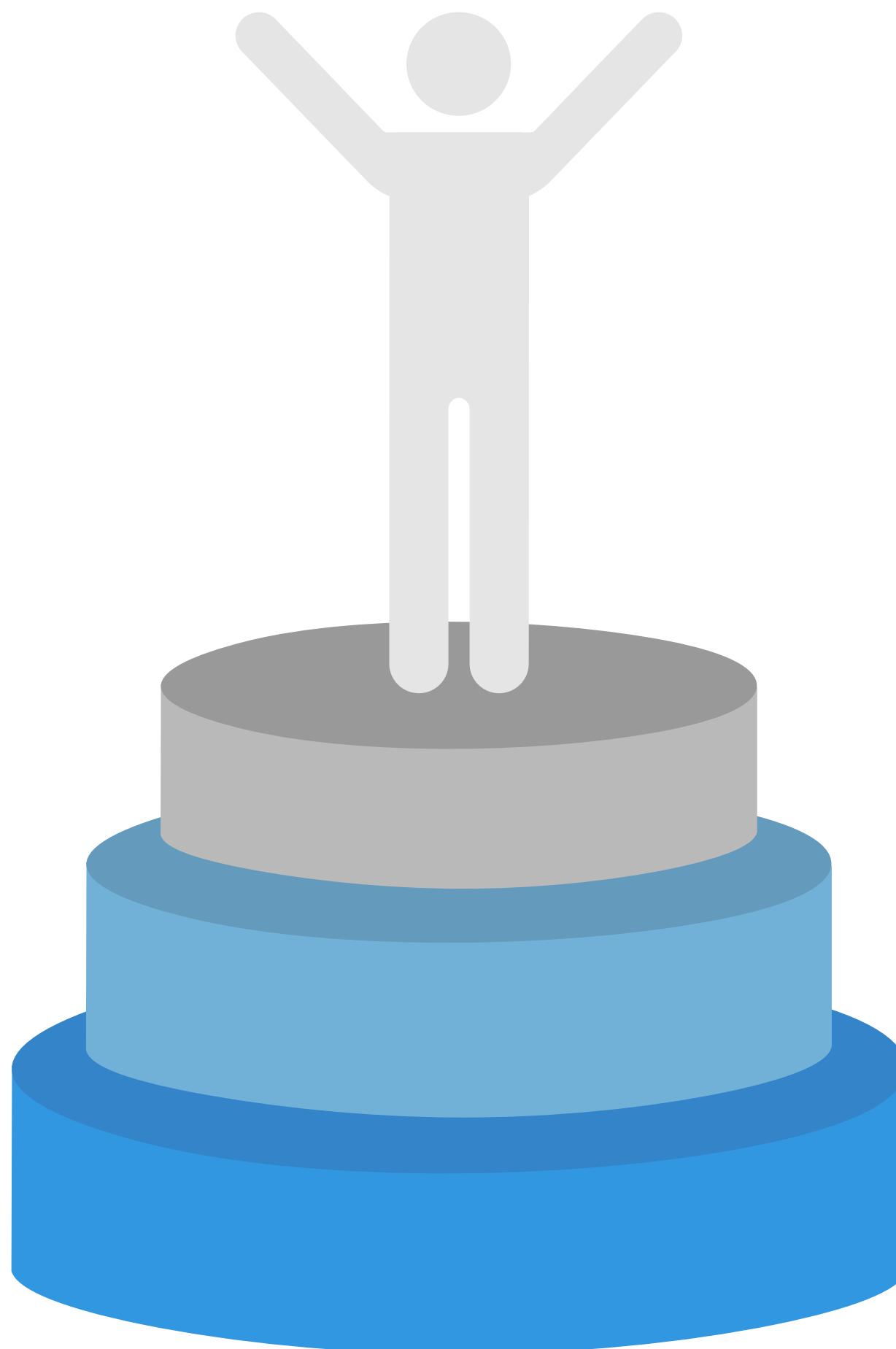


POPULARNOŚĆ JĘZYKÓW PROGRAMOWANIA





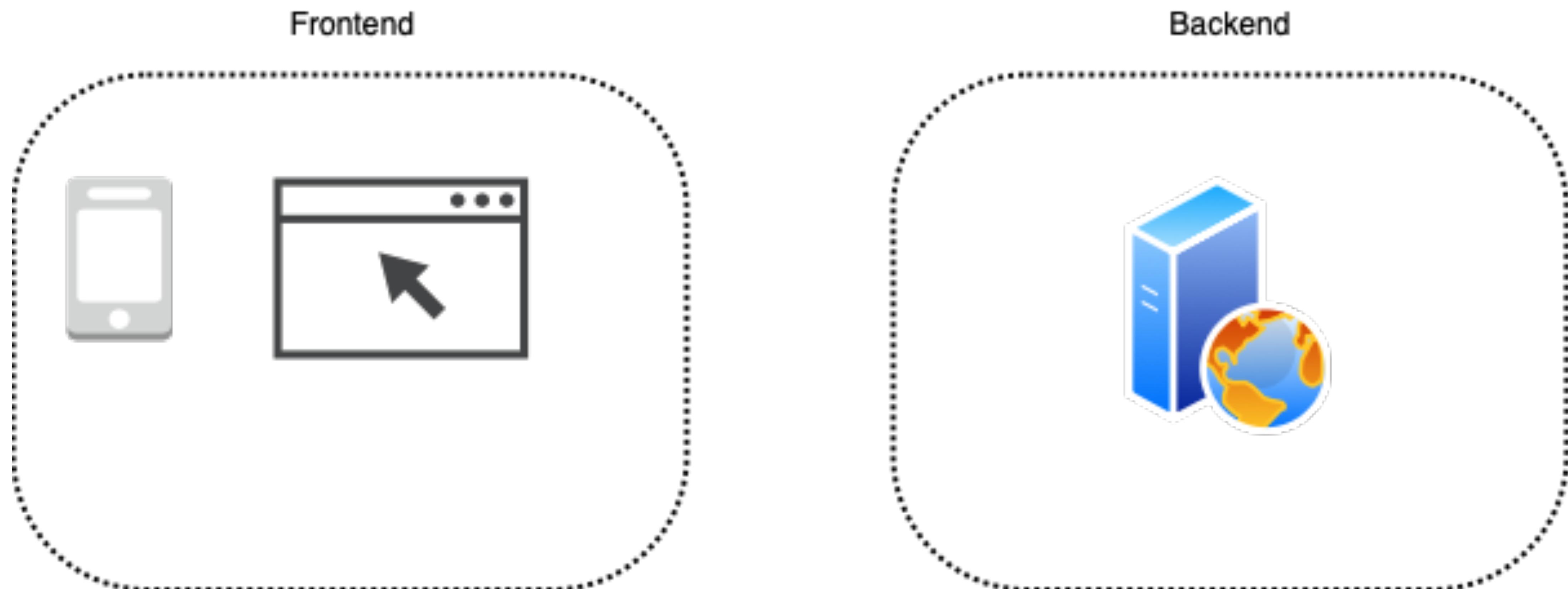
Prosty i przyjemny



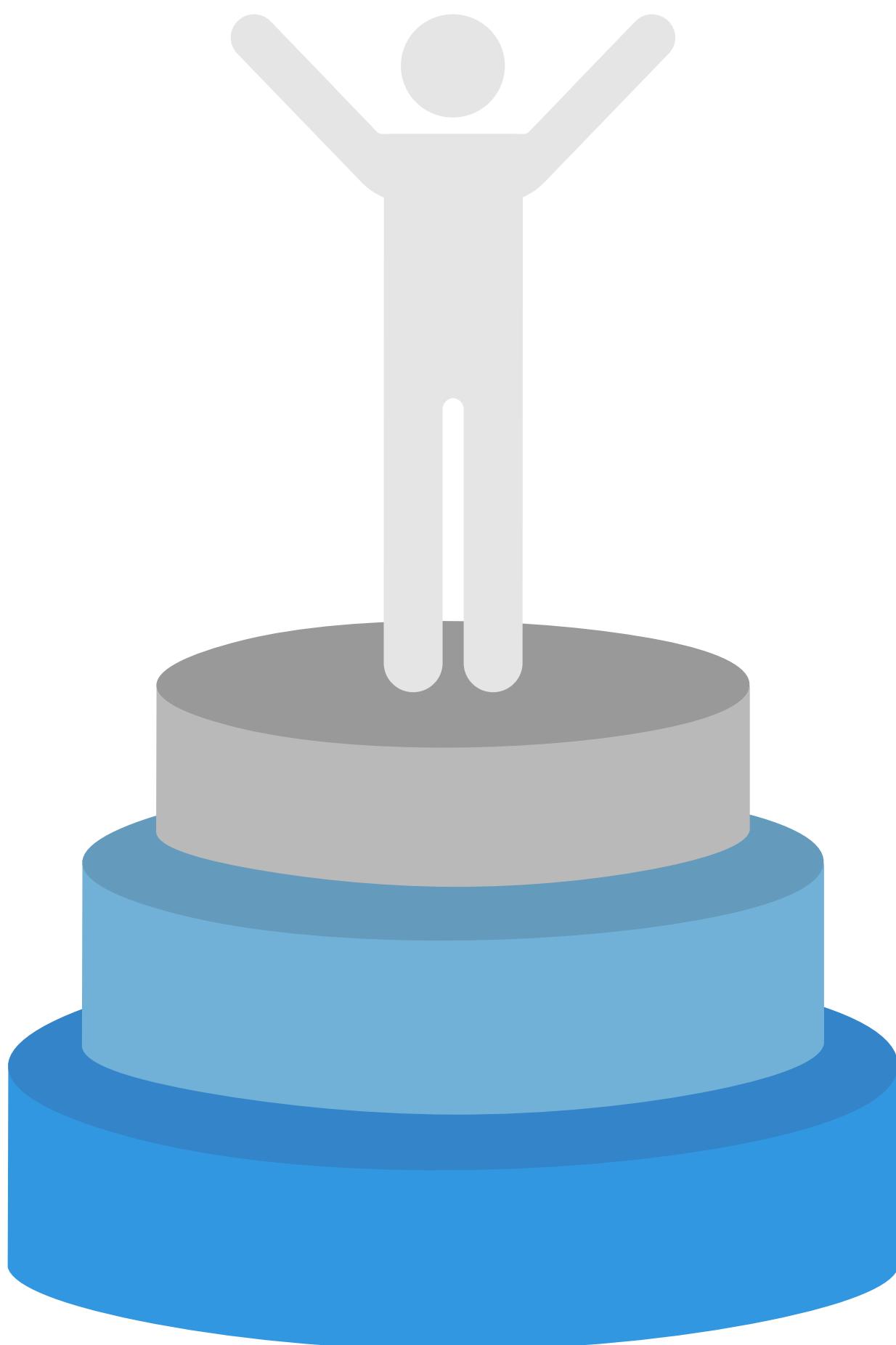
Popularność



Zarobki



ILE ZARABIA PROGRAMISTA JAVA?



Prosty i przyjemny



Popularność



Zarobki

ILE ZARABIA PROGRAMISTA JAVA?

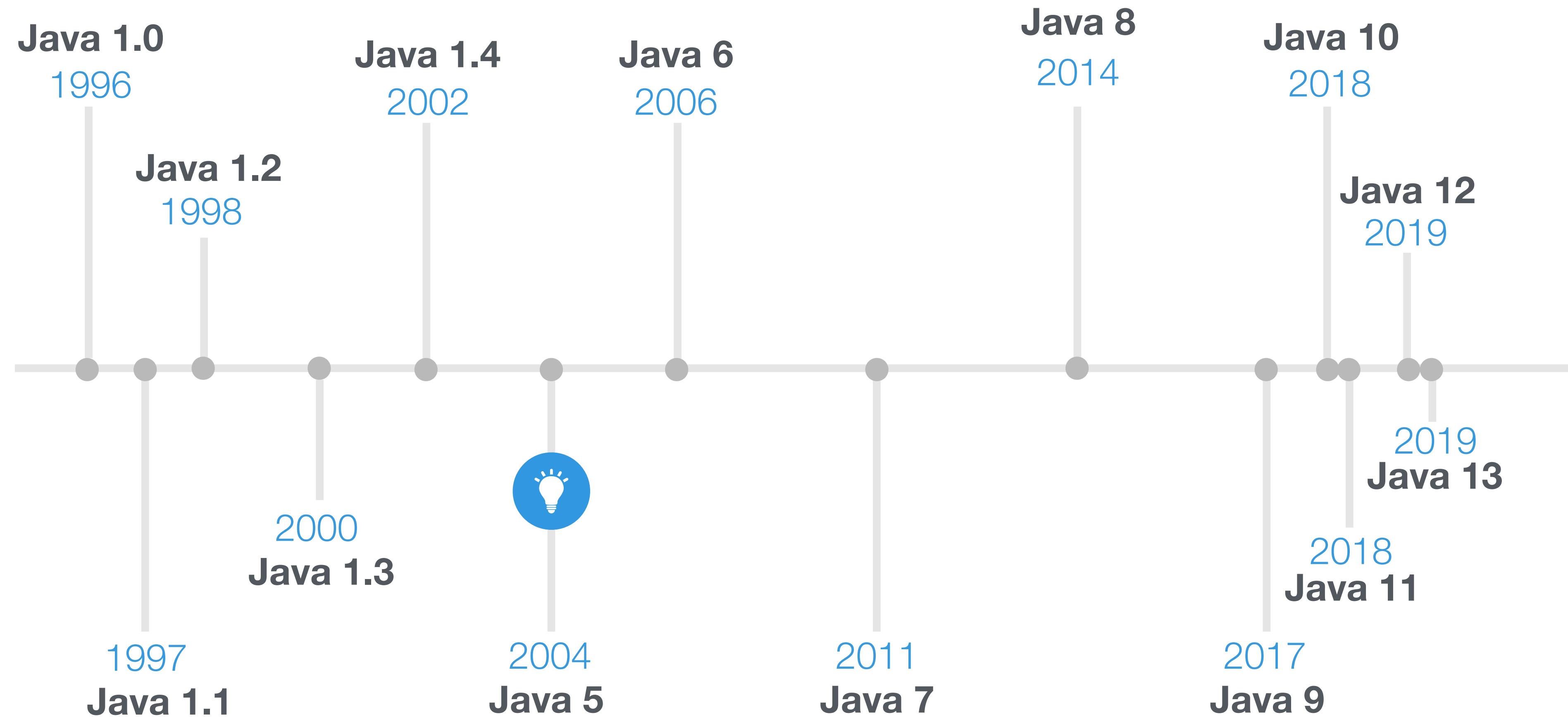
**Badanie
społeczności IT
2020**

Pod patronatem

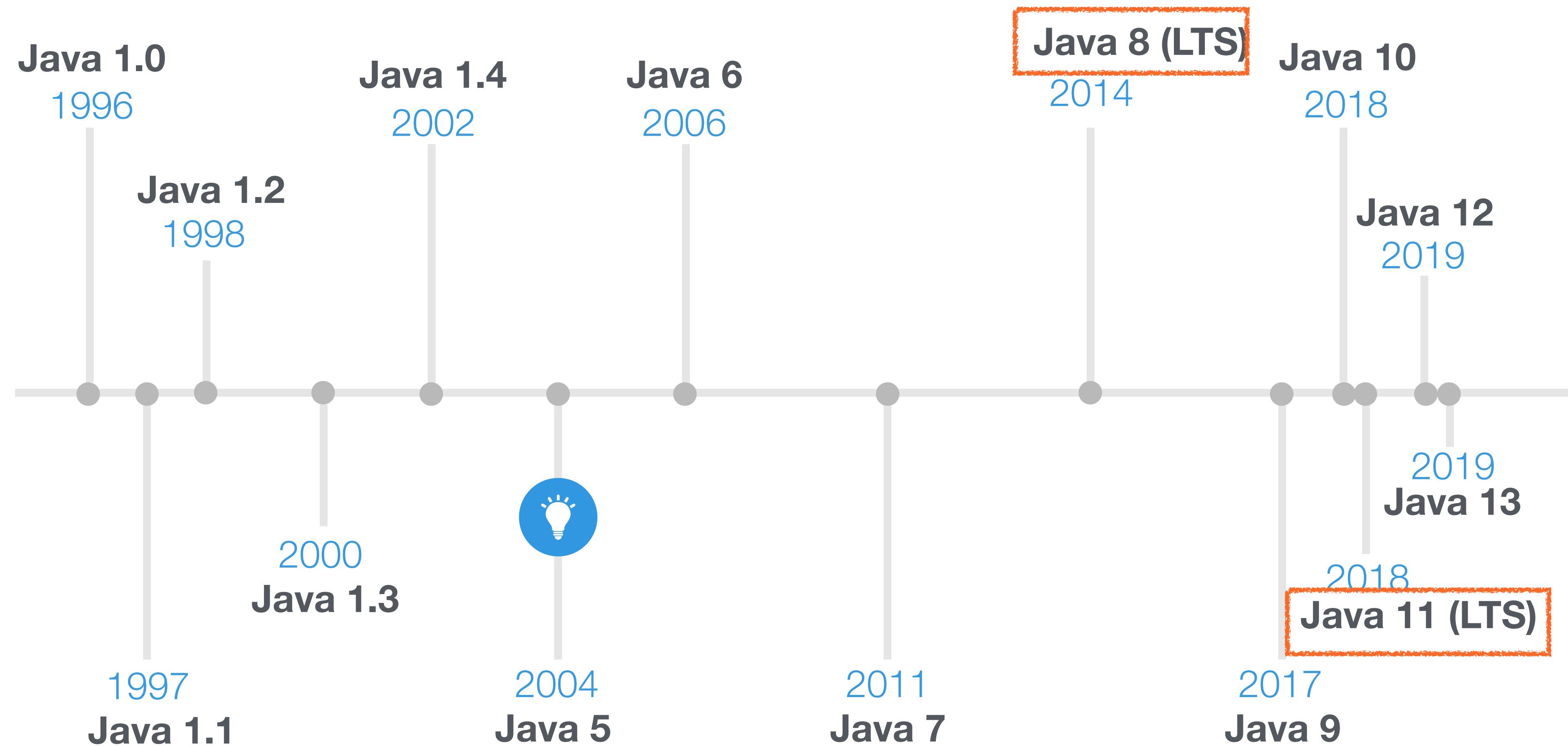


UCZ SIĘ PROGRAMOWAC!!!!

KROTKA HISTORIA JAVA



KROTKA HISTORIA JAVA



SKŁADOWE PLATFORMY JAVA

JAVA SE

- JDK
- API

JAVA EE

- JDK
- API

JAVA ME

- JDK
- API

JAVA FX

- JDK
- API

Java Standard Edition

Interfejs API Java SE zapewnia podstawową funkcjonalność języka programowania Java. Definiuje wszystko, od podstawowych typów i obiektów języka programowania Java do klas wysokiego poziomu używanych do pracy w sieci, bezpieczeństwa, dostępu do bazy danych, programowania graficznego interfejsu użytkownika (GUI) i analizy składni XML.

