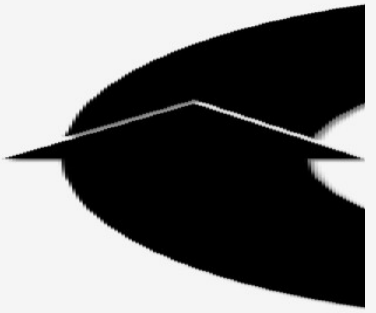
- Wykonują umieszczony blok kodu dopóki warunki są spełnione
- Kod wykonywany jest przez skończoną liczbę obiegów (pętli)
- Pętle które nie mają końca nazywamy pętlami nieskończonymi



Pętle – pętla for

Pętla for

```
int warunekWyjsciaZPetli =10;
int warunekPoczatkowy = 0;

for(int i= warunekPoczatkowy; i<warunekWyjsciaZPetli; i++){
    //instrukcje wewnatrz petli
}
```



Pętle – pętla for

Pętla for

```
int warunekWyjsciaZPetli =10;
int warunekPoczatkowy = 0;

for(int i= warunekPoczatkowy; i<warunekWyjsciaZPetli; i++){
    //instrukcje wewnatrz petli
}
```

Wykonywane raz przy inicjalizacji pętli

Pętle – pętla for

Pętla for

Sprawdzenie warunku:

dla wartości ‚true’ kod wewnątrz pętli jest wykonywany

dla wartości ‚false’ kod wewnątrz pętli nie jest wykonywany

```
int warunekWyjsciaZPetli =10;
int warunekPoczatkowy = 0;

for(int i= warunekPoczatkowy; i<warunekWyjsciaZPetli; i++){
    //instrukcje wewnatrz petli
}
```

Wykonywane raz przy inicjalizacji pętli

Pętle – pętla for

Pętla for

Sprawdzenie warunku:

dla wartości ‚true' kod wewnątrz pętli jest wykonywany

dla wartości ‚false' kod wewnątrz pętli nie jest wykonywany

```
int warunekWyjsciaZPetli =10;
int warunekPoczatkowy = 0;

for(int i= warunekPoczatkowy; i<warunekWyjsciaZPetli; i++){
    //instrukcje wewnatrz petli
}
```

Wykonywane raz przy inicjalizacji pętli

Po wykonaniu pętli, aktualizacja licznika

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, który pobierze od użytkownika całkowitą liczbę dodatnią.

Następnie przy użyciu wyświetl na ekranie Odliczanie z tekstem "Bomba wybuchnie za ... " gdzie w miejsce dwukropka mają się pojawić liczby od podanej przez użytkownika do 0. Napisz program przy użyciu pętli for.

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, który przyjmuje od użytkownika dzielnik oraz liczbę, a następnie drukuje na ekranie wszystkie liczby mniejsze od zadanej liczby podzielne przez zadany dzielnik.

Napisz program wyznaczający potęgę liczby n i m –obie zadane przez użytkownika i drukujący w czytelny sposób wynik na ekranie konsoli.

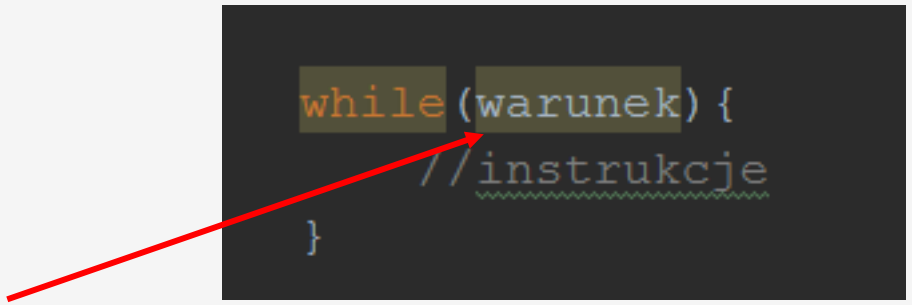
Pętle – pętla while

Pętla while

```
while (warunek) {
    //instrukcje
}
```

Pętle – pętla while

Pętla while



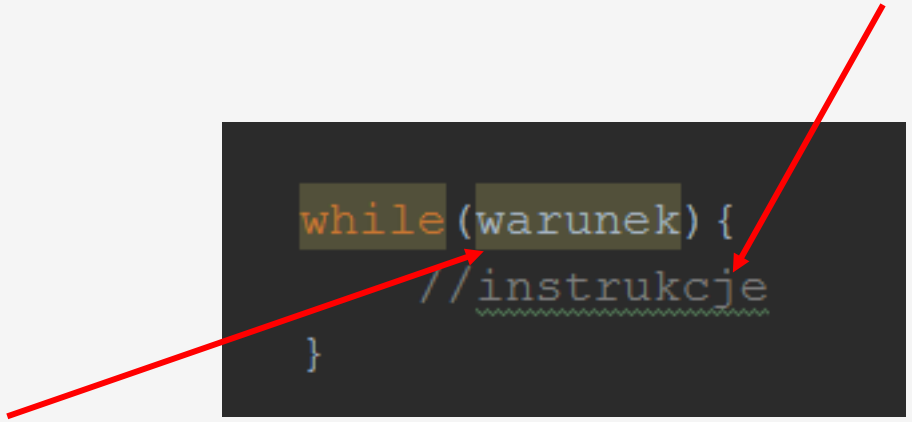
Instrukcje są wykonywane dopóki warunek logiczny ma wartość ‚true’

Pętle – pętla while

Pętla while

Aby wyjść z pętli należy

- sprawić aby warunek zmienił wartość na false
- dodać słowo kluczowe „break” wewnątrz instrukcji



Instrukcje są wykonywane dopóki warunek logiczny ma wartość „true”.

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, który pobierze od użytkownika całkowitą liczbę dodatnią.

Następnie przy użyciu wyświetl na ekranie Odliczanie z tekstem "Bomba wybuchnie za ... " gdzie w miejsce dwukropka mają się pojawić liczby od podanej przez użytkownika do 0. Napisz program przy użyciu pętli while.

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, który oblicza sumę wszystkich liczb poprzedzających zadaną przez użytkownika liczbę –dla liczby 100 będzie to suma liczb od 0 do 100 czyli 5050.

Napisz program, który oblicza silnię dla zadanej liczby przez użytkownika (do n=12) korzystając z pętli while https://pl.wikipedia.org/wiki/Silnia

Pętle – pętla do while

Pętla do while

```
boolean warunek = true;

do{
  //instrukcje
}while(warunek);
```

Pętle – pętla do while

Pętla do while

```
boolean warunek = true;

do{
  //instrukcje
}while(warunek);
```

Blok do wykonywany jest w 1 kroku, niezależnie od wartości warunku.

Pętle – pętla do while

Pętla do while

Następnie sprawdzany jest warunek logiczny w przypadku braku spełnienia pętla jest przerywana

```
boolean warunek = true;

do{
  //instrukcje
}while(warunek);
```

Blok do wykonywany jest w 1 kroku, niezależnie od wartości warunku.

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, który pobierze od użytkownika całkowitą liczbę dodatnią.

Następnie przy użyciu wyświetl na ekranie Odliczanie z tekstem "Bomba wybuchnie za ... " gdzie w miejsce dwukropka mają się pojawić liczby od podanej przez użytkownika do 0. Napisz program przy użyciu pętli do while.

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, który wypisuje „Hello World” zadaną ilość razy, dopóki użytkownik wpisuje liczby większe od 0.

Napisz program, który oblicza wartość pierwiastka z wprowadzonej przez użytkownika liczby, dopóki ta przyjmuje wartości większe od 0 (dla uproszczenia przyjmij że użytkownik wprowadza liczby całkowite).

Tablice

Co to jest tablica w Javie?

Tablica - kontener uporządkowanych danych takiego samego typu, w którym poszczególne elementy dostępne są za pomocą kluczy (indeksu). Rozmiar tablicy ustalany jest z góry.

Tablice

Cechy Tablic

- Przechowują elementy tego samego typu w sposób uporządkowany
- Wielkość tablicy jest deklarowana przy jej **inicjalizacji**
- Wielkość tablicy pozostaje niezmienna
- Do elementów tablicy odwołujemy się przez indeks, liczony od 0 do (n-1), gdzie n -to ilość elementów w tablicy

Tablice

Deklaracja tablicy

```
String[] pasazerowie;
```

Pusta tablica znaków

Tablice

Inicjalizacja tablicy

```
pasazerowie = new String[4];
```

Tablice

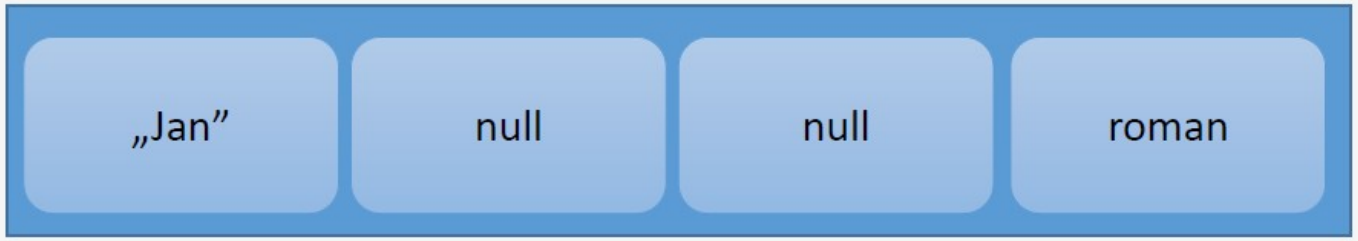
Przypisywanie elementów do tablicy

```
pasazerowie[0] = "Jan";
```

