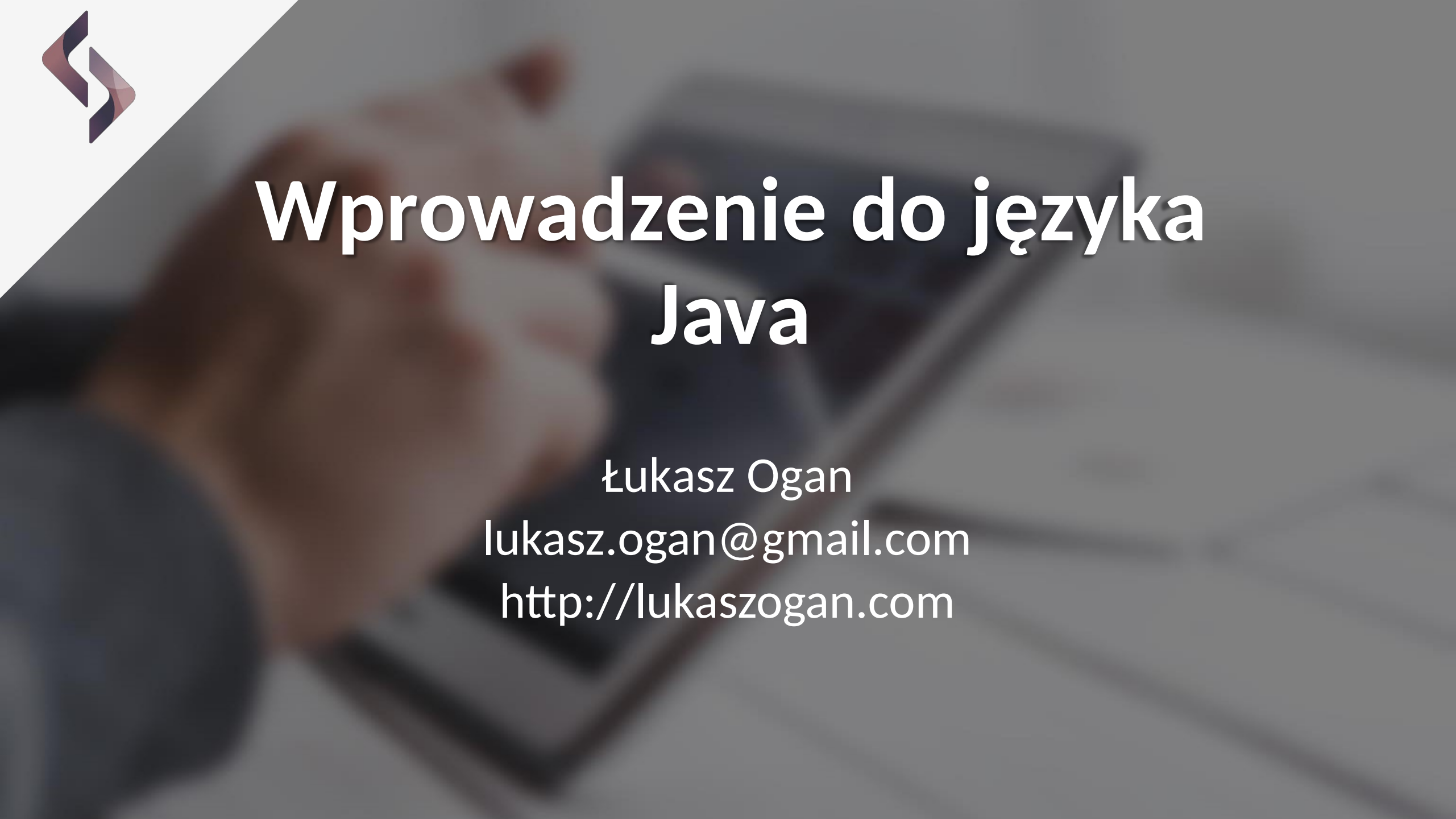
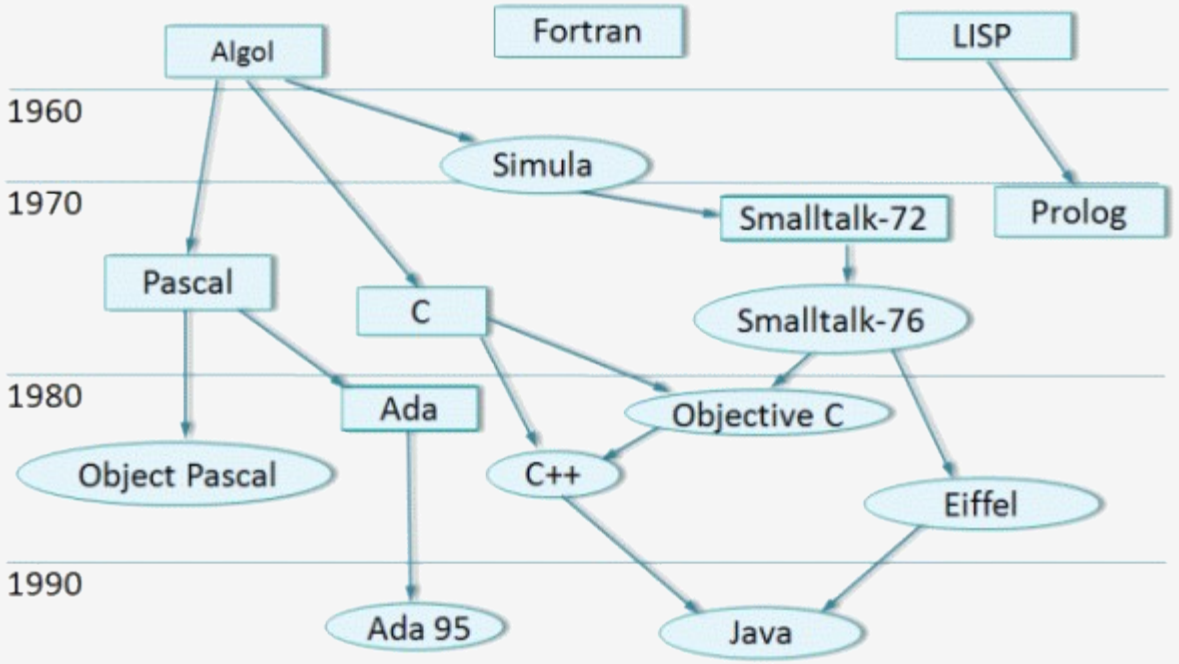


Wprowadzenie do języka Java

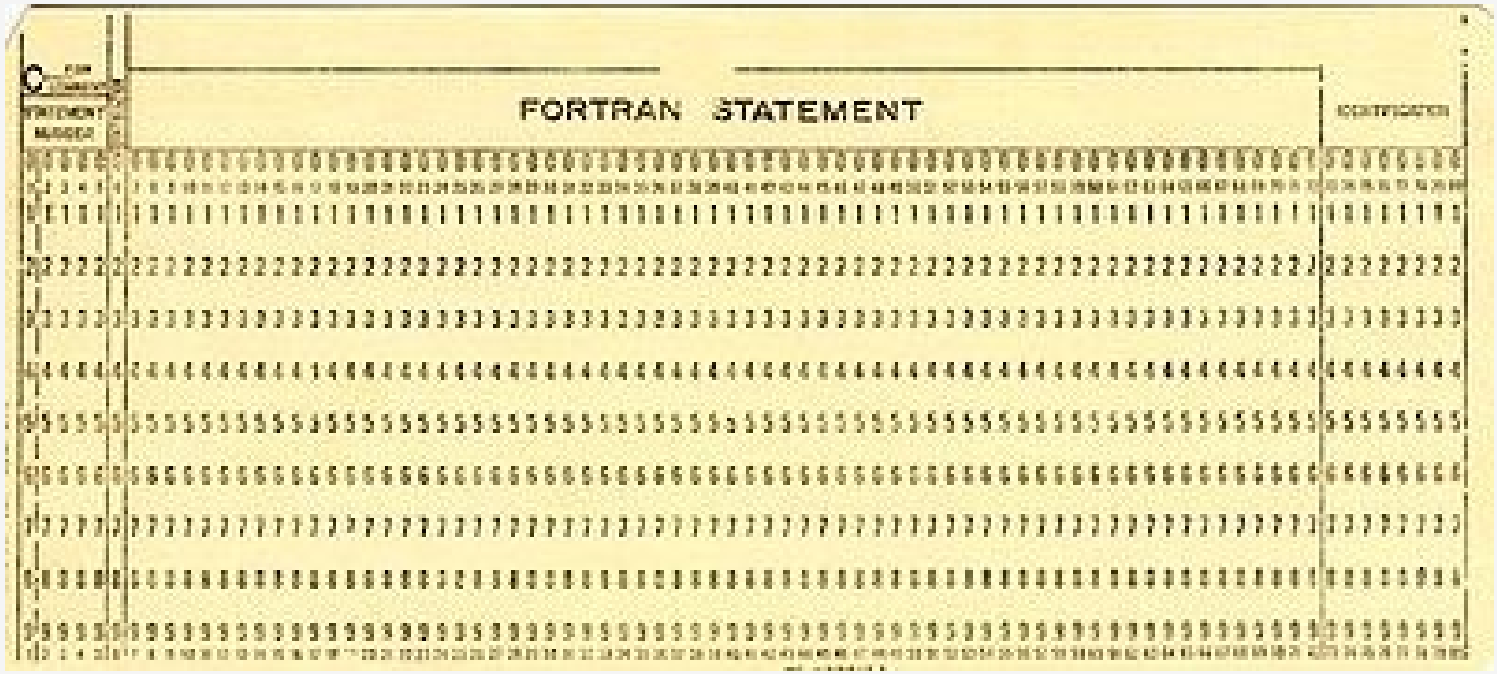
Łukasz Ogan
lukasz.ogan@gmail.com
http://lukaszogan.com

Historia programowania



Wraz z upływem czasu, komputery stawały się coraz szybsze oraz coraz tańsze. Karty perforowane po prostu przestały wystarczać. Pojawiły się urządzenia wejścia (klawiatury) i wyjścia (monitory, drukarki), a przede wszystkim, umożliwiono przechowywanie danych (pierwsze pamięci masowe). Dzięki temu możliwe stało się nie tylko wykonywanie, ale również przygotowanie programów na samych komputerach.

Fortran




```
IMPLICIT NONE
REAL(8) :: a,b
READ *,a,b
PRINT *,'wynik',a+b
END
```

Asembler

```
__asm {

    push    5                   // zapisz na stosie liczbę 5 (32bit)
    push    eax                 // zapamiętuje na stosie zawartość rejestru EAX
    push    dword ptr[edx]      // zapamiętuje wartość wskazywaną przez wskaźnik zapisany
                                // w rejestrze EDX

    sub     esp,4               // odpowiednik instrukcji 'push 5'
    mov     dword ptr[esp],5

    sub     esp,4               // odpowiednik instrukcji 'push eax'
    mov     dword ptr[esp],eax
}
```

C

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

WHICH PROGRAMMING LANGUAGE SHOULD I LEARN FIRST?

WHAT IS PROGRAMMING?

Writing very specific instructions to a very dumb, yet obedient machine.

LANGUAGES

PYTHON · JAVA · C · PHP · C++
JAVASCRIPT · C# · RUBY · OBJECTIVE-C

START

WHY DO YOU WANT TO LEARN PROGRAMMING?

- I don't know, just pick one for me
- Just for fun
- I'm interested
- Improve myself
- Make money
- I have a startup idea!

Start with Scratch, then move on to... ← For my kids

Have a brilliant idea/platform in mind?

- Nope. Just want to get started → I prefer to learn things...
  - The easy way
  - The best way
  - The slightly harder way
  - Auto or Manual car?
    - Auto
    - Manual
- YES → Which platform/field?

Which platform/field?

- I want to work for big tech companies
- Front-end (web interface)
- Web
  - Back-end ("brain" behind a website)
    - I want to work for...
      - Corporate
        - What do you think about Microsoft?
          - I'm a fan!
          - Not Bad
          - Suck
      - Startup
- 3D/Gaming
- Mobile
  - Which OS?
    - iOS
    - Android
- Enterprise
- Web
  - Does your web app provides info in real-time, like twitter?
    - YES
    - NO
- Doesn't matter, I just want $$$
- Get a job

Do you want to try something new with huge potential, but less mature?

- NOT SURE
- NO

Which one is your favourite toy?

- Lego
- Play-Doh
- I've an old & ugly toy, but i love it so much!
- The really hard way (but easier to pick up other languages in the future)

Python
The Ent

DIFFICULTY ★☆☆☆☆

Help little Hobbits (beginners) to understand programming concepts

Help Wizards (computer scientists) to conduct researches

Widely regarded as the best programming language for beginners

Easiest to learn

Widely used in scientific, technical & academic field, i.e. Artificial Intelligence

You can build website using Django, a popular Python web framework

POPULARITY ★★★★½

USED TO BUILD YouTube, Instagram, Spotify

AVG. SALARY $107,000

Java
Gandalf

DIFFICULTY ★★★☆☆

Wants peace & works with everyone (portable)

Very popular on all platforms, OS, and devices due to its portability

One of the most in demand & highest paying programming languages

Slogan: write once, work everywhere

POPULARITY ★★★★★

USED TO BUILD Gmail, Minecraft, Most Android Apps, Enterprise applications

AVG. SALARY $102,000

C
One Ring

DIFFICULTY ★★★☆☆

The power of C is known to them all

Everyone wants to get its Power

Lingua franca of programming language

One of the oldest and most widely used language in the world

Popular language for system and hardware programming

A subset of C++ except the little details

POPULARITY ★★★★★

USED TO BUILD Operating systems and hardware

AVG. SALARY $102,000

C++
Saruman

DIFFICULTY ★★★★☆

Everyone thinks that he is the good guy

But once you get to know him, you will realize he wants the power, not good deeds

Complex version of C with a lot more features

Widely used for developing games, industrial and performance-critical applications

Learning C++ is like learning how to manufacture, assemble, and drive a car

Recommended only if you have a mentor to guide you

POPULARITY ★★★★½

USED TO BUILD Operating systems, hardware, and browsers

AVG. SALARY $104,000

JavaScript
Hobbit

DIFFICULTY ★★☆☆☆

Frequently underestimated (powerful)

Well-known for the slow, gentle life of the Shire (web browsers)

"Java and Javascript are similar like Car and Carpet are similar" - Greg Hewgill

Most popular clients-side web scripting language

A must learn for front-end web developer (HTML and CSS as well)

One of the hottest programming language now, due to its increasing popularity as server-side language (node.js)

POPULARITY ★★★★☆

USED TO BUILD Paypal, front-end of majority websites

AVG. SALARY $99,000

C#
Elf

DIFFICULTY ★★★☆☆

Beautiful creature (language), used to stay in their land, Rivendell (Microsoft Platform), but recently started to open up to their neighbours (open source)

A popular choice for enterprise to create websites and Windows application using .NET framework

Can be used to build website with ASP.NET, a web framework from Microsoft

Similar to Java in basic syntax and some features

POPULARITY ★★★★½

USED TO BUILD Enterprise and Windows applications

AVG. SALARY $94,000

Ruby
Man (Middle Earth)

DIFFICULTY ★★☆☆☆

Very emotional creature

They (some Ruby developers) feel they are superior & need to rule the Middle Earth

Mostly known for its popular web framework, Ruby on Rails

Focuses on getting things done

Designed for fun and productive coding

Best for fun and personal projects, startups, and rapid development

POPULARITY ★★★½☆

USED TO BUILD Hulu, Groupon, Slideshare

AVG. SALARY $107,000

PHP
Orc

DIFFICULTY ★★☆☆☆

Ugly guy (language) and doesn't respect the rules (inconsistent and unpredictable)

Big headache to those (developers) to manage them (codes)

Yet still dominates the Middle-earth (most popular web scripting language)

Suitable for building small and simple sites within a short time frame

Supported by almost every web hosting services with lower price

POPULARITY ★★★★☆

USED TO BUILD Wordpress, Wikipedia, Flickr

AVG. SALARY $89,000

Objective-C
Smaug

DIFFICULTY ★★★☆☆

Lonely and loves gold

Primary language used by Apple for Mac OS X & iOS

Choose this if you want to focus on developing iOS or OS X apps only

Consider to learn Swift (newly introduced by Apple in 2014) as your next language

POPULARITY ★★★☆☆

USED TO BUILD Most iOS Apps and part of Mac OS X

AVG. SALARY $107,000

O języku Java

- Obiektowy język programowania
- Istnieje od 1995, stworzony przez Jamesa Goslinga z firmy Sun Microsystems
- Oparty na podstawie koncepcji języka Smalltalk i C++
- Obecnie możemy używać Javy w wersji 8 (stabilna) i 9 (early access)
- Dziś Java jest wspierana przez firmę Oracle

- Aplikacje uruchamiane są na maszynie wirtualnej JVM
- Zapewnia to niezależność od platformy. Wiele platform języka (Java Standard Edition, Java Enterprise Edition, Java ME, Java FX, Android SDK)
- Środowisko uruchomieniowe (JRE) Java jest zainstalowane na większości urządzeń PC.
- Początkowo zaprojektowana do tworzenia aplikacji przeglądarkowych (aplety), obecnie ma szerokie zastosowanie

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 języka Java

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.