SKŁADOWE PLATFORMY JAVA

JAVA SE

- JDK
- API

JAVA EE

- JDK
- API

JAVA ME

- JDK
- API

JAVA FX

- JDK
- API

Java Enterprise Edition

Java EE, jest zbudowana na platformie Java SE i służy do tworzenia aplikacji internetowych, które działają na serwerze aplikacji i serwerach WWW. Platforma Java EE zapewnia różne interfejsy API i środowisko wykonawcze do tworzenia skalowalnych, niezawodnych, wielowymiarowych i bezpiecznych aplikacji sieciowych na dużą skalę. Java Servlet, JSP, JSF, Primefaces, EJB, JPA, Oracle ADF to technologie Java EE.

SKŁADOWE PLATFORMY JAVA

Java SE

- JDK
- API

Java EE

- JDK
- API

Java ME

- JDK
- API

Java FX

- JDK
- API

Java Micro Edition

Jest to platforma Java dla aplikacji działających na niewielkich urządzeniach, takich jak telefony komórkowe. Java ME służy do opracowywania różnych aplikacji i gier na telefony komórkowe. Aplikacje opracowane za pomocą Java ME mogą działać na różnych systemach operacyjnych telefonów komórkowych, takich jak Nokia Symbian, Android, Iphone i Windows Phone itp. Java ME API to podzbiór Java SE API, a także specjalne biblioteki klas przydatne w aplikacjach na małe urządzenia rozwój. Aplikacje Java ME są często klientami usług platformy Java EE.

SKŁADOWE PLATFORMY JAVA

Java SE

- JDK
- API

Java EE

- JDK
- API

Java ME

- JDK
- API

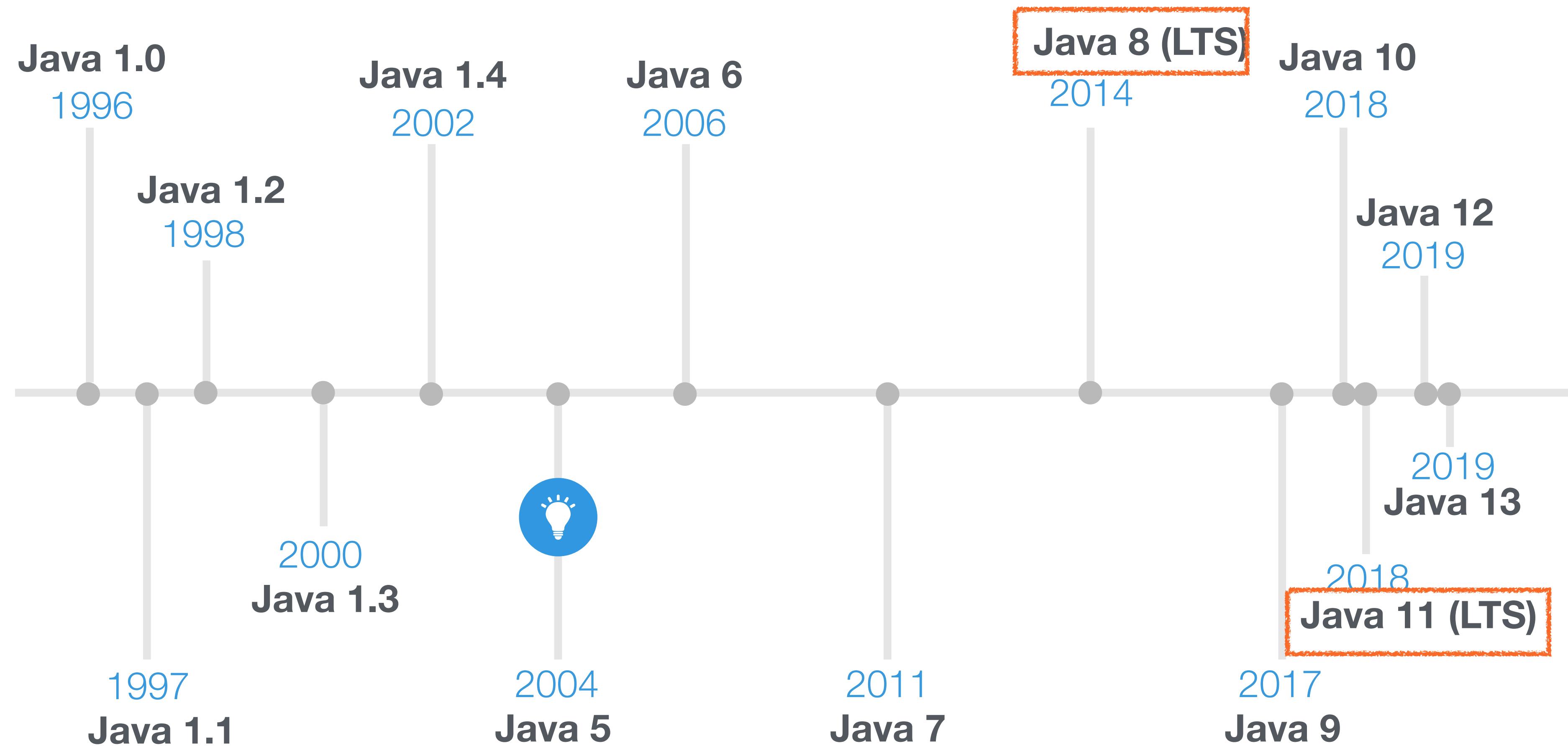
Java FX

- JDK
- API

Java FX

Podobnie jak Java SE i Java EE, JavaFX jest potężną platformą UI (interfejsu użytkownika) opartą na Javie do tworzenia dużych aplikacji biznesowych opartych na danych i bogatych aplikacji internetowych przy użyciu lekkiego interfejsu API interfejsu użytkownika. Aplikacje zbudowane w JavaFX wykorzystują akcelerację sprzętową grafiki i silniki multimedialne dla klientów o wysokiej wydajności i nowoczesnych bogatych interfejsów. Aplikacja JavaFX może także służyć jako klient usług Java EE.

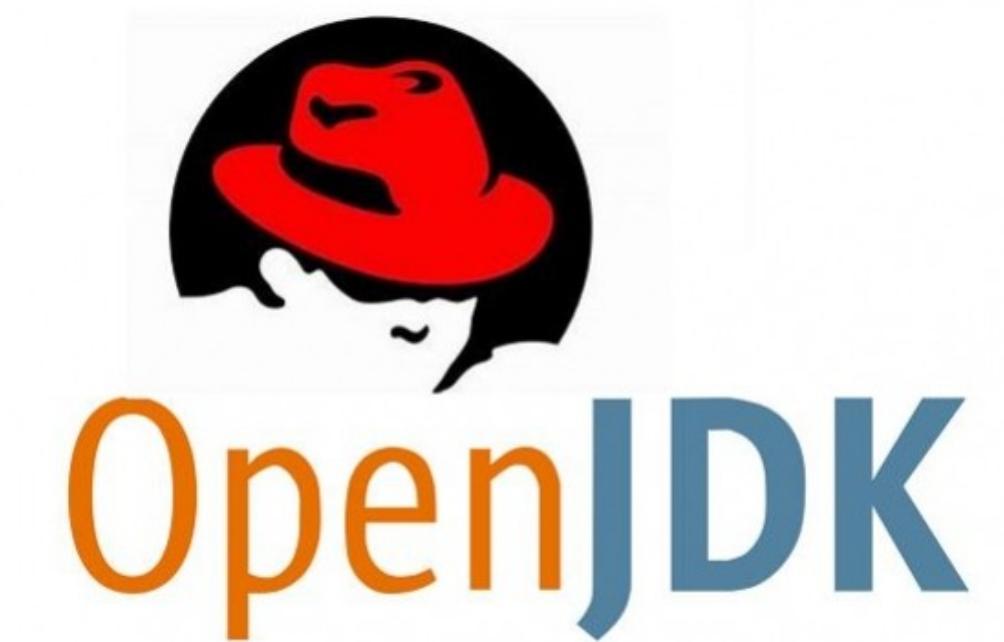
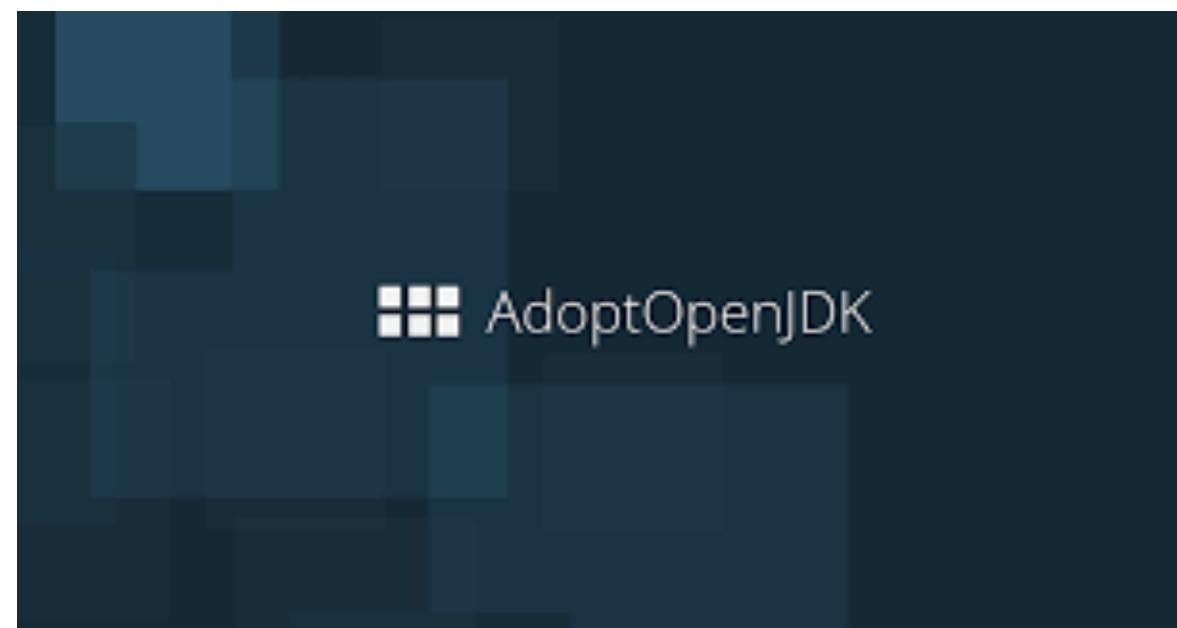
KROTKA HISTORIA JAVA