Tablice

Przypisywanie elementów do tablicy

```
String roman = "Roman";

pasazerowie[3] = roman;
```

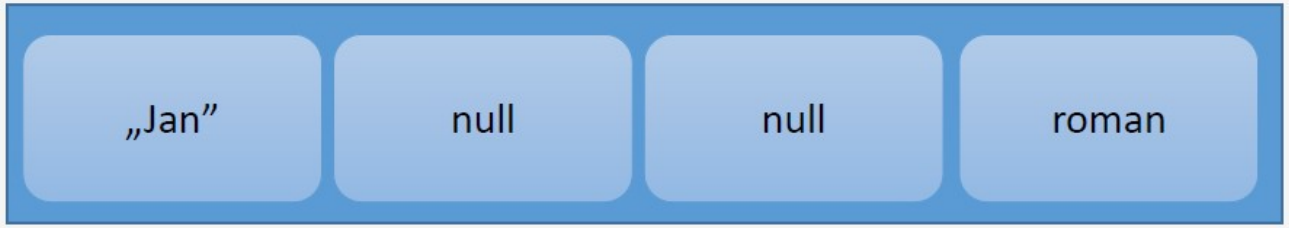


Tablice

Używanie elementów z tablicy

```
System.out.println(pasazerowie[0]);
```

→ Jan

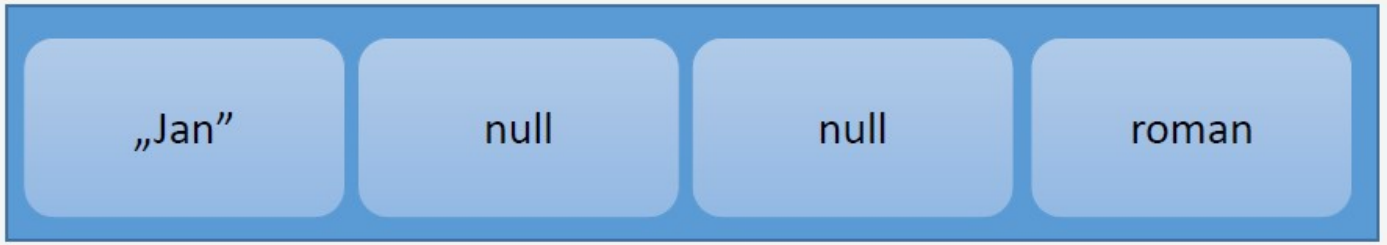


Tablice

Używanie elementów z tablicy

```
String drugiPasazer = pasazerowie[1];
System.out.println(drugiPasazer);
```

null



Tablice

Określanie wielkości tablicy

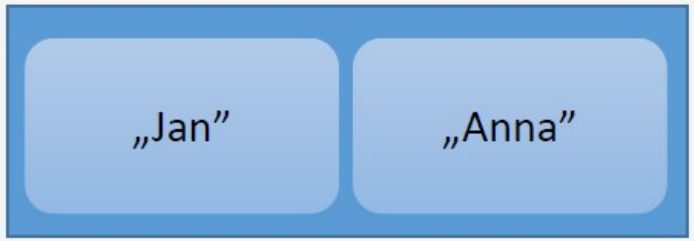
```
System.out.println(pasazerowie.length);
```

→ 4

Tablice

Inicjalizacja tabeli wraz z elementami

```
String[] pasazerowie = new String[]{"Jan", "Anna"};
```



Ćwiczenie

Zadania do samodzielnego wykonania

Napisz program, w którym zadeklarujesz i utworzysz pięcioelementową tablicę odpowiedniego typu. W pętli pobierzesz od użytkownika 5 imion i je w niej zapiszesz. Następnie wyświetl na ekranie powiadomienia "Witaj imie_z_tablicy" dla każdego z podanych parametrów.

Ćwiczenie

Zadania do samodzielnego wykonania

Utwórz tablicę typu int przechowującą n elementów- gdzie n jest parametrem

zawartość na ekranie przy pomocy pętli while.

Ćwiczenie

Zadania do samodzielnego wykonania

1. Utwórz tablicę liczb {1,3,5,10}
2. Wypisz wszystkie elementy po kolei
3. Wypisz elementy w pętli
4. Wypisz tylko liczby o parzystym indeksie
5. Wypisz tylko liczby parzyste
6. Wypisz elementy w odwróconej kolejności

Tablice wielowymiarowe

Schematyczna deklaracja

```
typ[][] nazwa_tablicy; //deklaracja
nazwa_tablicy = new typ[liczba1][liczba2]; //przypisanie (utworzenie)
typ[][] nazwa_tablicy2 = new typ[liczba1][liczba2]; //deklaracja i przypisanie (utworzenie)
```

Odwołanie do elementów

```
int[][] tablica = new int[3][3];
tablica[2][1] = 5;
int zmienna = tablica[2][1];
```

Ćwiczenie

Zadania do samodzielnego wykonania

Wyświetl tablicę wyglądającą następująco:

0 0 0 0 0
0 0 0 0 0
1 1 1 1 1
0 0 0 0 0
0 0 0 0 0

Ćwiczenie

Zadania do samodzielnego wykonania

Wyświetl tablicę wyglądającą następująco:

```
0 0 0 0 0
0 1 0 0 0
0 0 2 0 0
0 0 0 3 0
0 0 0 0 4
```

Ćwiczenie

Zadania do samodzielnego wykonania

Przy użyciu pętli i tablic przechowujących liczby całkowite zaprezentuj poniższą treść:

tab[0,0] = 0; tab[0,1] = 1;

tab[0,2] = 2; tab[1,0] = 3;

tab[1,1] = 4; tab[1,2] = 5;

Wykorzystuj przy tym własność length.

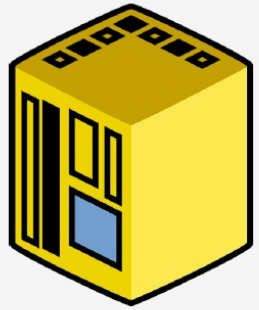
Zapis i odczyt z plików

Podstawowa klasa, która pozwoli nam utworzyć obiekt przechowujący dane pliku to **File**.

```
1 | File plik = new File("nazwa_pliku.txt");
```

Aby utworzyć strumień, należy użyć konstrukcji:

```
1 | Scanner odczyt = new Scanner(new File("nazwa_pliku.txt"));
```



Ćwiczenie

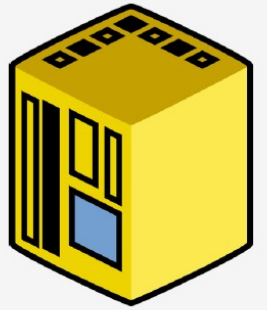
Zadania do samodzielnego wykonania

Utwórz plik, w którym zapiszesz jedną dowolną linię tekstu, przykładowo "Ala ma kota, bo nie wzięła leków". Zapisz go jako *ala.txt*.

Zapis i odczyt z plików

Kod programu do odczytania tego zdania wyglądałby następująco

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Scanner;
4
5  public class Odczyt{
6    public static void main(String[] args) throws FileNotFoundException{
7        File file = new File("ala.txt");
8        Scanner in = new Scanner(file);
9
10       String zdanie = in.nextLine();
11       System.out.println(zdanie);
12   }
13 }
```



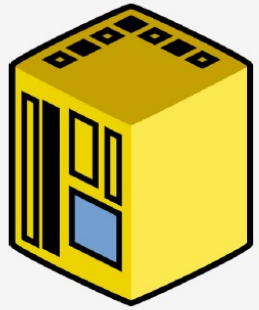
Zapis i odczyt z plików

Zapis do pliku

```
1 | PrintWriter zapis = new PrintWriter("nazwa_pliku.txt");
```

Zapiszemy do pliku *ala.txt* zdanie "Ala ma kota, a kot ma Alę" przy pomocy metody ***print()***.

```
1  import java.io.FileNotFoundException;
2  import java.io.PrintWriter;
3
4  public class Zapis{
5    public static void main(String[] args) throws FileNotFoundException{
6        PrintWriter zapis = new PrintWriter("ala.txt");
7        zapis.println("Ala ma kota, a kot ma Alę");
8        zapis.close();
9    }
10 }
```



Ćwiczenie

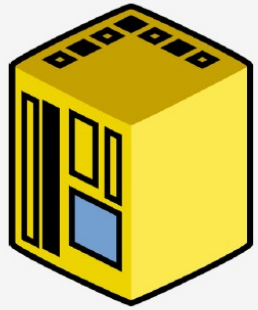
Zadania do samodzielnego wykonania

Napisz program, w którym wprowadzisz w konsoli swoje imię, następnie zapiszesz je do pliku. Odczytaj je z powrotem z pliku i bez użycia dodatkowej zmiennej wyświetl na ekranie.

Metody o zmiennej liczbie argumentów (varargs)

Od piątej wersji Javy, mamy do dyspozycji mechanizm pozwalający na tworzenie metod o zmiennej ilości argumentów, tzw. **varargs**, bez konieczności wykorzystywania tablic przechowujących te argumenty.