Kolejne edycje Javy, 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), typy enum, autoboxing, varargs, adnotacje.

Historia wersji języka JAVA

JDK Beta	1994	
JDK 1.0	1996	Wersja inicjalna
JDK 1.1	1997	Klasy wewnętrzne, JDBC, RMI, refleksja, JIT, AWT
J2SE 1.2	1998	Swing, kolekcje
J2SE 1.3	2000	JNDI, debugger
J2SE 1.4	2002	Nowa obsługa operacji Input/Output, wyrażenia regularne, parser XML, logowanie, exception chaining
J2SE 5.0	2005	Typy generyczne, typy wyliczeniowe, autoboxing, varargs, pętla foreach, importy statyczne, zmiany w wielowątkowości
Java SE 6	2006	Zakończenie obsługi Windows 9x, zmiany w Swingu, usprawnienie JDBC, JAX-WS, zmiany w JVM
Java SE 7	2011	Zmiana obsługi plików, zmiany w języku (switch, operator diamond, catch wielu wyjątków). Nowości w wielowątkowości.
Java SE 8(LTS)	2014	Wyrażenia lambda, strumienie, nowe api dla dat i czasu. Zakończenie obsługi Windows
Java SE 9	2017	XP
Java SE 10 (LTS)	2018	
Java 11	2018	
Java 12	2019	

Dlaczego Java?

1. Język obiektowy
2. Niezależność od platformy (*Write Once, Run Anywhere)*
3. Prostota
4. Czy Java jest powolna?
5. Java jest "duża".
6. Podobieństwo do C#
7. Duża ilość literatury
8. Duża społeczność

Środowisko do programowania

- Zainstalowane Java Development Kit w odpowiedniej wersji
  Sposób instalacji zależny od systemu operacyjnego. JDK można pobrać z strony Oracle lub OpenJDK.
  Ustawienie zmiennej środowiskowej JAVA_HOME

- Środowisko uruchomieniowe Java (JRE)
- Środowisko deweloperskie - IDE
  Darmowe narzędzia: IntelliJ IDEA, Eclipse, NetBeans

Pojęcia

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.

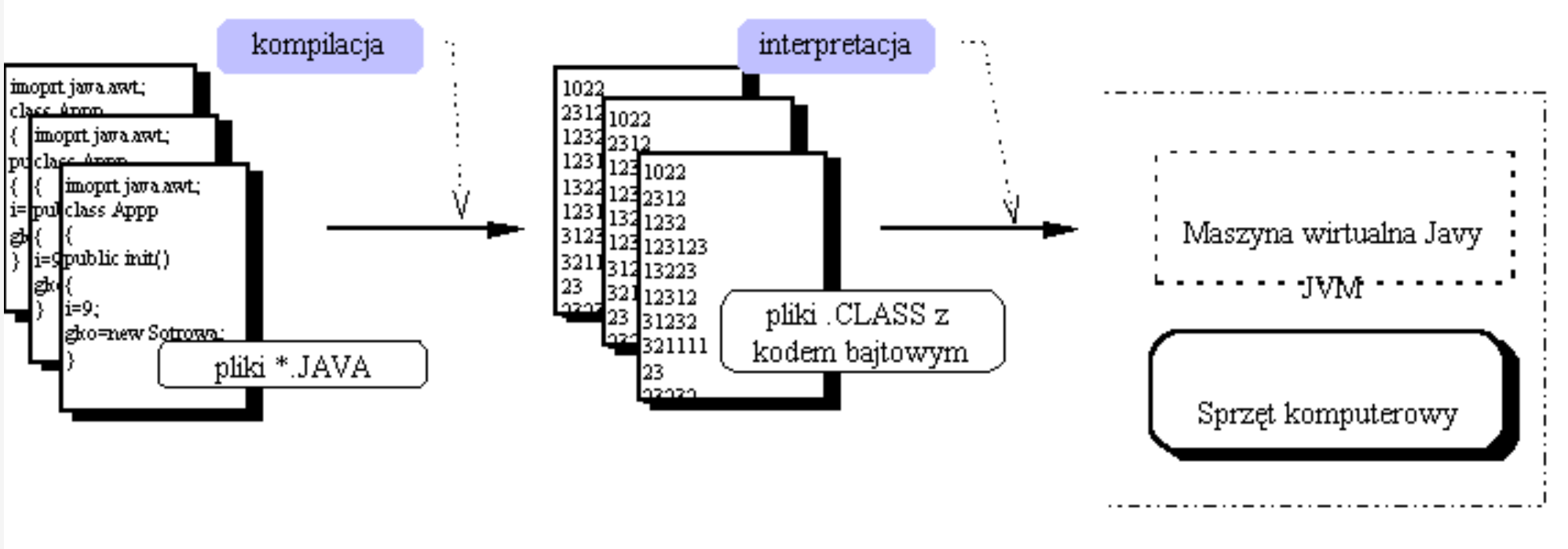
Kompilator - program, który kod źródłowy zamienia w kod

napisany w innym języku. Oprócz tego kompilator ma za zadanie

odnaleźć błędy leksykalne i semantyczne oraz dokonać optymalizacji kodu.

Kod bajtowy - bytecode, wynik kompilacji programu

Jak działa język programowania



Instalacja JDK i JRE

Należy pobrać środowisko JDK ze strony Oracle (zajmuje ono do 200MB pamięci).

Interesuje nas wersja 8.

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

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*.**

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.

Reprezentacja liczb

Najprostszym układem pozycyjnym jest system binarny. Elementami zbioru znaków systemu binarnego jest para cyfr: 0 i 1. Znak dwójkowy (0 lub 1) nazywany jest bitem. Liczby naturalne w systemie dwójkowym zapisujemy analogicznie jak w systemie dziesiętnym - jedynie zamiast kolejnych potęg liczby dziesięć, stosujemy kolejne potęgi liczby dwa.

$$9 = 1 \cdot 23 + 0 \cdot 22 + 0 \cdot 21 + 1 \cdot 20$$

$$9_{(10)} = 1001_{(2)}$$

9 | 1
4 | 0
2 | 0
1 | 1

Dziele przez 2 i resztę zapisuje po lewej stronie. Wynik jest czytany od góry.

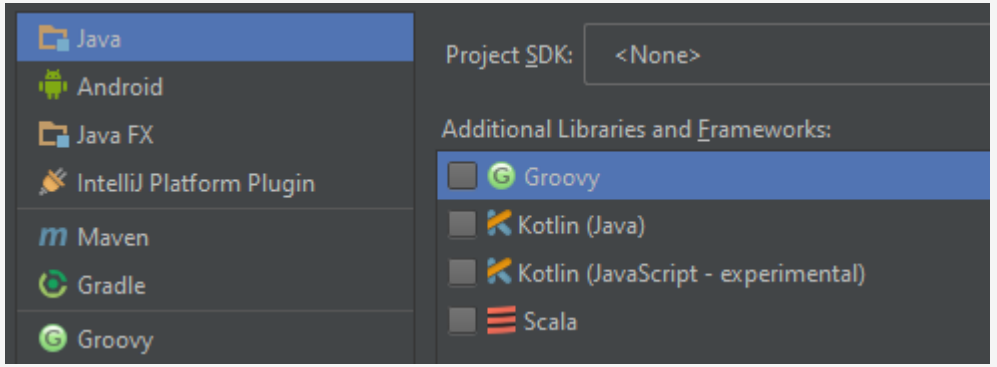
Pobieranie oraz instalacja

Środowisko można pobrać z oficjalnej strony JetBrains:

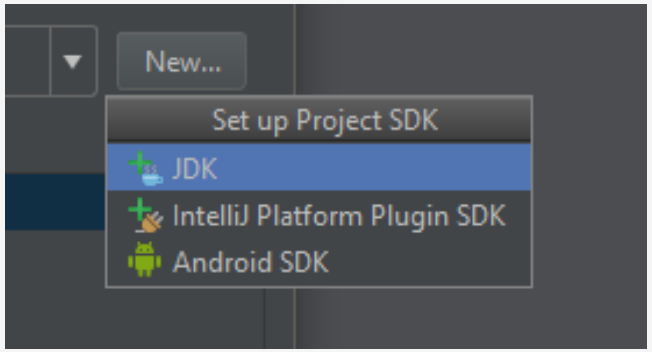
https://www.jetbrains.com/idea/download/

Pierwszy projekt

Rodzaj projektu: wybieramy opcję **Java Project**



Konfiguracja JDK: wskazujemy lokalizację instalacji



Stworzenie paczki – pl.sda.hello

Klasy pogrupowane są w tzw. **pakiety**. Klasy dzielimy na pakiety by pogrupować je według ich znaczenia – analogicznie do tego, jak dzielimy pliki na katalogi na dysku twardym naszego komputera.

Pakiety – podobnie jak katalogi – mają strukturę hierarchiczną każdy z pakietów może zawierać kolejne pakiety – podobnie jak katalogi mogą zawierać podkatalogi. Każdy pakiet, oprócz dowolnej liczby innych pakietów może zawierać także dowolną liczbę klas – podobnie do katalogu, który oprócz innych katalogów może zawierać dowolnie wiele plików. O klasach które nie należą do żadnego pakietu (mówi się o nich czasem, że należą do pakietu domyślnego lub głównego) możemy myśleć jak o plikach które znajdują się bezpośrednio na dysku *C* naszego komputera – one również nie należą do żadnego katalogu. Nazwy pakietów zwyczajowo pisze się małymi literami. Kolejne poziomy zagnieżdżenia w hierarchii pakietów oddziela się od siebie kropkami, podobnie do tego jak nazwy kolejnych katalogów w ścieżce dostępu oddziela się od siebie ukośnikami.

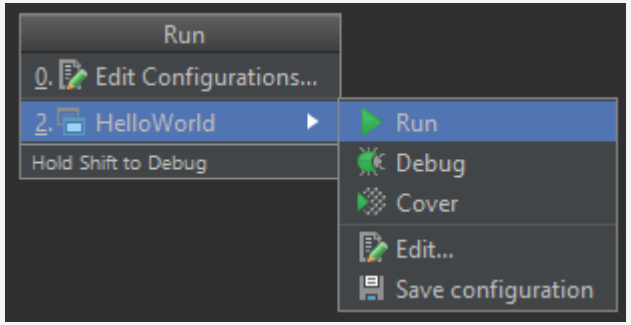
Pierwszy projekt

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.

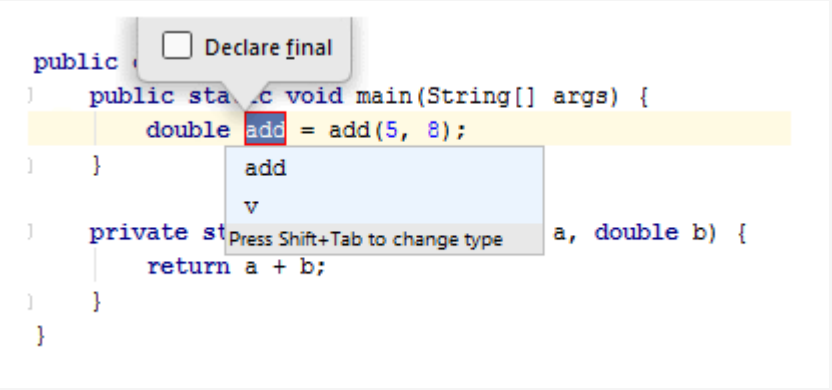


IntelliJ IDEA - skróty klawiaturowe

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*

Pierwszy program

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.

Uruchomienie programu - terminal

Kompilacja: *javac nazwa_klasy.java*
Uruchomienie: *java nazwa_klasy*

Zadanie

Napisz program, który wyświetli w 3 kolejnych liniach trzy imiona: Jan, Maciej, Ola.

Komentarze

Dwa rodzaje komentarzy:

- *//text* - tekst umieszczony za podwojonym znaku // jest uznawany za komentarz jednolinijkowy
- */* text *** - tekst umieszczony w takich znacznikach jest traktowany jako komentarz blokowy (wiele linii kodu)

Typy danych

W Javie podobnie jak w innych językach wyróżniamy wiele typów danych mogących przechowywać zarówno liczby stało i zmiennoprzecinkowe, znaki, ciągi znaków oraz typ logiczny. Java posiada ścisłą kontrolę typów, czyli mówiąc prościej każdy obiekt musi mieć określony typ.

byte - 1 bajt - zakres od -128 do 127

short - 2 bajty - zakres od -32 768 do 32 767

int - 4 bajty - zakres od -2 147 483 648 do 2 147 483 647

long - 8 bajtów - zakres od -2^63 do (2^63)-1 (posiadają przyrostek L, lub l)

float - 4 bajty - max ok 6-7 liczb po przecinku (posiadają przyrostek F, lub f)

double - 8 bajtów - max ok 15 cyfr po przecinku (posiadają przyrostek D, lub d)

char – odpowiada jednemu znakowi (np. literze), może przechowywać liczby całkowite z zakresu

od 0 do 65 535.

boolean – wartość *logiczna*, może przyjąć jedną z wartości true (oznaczającą prawdę) lub false (oznaczającą fałsz).

Podstawowe typy danych – liczby całkowite

Byte - 1 bajt

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

Podstawowe typy danych – liczby całkowite

Short - 2 bajty

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

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

Podstawowe typy danych – liczby całkowite

Long - 8 bajtów

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

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**)

Podstawowe typy danych – liczby zmiennoprzecinkowe

Double - 8 bajtów

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

Operatory matematyczne

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

Operatory porównawcze

== sprawdza równość

!= różny

>= większy równy

<= mniejszy równy

> , < większy, mniejszy

Nazwa	Język Java	Opis
i	&&	Iloczyn logiczny - wszystkie wartości muszą być prawdziwe, aby została zwrócona prawda.
lub	||	Suma logiczna - co najmniej jedna z wartości musi być prawdziwa, aby została zwrócona prawda.
negacja	!	Zanegowanie wartości - czyli zwrócenie wartości przeciwnej.