IMPLEMENTACJE JAVA



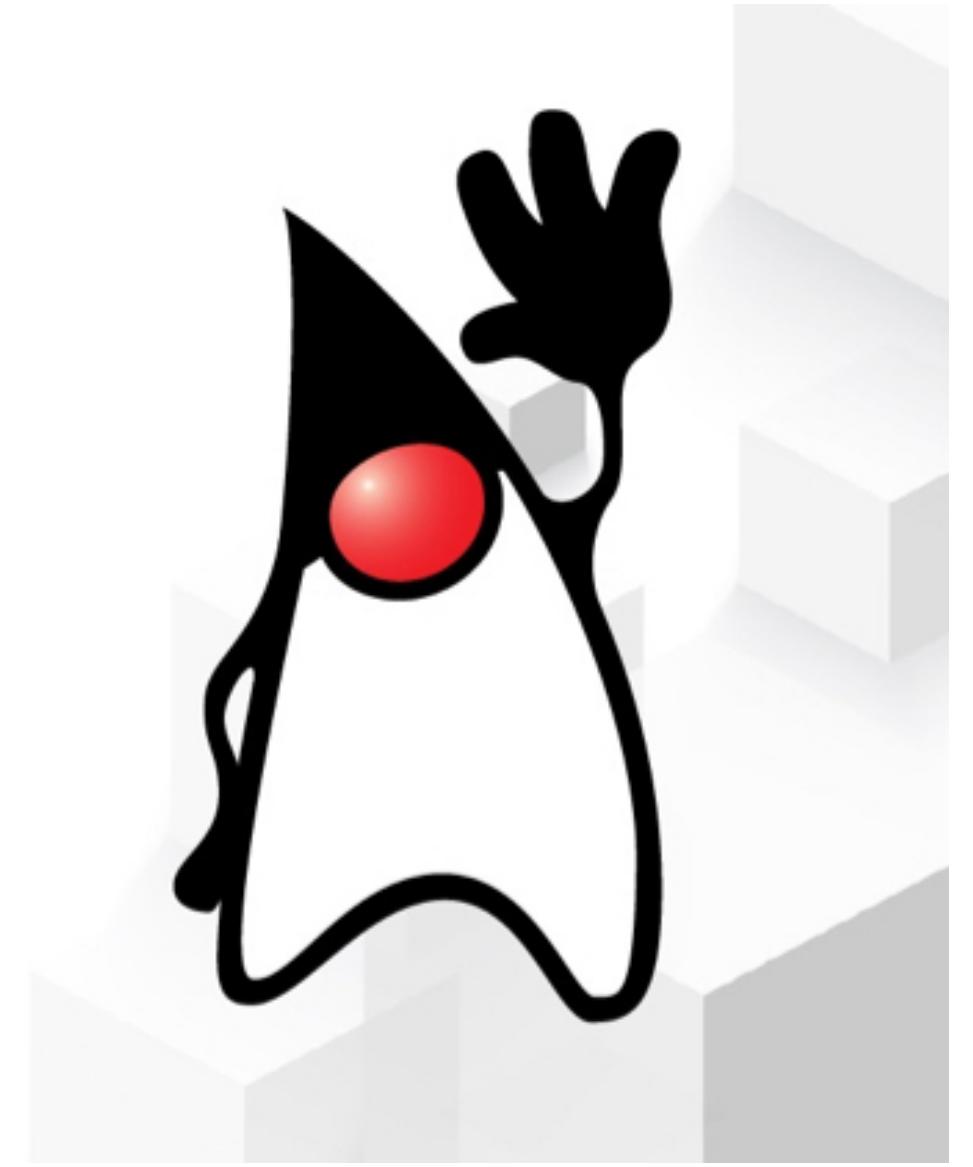
OpenJDK



AMAZON CORRETTO

Poprawki i ulepszenia w Corretto umożliwiają firmie Amazon rozwiązywanie poważnych problemów związanych z usługami, spełniając wysokie wymagania dotyczące wydajności i skalowalności. Udostępniamy je klientom bezobsługowym, długoterminowym wsparciem, z kwartalnymi aktualizacjami, w tym poprawkami błędów i poprawkami bezpieczeństwa. AWS zapewni również pilne poprawki klientom poza harmonogramem kwartalnym.

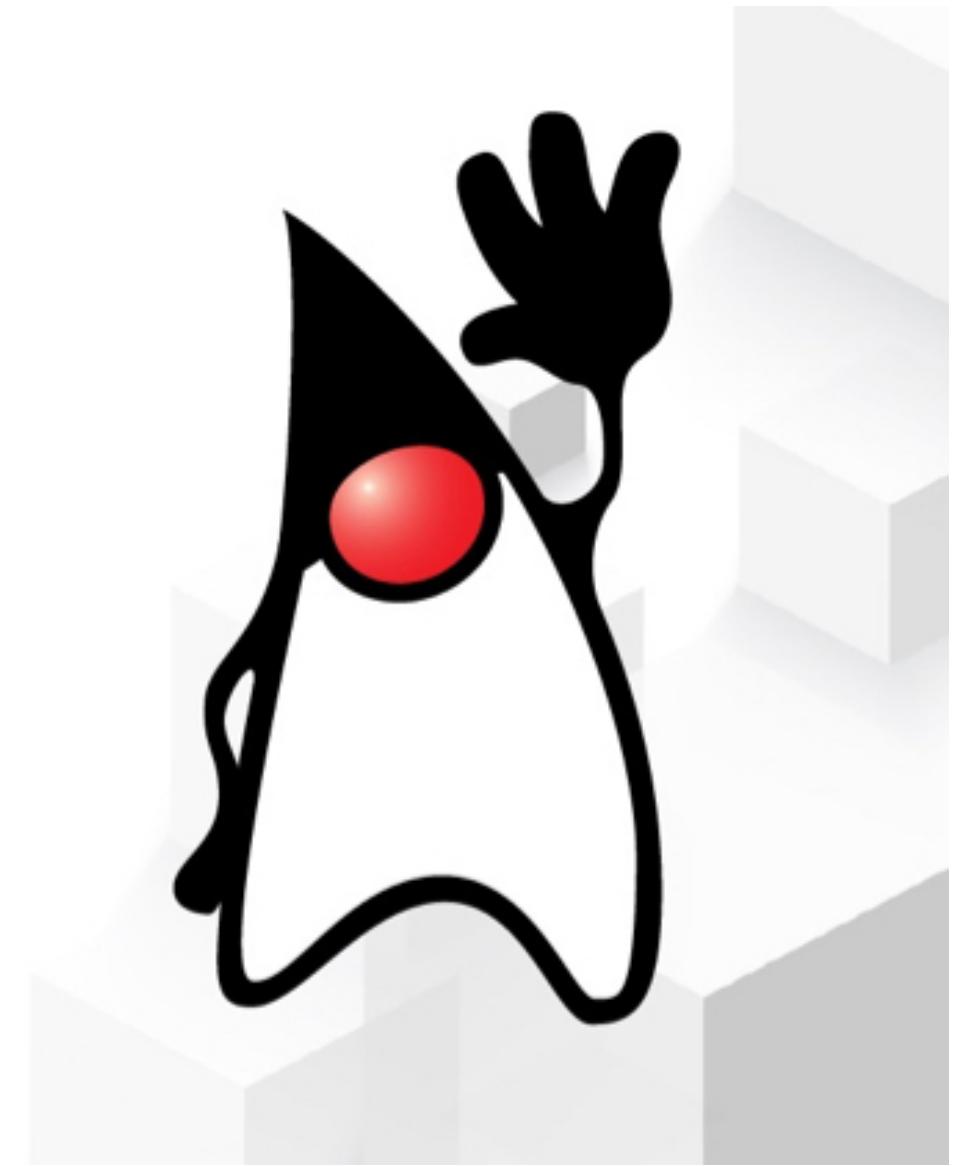
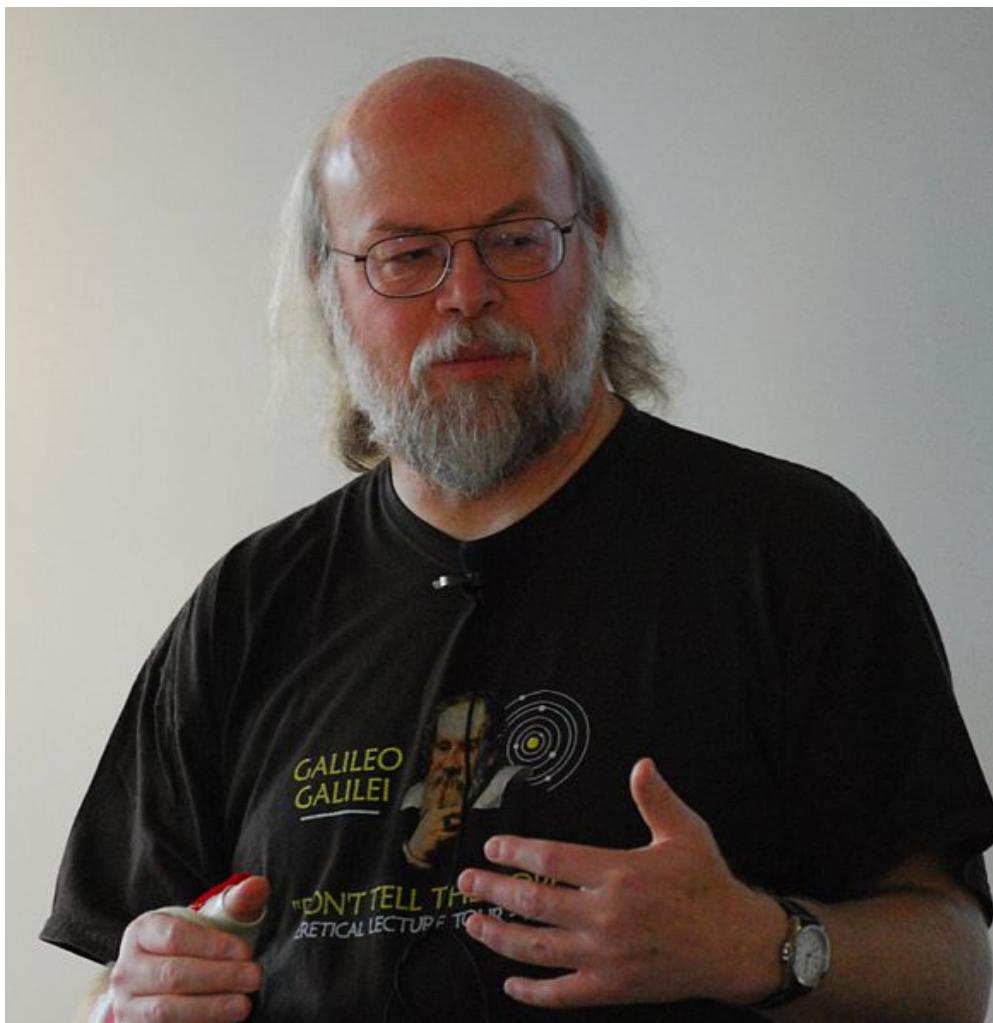
Introducing Amazon Corretto