```
[typ] metoda(typ ArgumentStały, typ...GrupaArgumentow)
```



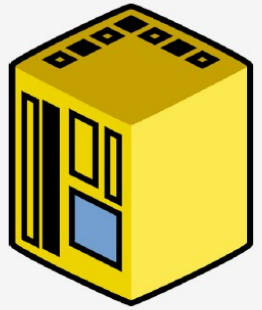
Metody o zmiennej liczbie argumentów (varargs)

Przykładowa implementacja

```
1  public int suma(int arg0, int...args) {
2
3      int wynik = arg0;
4
5      for(int i=0; i<args.length; i++) {
6
7          wynik += args[i];
8      }
9      return wynik;
10 }
```

Przykładowe wywołanie

```
1  System.out.print(wypisz(3) + " ");
2  System.out.print(wypisz(3,3) + " ");
3  System.out.print(wypisz(3,3,3) + " ");
4  System.out.print(wypisz(3,3,3,3,3,3) + " ");
```

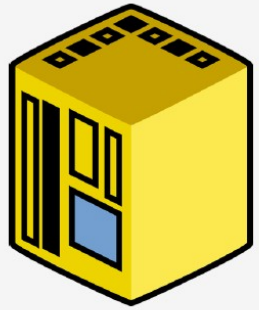


Obiekt

- **Obiekt** - odzwierciedlenie rzeczywistej rzeczy np. mieszkanie, pokój, pralka ,samochód
- **Pole obiektu** – pojedyncza informacja o obiekcie np. kolor, waga, wielkość, cena
- **Metoda** – czynności, jakie możemy wykonać na danym obiekcie np. otwarcie, uruchomienie, hamowanie

```
1  specyfikator_dostepu typ_Obiektu nazwaObiektu = new typ_Obiektu();
```

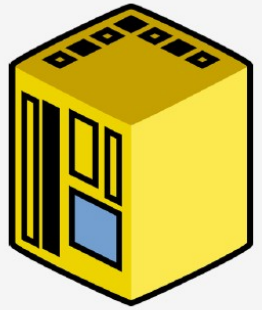
Typem obiektu jest oczywiście klasa, którą on reprezentuje.



Wyjątki - blok try catch

Schemat

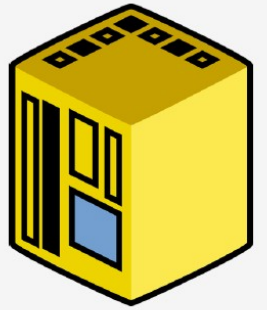
```
try{
    kod programu mogący generować wyjątki
}
catch (TypWyjątku1 a){ Obsługa wyjątku a }
catch (TypWyjątku2 b){ Obsługa wyjątku b }
...
finally{ Blok instrukcji, który wykona się niezależnie, czy wyjątki wystąpią, czy nie }
```



Wyjątki - blok try catch

Zastosowanie

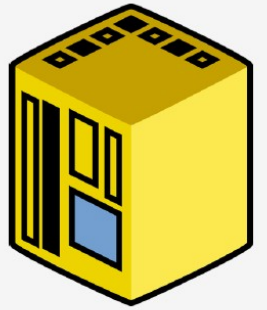
```
1  import java.util.Scanner;
2
3  public class Odczyt{
4    public static void main(String[] args){
5        int tab[] = {1,2,3,4,5};
6        Scanner odczyt = new Scanner(System.in);
7        int index = -1;
8
9        System.out.println("Podaj indeks tablicy, który chcesz zobaczyć: ");
10       index = odczyt.nextInt();
11
12       try {
13           System.out.println(tab[index]);
14       } catch (ArrayIndexOutOfBoundsException e) {
15           System.out.println("Niepoprawny parametr, rozmiar tablicy to: "+tab.length);
16       }
17   }
18 }
```



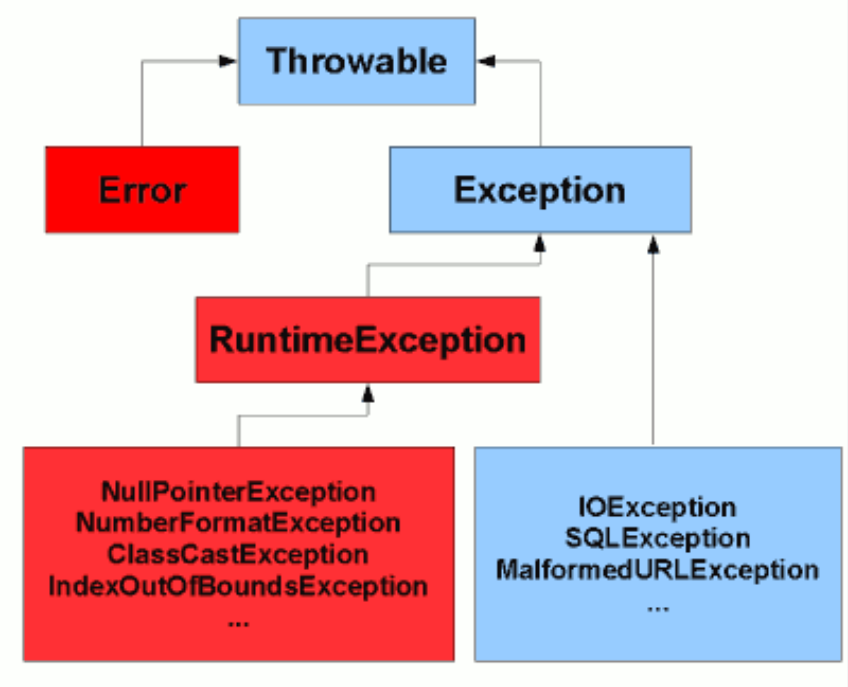
Zgłaszanie wyjątków - instrukcja throw

Schematycznie użycie instrukcji **throw** wygląda następująco:

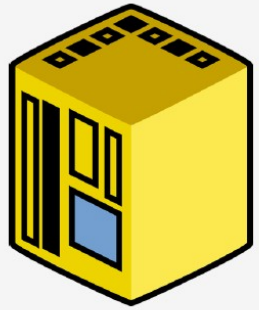
```
[static][final] nazwa_metody() <strong>throws</strong> typ_wyjątku1, typ_wyjątku2, ... {
                kod...
                <strong>throw</strong> new typ_wyjatku1();
                kod
...
```



Hierarchia wyjątków



Na powyższym schemacie można wyróżnić wyjątki nie wymagające obsługi (uncaught - czerwone), oraz te, które musimy zgłosić, lub obsłużyć w jeden z poznany przez nas sposobów (niebieskie).

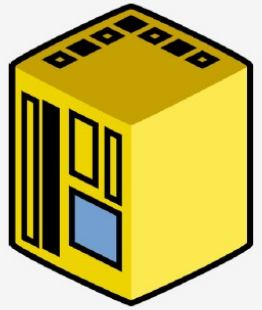


Zgłaszanie wyjątków - instrukcja throw

może zgłosić wyjątek typu *ArithmeticException*

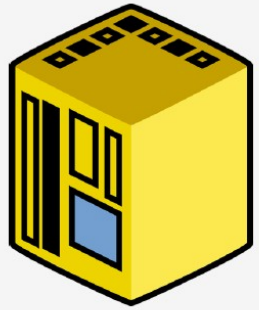
```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] agrs) throws ArithmeticException{
5          int x=10;
6          int y;
7          Scanner sc = new Scanner(System.in);
8          System.out.print("Podaj dzielnik: ");
9          y = sc.nextInt();
10         if(y==0)
11             throw new ArithmeticException("Nie mozna dzielić przez 0");
12         else
13             System.out.println(x/(double)y);
14     }
15 }
```

zgłoszenie wyjątku



Klasa a obiekt

- **Obiekt** to instancja klasy
- **Klasa** to „szablon” opisu jak ma wyglądać obiekt. Pokazuje, co będzie zawierał utworzony obiekt ale nie daje informacji jakie wartości będą przechowywane w polach
- **Obiekt** to „konkretny przypadek” danej klasy – klasa oraz konkretne informacje, które składają się na stan obiektu
- **Klasy** mogą zawierać zarówno metody jak i zmienne, klasę rozpoczyna słowo kluczowe *class*, a te z kolei poprzedza specyfikator dostępu

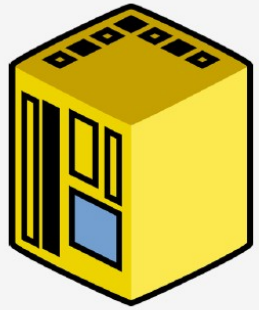


Podstawowa klasa

```
1  public class Hello{
2    public static void main(String[] args){
3      System.out.print("Hello World");
4    }
5  }
```

Schematycznie wygląda to tak

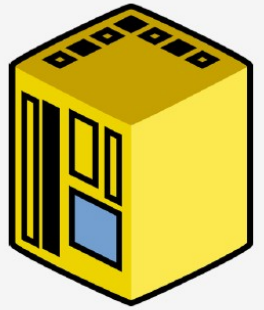
```
1  specyfikator_dostepu class NazwaKlasy{
2
3  specyfikator_dostepu typ_zwracany nazwaMetody(parametry_po_przecinku){
4    jeśli typ zwracany inny niż void to trzeba użyć słówka return i zwrócić wynik
5  }
6  }
```



Klasa z właściwością / polem

Klasa posiadające pole / właściwość

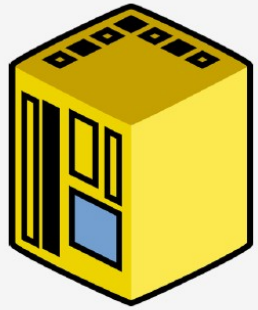
```
public class Pojazd {
    int liczbaKol;
    String kolor;

    public static void main(String[] args) {
        System.out.println("test");
    }
}
```



Gettery i settery

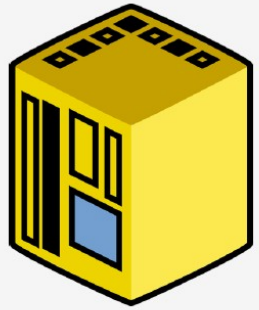
ograniczony dostęp do pola