Zmienne

Deklaracja - określamy typ i nazwę zmiennej

Inicjalizacja - nadanie wartości zmiennej

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

  int liczba; // Deklaracja

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

Instrukcja warunkowa

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

Warunki można łączyć z wykorzystaniem operatorów logicznych

Instrukcja switch-case

```
switch ({wyrażenie wyboru}) {
    case {wartość wyboru}:
        {ciąg instrukcji dla danego wariantu}
            break;
        case {inna wartość wyboru}:
        {ciąg instrukcji dla danego wariantu}
            break;
        default:
            {ciąg instrukcji dla wariantu
        domyślnego}
            }
```

Pętle

- While – sprawdza warunek logiczny przed wykonaniem pętli
- Do – while – jedna iteracja pętli wykona się zawsze
- For – służy do iterowania poprzez kolekcję

Pętla while

```
while(warunek  logiczny){
 //rozpocznij iterację jeśli warunek jest
 prawdziwy
    instrukcja1;instrukcja2;
}
```

Pętla do while

```
do{
            instrukcja1; instrukcja2;
}
while(warunek logiczny) //powtórz pętlę
jeśli
warunek  jest prawdziwy
```


while (not edge) {
run();
}
do {
run();
} while (not edge);

Pętla for

```
for(elementKolekcji  : kolekcja){ //iteruj
poprzez elementy  kolekcji
    instrukcja1;
    instrukcja2;
}
```

Pętla for

Schemat pętli for:

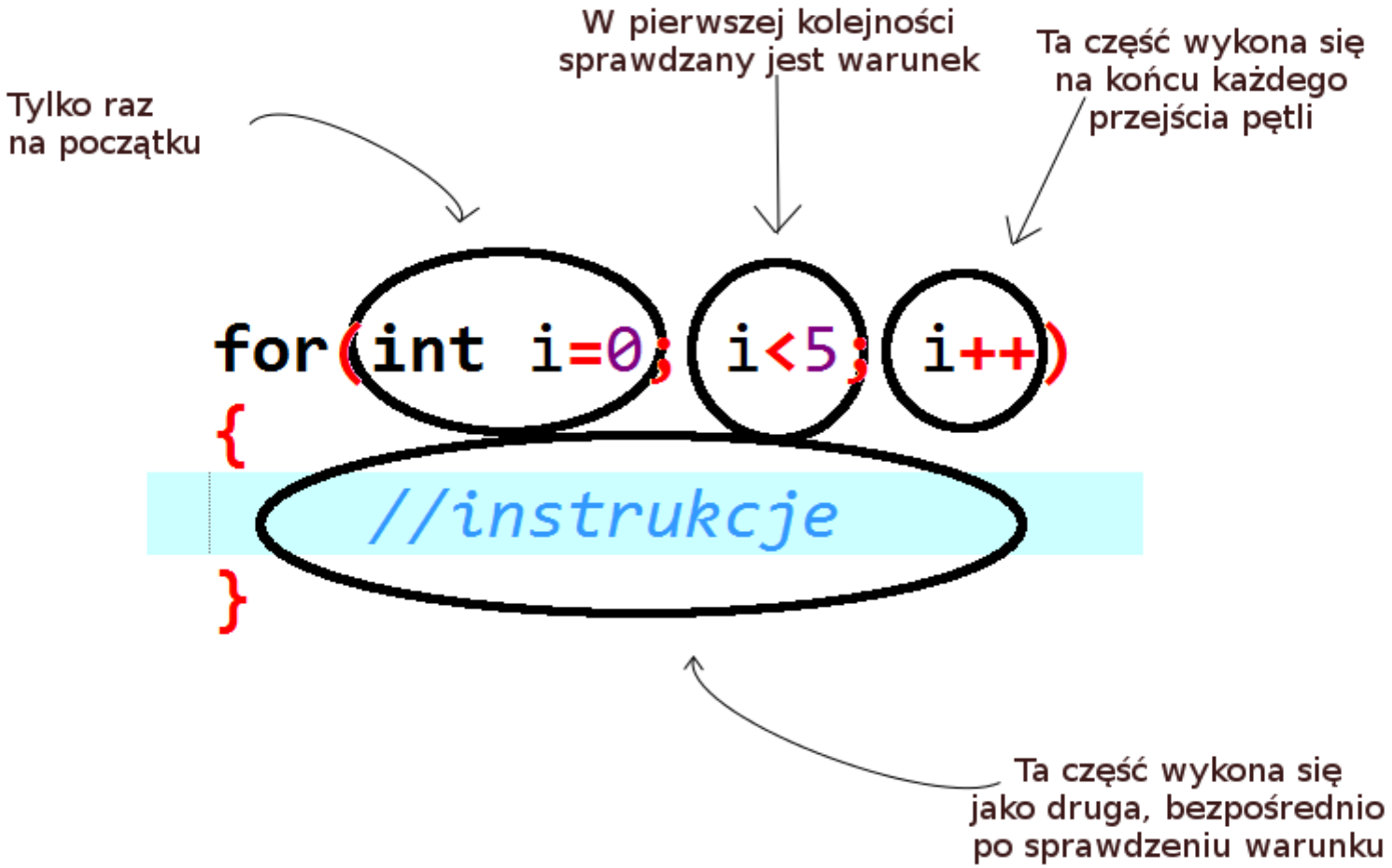
```
for(wyrażenie początkowe ;
warunek ;
  instrukcje do wykonania
}
```

```
modyfikator_licznika
){
```

Przykład:

```
for(int   i=0;  i<10; i++)
    { System.out.println("To jest
    pętla");
}
System.out.println("Koniec pętli");
```

Pętla for


W pierwszej kolejności sprawdzany jest warunek
Ta część wykona się na końcu każdego przejścia pętli
Tylko raz na początku
for(int i=0; i<5; i++)
{
//instrukcje
}
Ta część wykona się jako druga, bezpośrednio po sprawdzeniu warunku

Pętla while

```
while(warunek){
  instrukcje  do wykonania
}
```

Pętlę while najczęściej wykorzystuje się w miejscach, gdzie zakładana ilość powtórzeń jest bliżej nieokreślona, ale znamy warunek jaki musi być spełniony. Jej schematyczną postać przedstawiono poniżej:

```
int   licznik  =  0;

while(licznik<10){ System.out.println("To  jest  petla"); licznik++;
}
System.out.println("Koniec pętli");
```

Pętla do while

Różni się ona od pętli while przede wszystkim tym, że to co znajduje się w jej wnętrzu wykona się przynajmniej raz, ponieważ warunek jest sprawdzany dopiero w drugiej kolejności.

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

Przykład:

```
int   licznik
   =  0;
do{
    System.out.println("To  jest
    petla");
    licznik++;
}
while(licznik<10);
System.out.println("Koniec pętli");
```

Przydatne rzeczy

Pobranie liczby od użytkownika:

```
int liczba;
Scanner wejscie = new Scanner(System.in);
liczba= wej.nextInt();
```

Pierwiastek z liczby:

```
int wynik = Math.sqrt(9);  //oblicza
pierwiastek  z liczby 9
```

Zadanie

Zadania do samodzielnego wykonania

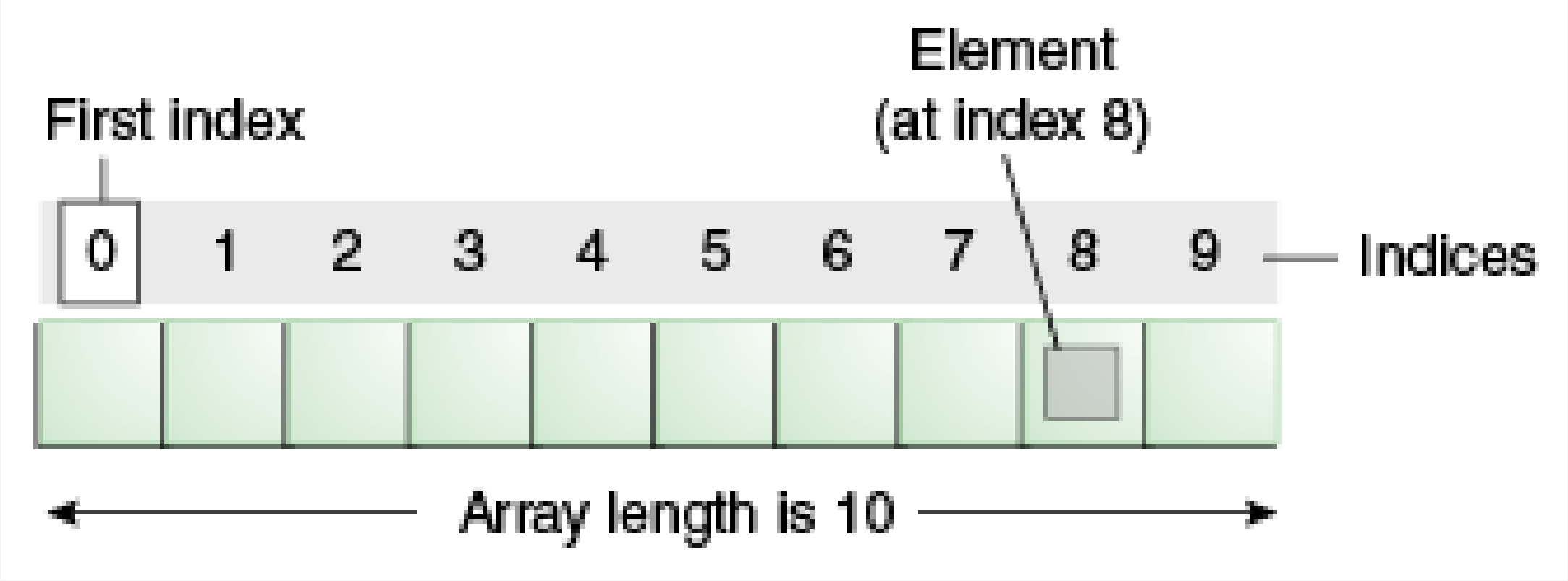
Napisz program, który pobierze od użytkownika całkowitą liczbę dodatnią.
Następnie przy użyciu wyświetl na ekranie Odliczanie z tekstem "Bomba wybuchnie za ... " gdzie w miejsce dwukropka mają się pojawić liczby od podanej przez użytkownika do 0. Napisz program przy użyciu pętli do while.

Zadanie

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).

Tablica




```
int[]  tablica = new
int[10];
```

Tablice jednowymiarowe

Tablice to struktury, które pozwalają nam gromadzić większą ilość danych w uporządkowanej formie. Jeśli potrzebujemy przechować 100 imion, czy liczb zamiast deklarować 100 zmiennych możemy do tego użyć tablicy. W Javie istnieją zarówno tablice jedno jak i wielowymiarowe.

```
typ[] nazwa_tablicy = new typ[liczba_elementów];
lub
typ nazwa_tablicy[] = new typ[liczba_elementów];
```

od elementy:

```
typ[] tablica = {wartosc1, wartosc2, wartosc3, ...};
```

Tablice

Pamiętajmy, że do utworzenia tablicy potrzebny jest operator new

```
int[] tablica = new int[100];
```

Powyższa instrukcja tworzy i inicjuje tablicę, w której można zapisać 100 liczb całkowitych. Elementy tablicy są numerowane od 0 (w przypadku wyżej od 0 do 99)

Przypisywanie wartości do tablicy

```
int[] tablica = new int[5]; //deklaracja
tablicv
 tablica[  = 23; //przypisanie
 0]          wartosci
 tablica[  = 232;
 1]
 tablica[  = 242;
 2]                                  w miejscu o indeksie  +
 tablica[  = 1;                                            tablica[2]);
 3]
 tablica[  = 41;
 4]
```

Wypełnienie tablicy wartościami od 0 do 99 – pętla for

```
int[] tablica = new int[100];

for(int i=0;i<100;i++)
{
    tablica[i] = i; //zapełnia tablice wartościami
    od 0
do99
}
```

Zadanie

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
7. Posumuj wszystkie liczby
8. Oblicz iloczyn liczb z tablicy
9. Znajdź element najmniejszy i największy

Pętla typu for each

W języku Java dostępny jest bardzo użyteczny rodzaj pętli umożliwiającej przeglądanie tablic bez stosowania indeksów.

```
for (zmienna: kolekcja)
{
    Instrukcje
}
```

For each przykład

```
int[] tablica = new
tablica[  = 232;
0]
tablica[  = 2311
1]        2;
for (int wartosc: tablica)
{
    System.out.println(„Wartosci: ” +
    wartosc);
}
```

Tablice wielowymiarowe

Różnią się przede wszystkim sposobem deklaracji i odwoływania do jej
elementów. Za jej pomocą można na przykład w łatwy sposób wyobrazić sobie grę w statki - można przechowywać za pomocą współrzędnych miejsce położenia statków. Tablice wielowymiarowe w języku Java to tak naprawdę tablice tablic.

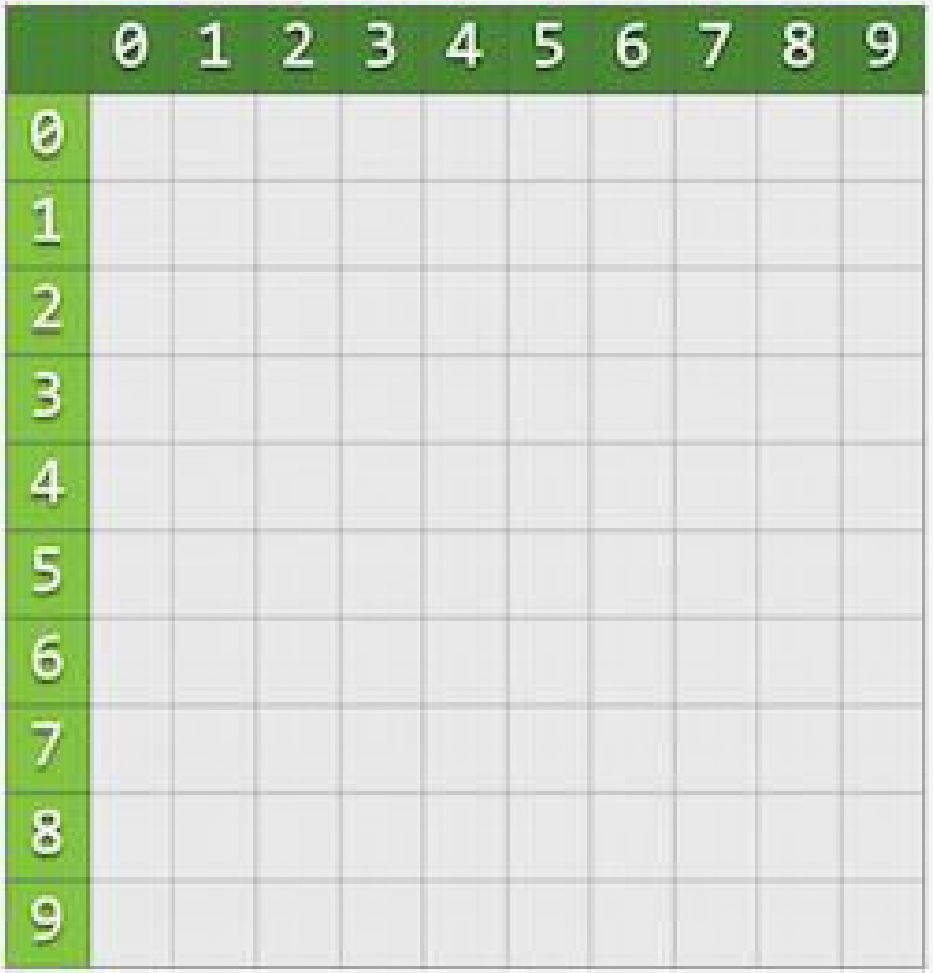
```
typ[][] nazwa_tablicy; //deklaracja
nazwa_tablicy = new typ[liczba1][liczba2]; //przypisanie (utworzenie)
typ[][] nazwa_tablicy2 = new typ[liczba1][liczba2]; //deklaracja i przypisanie
(utworzenie)

int[][] tablica = new int[3][3]; // deklarujemy siatke 3x3
tablica[2][1] = 5;
int zmienna = tablica[2][1];
```