AMAZON CORRETTO

Poprawki i ulepszenia w Corretto umożliwiają firmie Amazon rozwiązywanie poważnych problemów związanych z usługami, spełniając wysokie wymagania dotyczące wydajności i skalowalności. Udostępniamy je klientom bezobsługowym, długoterminowym wsparciem, z kwartalnymi aktualizacjami, w tym poprawkami błędów i poprawkami bezpieczeństwa. AWS zapewni również pilne poprawki klientom poza harmonogramem kwartalnym.

Introducing Amazon Corretto



WSKAZÓWKI

1. Skróty klawiszowe

WSKAZÓWKI

1. Skróty klawiszowe
2. Notatki

WSKAZÓWKI

1. Skróty klawiszowe
2. Notatki
3. Projekt: Wniosek kredytowy

WSKAZÓWKI

1. Skróty klawiszowe
2. Notatki
3. Projekt: Wniosek kredytowy
4. Język angielski

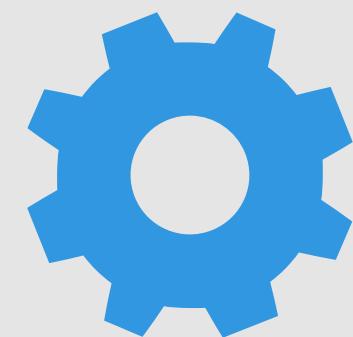
WSKAZÓWKI

1. Skróty klawiszowe
2. Notatki
3. Projekt: Wniosek kredytowy
4. Język angielski

pawel.wasowski@javaskills.pl

RZECZY DO ZAPAMIĘTANIA

Program Java



Każdy program Java składa się z kolekcji klas.

W każdym pliku źródłowym musi być zadeklarowana klasa o dokładnie takiej samej nazwie co plik źródłowy.

Conajmniej jednak klasa musi mieć metodę main.

PIERWSZE KROKI - KLASY I OBIEKTY



PIERWSZE KROKI - KLASY I OBIEKTY



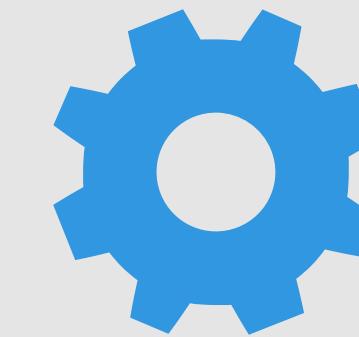
KLASA



OBIEKT (INSTANCJA KLASY)

RZECZY DO ZAPAMIĘTANIA

Klasa

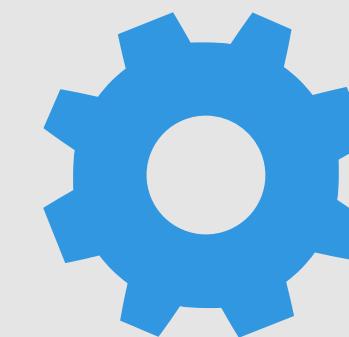


Klasa jest definicją, za pomocą której będziemy mogli tworzyć obiekty, czyli konkretne instancje danej klasy.

Klasa składa się z metod i zmiennych (narazie).

RZECZY DO ZAPAMIĘTANIA

Klasa



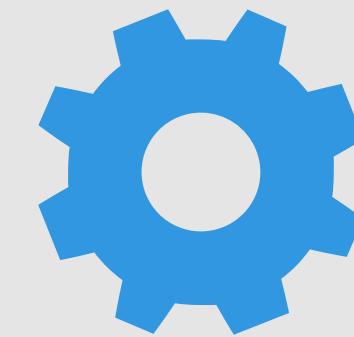
Każda klasa ma domyślny bezargumentowy konstruktor. Nie trzeba go jawnie tworzyć, kompilator generuje go automatycznie.

Każda klasa dziedziczy po klasie Object.

Klasa może mieć 2 typy elementów: static i instance.

RZECZY DO ZAPAMIĘTANIA

Obiekt

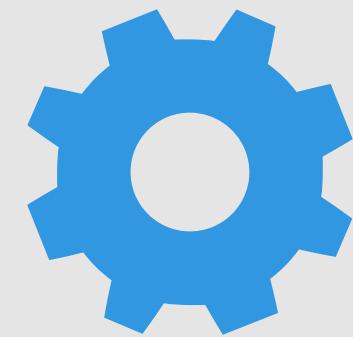


Do tworzenia obiektów, używamy operatora NEW.

Metoda `toString` zwraca słowo, które ma słownie opisywać ten obiekt. Metoda jest dziedziczona z klasy `Object`.

RZECZY DO ZAPAMIĘTANIA

Konstruktor



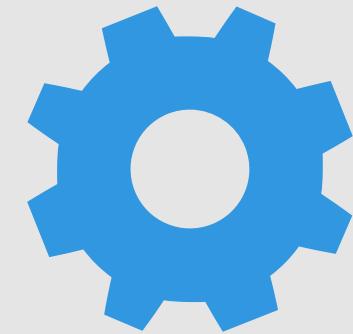
Każda klasa ma domyślnie konstruktor, nie trzeba go jawnie deklarować.

Konstruktor niestandardowy to konstruktor, który przyjmuje jakieś argumenty.

Po zdefiniowaniu konstruktora niestandardowego, konstruktor domyślny nie zostanie wygenerowany.

RZECZY DO ZAPAMIĘTANIA

Pola final

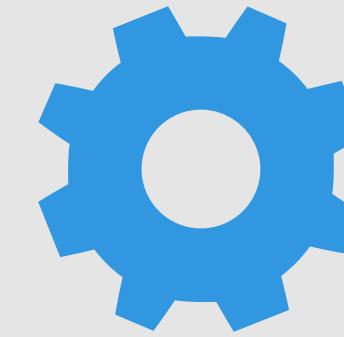


Słowo kluczowe final wymusza, że zmienna może być zainicjalizowana tylko raz.

Zmienne oznaczone jako final muszą być zainicjalizowane przed utworzeniem obiektu.