```
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public int getLiczbaKol(){
    return liczbaKol;
  }

  public void setLiczbaKol(int liczbaKol){
    this.liczbaKol = liczbaKol;
  }

}
```



Konstruktor

- Każda klasa, nawet abstrakcyjna, musi posiadać **konstruktor**
- Nazwa konstruktora jest taka sama jak nazwa klasy
- Inicjalizuje zmienne instancyjne obiektu
- Jest wywoływany podczas tworzenia obiektu poprzez operator *new*
- Klasa może mieć **kilka konstruktorów**
- Jeśli w kodzie klasy nie ma konstruktora, **domyślny konstruktor** zostanie wygenerowany przez kompilator

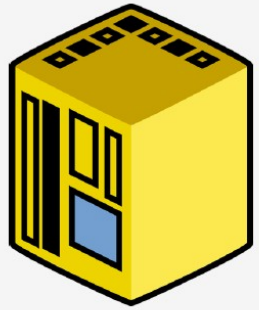


Konstruktor bezargumentowy

W konstruktorze znajdują się operacje wykonywane w trakcje inicjalizowania obiektu.

konstruktor bezargumentowy

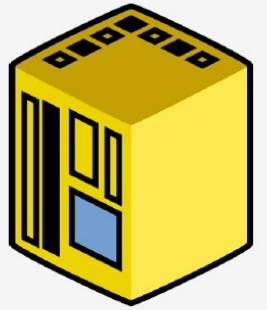
```
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public Pojazd() {
  }

  public int getLiczbaKol(){
    return liczbaKol;
  }

  public void setLiczbaKol(int liczbaKol){
    this.liczbaKol = liczbaKol;
  }

}
```



Konstruktor z argumentem

konstruktor z argumentem

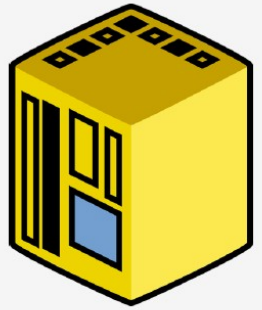
```
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public Pojazd(int liczbaKol) {
    this.liczbaKol = liczbaKol;
  }

  public int getLiczbaKol(){
    return liczbaKol;
  }

  public void setLiczbaKol(int liczbaKol){
    this.liczbaKol = liczbaKol;
  }

}
```



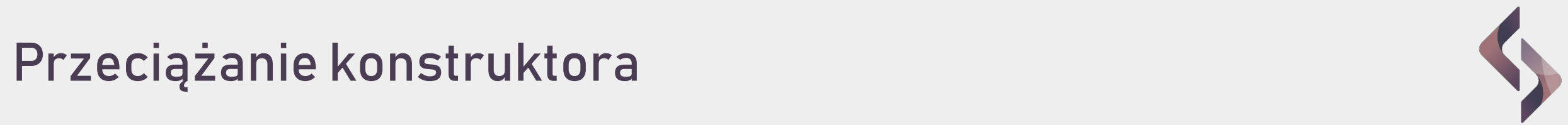
Konstruktor z argumentami

konstruktor z argumentami

```java
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public Pojazd(int liczbaKol, String kolor) {
    this.liczbaKol = liczbaKol;
    this.kolor = kolor;
  }

  public int getLiczbaKol(){
    return liczbaKol;
  }

  public void setLiczbaKol(int liczbaKol){
    this.liczbaKol = liczbaKol;
  }

}
```

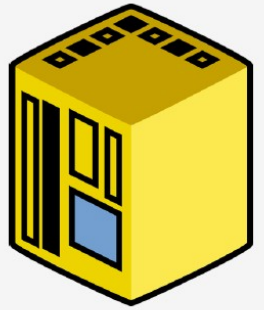


Przeciążanie konstruktora



```java
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public Pojazd() {
  }

  public Pojazd(int liczbaKol) {
    this.liczbaKol = liczbaKol;
  }

  public Pojazd(int liczbaKol, String kolor) {
    this.liczbaKol = liczbaKol;
    this.kolor = kolor;
  }

  public int getLiczbaKol(){
    return liczbaKol;
  }
```

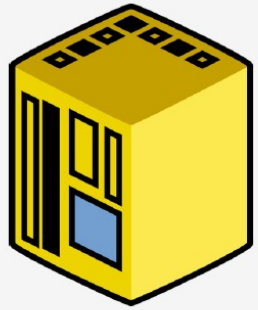
konstruktory



Tworzenie obiektu

utworzenie obiektu

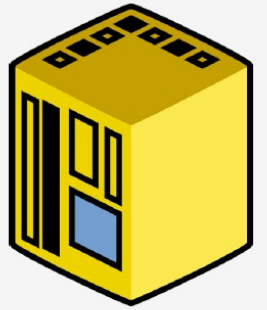
```
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public String getKolor() {
    return kolor;
  }

  public static void main(String[] args) {
    Pojazd pojazd = new Pojazd();
    System.out.println(pojazd.getKolor());
  }

}
```



Tworzenie obiektu

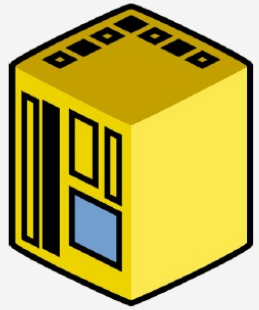
utworzenie obiektu

```
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public String getKolor() {
    return kolor;
  }

  public Pojazd(String kolor) {
    this.kolor = kolor;
  }

  public static void main(String[] args) {
    Pojazd pojazd = new Pojazd("czerwony");
    System.out.println(pojazd.getKolor());
  }

}
```



Konwencje nazewnictwa

- Klasy, interfejsy -z wielkiej litery, następnie „camelCase”
- Metody - z małej litery, następnie „camelCase”
- Zmienne - tak jak metody
- Stałe - wielkimi literami używając „_” jako separatora słów
- Notacja JavaBeans - getXXX(), setXXX(), isXXX()

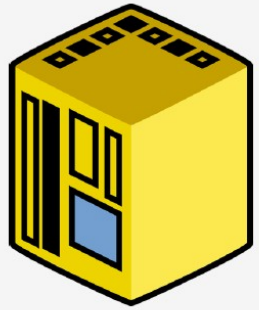


Metoda statyczna

Metoda, którą można wywołać bez utworzenia obiektu. Wywoływana jest poprzez identyfikator

Klasy.nazwaMetody();

```
public class Pojazd {

  private int liczbaKol;
  private String kolor;

  public static void jedz() {
  }

  public static void main(String[] args) {
    Pojazd.jedz();
  }

}
```



Ćwiczenie

Zadania do samodzielnego wykonania

Klasa Pojazd powinna zawierać trzy pola: liczbaKol, kolor oraz predkosc.

Utwórz konstruktor z wszystkimi argumentami.

Utwórz nowy obiekt, w konsoli wyświetl: „Pojazd ma 4 koła, jest czerwony i może jechać 200km/h”.

Zmodyfikuj klasę Pojazd, jeśli w konstruktorze nie zostanie podana prędkość, domyślna wartość powinna wynosić 100.

Ćwiczenie

Zadania do samodzielnego wykonania

Napisz klasę Pracownik, która przechowuje trzy pola: Imię, Nazwisko, Wiek.

Następnie utwórz klasę Firma, w której wykorzystasz klasę pracownik do utworzenia dwóch obiektów przechowujących dane pracowników (wymyśl sobie jakieś) i później wyświetlasz je na ekran.

Zmodyfikuj powyższy program tak, aby utworzyć trzech pracowników, a odpowiednie pola zainicjuj wartościami z wcześniej utworzonych tablic (dowolne dane) przy użyciu pętli.

Ćwiczenie

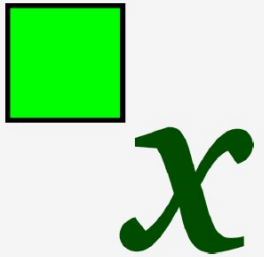
Zadania do samodzielnego wykonania

Napisz program, który będzie się składał z dwóch klas:

Pracownik- przechowująca takie dane jak imię, nazwisko i wiek pracownika, oraz co najmniej trzy konstruktory, które posłużą do inicjowania obiektów z różnymi parametrami- w przypadku gdy przykładowo konstruktor przyjmuje tylko 1 parametr, zainicjuj pozostałe pola jakimiś domyślnymi wartościami.

Firma- klasa testowa, w której utworzysz kilka obiektów klasy Pracownik i wyświetlisz dane na ekran.

Argumenty metod

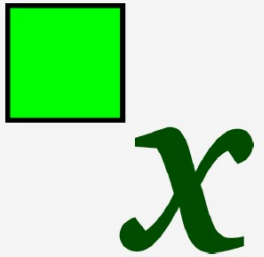


Typy proste jako argumenty:

```
1  String metodaNapis(String str){
2    System.out.println(str);
3    return str;
4  }
5
6  int sumaLiczb(int a, int b, int c){
7    return a+b+c;
8  }
9
10 double sumaLiczb2(int a, short b, double c){
11   return a+b+c;
12 }
```

- wyświetla napis podany jako parametr i go zwraca
- oblicza sumę trzech liczb typu int
- oblicza sumę trzech liczb różnych typów

Argumenty metod



Typy proste jako argumenty:

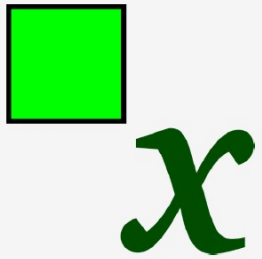
- Kolejne argumenty mogą być różnych typów
- Kolejne argumenty oddzielamy przecinkiem
- Zmiany dokonane na argumentach nie wpływają na jego oryginalną wartość!