Pomocna jest interpretacja tablicy 2d jako wiesz/kolumna

Tablica 2d - przykład

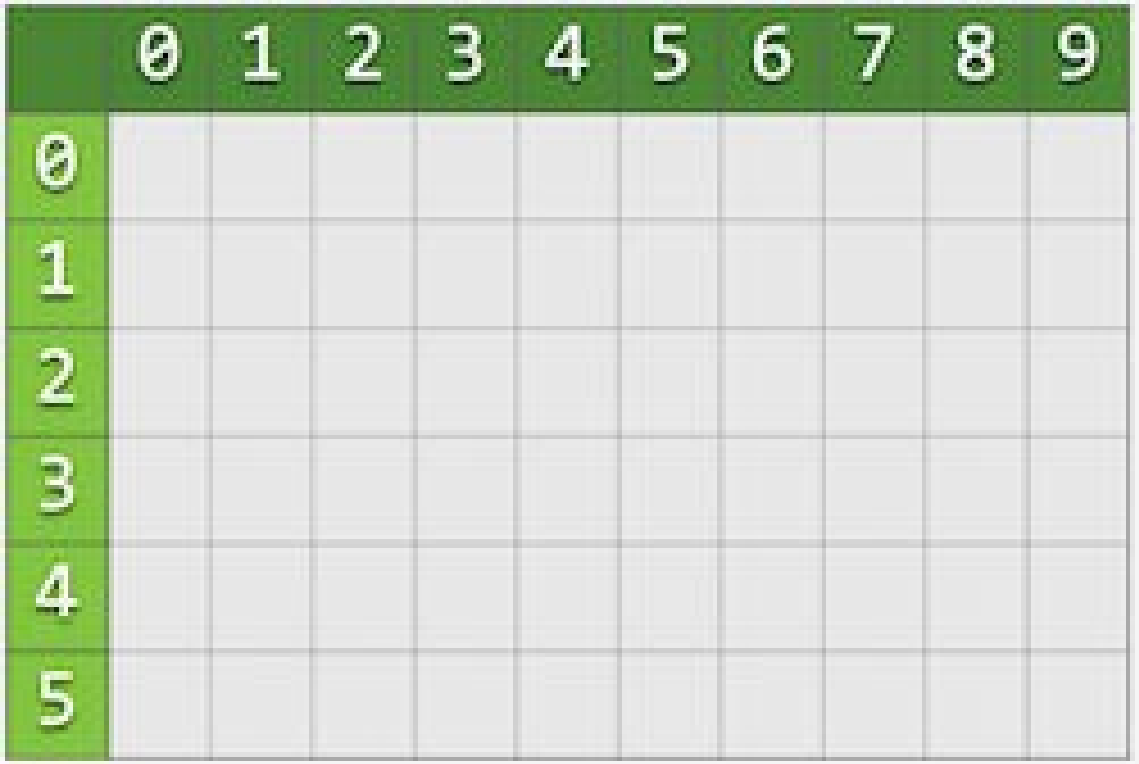
```
int[][] tablica2D = new int[10]
[10];
```



Tablice 2d - przykład

Liczba rzędów i kolumn nie musi być taka sama:

```
int [][] tablica= new int[6][10]
```



Tablice 2d = dwie pętle for

```
//Tworzenie  tablicy
int[][]  tablica2D  = new int[10][10];
// pętla zewnętrzna generuje indeksy rzędów
for (inti = 0;i < 10;  i++){
    // pętla wewnętrzna generuje indeksy kolumn
    for (intj = 0;j < 10;  j++){
        // możemy tak wyliczyć kolejną liczbę,
        ponieważ
        // każdy rząd odpowiada kolejnej dziesiątce
        tablica2D[i][j]= i*10+ j;
    }
}
```

Zadanie

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

Metody

W języku możemy definiować własne funkcje (metody), metody mogą przyjmować argumenty.

```
void metoda1(){
  System.out.println("Ta metoda nic nie zwraca, ale wyświetla ten tekst");
}

int metoda2(){
  return    //ta metoda zwraca liczbę 2 typu
} 2;        int

String metoda3(){
  return "Jakis         //ta metoda zwraca String "Jakis
  napis";               napis"
}
```

Własne metody definiujemy poza funkcją main, wywołać możemy je
wewnątrz funkcji main.

Varargs

- Specyficzny sposób przekazywania parametrów do funkcji, kiedy znamy typ parametru, a chcemy pozostawić w dowolności liczbę tych parametrów, tzn. w jednym miejscu wywołamy funkcję z 2 parametrami, w innym z 5
- Definicja funkcji:

  public String concat(int numberPrefix, String… others)
- Operator … wskazuje że funkcja przyjmie dowolną wartość obiektów String
- Varargs zawsze jest ostatnim parametrem funkcji !
- W ciele funkcji varargs traktujemy jako tablicę typu określonego przed …
- Wywołanie funkcji:

  concat(1, ”Kot”, ”Pies”, ”Mysz”);

Najpopularniejszym przykładem varags jest parametr w funkcji main.

PROGRAMOWANIE OBIEKTOWE

Programowanie obiektowe

- Sposób programowania oparty o modelowaniu rzeczywistych bytów w elementy posiadające stan i zachowania
- W programie następuje interakcja pomiędzy obiektami
- Ścisłe powiązanie danych z procedurami
- Dane to wszystko to co opisuje modelowany byt za pomocą wartości określonego typu (liczbowe, znakowe)
- Zachowania to zbiór funkcji określających odpowiedzialność tego bytu – co potrafi wykonać.

Klasa

- Klasa w języku Java stanowi definicję obiektu
- Składa się pól – opis stanu
- Oraz funkcji – opis zachowania
- Instancja klasy to obiekt

Paczki - package

- Programy Javowe nie stanowią monolitu.
- Poleca się, aby klasy dzielić na pakiety według ich odpowiedzialności

- Słówko kluczowe package określa względną lokalizację klasy w drzewie projektu, gdzie każdy kolejny poziom jest oddzielony kropką.

- Pakiety pozwalają na uporządkowanie kodu programu oraz zapobiegają konfliktom nazw.

Import

•Aby klasa mogła w swojej definicji wykorzystać inne klasy, konieczne jest zaimportowanie tej klasy poprzez instrukcję import ...;

- Import wskazuje na pakiet z jakiego pochodzi potrzebna klasa.
- Możemy importować pojedyncze klasy.
- Możemy importować całe pakiety (import nazwapakietu.*)

Klasa w języku Java – ciało klasy

- Ciało klasy zawarte jest w nawiasach klamrowych
- Język Java nie pozwala tworzyć pól
lub metod znajdujących się poza klasą

- Ciało klasy stanowią pola (dane) oraz metody (zachowania)
- Każdy element klasy może posiadać inny modyfikator widoczności

Programowanie obiektowe – konstruktor klasy

- Specyficzne metody wewnątrz klasy, które są odpowiedzialne za utworzenie instancji klasy w odpowiedni sposób (utworzenie stanu)
- Klasa może posiadać dowolną liczbę konstruktorów, ważne, aby były tworzone zgodnie
  z zasadą przeciążania metod
- Jeśli nie został jawnie utworzony konstruktor, zostanie utworzony bezparametrowy
  konstruktor
- Jeśli klasa dziedziczy po innej to w konstruktorze powinien zostać wywołany
  konstruktor klasy nadrzędnej

Klasa w języku Java – konstruktor

- Konstruktor może odwoływać się do pól oraz metod
- Aby odwołać się do elementu klasy należy użyć słówka kluczowego this
  - Aby odwołać się do elementu klasy nadrzędnej należy użyć słówka kluczowego super

public, private, protected

•Klasy, pola oraz funkcje mogą mieć zdefiniowany zakres widoczności, pozwala to programiście ukryć część implementacji klasy, która zgodnie z zasadą hermetyzacji nie powinna wyjść poza daną klasę.

- Private – element jest widziany tylko wewnątrz klasy.
- Default – brak użycia modyfikatora widoczności powoduje, że element
  jest widoczny dla klas tego samego pakietu.
- Protected – element jest widziany w klasach z tego samego pakietu oraz
  w klasach, które po niej dziedziczą
- Public – element jest widoczny na zewnątrz

Programowanie obiektowe – pola klasy

- Pole klasy opisuje w sposób określony co do typu jedną z własności klasy
- Definicja pola klasy składa się z:

<modyfikator widoczności> <typ> <nazwa>;

Np. `private int groupSize;`

Co oznacza, że pole o nazwie groupSize typu całkowitego jest prywatne.

- Pole klasy może mieć wartość domyślną:

```
private int groupSize = 0;
```

Klasa w języku Java – extends …

- Opcjonalne słówko kluczowe oznaczające że dana klasa rozszerza
  inną klasę, która musi tu zostać określona za pomocą nazwy
  - Każda klasa Javy niejawnie dziedziczy po klasie Object
  - Klasa może dziedziczyć po maksymalnie jednej klasie
  - Możliwe jest dziedziczenie wielopoziomowe,
  tzn. A extends B extends C
- Dzięki dziedziczeniu klasa otrzymuje funkcjonalność klasy rozszerzanej o widoczności public i protected

Przykład

```
class  nazwa  {
//deklaracje pól
typ  pole1;
...
typ  poleN;
//deklaracje metod
typ  metoda1(lista-
parametrów) {
//treść metody
}
...
typ  metodaM(lista-
parametrów) {
//treść metody
}
```

Klasa, która zawiera tylko trzy pola danych:

```
class Pudelko
{double
szerokosc;
double wysokosc;
double
glebokosc;
}
```

Klasa w języku Java – metody

- Opis zachowań klasy stanowi zbiór metod (funkcji)
  - W Javie definicja funkcji wygląda następująco:

<modyfikator widoczności> <zwracany typ> <nazwa>(lista parametrów)

public int getGroupSize();

- Lista parametrów może być pusta, lub zawierać dowolną liczbę elementów oddzielonych przecinkami. Każdy parametr musi mieć określony typ i nazwę (np. String name)
- Zwracany typ oznacza jakiego typu obiekt zostanie przez metodę zwrócony. Jeśli funkcja ma nie zwracać wartości należy ustawić jej zwracany typ jako *void*.

Klasa w języku Java – metody - przykład

```
public String getFullName(String name, String surName) {
        return name + ” ” + surName;
}
```

- Ciało metody, podobnie jak każdy blok kodu w Javie znajduje się w nawiasach klamrowych.
- Jeśli metoda zwraca wartość (typ inny niż void) to w ciele metody musi znaleźć się zapis
  return <obiekt>;
- Pisanie kodu po słowie return spowoduje błąd kompilacji lub jego niewykonanie.
- W metodach, które nie zwracają wartości (typ void) użycie słówka

Dziedziczenie

- Jedno z głównych założeń obiektowego programowania

- Obiekt A może dziedziczyć po obiekcie B co oznacza, że obiekt A posiada pełną funkcjonalność obiektu B oraz może ją rozszerzyć (dodać nowe funkcjonalności)
lub nadpisać (edytować istniejące funkcjonalności)

- Dziedziczenie prowadzi do powstawania drzew obiektów

- W języku Java wszystkie klasy dziedziczą po klasie Object.

Hermetyzacja

•Stan obiektu powinien być ukryty. Obiekty nie są uprawnione do zmiany danych wewnątrz innych obiektów

- Do zmiany stanu obiektu powinny służyć tylko i wyłącznie jego zachowania

- W języku Java realizujemy tą zasadę poprzez odpowiednie nadanie modyfikatorów widoczności.

Polimorfizm

- Wywołanie funkcji obiektu spowoduje zachowanie odpowiednie dla pełnego typu obiektu wywoływanego. Jeśli dzieje się to w czasie działania programu, to nazywa się to *późnym wiązaniem* lub *wiązaniem dynamicznym*.

Przykład:
Obiekty Pies i Kot dziedziczą po obiekcie Zwierzak. Wywołanie funkcji dajGlos() na obiekcie Zwierzak, spowoduje wykonanie odpowiedniego zadania („hau” lub „miau”) zależnie od tego jakiego konkretnego typu był obiekt Zwierzak.

Polimorfizm

Polimorfizmem w językach obiektowych nazywa się również sytuację,
gdzie dwie funkcje ze zbioru zachowań danego obiektu mają taką samą nazwę,
ale do ich wywołania potrzebne są różne parametry wejściowe (typ, ilość lub kolejność)

Inaczej nazwany *statycznym wiązaniem* lub *przeciążaniem metod*.

Przykład:
Obiekt Pies posiada dwie metody dajGlos, jedna, zakładająca nagrodę dla psa za wykonanie polecenia, druga nie przewidująca takiej nagrody.

dajGlos() {..}

dajGlos(nagroda) {..}

Klasa w języku Java – static

- Omawiany do tej pory stan klasy jest charakterystyczny dla danej instancji klasy (rozmiar grupy charakterystyczny dla grupy)
- Język Java pozwala na definiowanie pól w klasie, które będą współdzielone przez wszystkie instancje. Wartość takiego pola zmieniona w ramach jednego obiektu, spowoduje zmianę w pozostałych

  - Pole statyczne nie jest powiązane z konkretną instancją klasy
- Pole statyczne tworzymy poprzez dodanie słówka kluczowego static w definicji pola: private static String trainingName;

Static - przykład

```
class Test{
  static void zwieksz(int
      liczba){liczba++;
}
}

classMain{
public static void main(String[]
  args)  {
 int a = 5;Test.zwieksz(a);
     System.out.println(a);
}
}
```

Przykłady należy przepisywać i sprawdzać ich działanie. Nie polecam kopiowania kodu.

Klasa w języku Java – static

- Podobnie jak stan klasy tak jego zachowania, także są powiązane z konkretną instancją klasy.
  - Metody statyczne pozwalają na wykonanie funkcji, nie tworząc instancji klasy
  - Metody statyczne nie mogą korzystać z niestatycznych zachowań i stanu klasy (nie są wywoływane w kontekście konkretnej instancji).
- Elementy statyczne kody mogą być wywoływane z niestatycznej funkcji
- Dostęp do statycznych elementów można uzyskać poprzez obiekt lub bezpośrednio po odwołaniu do nazwy klasy

Klasa w języku Java – final

- Słówko kluczowe final zależnie od użycia może mieć różny kontekst
  - Dla klas – oznacza, że klasa ta nie może zostać rozszerzona
  - Dla metod – oznacza, że dana metoda nie może zostać nadpisana w klasie dziedziczącej
- Dla pól – oznacza, że obiekt przypisany do tej referencji nie zmieni się w cyklu życia obiektu (co nie oznacza że stan obiektu na jaki wskazuje referencja nie może się zmienić !).
  - Połączenie słówek final i static powoduje powstanie ***stałej.***

  private static final String POLISH_PHONE_CODE=„0048”;

Jak nazywać klasy, pola, metody? Konwencja

•Pola, metody, klasy, zmienne nie mogą nazywać się jak słówka kluczowe języka (np. return, ale returnAmount tak);
•Nazwy klas piszemy zgodnie z konwencją ***camelCase*** zaczynając od dużej litery (inicjał każdego wyrazu z dużej litery, reszta mała, nie używamy polskich znaków). Nazwa klasy powinna mówić co dana klasa modeluje.