RZECZY DO ZAPAMIĘTANIA

Słowo kluczowe this

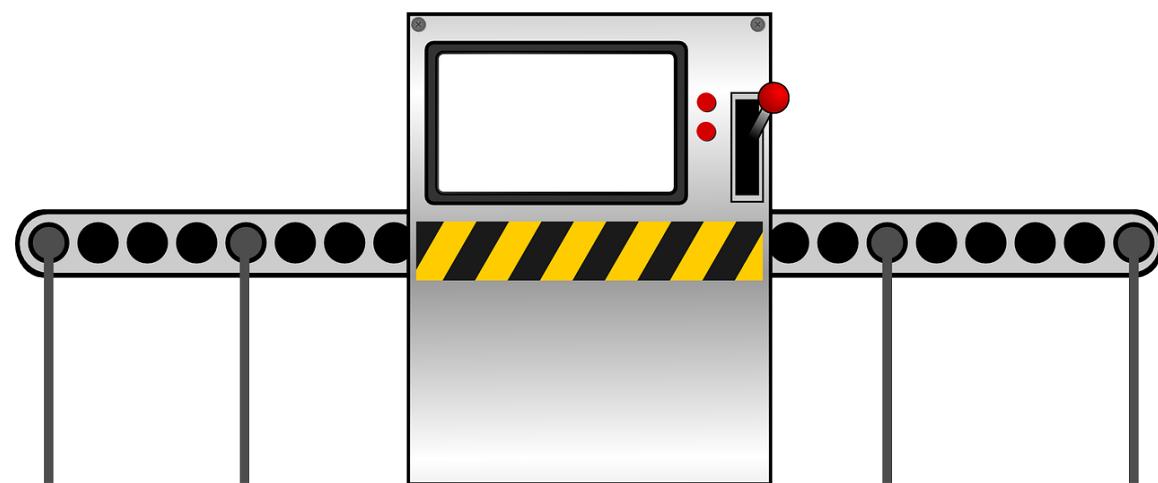


Służy do odwołania się do elementów klasy, w której zostało użyte.

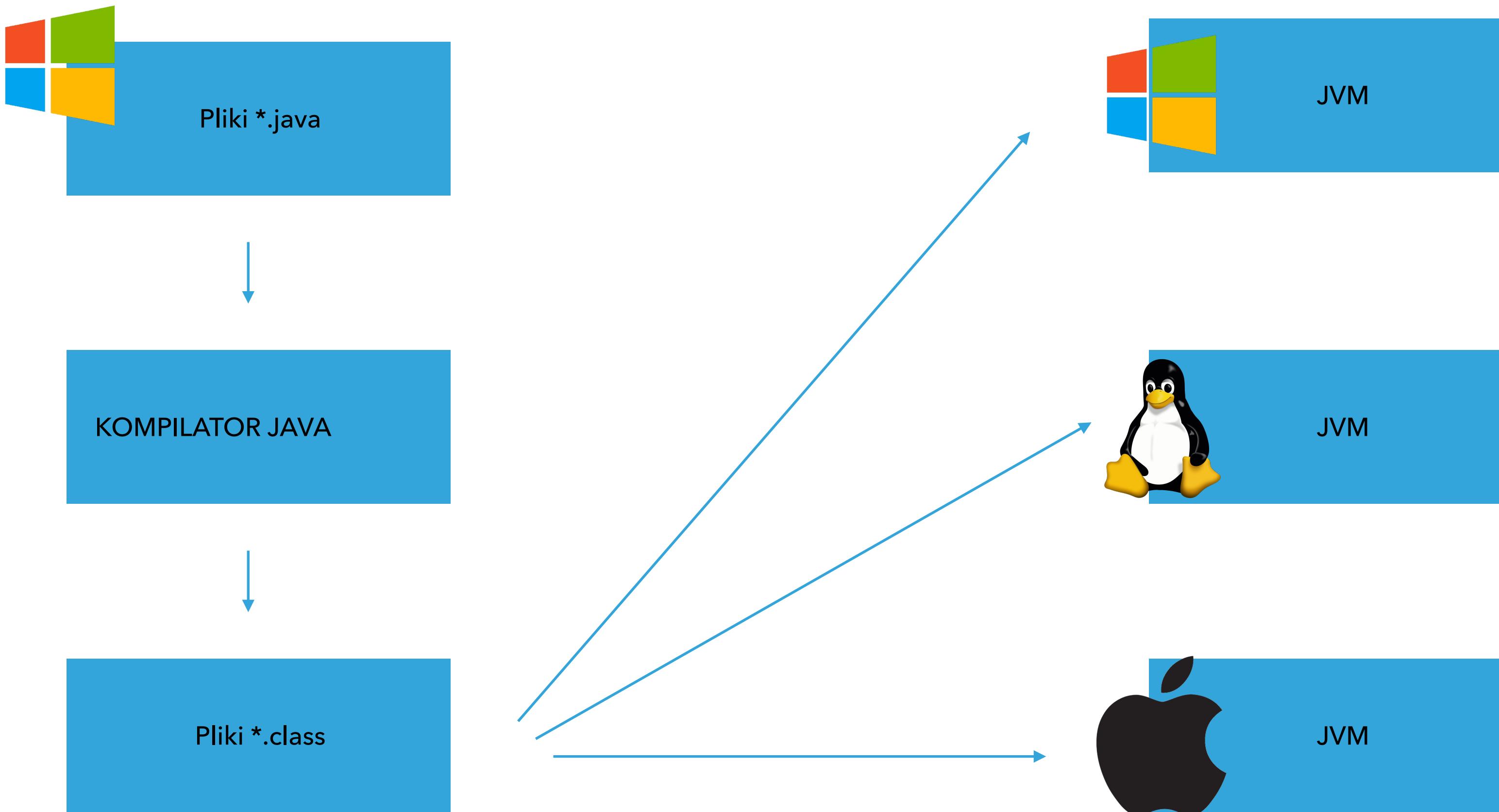
JAK DZIAŁA PROGRAM JAVA?



JAK DZIAŁA PROGRAM JAVA?

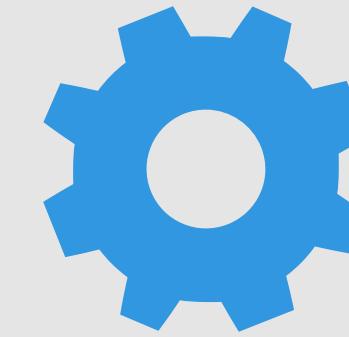


CO SIĘ DZIAŁA PODCZAS URUCHAMIANIA PROGRAMU?



RZECZY DO ZAPAMIĘTANIA

Kod źródłowy

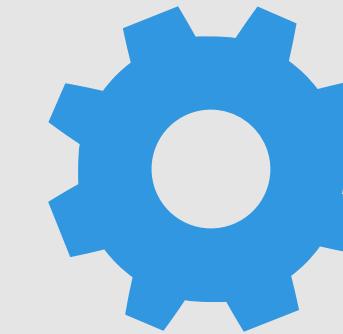


Kod źródłowy znajduje się w plikach o rozszerzeniu **java**.

Prze uruchomieniem kod źródłowy musi zostać przekonwertowany do języka pośredniego - **bytecode**.

RZECZY DO ZAPAMIĘTANIA

Kompilacja

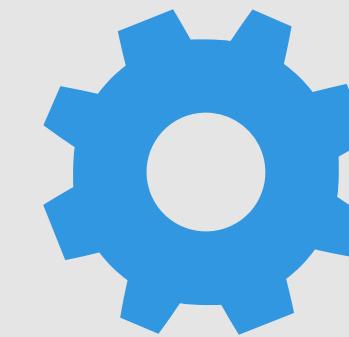


Proces konwersji kodu do języka pośredni, nazywa się komplikacją i wykonuje go Kompilator Java.

Wynikiem komplikacji są pliki o rozszerzeniu **.class**, w których znajduje się **bytecode**.

RZECZY DO ZAPAMIĘTANIA

Wirtualna Maszyna Java (JVM)

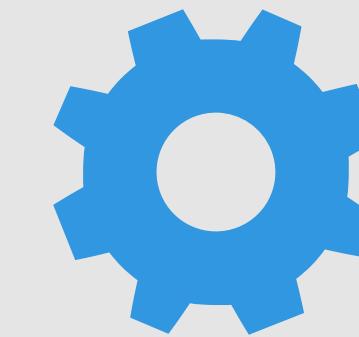


Odpowiada za wykonanie **bytecode**, na konkretnym systemie operacyjnym.

Dzięki **bytecode** oraz **JVM**, java jest językiem niezależnym od platformy.

RZECZY DO ZAPAMIĘTANIA

PAKIETY



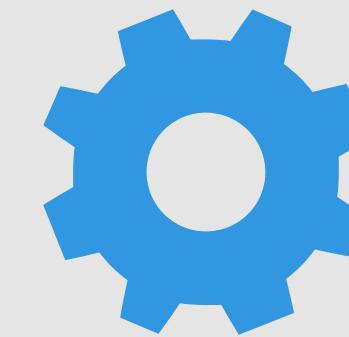
Pakiety służą do grupowania klas.

Wszystkie klasy w głównym folderze źródłowym mają **Pakiet Domyślny**.

Pakiet `java.lang` jest automatycznie importowany w każdej klasie.

RZECZY DO ZAPAMIĘTANIA

Pakiety



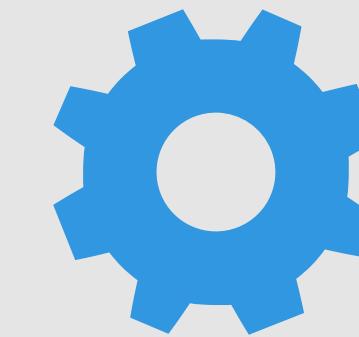
Klasy z tego samego **Pakietu** są domyślnie zimportowane.

Do deklaracji **pakietu** służy słowo kluczowe **PACKAGE**.