```
1  void wyswietl(int liczba){
2    liczba++;
3    System.out.println(liczba);
4  }
```

Argumenty metod

Typy obiektowe:

- Argumenty w języku Java są przekazywane przez wartość, ale taką wartością są też referencje (jakaś liczba).
- Jeśli przekażemy jakiś obiekt jako argument metody i zmodyfikujemy go w jej wnętrzu, to obiekt też zostanie zmodyfikowany, ponieważ operujemy na tym samym obszarze pamięci (kopii referencji wskazującej na ten sam obiekt).

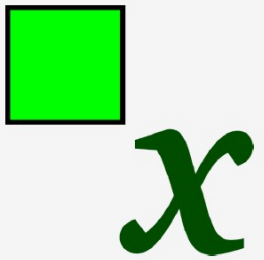


Argumenty metod

```java
public class Punkt {
    int x;
    int y;
}
```

```java
public class Test {
    static void zmien(Punkt pkt){
        pkt.x++;
        pkt.y++;
    }
}
```

```java
public class Main{

    public static void main(String args[]){
        Punkt punkt = new Punkt();
        punkt.x = 5;
        punkt.y = 5;

        Test.zmien(punkt);

        System.out.println("Współrzędne to: "+punkt.x+" "+punkt.y);
    }
}
```



Ćwiczenie

Zadania do samodzielnego wykonania

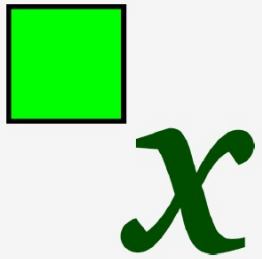
Utwórz klasę Punkt, która przechowuje dwie wartości typu int- współrzędne punktu na powierzchni. Napisz w niej także metody które: zwiększają wybraną współrzędną o 1, zmieniają wybraną zmienną o dowolną wartość, zwracają wartość współrzędnych (oddzielne metody), wyświetla wartość współrzędnych.

Napisz także klasę, w której przetestujesz działanie metod wyświetlając działanie metod na ekranie.

Przeciążanie metod

Pozwala to najprościej mówiąc na tworzenie metod o takich samych nazwach, ale różnych parametrach. Analogicznie nieprawidłowe jest utworzenie w jednej klasie dwóch metod o identycznej nazwie i przyjmującej takie same parametry, a także metody o takiej samej nazwie i parametrach, ale różniące się **tylko** zwracanym typem.

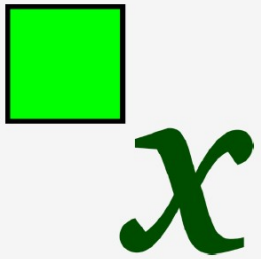
```
1  class Test{
2    int dodaj(int a, int b){
3      return a+b;
4    }
5
6    double dodaj(double a, double b){
7      return a+b;
8    }
9  }
```



Przeciążanie metod

Niepoprawna implementacja!

```
1  class Test2{
2    int dodaj(int a, int b){
3      return a+b;
4    }
5
6    double dodaj(int a, int b){
7      return a+b;
8    }
9  }
```



Ćwiczenie

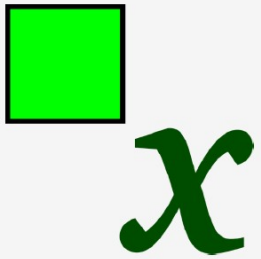
Zadania do samodzielnego wykonania

Napisz program składający się z dwóch klas. Pierwsza niech zawiera kilka metod o nazwie dodaj(), ale zwracających różne typy wyników i przyjmujących po minimum dwa parametry typów liczbowych wybranych przez Ciebie. Ich zadaniem jest zwrócenie, lub wyświetlanie sumy podanych argumentów. W drugiej klasie Testowej utwórz obiekt tej klasy i sprawdź działanie swoich metod, wyświetlając wyniki działań na ekranie. Dodatkowo każda z metod niech wyświetla swój typ zwracany i rodzaj argumentów, abyś wiedział, która z nich zadziałała.

Modyfikatory dostępu

Określenie w jaki sposób inne obiekty mogą otrzymać dostęp do danego pola, metody, definicji klasy czy interfejsu

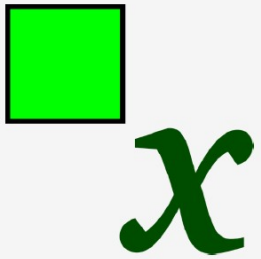
- Public – widoczny i dostępny wszędzie
- Protected – dostępny w danym pakiecie oraz w klasach dziedziczących
- Private – niewidoczny poza obszarem definicji właściciela
- Default(package private) – widoczny w danym pakiecie



Dziedziczenie

Domyślnie, każda klasa w języku Java dziedziczy po klasie Object. Dzięki temu, w każdej klasie dostępne są metody

- clone()
- equals()
- finalize()
- getClass()
- hashCode()
- notify()
- notifyAll()
- toString()
- wait()

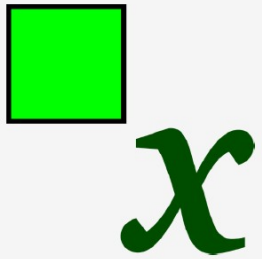


Dziedziczenie

W języku Java w przeciwieństwie do innych nie występuje dziedziczenie wielokrotne. To znaczy, że klasa potomna może rozszerzać tylko jedną klasę bazową. Projektanci Javy uznali, że mechanizm taki jest prostszy i nie wprowadza niepotrzebnego chaosu.

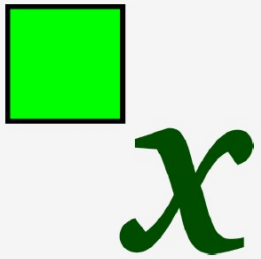
Aby rozszerzyć jakąś klasę należy użyć słowa kluczowego **extends** w nagłówku klasy:

```
1  public class Szef extends Pracownik{
2  ...
3  }
```



Dziedziczenie

```java
class Pracownik{
    String imie;
    String nazwisko;
    int wyplata;

    public Pracownik(){
        imie = "";
        nazwisko = "";
        wyplata = 0;
    }

    public Pracownik(String i, String n, int w){
        imie = i;
        nazwisko = n;
        wyplata = w;
    }
}
```



Ćwiczenie

Zadania do samodzielnego wykonania

Stwórz klasę Punkt2D, która przechowuje informacje na temat punktu na przestrzeni dwuwymiarowej (współrzędne x oraz y). Zawierająca dwa konstruktory: bezparametrowy ustawiający pola na wartość 0, oraz przyjmujący dwa argumenty i ustawiający pola obiektu zgodnie z podanymi parametrami.

Napisz klasę Punkt3D dziedziczącą po Punkt2D, reprezentującą punkt w trójwymiarze (dodatkowe pole z).

W klasie testowej utwórz obiekty obu klas i przetestuj działanie.

Ćwiczenie

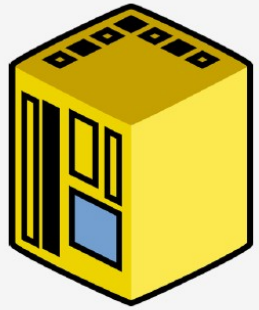
Zadania do samodzielnego wykonania

Proszę napisać klasę Circle rozszerzającą klasę Shape. Nowo utworzona klasa powinna zawierać metodę obliczającą pole powierzchni.

Wzór na pole koła: PI*R^2

Klasy abstrakcyjne

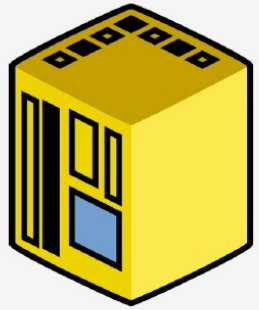
- Mogą zawierać metody abstrakcyjne, czyli takie, które nie posiadają implementacji (ani nawet nawiasów klamrowych)
- Może zawierać stałe (zmienne oznaczone jako public static final)
- Mogą zawierać zwykłe metody, które niosą jakąś funkcjonalność, a klasy rozszerzające mogą ją bez problemu dziedzic
- Klasy rozszerzające klasę abstrakcyjną muszą stworzyć implementację dla metod oznaczonych jako abstrakcyjne w klasie abstrakcyjnej
- Metod abstrakcyjnych nie można oznaczać jako statyczne (nie posiadają implementacji)
- Podobnie jak w przypadku interfejsów nie da się tworzyć instancji klas abstrakcyjnych



Klasy abstrakcyjne

Poprawna implementacja

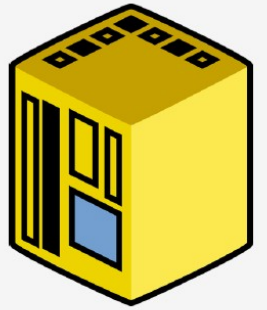
```java
public abstract class Emeryt {
    public static final int ILOSC_OCZU = 2; //stałe są ok

    //metoda abstrakcyjna
    public abstract String krzyczNaDzieci();

    //zwykła metoda z implementacją
    public static void biegnijDoSklepu(int odleglosc, int predkosc) {
        double czas = (double)odleglosc/predkosc;
        System.out.println("Biegne po kiełbase bede za "+czas);
    }
}
```



Klasy abstrakcyjne

Błędna implementacja!