- Nazwy metod piszemy zgodnie z konwencją camelCase zaczynając od małej litery. Nazwa metody powinna jednoznacznie

określać co dana metoda ma robić (nazwy powinny być możliwie zwięzłe – max 65535 ⏪).
•Nazwy pól podobnie jak metody nazywamy zgodnie z camelCase zaczynając z małej litery. Należy określić jaką wartość dane pole reprezentuje. W nazwie pola nie należy powielać nazwy klasy. (w klasie Cat, pole dotyczące koloru, może nazywać się color, a nie catColor).

- Nazwy zmiennych wewnątrz metod nazywamy podobnie jak pola.
- Do nazywania stałych należy używać wersalików z wyrazami oddzielonymi podkreśleniami, np. PI_NUMBER

Jak utworzyć nową instancję klasy?

Aby utworzyć nową instancję musimy wywołać konstruktor

- Każda klasa, w której nie został jawnie utworzony konstruktor posiada konstruktor bezparametrowy
- Chcąc utworzyć nowy obiekt klasy Cat i przypisać go do zmiennej cat muszę napisać kod:

```
Cat cat = new Cat();
```

- Jeśli chcemy skorzystać z innego konstruktora (o ile został utworzony)

należy napisać kod:

```
Cat cat = new Cat(color);
```

Gdzie color to obiekt mówiący jakiego koloru jest kot.

Posiadam instancję klasy, jak odwołać się do pól/metod?

```
Cat cat = new Cat();
```

- Aby wywołać dowolną metodą na zmiennej cat musimy napisać kod:

```
cat.sleep();
cat.eat();
```

- Aby odwołać się do stanu (pola) zmiennej cat (o ile pozwala na to modyfikator widoczności!) należy napisać kod:

```
int age = cat.length;
```

Rozwiązaniem problemu hermetyzacji jest utworzenie metody zwracającej wartość pola:

```
int age = cat.getAge();
```

Typy danych w Javie – rozszerzenie

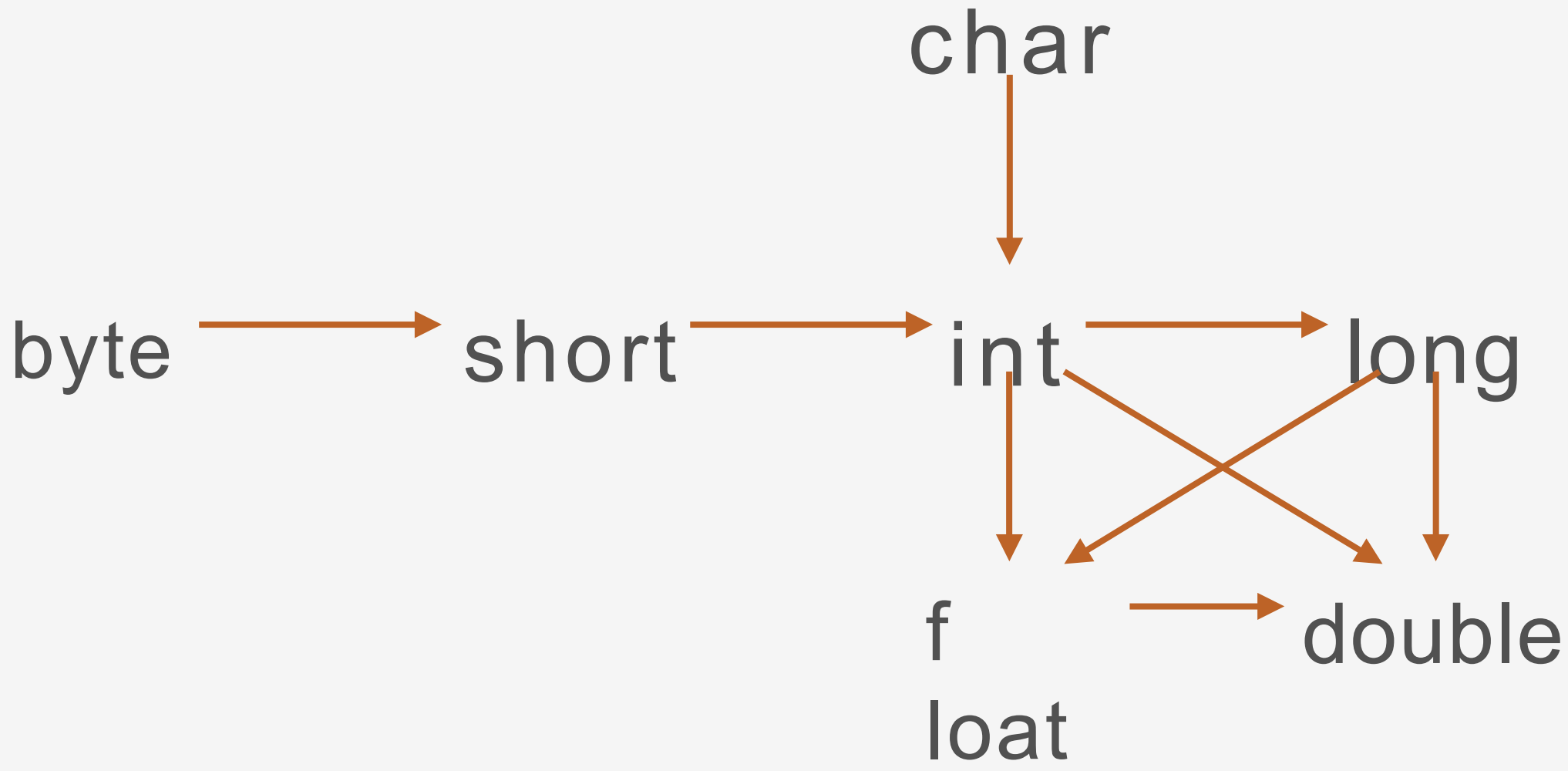
Typy podstawowe

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

Typy obiektowe

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

Typy danych -



Typ String

String jest jednym z najczęściej używanych typów zmiennej w Javie. Reprezentuje on łańcuch znakowy. String jest reprezentowany w kodzie przez łańcuch znakowy rozpoczęty i zakończony cudzysłowem.

Zmienną typu String możemy utworzyć na dwa sposoby:

- Przez konstruktor : String name = new String();
- Przez przypisane : String name = ”Adam”;

Klasa String posiada wiele wbudowanych funkcji (trim, replace, substring, length).

Aby utworzyć nową instancję klasy String poprzez sklejenie innych należy użyć operator +
Np. name + ” ” + surName;

Każda klasa posiada funkcję toString(), która umożliwia konwersję instancji do łańcucha znakowego (nie musi być jawnie określona).

Klasa Object

Każda klasa Java niejawnie dziedziczy po klasie Object.

Dzięki temu każda klasa niejawnie posiada zestaw metod opisanych w dokumentacji:
https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Najistotniejsze w tym momencie są funkcje:

- hashCode() – wyliczająca (zawsze ten sam) wartość całkowitą stanowiącą skrót obiektu
- equals(Object object) – zwracającą zmienną typu boolean, określającą czy dane obiekty są
  sobie równe
- toString() – konwersja instancji do String'a
- finalize() – metoda wywoływana w momencie niszczenia instancji w pamięci przed odśmiecacz pamięci

Każdą z tych metod można nadpisać (a oprócz finalize – należy) w swojej klasie!

HashCode i Equals - kontrakt

W momencie porównania dwóch instancji klasy na początku liczone są wartości hashCode dla obu:

- Jeśli nie są takie same to oznacza że instancje z całą pewnością są różne
- Jeśli są takie same to należy wywołać metodę equals, aby dokładnie porównać instancje.

Kontrakt:

a.hashCode() == b.hashCode() nie oznacza że a.equals(b) == true

,ale

a.equals(b) ==> a.hashCode() == b.hashCode()

Przykład, a =4, b=35, funkcja hashCode liczy x mod 31

a.hashCode() równa się 4, b.hashCode() równa się 4, ale a nie równa się b

Interfejs

- Interfejs reprezentuje listę zachowań obiektu (tylko ich definicje, bez implementacji!)
- Interfejs może posiadać stałe
- Interfejs nie posiada stanu!
- Nie można utworzyć instancji interfejsu
- Interfejs nie posiada implementacji metod (odstępstwem są domyślne
  implementacje wprowadzone w Javie 8).
- Klasa może implementować interfejs – nie rozszerzać (nie używamy extends w stosunku do interfejsów).
- Interfejs może rozszerzać inny interfejs (wtedy używamy extends)
- Klasa, która implementuje interfejs musi posiadać implementacje wszystkich metod zawartych w interfejsie !

Interfejs

```java
package com.sdajava.interfejs;

import java.io.Serializable;

public interface MainInterface extends Serializable{

    void getGroupSize();

    default String getCity(){
        return "Honolulu";
    }
}
```

```java
package com.sdajava.interfejs;

public class MainInterfaceImpl implements MainInterface {
    @Override
    public void getGroupSize() {
        System.out.printf("10");
    }

    @Override
    public String getCity() {
        return "Torun";
    }
}
```

Definicja interfejsu jest podobna do definicji klasy, słowo kluczowe class zostaje zastąpione przez interface.

Pierwsza metoda ma określony typ, nazwę i listę parametrów, ale nie posiada ciała, które musi zostać zaimplementowane w klasie implementującej tej interfejs.

Druga metoda posiada domyślną implementację zwracającą pewną wartość. Może być nadpisana w klasie implementującej interfejs.
Interfejs rozszerza inny interfejs.

Klasa implementuje podany interfejs, co oznacza że musi posiadać nadpisane wszystkie funkcje interfejsu, które nie posiadają domyślnej implementacji.

Dodatkowo nadpisana została metoda getCity(), która w wypadku wywołania metody na instancji tego typu spowoduje zwrócenie wartości Torum zamiast Honolulu

Zadanie: Utwórz interfejs zawierający zbiór zachowań klasy z poprzedniego zadania. Utwórz chociaż jedną metodą z domyślną implementacją

Klasy abstrakcyjne

- Klasa abstrakcyjna może posiadać stan
- Nie można utworzyć instancji klasy abstrakcyjnej
- Klasa abstrakcyjna może posiadać część metod zaimplementowanych, a część znanych tylko z sygnatury jak w przypadku interfejsu
- Klasa może rozszerzać klasę abstrakcyjną (poprzez extends), musi wtedy posiadać zaimplementowane wszystkie metody klasy abstrakcyjnej nie posiadające implementacji
- Klasa abstrakcyjna może rozszerzać inną klasę (niekoniecznie abstrakcyjną)
  lub implementować interfejs
- Dobrym zwyczajem jest rozpoczynanie nazwy klasy abstrakcyjnej od słówka Abstract

Klasy abstrakcyjne

```java
package com.sdajava.interfejs;

public abstract class AbstractMainClass
    implements MainInterface {

    private String courseName = "Java";

    public abstract String getTeacherName();

}
```

```java
package com.sdajava.interfejs;

public class MainImpl
    extends AbstractMainClass{

    @Override
    public String getTeacherName(){
        return "Lukasz";
    }

    @Override
    public void getGroupSize(){

    }
}
```

Definicja klasy abstrakcyjnej od definicji klasy różni się obecnością
słówka kluczowego abstract.
Każda metoda, która ma zostać niezaimplementowana musi także w swojej sygnaturze posiadać słówko abstract oraz nie posiadać ciała.
Klasa abstrakcyjna może mieć stan.

Klasa rozszerzająca klasę abstrakcyjną posiada implementację abstrakcyjnej metody z klasy abstrakcyjnej oraz implmentację metody z interfejsu, który implementuje klasa abstrakcyjna!

Typy wyliczeniowe - enum

Typ zawierający listę wartości jaką zmienna danego typu może przyjąć,
np. status książki w wypożyczalni.

```
public enum BookStatus{
	WYPOZYCZONY,
	W_BIBLIOTECE,
	ZNISZCZONA,
	ZGUBIONA;
}
```

Do wartości enuma odwołujemy się jak do pola statycznego w klasie, ti

```
BookStatus status =
BookStatus.ZAGUBIONA;
```

Klasa anonimowa

Klasa anonimowa to klasa nie posiadająca nazwy. Definicja takiej klasy jest tworzona zawsze w momencie tworzenia nowej instancji. Klasy anonimowe najczęściej są jednokrotną, nigdzie indziej nie używaną implementacją klasy abstrakcyjnej lub interfejsu.

Klasa wewnętrzna

Klasa wewnętrzna to klasa, której definicja znajduje się wewnątrz innej klasy. Klasa wewnętrzna może mieć stan, metody, konstruktory.

Jeśli klasa A znajduje się w klasie B, to odwołanie do niej następuje przez kropkę: A.B.poleStatyczne

Utworzenie obiektu: new A().new B();

Klasy wewnętrzne zależnie od modyfikatory widoczności mogą być nie widoczne na zewnątrz klasy otaczającej.

Adnotacje

Znacznik w kodzie, stanowi metainformację dla kompilatora, może być umieszczony na klasie,
metodzie, polu, zmiennej.

Posiada formę @NazwaAdnotacji. Popularne adnotacje:
@Override – oznacza, że metoda nadpisuje metodę z klasy nadrzędnej
@SuppressWarnings – oznacza, że programista jest świadomy zagrożeń w kodzie i prosi kompilator o wygaszenie ostrzeżeń.

Typy generyczne

- Generyczność pozwala nam parametryzować klasy i metody
- Zwiększa uniwersalność klasy
- Jest to jeden ze sposobów na wprowadzenie polimorfizmu w kodzie
- Typ parametru metody lub typ pola klasy wybieramy dopiero w momencie jej użycia, a nie konstruowania
- Do użycia typów generycznych służy operator <>

Typy generyczne

- Generyczność możemy wprowadzać na poziomie klas i metod
- Klasa – generyczny kubek – nie wiemy co jest w środku, tworząc instancję musimy zadecydować czym będzie zawartość

Konwencja nazewnictwa:

- E - Element (stosowany z Java Collections Framework)
- K – Key
- N – Number
- T – Type
- V – Value
- S, U, V etc. - 2nd, 3rd, 4th types

https://docs.oracle.com/javase/tutorial/java/generics/types.html

Typy generyczne

- Klasa – generyczny kubek – w środku powinno znajdować się cokolwiek co jest napojem, ale zawsze napój tego samego typu

```
public class GenericCup <T extends Drink> {

    private T content;
}
```

Kubek może zostać sparametryzowany, aby przyjmować napój
tylko konkretnego typu, np. mój ulubiony kubek na kawę.
W takim przypadku kubek może zawierać tylko kawę, nie możliwe jest
wlanie do niego herbaty.

Aby utworzyć nowy kubek na kawę, musimy wywołać kod:

GenericCup<Coffee> coffeeCup = new GenericCup<>();

Klasa, która rozszerza generyczny kubek powinna określić jakiego typu
zawartość będzie w kubku:

Typy generyczne

- Metody – sparametryzować możemy zwracany typ:

public <T extends Drink> fillCup(String name, Class<T> type)
Konieczne jest przekazanie jako parametr funkcji typu generycznego (obiekt Class można pobrać z każdej klasy i obiektu poprzez wywołanie metody getClass())

- Parametry :

public static <T> void copy(List<T> dest, List<? extends T> src)
Składnie <? extends T> oznacza że akceptowany jest dowolny typ rozszerzający typ generyczny T.