Nazwy **Pakietów** odzwierciedlają fizyczną organizację plików w podfoldery na dysku.

RZECZY DO ZAPAMIĘTANIA

Importowanie klas



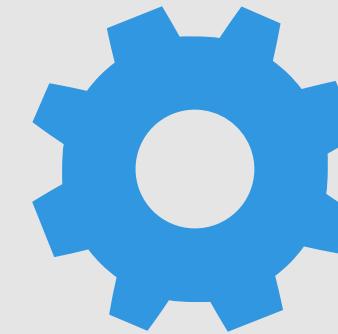
Klasy z innych **pakietów**, muszą zostać jawnie zimportowane.

Do deklaracji importu służy słowo kluczowe **IMPORT**.

Pakiet i nazwa klasy muszą jednoznacznie identyfikować klasę w projekcie.

RZECZY DO ZAPAMIĘTANIA

Konwencje nazewnicze



Do nazywania klas, zmiennych i metod używa się notacji **camelCase**.

Nazwy wszystkich zmiennych i metod zaczynamy z małej litery i każda kolejna litera nowego słowa pisana jest dużą literą.

Nazwy klas zaczynamy od garba, czyli pierwsza litera zawsze jest duża.

RZECZY DO ZAPAMIĘTANIA

Konwencje nazewnicze



Do nazywania klas używa się rzeczowników.

Nazwa metody powinna zawierać w sobie czasownik i wyrażać jakąś czynność.

RZECZY DO ZAPAMIĘTANIA

Komentarze



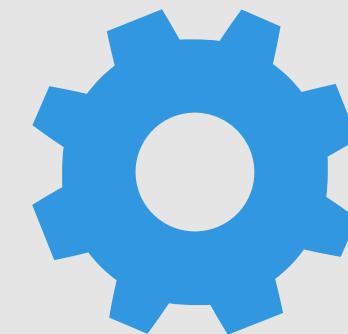
Są trzy rodzaje komentarzy w JAVA:

1. Dokumentujący `/** */`
2. Jednolinijkowy `//`
3. Wielolinijkowy `/* */`

Kod powinien być samo komentującym się.

RZECZY DO ZAPAMIĘTANIA

Zmienne



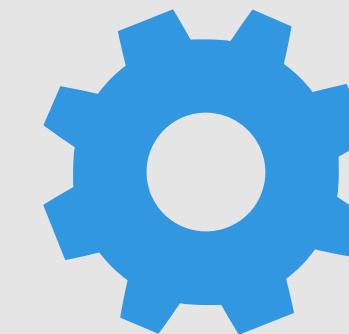
Zmienne na czas działania programu zapisane są w pamięci RAM.

Każdy typ prymitywny ma zdefiniowany zakres bitowy, na którym zapisane są zmienne tego typu.

Nazwa zmiennej jest etykietą, która wskazuje na adres w pamięci RAM, pod którym zapisana jest wartość tej zmiennej.

RZECZY DO ZAPAMIĘTANIA

Deklaracja i inicjalizacja



Java jest językiem silnie typizowanym - każda zmienna musi mieć typ.

Zmienne można deklarować i inicjalizować w jednej linijce, albo w osobnych.

Składnia pozwala na deklarację i inicjalizację wielu zmiennych w jednej linijce.

RZECZY DO ZAPAMIĘTANIA

Deklaracja i inicjalizacja

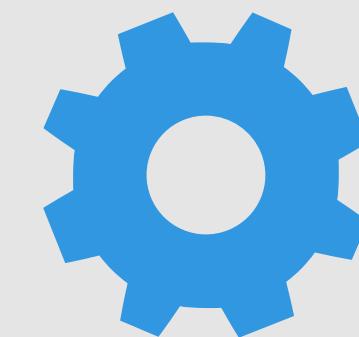


Każda zmienna w metodzie musi zostać jawnie zainicjalizowana przed użyciem.

Słowo kluczowe VAR służy do deklaracji zmiennej bez typu, jednak jest to tylko likier składniowy. Kompilator ustala typ z kontekstu. W skompilowanym kodzie zmienne var mają jawnie zadeklarowany typ.

RZECZY DO ZAPAMIĘTANIA

Deklaracja i inicjalizacja

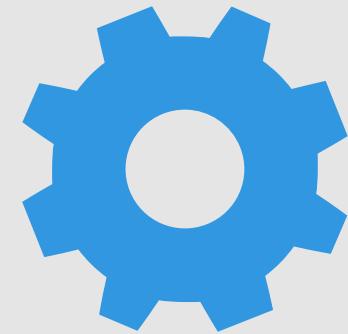


Wszystkie zmienne zadeklarowane w ciele metody są widoczne tylko w tej metodzie.

Zmienne zadeklarowane w metodzie nie przyjmują wartości domyślnych, trzeba je jawnie zainicjalizować.

RZECZY DO ZAPAMIĘTANIA

Typy prymitywne



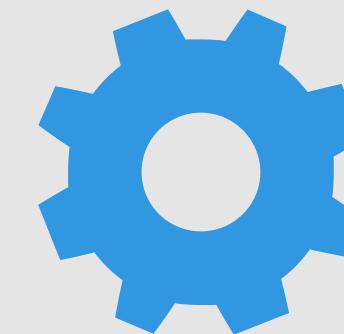
Typy prymitywne to podstawowe typy danych, które służą do zapisu wartości w pamięci RAM.

Typy prymitywne nie są obiektami.

Za pomocą typów prymitywnych można zapisać, w pamięci RAM, wartość każdego obiektu.

RZECZY DO ZAPAMIĘTANIA

Zakres typów prymitywnych



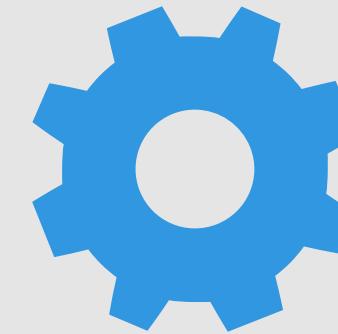
Wzór na wyliczenie zakresu typu całkowitego:

od -2^{n-1} do $2^{n-1} - 1$, gdzie n ilość bitów, na których zapisany jest ten typ.

Typ	Pamięć	Zakres
byte	1 bajt	od -128 do 127
short	2 bajty	od -32,768 do 32,767
int	4 bajty	od -2,147,483,648 do 2,147,483,647 (niewiele ponad 2 miliardy)
long	8 bajtów	od -9,223,372,036,854,775,808 do 9,223,372,036,854,775,807

RZECZY DO ZAPAMIĘTANIA

Sposoby zapisu wartości typów całkowitych



- notacja dziesiętna: int x = 10;
- notacja binarna: int x = 0B1010;
- notacja ósemkowa: int x = 023;
- notacja szesnastkowa: int x = 0XA;
- notacja z podkreśleniem: int x = 1_000_000_000;

RZECZY DO ZAPAMIĘTANIA

Operatory arytmetyczne

Operator	Nazwa	Przykład
+	Dodawanie	$x = 3 + 3$
-	Odejmowanie	$x = 3 - 3$
*	Mnożenie	$x = 3 * 3$
/	Dzielenie	$x = 3 / 3$
%	Modulo	$x = 7 \% 2$
++	Inkrementacja	$x++$
--	Dekrementacja	$x-$

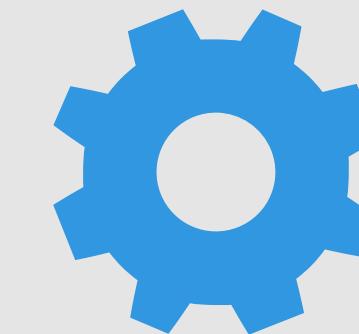
RZECZY DO ZAPAMIĘTANIA

Operatory przypisania

Operator	Przykład	Alternatywa
=	$x = 5$	$x = 5$
+=	$x += 5$	$x = x + 5$
-=	$x -= 5$	$x = x - 5$
*=	$x *= 5$	$x = x * 5$
/=	$x /= 5$	$x = x / 5$
%=	$x %= 5$	$x = x \% 5$

RZECZY DO ZAPAMIĘTANIA

Typy prymitywne zmiennoprzecinkowe

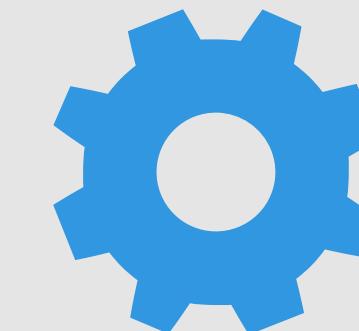


Istnieją 2 typy zmiennoprzecinkowe w java: **float** i **double**.

Typ	Pamięć	Zakres
float	4 bajty	W przybliżeniu $\pm 3.40282347E+38F$ (6–7 cyfr po przecinku)
double	8 bajtów	W przybliżeniu $\pm 1.79769313486231570E+308$ (15 cyfr po przecinku)

RZECZY DO ZAPAMIĘTANIA

Typy prymitywne zmiennoprzecinkowe



Domyślnie każda liczba zmiennoprzecinkowa ma typ **double**.

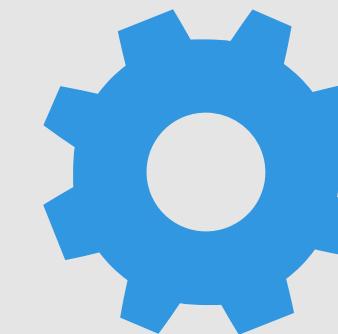
Aby używać typu **float** należy dodać przyrostek **f**. Opcjonalnie do deklaracji typu **double** można dodać przyrostek **d**.