```
1  public abstract class Emeryt {
2      public static final int ILOSC_OCZU = 2; //stałe są ok
3
4      //metoda abstrakcyjna nie może być statyczna
5      public abstract static String krzyczNaDzieci();
6
7      //metoda abstrakcyjna nie może posiadać implementacji
8      //ani nawet nawiasów klamrowych
9      public abstract void biegnijDoSklepu() {}
10 }
```



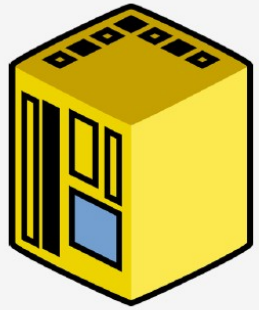
Enum

Pozwala on na definiowane wybranego zbioru możliwych wartości.

```
1  public enum Kolor {
2      CZERWONY, ZIELONY, NIEBIESKI;
3  }
```

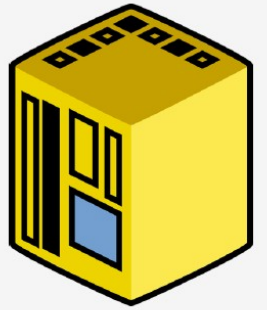
Własny konstruktor

```
1  public enum Kolor {
2
3      CZERWONY(false),
4      ZIELONY(true),
5      NIEBIESKI(true);
6
7      boolean ladny;
8
9      private Kolor(boolean czyLadny) {
10         ladny = czyLadny;
11     }
12 }
```



Enum - przykładowe zastosowanie

```
1  public class EnumTest {
2      public enum Kolor {
3
4          CZERWONY(false),
5          ZIELONY(true),
6          NIEBIESKI(true);
7
8          boolean ladny;
9
10         private Kolor(boolean czyLadny) {
11             ladny = czyLadny;
12         }
13     }
14
15     public static void main(String[] args) {
16
17         Kolor kolor = EnumTest.Kolor.CZERWONY;
18         System.out.println("Kolor czerwony jest "+czyLadny(kolor));
19
20         kolor = EnumTest.Kolor.ZIELONY;
21         System.out.println("Kolor zielony jest "+czyLadny(kolor));
22
23     }
24
25     public static String czyLadny(Kolor kolor) {
26         return (kolor.ladny) ? "ladny" : "brzydki";
27     }
28
29 }
```



Struktury danych – Jednokierunkowa lista wiązana

W Javie wyróżniamy dwa rodzaje listy:

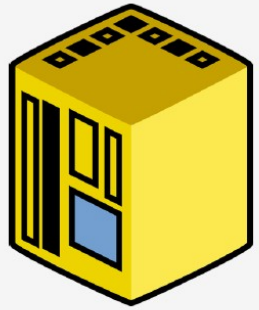
- **Listy wiązane**
- Listy tablicowe

wg innych kryteriów mogą one być:

- jednokierunkowe
- dwukierunkowe

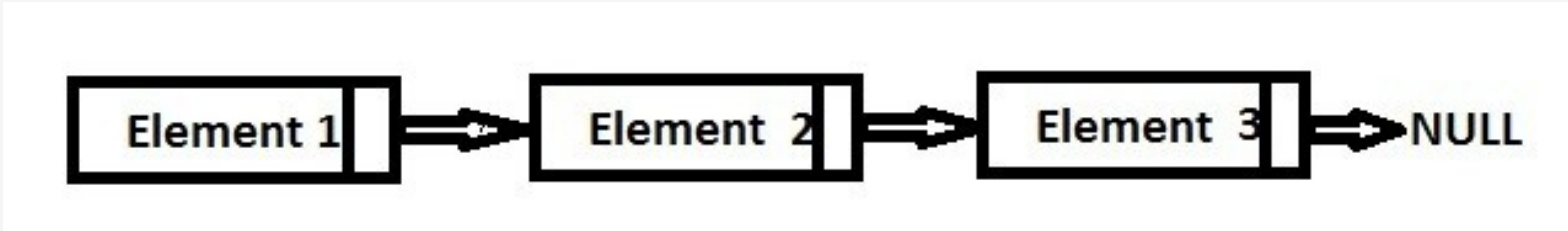
a także:

- cykliczne
- acykliczne

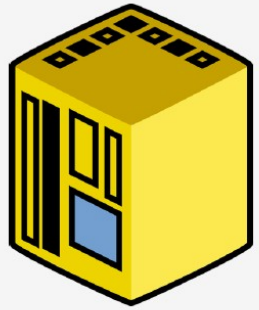


Struktury danych – Jednokierunkowa lista wiązana

Acykliczna jednokierunkowa lista wiązana

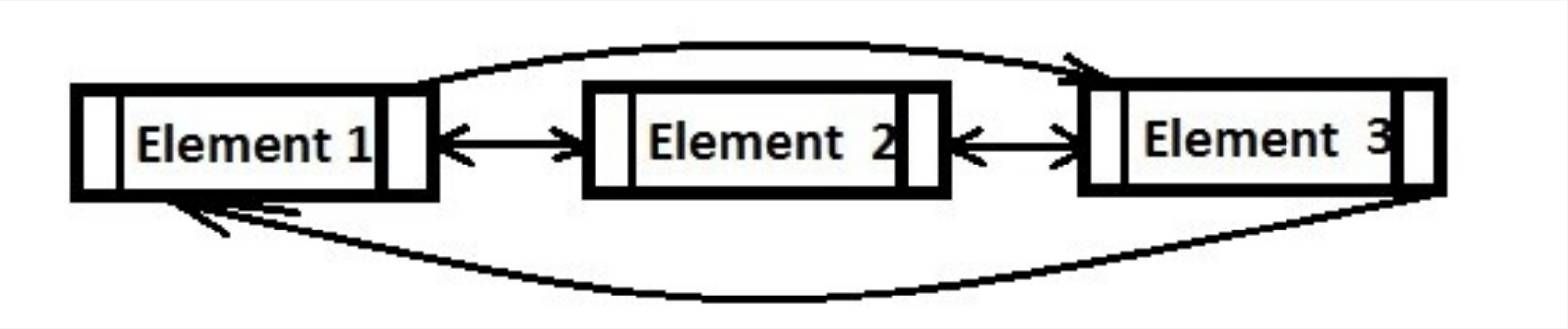


Ostatni element acyklicznej jednokierunkowej listy wskazuje na wartość **null**, która zwana jest głową, dzięki czemu przy jej przechodzeniu wiemy kiedy doszliśmy do końca.

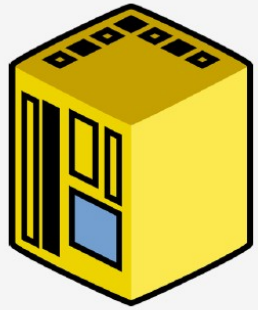


Struktury danych – Jednokierunkowa lista wiązana

Dwukierunkowa lista cykliczna



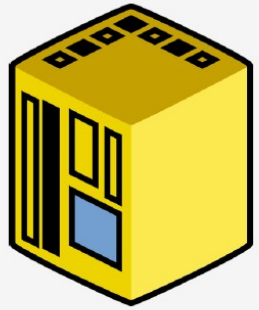
Przeglądając taką listę możemy poruszać się w obie strony i nie występuje tutaj zakończenie listy w postaci wartości **null**.



Struktury danych – Jednokierunkowa lista wiązana

Podstawowe metody jakie powinna mieć zaimplementowana lista to:

- add() - dodaje element do listy
- delete() - usuwa element przechowujący obiekt podany jako parametr
- get() - zwraca wartość obiektu z elementu listy o wskazanym indeksieset() - ustawia obiekt w elemencie o podanej pozycji
- size() - zwraca rozmiar listy
- isEmpty() - zwraca true, lub false, gdy lista jest, lub nie jest pusta - najlepiej wykorzystać metodę size()
- clear() - czyści listę. W przypadku listy z wartownikiem wystarczy ustawić jego następnik na wartość null



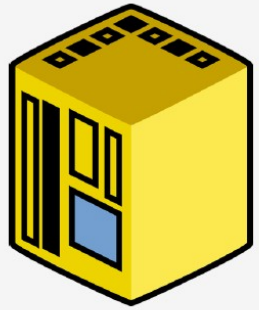
Struktury danych – Mapa

Mapa - obiekt mapujący wartości parametrów na podstawie klucza. W danej mapie może istnieć tylko jeden unikatowy klucz. Szeroko stosowany w komunikacji między aplikacjami, dzięki możliwości przypisanie wartości do stałych, ustalonych parametrów.

Od Javy 7 nie trzeba już deklarować podwójnie typów (diamond operator):
Map<String, Integer> Oceny = new HashMap<>();

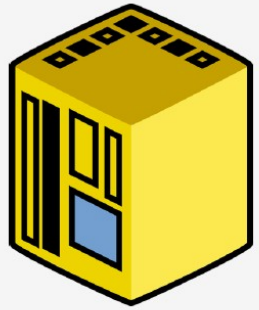
```
1  Map<String, Integer> Oceny = new HashMap<String, Integer>();
2  Oceny.put("Matematyka", 5);
3  Oceny.put("Polski", 2);
```

Mapy stosowane są przy przekazywaniu parametrów za pomocą metod HTTP GET oraz POST, oraz bardzo popularnym formacie JSON (JavaScript Object Notation).



Struktury danych – Stos

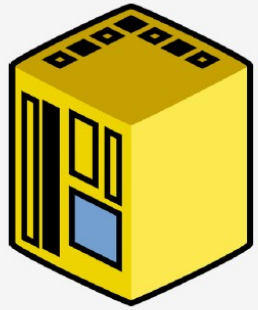
Stos - jest to liniowa struktura danych, wykorzystywana często w informatyce. Głównym założeniem stosu, jest strategia LIFO - Last In First Out. W wolnym tłumaczeniu polega to na tym, aby ostatnio położony element, był z niego w pierwszej kolejności zdejmowany.



Polimorfizm

- Z Greki „poly” (wiele), „morphe” (forma, postać, kształt)
- Oznacza, że dana referencja może mieć dostęp do wielu różnych form