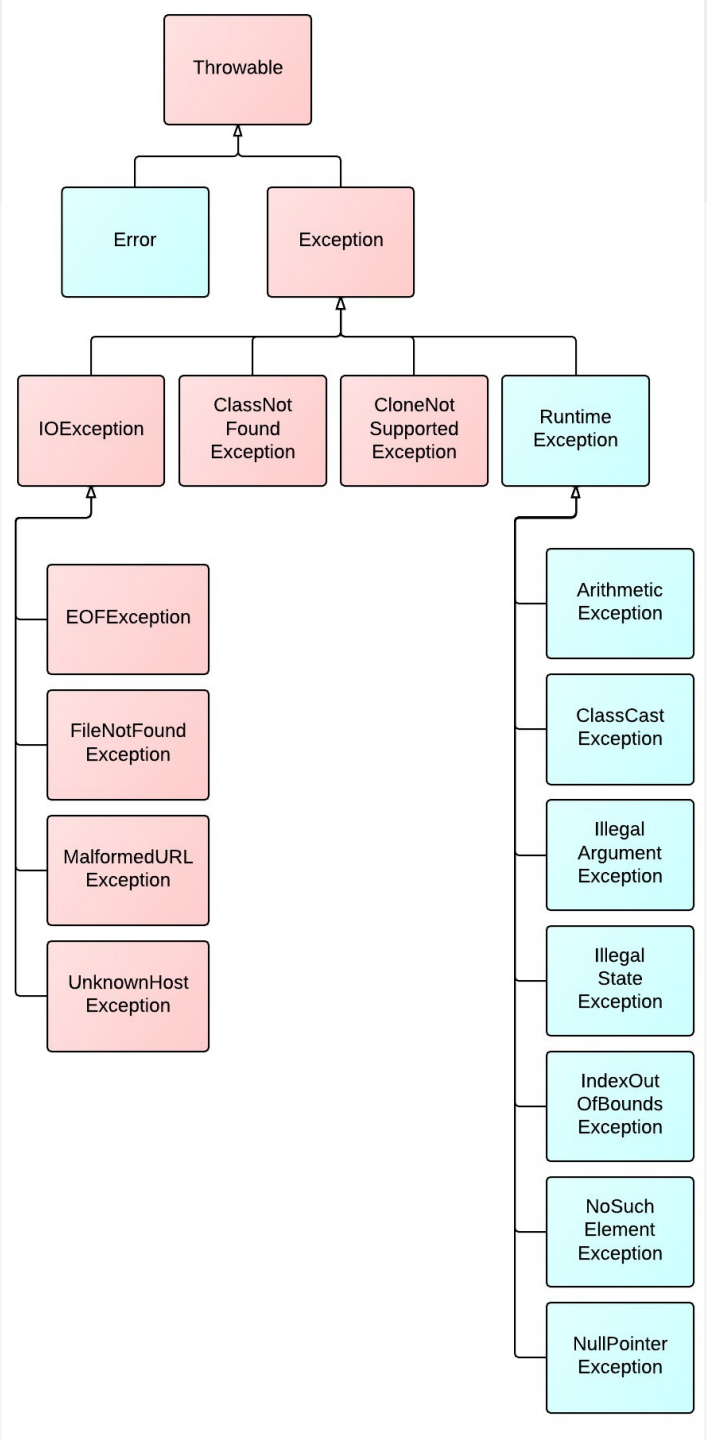
Symbole typów generycznych oznacza się najczęściej dużą pojedynczą literą (T, R, S).
Klasa lub metoda może posiadać wiele różnych typów generycznych (wymienione po przecinku)

Typy generyczne - ograniczenia

- tworzyć obiektów typów sparametryzowanych (new T())
- używać operatora instanceOf (z powodu j.w.),
- używać ich w statycznych kontekstach (bo statyczny kontekst jest jeden dla wszystkich różnych instancji typu sparametryzowanego) – dotyczy klas
- wywoływać metod z konkretnych klas i interfejsów, które nie są zaznaczone jako górne ograniczenia parametru typu (w najprostszym przypadku tą górną granicą jest Object, wtedy możemy używać tylko metod klasy Object).

Wyjątki w Javie

- Wyjątki jest to mechanizm obsługi błędów w języku Java.
- Każdy wyjątek musi być obiektem klasy Throwable lub klasy z niej
  dziedziczącej.
- Wyjątki dzielimy na dwie grupy: checked i unchecked.
- Wyjątki checked to wyjątki dziedziczące po klasie Exception.
- Są to wyjątki, których wystąpienie musi zostać jawnie określone i muszą zostać obsłużone.
- Wyjątki unchecked to wyjątki dziedziczące po klasie RuntimeException.
- Są to wyjątki, których deklaracja i obsłużenie jest dobrowolnie.


Throwable
Error
Exception
IOException
ClassNot Found Exception
CloneNot Supported Exception
Runtime Exception
EOFException
FileNotFound Exception
MalformedURL Exception
UnknownHost Exception
Arithmetic Exception
ClassCast Exception
Illegal Argument Exception
Illegal State Exception
IndexOut OfBounds Exception
NoSuch Element Exception
NullPointer Exception

Wyjątki w Javie – try catch

- Blok try – catch –finally zapewnia nam wykonanie bloku kodu w
  sposób zapewniający obsługę potencjalnych błędów

- try zawiera blok kodu, który może powodować błąd

- catch to blok zawierający kod, w którym obsługujemy błąd

- finally to blok kodu, który wykona się zawsze, nie zależnie czy w
  bloku try wystąpi błąd

Wyjątki w Javie – try catch

Bloków catch może być wiele, należy pamiętać aby je układać „od szczegółu do ogółu”.

Tj. jeśli zrobimy obsługę wyjątku nadrzędnego przed obsługą wyjątku dziedziczącego to wykona się blok kodu
dla ogólniejszego wyjątku

Od Javy 7 bloki catch dla różnych wyjątków można łączyć za pomocą operatora |.

Bloki catch i finally są opcjonalne, ale przynajmniej jeden z nich musi się pojawić.

```
import java.util.Scanner;

public class Odczyt{
  public static void main(String[] args){
      int tab[] = {1,2,3,4,5};
      Scanner odczyt = new Scanner(System.in);
      int index = -1;

      System.out.println("Podaj indeks tablicy, który chcesz zobaczyć: ");
      index = odczyt.nextInt();

      try {
          System.out.println(tab[index]);
      } catch (ArrayIndexOutOfBoundsException e) {
          System.out.println("Niepoprawny parametr, rozmiar tablicy to: "+tab.length);
      }
  }
}
```

Wyjątki w Javie – throws a throw

- Throws jest elementem sygnatury funkcji, określa jakie wyjątki może rzucić ta funkcja. Standard języka wymaga zapisania wyjątków checked, zapisanie wyjątków unchecked jest dobrowolne (pisze się je tylko w uzasadnionej sytuacji, aby nie zaciemniać kodu).

```
public String showArrayElements(int[] i) throws IndexOutOfBoundsException
```

- Throw służy do wywołania wyjątku. Aby go wywołać należy utworzyć nową instancję.

```
throw new MyBusinessException();
```

Wywołanie wyjątku w naszym kodzie może spowodować konieczność dodania throws w sygnaturze metody.

Wyjątki nie muszą zostać obsłużone w metodzie, która wywołała błędną funkcję. Dodanie throws na tej metodzie przeniesie odpowiedzialność za obsługę błędów do metody wyżej.

Wyjątki w Javie – własne wyjątki

- Wyjątek jest zwykłą klasą
- Chcąc napisać własny wyjątek tworzymy klasę rozszerzającą Exception lub RuntimeException

- Własne wyjątki wykorzystujemy i obsługujemy jak te wbudowane w język Java

Wyjątki w Javie – co zawiera wyjątek

Z wyjątku możemy odczytać:

- Klasę wyjątku
- Wiadomość zapisaną w wyjątku
- Dodane pola (w przypadku naszego wyjątku polską wiadomość)
- Stos wywołań funkcji (kolejne wywołania funkcji odłożone na stos – pomaga programiście
  w procesie debugowania kodu oraz określa miejsce gdzie nastąpiła sytuacja wyjątkowa)

- Wyświetlenie stosu wywołań na standardowym wyjściu – funkcja printStackTrace();

Zapis do plików

Do zapisu danych do pliku można użyć klasy **PrintWriter**, klasa posiada statyczną metodę **.println()**

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Notes{

     public static void main(String[] args) throws FileNotFoundException
     {
               PrintWriter tekstowy = new PrintWriter("plik.txt");
               tekstowy.println("Test zapisu do pliku");
               tekstowy.close(); //Należy pamiętać o zamknięciu pliku!
     }
}
```

Zapis do pliku – ZADANIE

Napisz program, który będzie pobierał od użytkownika wyniki meczów piłkarskich i zapisywał je linia po linii do pliku w postaci:

```
Drużyna1 <Bramek>(Karne) : (Karne)<Bramek>Drużyna2
```

Jeśli mecz się rozstrzygnął wynikiem bramkowym, to nie pytaj użytkownika o karne i nie zapisuj ich do pliku.

Przykład:

```
Brazylia <1> : <2>Belgia
Rosja <2>(3) : (4)<2>Chorwacja
Szwecja <1> : <0>Szwajcaria
```

Podpowiedź, przeczytaj od użytkownika najpierw nazwę pierwszej drużyny, potem nazwę drugiej drużyny. Następnie przeczytaj wynik jeden i drugi, jeżeli są takie same zapytaj o karne. Na koniec za pomocą operacji konkatenacji (+) połącz wszystkie dane i zapisz do pliku.

Odczyt z pliku

Odczyt danych następuje przy użyciu klasy Scanner (tej samej, której używaliśmy przy pobieraniu tekstu wpisanego z klawiatury). Konstruktor Scannera wymaga podania obiektu klasy File jak przedstawiono poniżej:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Read{
    public static void main(String[] args) throws FileNotFoundException{
        File file = new File("C:/SDA/demo.txt");
        Scanner input = new Scanner(file);
        String line = input.nextLine(); //można użyć pętli i przeczytać cały plik
        System.out.println(line);
   }
}
```

Java.NIO – New I/O

Poznane wcześniej sposoby pracy z plikami mają swoje wady, dlatego powstała nowa wersja w standardowej bibliotece języka Java do pracy z plikami jest to **java.nio**.

- Umożliwia nieblokujące odpytywanie (program może działać dalej oczekując na nadejście danych)
- Jest bardziej wydajne pamięciowo
- Umożliwia uproszczenie komunikacji między wątkami (poprzez zapis i odczyt do potoków)

Java.NIO – New I/O

Podstawowym obiektem jest **Path** reprezentujący ścieżkę do pliku:

```
Path sciezka = Paths.get("C:/SDA/test.txt");
```

Zaletą stosowania Path nad zwykłym stringiem jest dostępność wygodnych metod manipulacji m.in.:

`getFileName()` – zwraca nazwę pliku
`getName(int index)` – zwraca element ścieżki o podanym indeksie
`getParent()` – zwraca nadrzędną część ścieżki (jak wyjście folder wyżej)
`getRoot()` – zwraca katalog główny

Java.NIO – New I/O

Do operacji na pliku służy klasa **`Files`**. Operując na **`Path`** udostępnia ona metody manipulacji plikami i katalogami m.in.:

`exists(Path, LinkOption)` – Sprawdza czy ścieżka istnieje
`copy(Path, Path, CopyOption)` – Kopiuje z miejsca na miejsce
`move(Path, Path, CopyOption)` – Przenosi z miejsca na miejsce
`deleteIfExists(Path)` – Usuwa (jeśli istnieje)
`size(Path)` – Zwraca wielkość
`isDirectory(Path, LinkOption)` – Sprawdza czy jest katalogiem
`readAllLines(Path, Charset)` – Odczytuje wszystkie linie (małe pliki)
`newInputStream(Path, OpenOption…)` – Odczyt strumieniowy (jak w **java.io**)
`newOutputStream(Path, OpenOption…)` – Zapis strumieniowy (jak w **java.io**)

Data i czas

- W języku Java mamy obecnie 3 implementacje daty:
  java.util.Date,
  java.sql.Data i java.time.LocalDate
- Ostatnia implementacja pochodzi z najnowszej wersji Javy i jest obowiązująca.
- LocalDate reprezentuje datę, LocalDateTime posiada również reprezentacje czasu
- LocalDateTime.now() – zwróci obiekt zawierający aktualną datę (komputera).
- new LocalDateTime() jak wyżej

Data i czas - formater

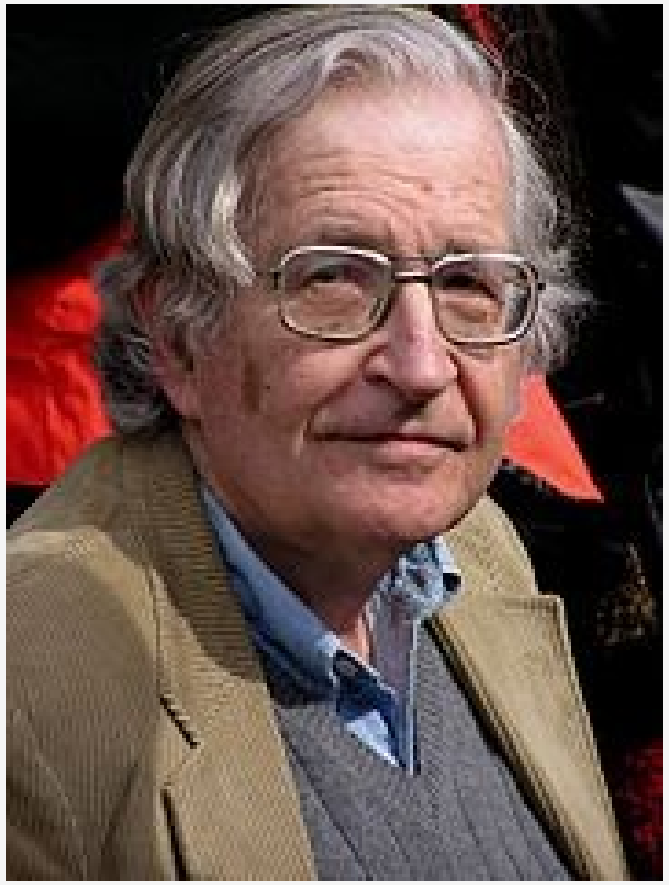
- Obiekt daty może być tworzony z dowolnego napisu w dowolnym formacie
- Wystarczy przekazać obiekt formatera podczas parsowania daty.

```
DateTimeFormatter formatter =
        DateTimeFormatter.ofPattern(„dd-MM-yyyy");
LocalDate date = LocalDate.parse(input, formatter);

String formattedDate = date.format(formatter);
```

Gramatyki

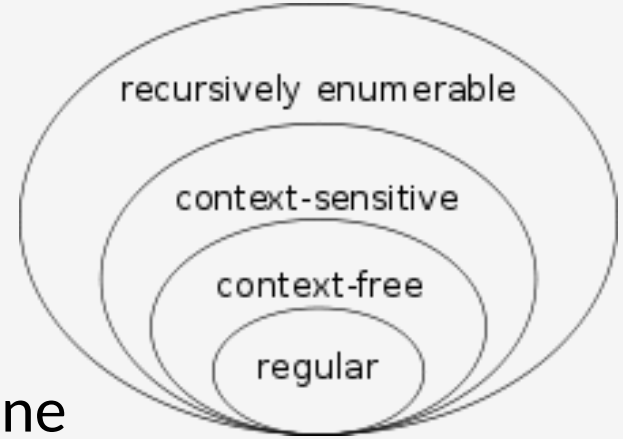
Każdy format, który można przeanalizować, musi mieć gramatykę deterministyczną składającą się ze słownictwa i reguł składni. Nazywa się to gramatyką bezkontekstową.



Hierarchia Chomsky'ego – stworzona przez Noama Chomsky'ego hierarchia klas języków formalnych.