```
Vehicle car = new Car();
car.go();              // go() z Car
car = new Sedan();
car.go();              // go() z Sedan
```

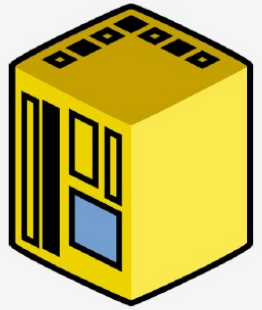


Polimorfizm

- Zmienna referencyjna może mieć tylko jeden typ i nie można go zmienić
- Typ referencji(zmiennej) określa jakie metody mogą być na niej wywoływane
- Typ obiektu określa, która przesłonięta metoda zostanie wywołana
- Typ referencji określa, która przeciążona metoda zostanie wywołana

```
Car myCar = new Sedan(„Mazda”);
```

typ referencji — zmienna referencyjna — typ obiektu



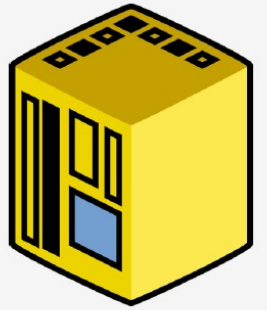
Polimorfizm

Przypisując obiekt B do referencji typu A mamy dostęp **tylko to metod, które znajdują się w typie referencji**, w tym przypadku interfejsu A.

```java
public interface A {
    public void x();
}
```

```java
public class B implements A{
    @Override
    public void x() {
        //do something
    }

    public void y() {
        //do something
    }
}
```

```java
public class Test {
    public static void main(String[] args) {
        A obiekt = new B();
        obiekt.x();
        obiekt.y(); //błąd, interfejs A nie posiada metody y()
        ((B)obiekt).y(); //ok, dzięki rzutowaniu uzyskujemy dostęp do metody y()
    }
}
```



Ćwiczenie

Zadania do samodzielnego wykonania

Korzystając z polimorfizmu i dziedziczenia, utwórz co najmniej po jednym obiekcie typu Circle oraz Square – typ referencji powinien wskazywać na Shape.

Na każdym obiekcie wywołaj metodę area();

Utwórz listę obiektów typu Shape, dodaj do niej kilka obiektów typu Cirle oraz Square. Wypisz w konsoli nazwy klas elementów umieszczonych na liście.

Java – podstawowe typy danych

Typy podstawowe

- byte
- short
- int
- long
- float
- double
- boolean
- char
- void

Boxing

Can't be null!

Typy obiektowe

- Byte
- Short
- Integer
- Long
- Float
- Double
- Boolean
- Character
- Void

Unboxing

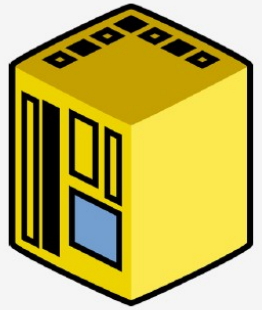
Can be null!

Interfejsy

- Interfejs jest podobny do klasy abstrakcyjnej, w której wszystkie metody są abstrakcyjne
- Umożliwia dziedziczenie wielobazowe
- Interfejs może rozszerzać wiele interfejsów
- Jeśli interfejs posiada pole, to jest ono z definicji **public**, **static**, **final**
- Aby w klasie użyć interfejsu należy go zaimplementować – w deklaracji nazwy klasy używamy słowa **implements**

```
1  public interface Pojazd {
2      public void jazda(int predkosc);
3      public void stop();
4  }
```

```
1  public class Samochod implements Pojazd {
2      @Override
3      public void jazda(int predkosc) { }
4
5      @Override
6      public void stop() { }
7
8      public void drift() { }
9  }
```

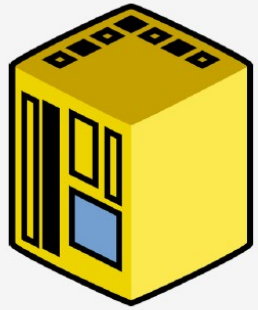


Interfejsy

- Interfejs musi być utworzony przy użyciu słowa kluczowego interface
- Interfejsy mogą być wykorzystywane polimorficznie, tzn można ich używać jako typu ogólniejszego klas, które go implementują
- Metody interfejsu nie mogą być zadeklarowane jako statyczne

```
1  Pojazd rower = new Rower();
2  Pojazd samochod = new Samochod();
```

W Javie niedozwolone jest dziedziczenie wielokrotne, jednak jak najbardziej poprawne jest implementowanie wielu interfejsów!



Ćwiczenie

Zadania do samodzielnego wykonania

Proszę napisać klasę Employer(pracodawca), która będzie reprezentowała pracodawcę. Klasa ta powinna dziedziczyć po klasie Person i dodatkowo zawierać metodę zwracającą nazwę firmy. Klasa Emplyer powinna implementować interfejs:

public interface Iemployer { String getCompany(); }

Ćwiczenie

Zadania do samodzielnego wykonania

Klasa Person powinna posiadać dwa pola: imię oraz nazwisko. Powinna implementować interfejs:

```
public interface IPerson{

String FirstName();

String LastName();}
```

Ćwiczenie

Zadania do samodzielnego wykonania

Proszę utworzyć klasę Market a w niej metodę main(). W metodzie tej proszę utworzyć instancję klasy Employeri wypisać na konsoli rezultat wywołania metody toString() na obiekcie klasy Employer.

Ćwiczenie

Zadania do samodzielnego wykonania

Proszę przysłonić metodę toString() klasy Object w klasie Employer. Metoda ta powinna zwracać imię, nazwisko oraz firmę w następujący sposób:

First Name: <tu imię osoby>

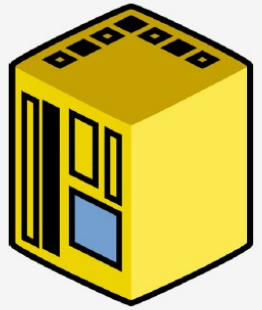
LastName: <tu nazwisko osoby>

Company: <tu firma>

Kompozycja

Kompozycję stosuje się wtedy, gdy zachodzi relacja „całość <-> część”, np. pojazd zawiera silnik.

```
public class Pojazd {

  private Silnik silnik;

  public Pojazd(Silnik silnik) {
    this.silnik = silnik;
  }

}
```



Ćwiczenie

Zadania do samodzielnego wykonania

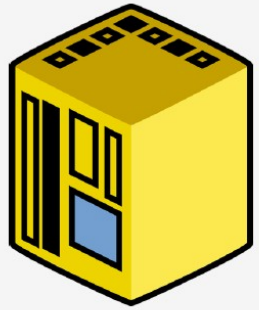
Proszę utworzyć klasę Employee(pracownik) implementujący interfejs

pracodawcy. Metoda getEmployer() powinna zwracać pracodawcę tego pracownika a calcSalary() wysokość jego wynagrodzenia.

public interface IEmployee{ Employer getEmployer(); double calcSalary(); }

Typy generyczne

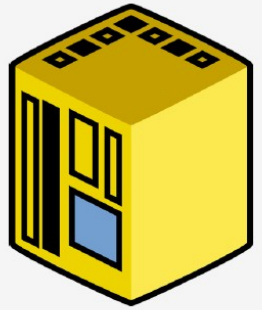
- W uproszczeniu można powiedzieć, że typy generyczne są “szablonami”.
- Dzięki typom generycznym możemy uniknąć niepotrzebnego rzutowania.
- Ponadto przy ich pomocy kompilator jest w stanie sprawdzić poprawność typów na etapie kompilacji



Typy generyczne

```
public class Apple {
}

public class AppleBox {
    private Apple apple;

    public AppleBox(Apple apple) {
        this.apple = apple;
    }

    public Apple getApple() {
        return apple;
    }
}
```

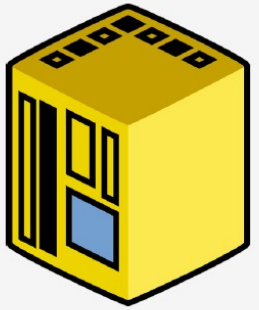
Klasa AppleBox „wie” jakiego typu obiekt może przechowywać, jest to obiekt typu Apple



Typy generyczne

Wymagane „rzutowanie” aby pobrać atrybut

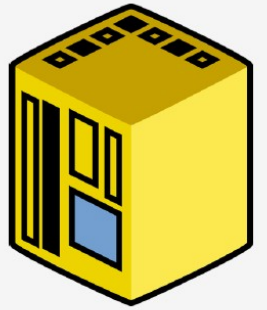
```java
public class FruitBox {
    private Object fruit;

    public FruitBox(Object fruit) {
        this.fruit = fruit;
    }

    public Object getFruit() {
        return fruit;
    }
}

public class Main {
    public static void main(String[] args) {
        FruitBox fruitBox = new FruitBox(new Orange());
        Orange fruit1 = (Orange) fruitBox.getFruit();
    }
}
```



Typy generyczne

Typ ten dostaje „tymczasową nazwę”

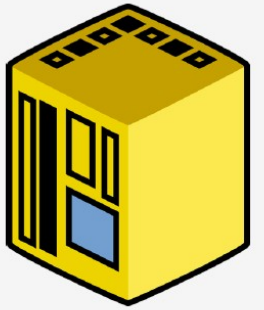
```
public class BoxOnSteroids<T> {
    public T fruit;

    public BoxOnSteroids(T fruit) {
        this.fruit = fruit;
    }

    public T getFruit() {
        return fruit;
    }
}

public class Main {
    public static void main(String[] args) {
        BoxOnSteroids<Apple> appleBox = new BoxOnSteroids<Apple>(new Apple());
        BoxOnSteroids<Orange> orangeBox = new BoxOnSteroids<Orange>(new Orange());

        Orange fruit = orangeBox.getFruit();
    }
}
```



Klasy generyczne i parametryzowanie

```
public class CookieCutter<T> {
    private T glaze;
}
```

parametry klasy generycznej są elementami uzupełnianymi w szablonie

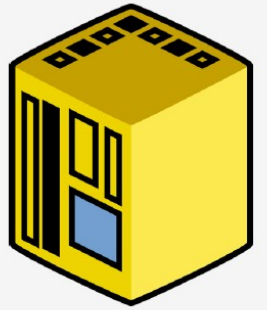


Definicja klasy generycznej

Klasę generyczną definiujemy w następujący sposób

```
class Name<T1, T2, ..., Tn> {
    /* body */
}
```

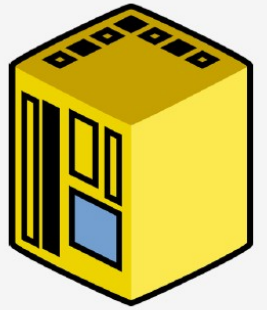
Zgodnie ze specyfikacją języka Java możesz użyć dowolnej nazwy która nadaje się na nazwę zmiennej istnieje konwencja nazewnicza sugerująca nazwy parametrów. Zwyczajowo do tego celu używa się wielkich liter T, K, U, V, E.



Metody z generycznymi argumentami – wildcard

Pisząc metody, które jako argumenty przyjmują typy generyczne nie zawsze chcesz dokładnie specyfikować typ. W takim wypadku z pomocą przychodzi znak ?, który może akceptować różne typy.

```
private static void method1(FancyBox<?> box) {
    Object object = box.object;
    System.out.println(object);
}

private static void plainWildcard() {
    method1(new FancyBox<>(new Object()));
    method1(new FancyBox<>(new Square()));
    method1(new FancyBox<>(new Apple()));
}
```



Wstęp do programowania funkcyjnego

Języki programowania takie jak C/C++/Java/Python są nazywane imperatywnymi językami programowania, ponieważ zawierają sekwencje zadań do wykonania. Programista jawnie, krok po kroku definiuje jak wykonać zadanie.

Programy funkcyjne składają się **jedynie z funkcji**. Funkcje są podstawowymi elementami języka funkcyjnego. Główny program jest funkcją, której podajemy argumenty, a w zamian otrzymujemy wyznaczoną wartość – wynik działania programu.

Główna funkcja składa się tylko i wyłącznie z innych funkcji, które z kolei składają się z jeszcze innych funkcji. Funkcje takie dokładnie odpowiadają funkcjom w czysto matematycznym znaczeniu – przyjmują pewną liczbę parametrów i zwracają wynik.

Wstęp do programowania funkcyjnego

Programując funkcyjnie z użyciem Java, nie mogą wystąpić efekty uboczne –przekazywane wartości muszą być oznaczone jako final.

Najważniejszą cechą jest możliwość pisania programów w kategoriach "co ma być osiągnięte", a nie - jak w programowaniu imperatywnym – określanie kolejnych kroków algorytmu do wykonania.

Poszczególne fragmenty kodu (funkcje) możemy traktować jak pełnoprawne obiekty, które mogą być przekazywane innym funkcjom i zwracane z innych funkcji.

Programowanie funkcyjne

```java
public String greet(List<String> names) {
  String greeting = "Welcome ";
  for (String name : names) {
    greeting += name + " ";
  }
  greeting += "!";
  return greeting;
}
```

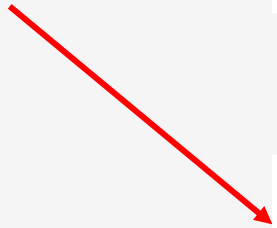
zapis funkcyjny

```java
public String greet(List<String> names) {
  String greeting = names.stream()
          .map(name -> name + " ")
          .reduce("Welcome ", (acc, name) -> acc + name);
  return greeting + "!";
}
```

Programowanie funkcyjne

```java
public List<Integer> addOne(List<Integer> numbers) {
  List<Integer> plusOne = new LinkedList<>();
  for (Integer number : numbers) {
    plusOne.add(number + 1);
  }
  return plusOne;
}
```

zapis funkcyjny



```java
public List<Integer> addOne(List<Integer> numbers) {
  return numbers
          .stream()
          .map(number -> number + 1)
          .collect(Collectors.toList());
}
```

Stream

Z listy możemy uzyskać stream(strumień/sekwencję) danych.

Na danych strumienia możemy wykonywać:

- przetwarzanie - metoda map
- filtrowanie - metoda filter
- uzyskać nowe kolekcje wyników powyższych operacji - metoda collect

```
public List<Integer> addOne(List<Integer> numbers) {
    return numbers
            .stream()
            .map(number -> number + 1)
            .collect(Collectors.toList());
}
```

Stream

- **map** – jako parametr podajemy wyrażenie lambda, wynikiem jest przetworzenie podanego parametru, metoda zwraca strumień,
- **filter** - jako parametr podajemy wyrażenie lambda, parametr jest przekształcany w wartość boolowską, metoda zwraca strumień danych, dla których wynik lambdy był true,
- **collect** – kończy przetwarzanie strumienia, jako parametr przyjmuje obiekt klasy Collector, który tworzy nową kolekcję i dodaje do niej dane ze strumienia. Popularny kolektor, budujący listę, można uzyskać za pomocą statycznej metody Collectors.toList()

```
public List<Integer> addOne(List<Integer> numbers) {
  return numbers
          .stream()
          .map(number -> number + 1)
          .collect(Collectors.toList());
}
```

Wyrażenie lambda

Wyrażenia lambda – bezpośrednio podawane fragmenty kodu, funkcje bez nazwy, mogą być przekazywane metodom.

n ->n < 10

Jest wyrażeniem lambda, które można traktować jako anonimową funkcję. Parametrem jest n, zwracany jest wynik wyrażenia n < 10.

n -> n*n

Jest wyrażeniem lambda. Parametrem jest n, zwracany wynik to n * n.

Ćwiczenie

Zadania do samodzielnego wykonania

Mając listę liczb, wypisz w konsoli liczby nieparzyste pomnożone przez 100List<Integer> num= Arrays.asList(1, 3, 4, 10, 9, 13, 6);

Ćwiczenie

Zadania do samodzielnego wykonania

Na podstawie listy słów, wypisz w konsoli wszystkie wyrazy zaczynają się literą „a”, wartości powinny posiadać dużą, pierwszą literęList<String> txt = Arrays.asList(„ala", „samochód", „kot”, „aleksandra", „pies", „azor");

Ćwiczenie

Zadania do samodzielnego wykonania

Na podstawie listy pracowników, wypisz w konsoli imiona i nazwiska wszystkich, którzy mają powyżej 30 lat i zarabiają mniej niż 4000.

List<Employee> employee= Arrays.asList(

new Employee("Kowal", "Jan", 34, 3400.0),new Employee("As", "Ala", 27, 4100.0),

new Employee("Kot", "Zofia", 33, 3700.0),new Employee("Puchacz", "Jan", 41 , 3600.0));

Dziękuję za uwagę!

Źródło / Opracowano na podstawie

https://pixabay.com/pl/kodowanie-komputer-1294361/ (dostęp 21.07.2017)

https://pl.wikipedia.org/wiki/Programowanie_komputer%C3%B3w (dostęp 22.07.2017)

Wprowadzenie do Języka Java (Łukasz Ognan, Marzec 2017)

https://pl.wikipedia.org/wiki/Java (dostęp 22.07.2017)

https://pl.wikipedia.org/wiki/Interpreter_(program_komputerowy) (dostęp 22.07.2017)

https://pixabay.com/pl/m%C5%82otek-narz%C4%99dzia-metalowe-celuj-w-33617/ (dostęp 22.07.2017)

https://pixabay.com/pl/matematyka-sub-zmienna-matematyczne-27893/ (dostęp 24.07.2017)

https://pixabay.com/pl/nagrobek-protokół-rip-martwe-śmierć-159792/ (dostęp 24.07.2017)

https://pixabay.com/pl/typu-mainframe-sprz%C4%99tu-serwer-mail-35647/ (dostęp 24.07.2017)

https://pixabay.com/pl/pole-przypadku-otwarte-pakiet-1295176/ (dostęp 24.07.2017)

Podstawy Java(Rafał Roppel)

https://pixabay.com/pl/puchar-kubek-kawa-pi%C4%87-herbata-42427/ (dostęp 24.07.2017)

https://pixabay.com/pl/zarejestruj-stanowisko-wy%C5%BCywienie-48703/ (dostęp 24.07.2017)

https://pixabay.com/pl/narzędzi-połowowych-funkcja-razem-818456/ (dostęp 24.07.2017)

https://pixabay.com/pl/skaner-palmtop-kod%C3%B3w-kreskowych-36385/ (dostęp 24.07.2017)

https://pl.wikipedia.org/wiki/P%C4%99tla_(informatyka) (dostęp 24.07.2017)

https://pixabay.com/pl/cykl-obieg-proces-zmiana-zmiany-2019535/ (dostęp 25.07.2017)

https://pixabay.com/pl/monitora-monitor-wall-big-screen-1054600/ (dostęp 25.07.2017)

https://pl.wikipedia.org/wiki/Tablica_(informatyka) (dostęp 25.07.2017)