Hierarchia składa się z czterech klas:

- języki typu 3 – regularne
- języki typu 2 – bezkontekstowe
- języki typu 1 – kontekstowe
- języki typu 0 – rekurencyjnie przeliczalne



Wyrażenia regularne

- **Wyrażenia regularne** pozwalają nam tworzyć pewien wzorzec do wyszukiwania w ciągu znaków.
  Wzorzec może zostać dopasowany jeden, wiele razy, lub wcale
- Mogą być wykorzystywane do wyszukiwania, edycji, manipulacji i usuwania tekstu.
- Wzorzec jest dopasowywany zawsze od lewej do prawej.
  Litera wykorzystana przy jednym wyszukaniu nie podlega już kolejnemu np. ciąg "aba" w tekście **aba**b**aba**ba zostanie dopasowany tylko dwa razy

Wyrażenia regularne

Wyrażenia regularne mają postać zwykłego literału Stringa

Przykłady prostych wyrażeń regularnych:

Regex	**Dopasowuje**
.	Pojedynczy znak
to jest tekst	Dopasowuje dokładnie tekst "to jest tekst"
^tekst	Początek linii, a potem "tekst" (czyli tekst na początku linii)
tekst$ [abc] [abc][xy]	"tekst", a potem koniec linii (czyli tekst na końcu linii) Dowolny znak ze zbioru (a lub b lub c). Znak a lub b lub c, a następnie znak x lub y (np. ay, cx)

Wyrażenia regularne

Regex	Dopasowuje
[^abc]	Gdy pierwszy wewnątrz nawiasów klamrowych [] daszek ^ zmienia znaczenie na negację (dowolny znak oprócz abc)
[a-d1-7]	Dowolny znak między "a" a "d" LUB dowolny znak między 1 a 7
X|Y	Szuka X lub Z
XY	znak X a następnie znak Y

Regex	Dopasowuje
.[a-z]X$	Dowolny znak, następnie mała litera od a-z, następnie X i koniec linii

Wyrażenia regularne

Meta znaki mają predefiniowane znaczenie (inaczej: są to skróty do pewnych wyrażeń)

Regex Dopasowuje
\d Dowolna cyfra (jak [0-9])
\D NIE cyfra (jak [^0-9])
\s Dowolny biały znak (jak [\t\n\x0b\r\f])
\S NIE biały znak (czyli dowolny inny znak)
\w Dowolny znak słowny (jak [a-zA-Z_0-9])
\W NIE dowolny znak słowny (jak [^\w]
\S+ Kilka nie białych znaków
\b Dopasowuje granice słowa, gdzie słowo składa się z [a-zA-Z_0-9]

Wyrażenia regularne

Oprócz meta znaków wyrażenia regularne tworzone są też przez **kwantyfikatory** mówiące **ile razy** znak może wystąpić.

Regex Przykład Dopasowuje

* X* Zero lub więcej razy (skrót dla {0,})

+ .+ Jeden lub więcej razy (skrót dla {1,})

? ? Jeden raz lub wcale (skrót dla {0,1})

{X} \d{3} Dokładnie X razy.

{X,Y} \d{1,4} Conajmniej X razy, ale maksymalnie Y razy

*? Powoduje, że regex stara się znaleźć najmniejsze dopasowanie i przestaje szukać dalej

Wyrażenia regularne

Jeśli w stringu występuje znak \ to musimy go zamienić na \\ bo pojedynczy \ ma specjalne zadanie.
Operatory nawiasów ( ) grupują wyrażenia (jak w matematyce). Pozwala to stosować kwantyfikatory do grup.
Zastosowanie regex do usuwania spacji przed kropką lub przecinkiem.

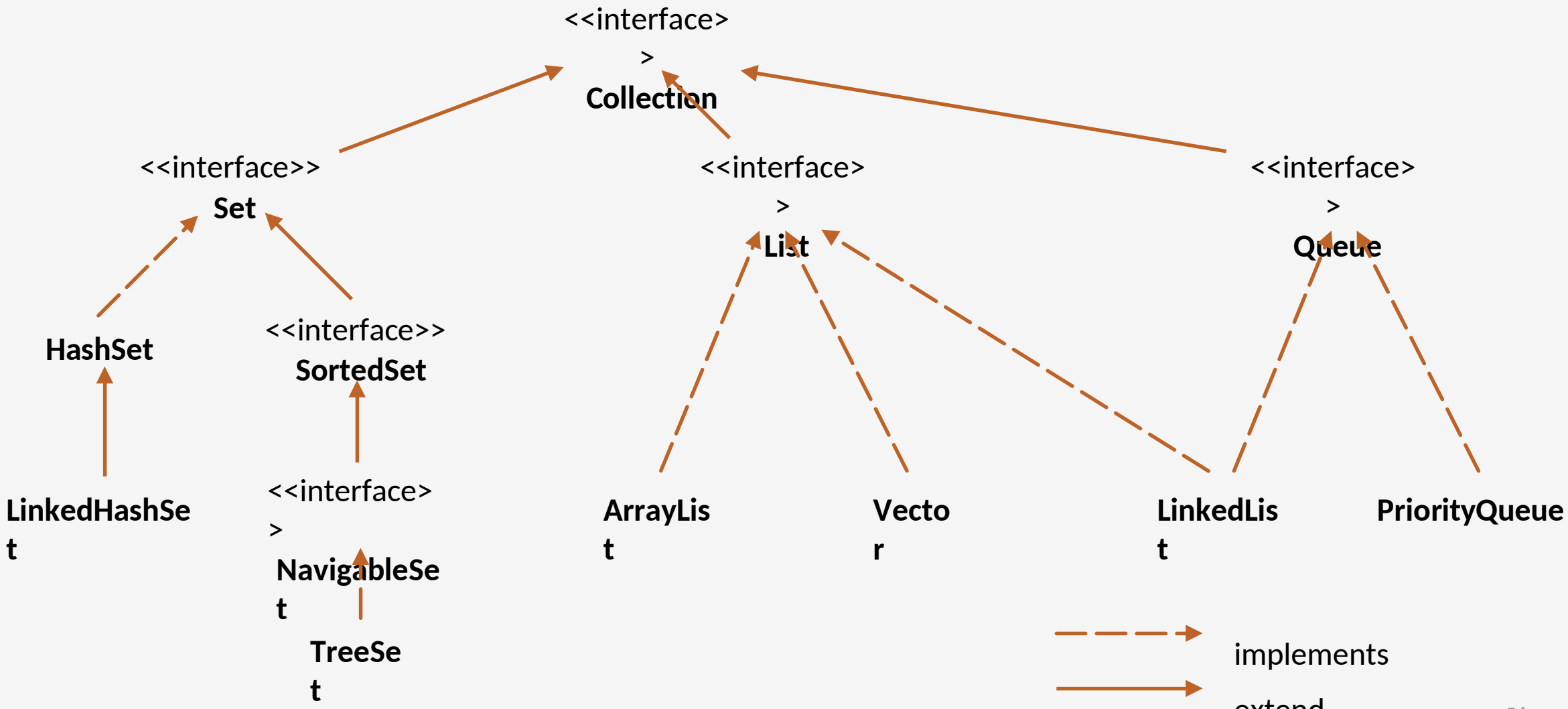
```
String wzor = "(\\w)(\\s+)([\\.,])";
System.out.println("Koniec . zzz .".replaceAll(wzor,"$1$3"));
```

Wyrażenia regularne

W klasie String występuje kilka niezwykle przydatnych funkcji do manipulacji tekstem, które przyjmują regex.

Funkcja	**Dopasowuje**
str.matches("regex")	Zwraca true gdy cały str pasuje do regex
str.split("regex")	Dzieli str na tablicę stringów dzieląc (i usuwając) to do czego regex się dopasował
str.replaceFirst("regex", "zastap")	Zastępuje pierwsze wystąpienie "regex" stringiem z drugiego parametru "zastap"
str.replaceAll("regex", "zastap")	Zastępuje wszystkie wystąpienia "regex" stringiem "zastap". Jeśli regex składa się z części to $1 odwołuje się do pierwszej $2

Kolekcje


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
HashSet
<<interface>>
SortedSet
LinkedHashSet
<<interface>>
NavigableSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
implements
extends

Kolekcje

- Wszystkie kolekcje w Javie implementują interfejs Collection
- Kolekcje służą jako magazyny obiektów
- Nie mogą przechowywać typów prostych (mogą obiekt po autoboxingu)
- W wyborze kolekcji należy uwzględnić do czego będzie wykorzystana, jakie operacje będą na niej najczęściej wykonywane
- Najbardziej znane kolekcje to List (ArrayList, LinkedList), Set

Kolekcje - List

- List – implementacje interfejsu List przechowują obiekty w postaci posortowanej listy,
  pozwalają przechowywać duplikaty

Klasa	Cechy	Zastosowanie
ArrayList	• w ‚tle' przechowuje dane w tablicy, której samodzielnie zmienia rozmiar wg potrzeb •dostęp do dowolnego elementu w czasie O(1) •dodanie elementu O(1), pesymistyczny przypadek O(n) •usunięcie elementu O(n)	Najczęstszy wybór z racji najbardziej uniwersalnego zastosowania. Inne imeplementacje mają przewagę tylko w bardzo specyficznych przypadkach. Jeśli nie wiesz, jakiej listy potrzebujesz, wybierz tą.
LinkedList	• przechowuje elementy jako lista powiązanych ze sobą obiektów (tj. pierwszy element wie, gdzie jest drugi, drugi wie, gdzie trzeci itd) •dostęp do dowolnego elementu O(n) (iteracyjnie O(1)) •dodanie elementu O(1) •usunięcie elementu O(1)	LinkedList ma przewagę w przypadku dodawania elementów pojedynczo, w dużej ilości, w sposób trudny do przewidzenia wcześniej, kiedy przejmujemy się ilością zajmowanej pamięci.W praktyce nie spotkałem się z sytuacją, w której LinkedList byłoby wydajniejsze od ArrayList
Vector	•Istnieje od początku Javy, z założenia miała to być obiektowa reprezentacja tablicy •Funkcjonalność i cechy analogiczne do ArrayList, ale znacznie mniej wyrafinowane	API oficjalnie zaleca korzystanie z klasy ArrayList zamiast Vector

Kolekcje - Set

- Set to kolekcja przechowująca unikalne wartości. Większość implementacji nie zachowuje kolejności.

Klasa	Cechy	Zastosowanie
HashSet	•zbiór nieposortowany •kolejność iteracji nieokreślona, może się zmieniać •dodanie elementu oraz sprawdzenie czy istnieje ma złożoność O(1) •pobranie kolejnego elementu ma złożoność O(n/h), gdzie h to parametr wewnętrzny (pesymistyczny przypadek: O(n))	Sytuacje, kiedy nie potrzebujemy dostępu do konkretnego elementu, ale potrzebujemy często sprawdzać, czy dany element już istnieje w kolekcji (czyli chcemy zbudować zbiór unikalnych wartości).W praktyce często znajduje zastosowanie do raportowania/zliczeń oraz jako ‚przechowalnia' obiektów do przetworzenia (jeśli ich kolejność nie ma znaczenia).
LinkedHashSet	•analogiczne, jak HashSet (dziedziczy po nim) •kolejność elementów używając iteratora jest deterministyczna i powtarzalna (zawsze będziemy przechodzić przez elementy w tej samej kolejności) •pobranie kolejnego elementu ma złożoność O(1)	Jesto to mniej znana i rzadziej stosowana implementacja, często może ona zastąpić HashSet poza specyficznymi przypadkami. W praktyce do iterowania w określonej kolejności często używane sa inne kolekcje (Listy, kolejki)
TreeSet	•Przechowuje elementy posortowane wg porządku naturalnego (jeśli implementują one interjfejs Comparable, w przeciwnym wypadku porządek jest nieokreslony) •Wszystkie operacje (dodanie, sprawdzenie czy istnieje oraz pobranie kolejnego elementu) mają złożoność O(log n)	Jeśli potrzebujemy, aby nasz zbiór był posortowany bez dodatkowych operacji (sortowanie nastepuje już w momencie dodania elementu) oraz iterować po posortowanej kolekcji.

Kolekcje - Map

- Map to kolekcja przechowująca pary klucz-wartość. Każdemu przechowywanemu w
  mapie obiektowi towarzyszy unikalny klucz (liczba, String, dowolny obiekt).

Klasa	Cechy	Zastosowanie
HashMap	•mapa nieposortowana •kolejność iteracji nieokreślona, może się zmieniać •dodanie elementu oraz sprawdzenie czy klucz istnieje ma złożoność O(1) •pobranie kolejnego elementu ma złożoność O(h/n), gdzie h to parametr wewnętrzny	Ogólny przypadek, tworzenie lokalnej pamięci podręcznej czy ‚słownika' o ograniczonym rozmiarze, zliczanie wg klucza.
LinkedHashMap	•analogiczne, jak HashMap (dziedziczy po niej) •kolejność kluczy używając iteratora jest deterministyczna i powtarzalna (zawsze będziemy przechodzić przez klucze w tej samej kolejności) •pobranie kolejnego elementu ma złożoność O(1)	Podobnie jak LinkedHashSet, rzadziej znana i stosowana. Przydatna, jeśli potrzebujemy iterować po kluczach w założonej kolejności.
TreeMap	•Przechowuje elementy posortowane wg porządku naturalnego kluczy (jeśli implementują one interjfejs Comparable, w przeciwnym wypadku porządek jest nieokreslony) •Wszystkie operacje (dodanie, sprawdzenie czy klucz istnieje oraz pobranie kolejnego elementu) mają złożoność O(log n)	Jeśli potrzebujemy, aby nasz zbiór był posortowany wg kluczy bez dodatkowych operacji (sortowanie nastepuje już w momencie dodania elementu) oraz iterować po posortowanej kolekcji.
Hashtable	• Historyczna klasa, która w Javie 1.2 została włączona do Java Collecion API	Oficjalnie zaleca się korzystanie z HashSet w większości wypadków zamiast Hashtable

Kolekcje – tworzenie i operacje

	List	Set	Map
Tworzenie	List<String> lista = new ArrayList<>();	Set<String> set = new HashSet<>();	Map<String, Object> mapa = new HashMap<>();
Dodawanie	lista.add(”obiekt1”); lista.addAll(innaLista);	set.add(”obiekt1”) ;	mapa.put(klucz, wartosc);
Usuwanie	lista.remove(obiekt); lista.removeAll(innaLista);	set.remove(obiekt);	mapa.remove(klucz);
Wyszukiwanie	lista.get(index)	Użycie klasy Iterator do pobrania kolejnego elementu. set.iterator().next();	mapa.get(klucz);

Kolekcje - iteracja

- Kolekcje podobnie jak tablice możemy w prosty sposób przeglądać za pomocą pętli foreach

- Dla listy i seta:

```
List<String> listaStringow = new ArrayList<>();
for (String element : listaStringow) { System.out.println(element);
}
```

- Dla mapy:

```
Map<String, String> map = new HashMap<>();
for (Map.Entry<String, String> entry : map.entrySet())
{
   System.out.println(entry.getKey() + "/" + entry.getValue());
}
```

WĄTKI

Proces to wykonujący się program wraz z dynamicznie przydzielanymi mu przez system zasobami (np. pamięcią operacyjną, zasobami plikowymi).
Każdy proces ma własną przestrzeń adresową.

Systemy wielozadaniowe pozwalają na równoległe (teoretycznie) wykonywanie wielu procesów, z których każdy ma swój kontekst i swoje zasoby.

Wątek to sekwencja działań, która wykonuje się w kontekście danego
procesu (programu)

Każdy proces ma co najmniej jeden wykonujący się wątek.
W systemach wielowątkowych proces może wykonywać równolegle (teoretycznie) wiele wątków, które wykonują

WĄTKI

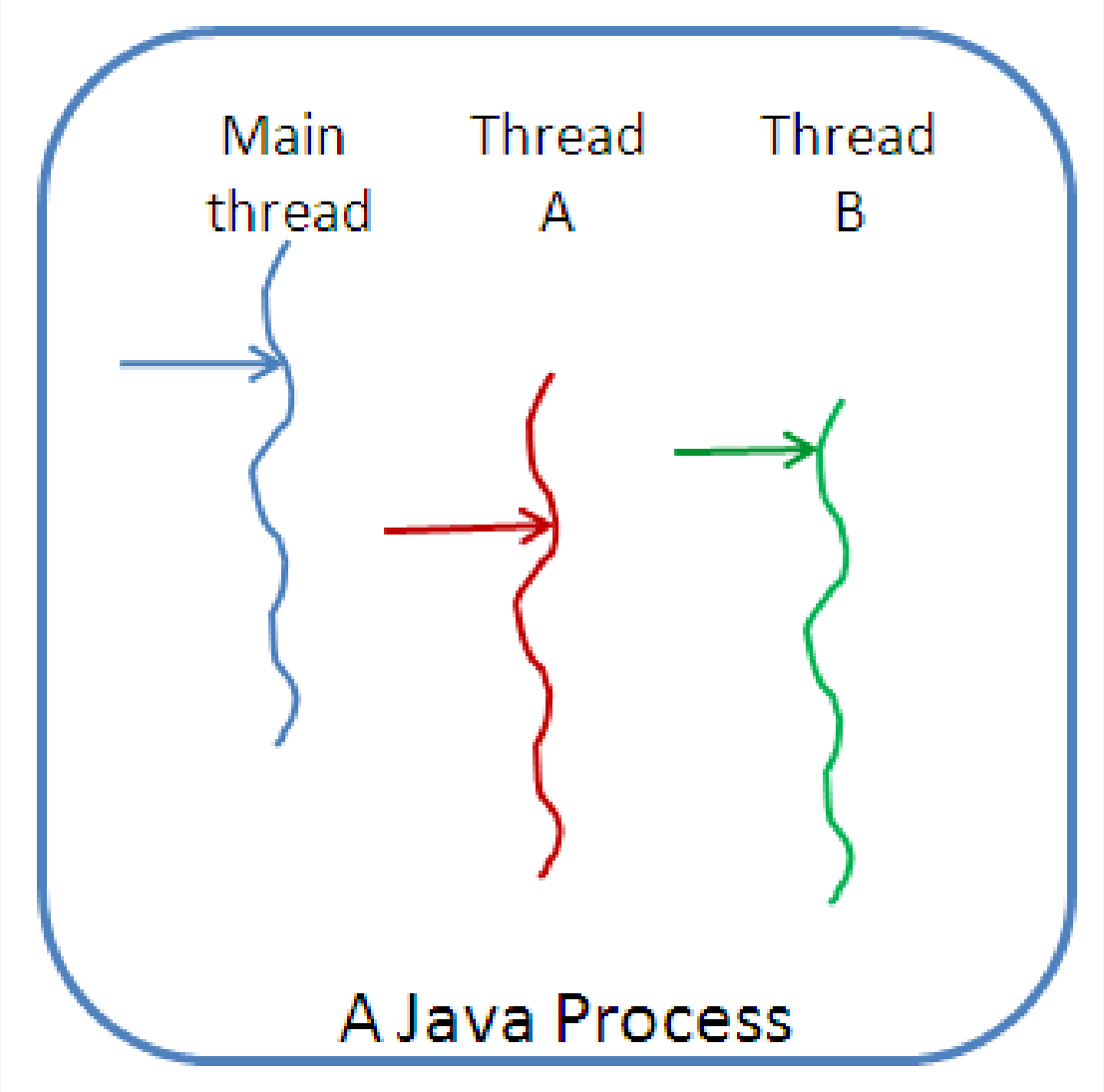
Równoległość działania wątków osiągana jest przez mechanizm przydzielania czasu procesora poszczególnym wykonującym się wątkom. Każdy wątek uzyskuje dostęp do procesora na krótki czas (kwant czasu), po czym „oddaje procesor” innemu wątkowi. Zmiana wątku wykonywanego przez procesor może dokonywać się na zasadzie:

współpracy (cooperative multitasking), wątek sam decyduje, kiedy oddać
czas procesowa innym wątkom,

wywłaszczania (pre-emptive multitasking), o dostępie wątków do procesora decyduje systemowy zarządca wątków, który przydziela wątkowi kwant czasu procesora, po upływie którego odsuwa wątek od procesora i przydziela kolejny kwant czasu innemu wątkowi.

Java jest językiem wieloplatformowym, a różne systemy operacyjne stosują różne mechanizmy udostępniania

WĄTKI



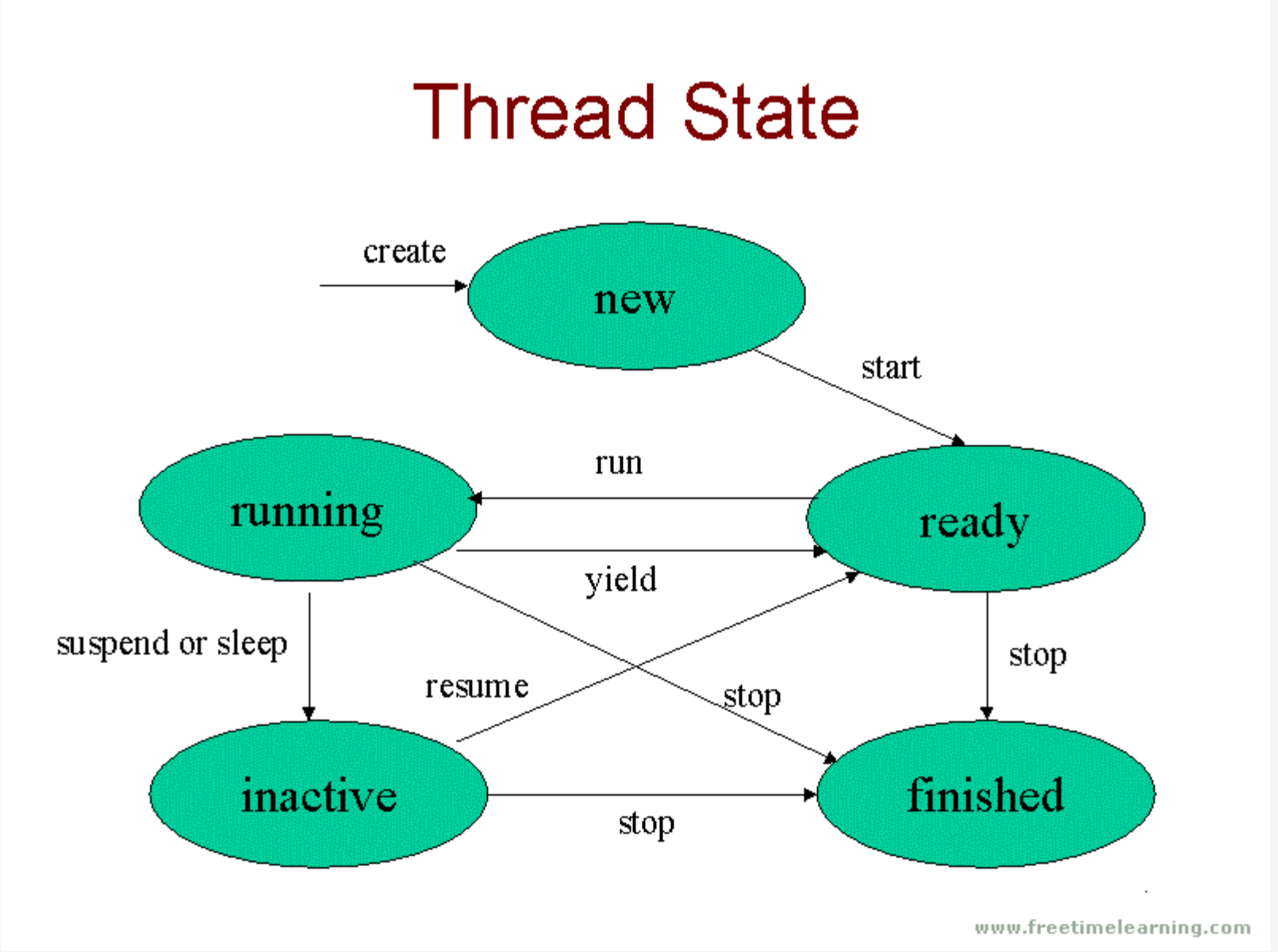
WĄTKI

• Przetwarzanie równoległe – jednoczesne wykonywanie wielu obliczeń w trakcie rozwiązywania jednego problemu. Oznacza wykonywanie kilku zadań w tym samym czasie przez odpowiednią liczbę procesów odpowiadającej ilości zadań.

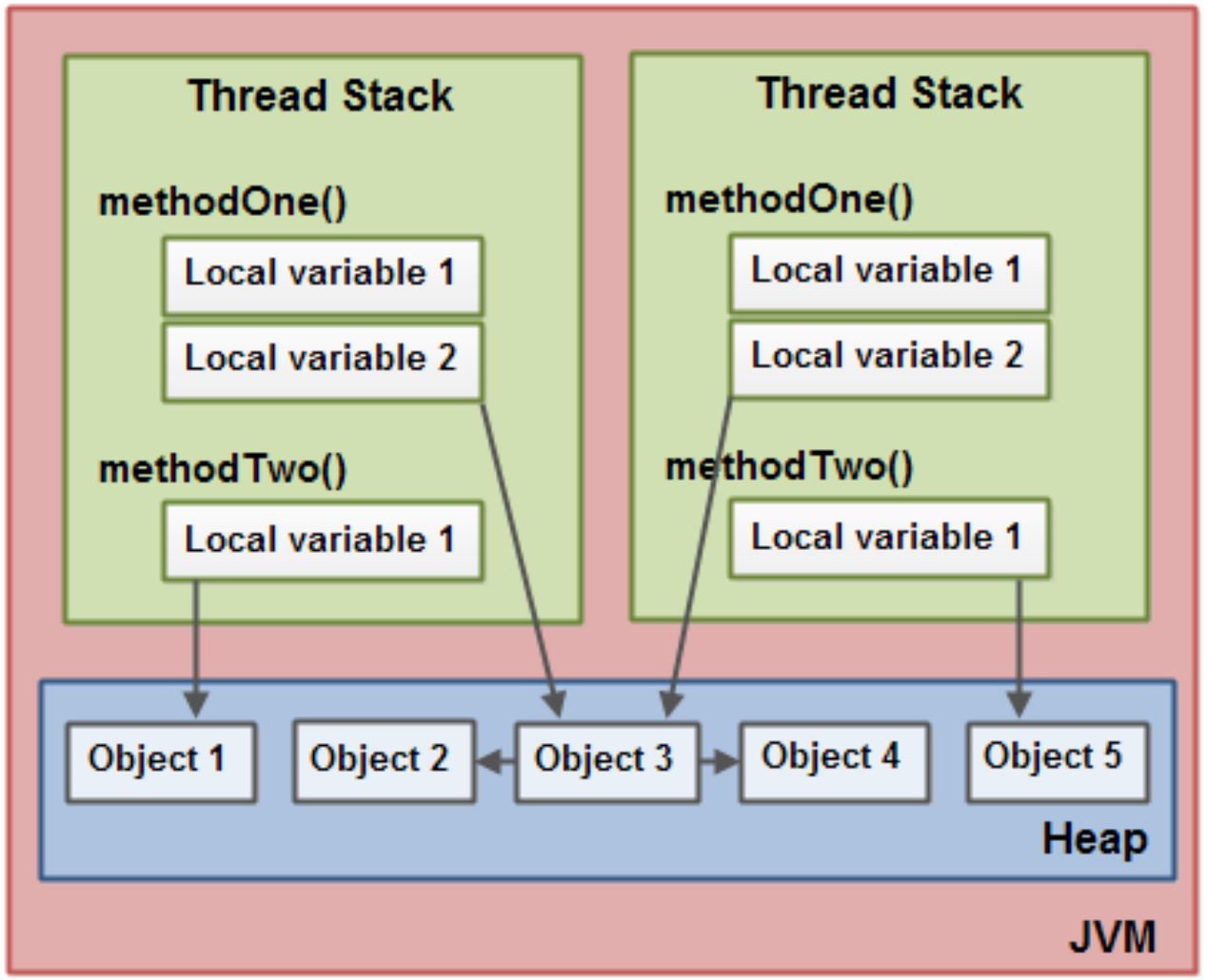
• Przetwarzanie współbieżne – jeden proces rozpoczyna się przed zakończeniem drugiego, oznacza wykonywanie kilku zadań przez procesor w tym samym czasie poprzez przeplatanie wątków (fragmentów zadań).

• Przetwarzanie rozproszone – informacja jest obrabiana jednocześnie przez wiele komputerów (procesorów), rozmieszczonych terytorialnie i połączonych ze sobą w sieć. Wykonują one osobno poszczególne etapy zadania i odsyłają wyniki do jednego wspólnego centrum nadzoru.

Stan wątku


Thread State
create
new
start
run
running
ready
yield
suspend or sleep
stop
resume
stop
inactive
finished
stop
www.freetimelearning.com

Java Memory Model



Główne metody klasy Thread

1. Uruchamianie i zatrzymywanie wątków:
   start - uruchomienie wątku,
   stop – zakończenie wątku (metoda niezalecana),
   run - kod wykonywany w ramach wątku.
2. Identyfikacja wątków:
   currentThread - metoda zwraca identyfikator wątku bieżącego,
   setName - ustawienie nazwy wątku,
   getName -odczytanie nazwy wątku,
   isAlive - sprawdzenie czy wątek działa,
   toString - uzyskanie atrybutów wątku.
3. Priorytety i szeregowanie wątków:
   getPriority - odczytanie priorytetu wątku,
   setPriority - stawienie priorytetu wątku,
   yield - wywołanie szeregowania.

Stany wątków

Wątek może znajdować się w jednym z czterech stanów:

utworzony (ang. new thread) - obiekt wątku został już utworzony ale nie wykonano metody start(), a więc wątek nie jest jeszcze szeregowany,

wykonywalny (ang. runnable) - Wątek posiada wszystkie zasoby aby być wykonywany. Będzie wykonywany gdy tylko procedura szeregująca przydzieli mu procesor,

zablokowany (ang. blocked) -Wątek nie może być wykonywany gdyż brakuje mu pewnych zasobów. Dotyczy to w szczególności operacji synchronizacyjnych (wątek zablokowany na wejściu do monitora, operacje wait, sleep, join) i operacji wejścia wyjścia,

zakończony (ang. dead) - Stan po wykonaniu metody stop(). Zalecanym sposobem kończenia wątku jest zakończeni metody run().

Programowanie fukncyjne

Składnia wyrażenia lambda to
() -> {}
Gdzie () to lista parametrów
{} to blok kodu zwracający wartość (może być to wyrażenie lub blok kodu z return)

Strumienie

Wyrażenia lambda mają swoje zastosowanie w tak zwanych strumieniach

Strumień pozwala na opakowanie dowolnego zestawu danych i pracę na nich w sposób:

- Skoncentrowany na danych, nie na algorytmie
- Bardziej czytelny
- Szybszy w działaniu

```
for(Samochod a : auta){
    if(a.mocKM>100){
        if(a.przebieg < 200000){
            if(a.cena<30000){
                System.out.println(a.nazwa.toUpperCase());
            }
        }
    }
}
```

Strumienie

```
auta.stream()
.filter(a -> a.mocKM >100)
.filter(a -> a.przebieg <200000)
.filter(a -> a.cena < 30000)
.map(a -> a.nazwa.toUpperCase())
.forEach(System.out::println);
```