Typu **float** i **double** używa się tylko do przechowywania wartości.

Do obliczeń na liczbach zmiennoprzecinkowych używa się klasy **BigDecimal**.

RZECZY DO ZAPAMIĘTANIA

Sposoby zapisu wartości typów zmiennoprzecinkowych

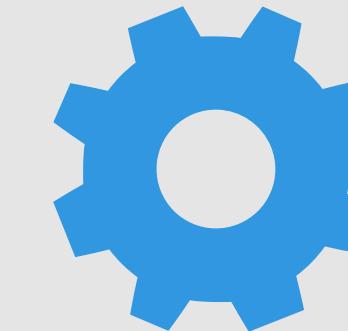


- notacja dziesiętna: float x = 1.0f;
- notacja szesnastkowa: float x = 0XAAp-2;
- notacja z podkreśleniem: float x = 1_000.00f;

RZECZY DO ZAPAMIĘTANIA

Typ prymitywny char

Służy do zapisy pojedynczego symbolu.



Zapisany jest 2 bajtach, co daje 65536 różnych symboli.

Każdy symbol ma unikalny kod unicode. Jest to standard, przestrzegany na całym świecie.

Typy zmiennoprzecinkowe i całkowite można rzutować do typu char, ale tylko z jawnym użyciem operatora rzutowania.

RZECZY DO ZAPAMIĘTANIA

Typ prymitywny boolean



Służy do zapisy prawdy albo fałszu.

Zapisany jest na 1 bicie.

Nie można rzutować typu boolean do innych typów prymitywnych i odwrotnie.

RZECZY DO ZAPAMIĘTANIA

Typ	Pamięć	Zakres
byte	1 bajt	od -128 do 127
short	2 bajty	od -32,768 do 32,767
int	4 bajty	od -2,147,483,648 do 2,147,483,647 (niewiele ponad 2 miliardy)
long	8 bajtów	od -9,223,372,036,854,775,808 do 9,223,372,036,854,775,807
float	4 bajty	W przybliżeniu ±3.40282347E+38F (6–7 cyfr po przecinku)
double	8 bajtów	W przybliżeniu ±1.79769313486231570E+308 (15 cyfr po przecinku)
char	2 bajty	Kody znaków unicode, od 0 do 65 535
boolean	1 bit	true/false

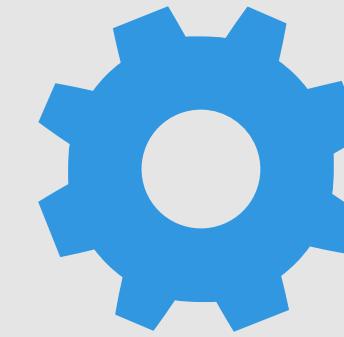
RZECZY DO ZAPAMIĘTANIA

Operatory logiczne i relacyjne

Operator	Nazwa	Przykład
<	Mniejsze od	$x < 0$
\leq	Mniejsze równe od	$x \leq 0$
>	Większe od	$x > 0$
\geq	Większe równe od	$x \geq 0$
\neq	Nierówny	$x \neq 0$
\equiv	Równy	$x \equiv 0$
$\&\&$	Koniunkcja (iloraz logiczny)	<code>true && x > 0</code>
$\ $	Alternatywa (suma logiczna)	<code>false \ x < 0</code>
!	Negacja	$!(x > 0)$
$? :$	Trójargumentowy	$x > 0 ? "X większe od 0" : "X mniejsze bądź równe 0"$

RZECZY DO ZAPAMIĘTANIA

Typy prymitywne i Wrappery



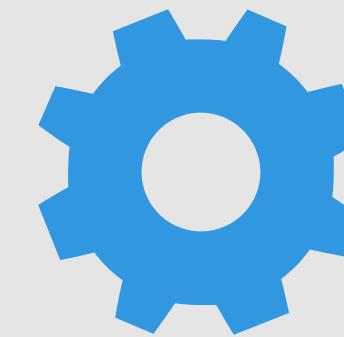
Wrapper typu prymitywnego to klasa, która opakowuje ten typ prymitywny.

Wrappery używa się, gdy:

- chcemy użyć typów prymitywnych w kolekcjach.
- chcemy sparsować typ **String** do typów danego typu.
Służy do tego metoda **parseXXX()**.
- chcemy użyć wartości **null**.

RZECZY DO ZAPAMIĘTANIA

Autoboxing i Unboxing



To automatyczny mechanizm do konwersji między typami prymitywnymi, a Wrapperami.

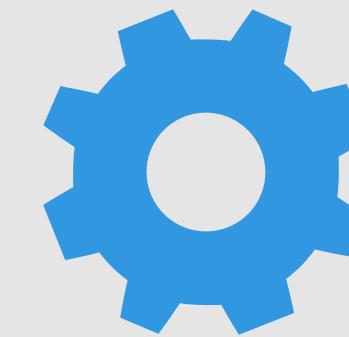
Mechanizm działa automatycznie za każdym razem, gdy oczekiwany jest obiekt, ale typ prymitywny jest dostępny, i vice versa

RZECZY DO ZAPAMIĘTANIA

Typ prymitywny	Wrapper
byte	java.lang.Byte
short	java.lang.Short
int	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
char	java.lang.Character
boolean	java.lang.Boolean

RZECZY DO ZAPAMIĘTANIA

Zmienne lokalne



Deklarujemy je w metodzie, konstruktorze albo bloku.

W pamięci tworzone są dopiero na wejście do metody, konstruktora albo bloku.

Są widoczne tylko w metodzie, konstruktora albo bloku.

Nie mają wartości domyślnych. Muszą być zainicjalizowane przed użyciem.

RZECZY DO ZAPAMIĘTANIA

Zmienne instancji



Deklarujemy je w klasie, zwykle jak **private**.

W pamięci tworzone są po utworzeniu konkretnej instancji klasy.

Ich widoczność zależy od modyfikatora dostępu.

Mają wartość domyślną.

RZECZY DO ZAPAMIĘTANIA

Zmienne klasy



Deklarujemy je w klasie, trzeba je oznaczyć jako **static**.

W pamięci tworzone są po uruchomieniu programu.

Ich widoczność zależy od modyfikatora dostępu.

Mają wartość domyślną, analogicznie jak zmienne instancji.

RZECZY DO ZAPAMIĘTANIA

Stałe



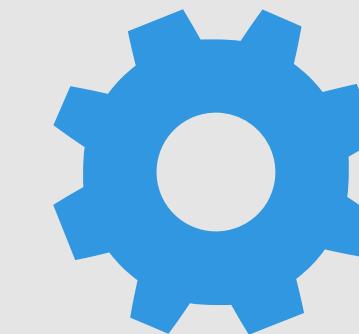
Służą do zapisu wartości, które są niezmienne w czasie życia programu.

Deklarujemy je przy użyciu słów kluczowych **static final**.

Zapisane są w pamięci z chwilą uruchomienia programu.

RZECZY DO ZAPAMIĘTANIA

Typy enumerowane



Służą do definiowania kolekcji stałych logicznie powiązanych ze sobą wartości.

Typ **enum** jest obiektem i dziedziczy wszystkie metody z klasy **Object**.

Enum zapisany jest w pamięci z chwilą uruchomienia programu.

RZECZY DO ZAPAMIĘTANIA

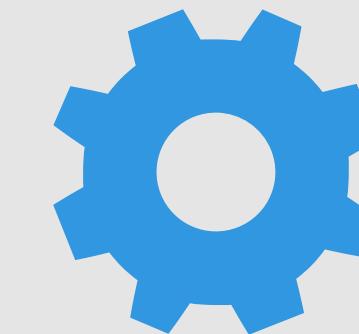
Operator	Nazwa	Przykład
&	Bitowy AND	4 & -6
	Bitowy OR	4 -6
^	Bitowy XOR	4 ^ -6
~	Bitowa negacja	~4
<<	Przesunięcie bitowe w lewo ze znakiem	4 << 2
>>	Przesunięcie bitowe w prawo ze znakiem	4 >> 2
>>>	Przesunięcie bitowe w prawo bez znakiem	4 >>> 2

RZECZY DO ZAPAMIĘTANIA

Priorytet	Operatory
1	expr++ expr-
2	++expr --expr +expr -expr ~ !
3	* / %
4	+ -
5	<< >> >>>
6	< > <= >= instanceof
7	&
8	^
9	
10	&&
11	
12	? :
13	= += -= *= /= %= &= = ^= = <<= >>= >>>=

RZECZY DO ZAPAMIĘTANIA

Switch



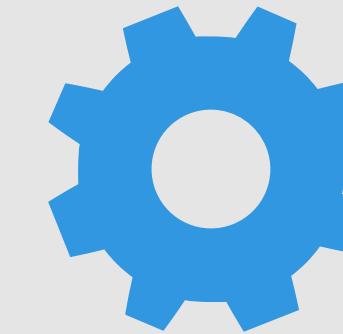
Switch stosujemy dla typów **byte**, **short**, **int**, **char** oraz dla **Wrapperów** tych typów **Byte**, **Short**, **Integer** i **Character**.

Switch działa dla typu **String** oraz typów **enumerowanych**.

Switch testuje wyrażenie tylko na typie całkowitym, słowie albo typie enumerowanym.

RZECZY DO ZAPAMIĘTANIA

Switch

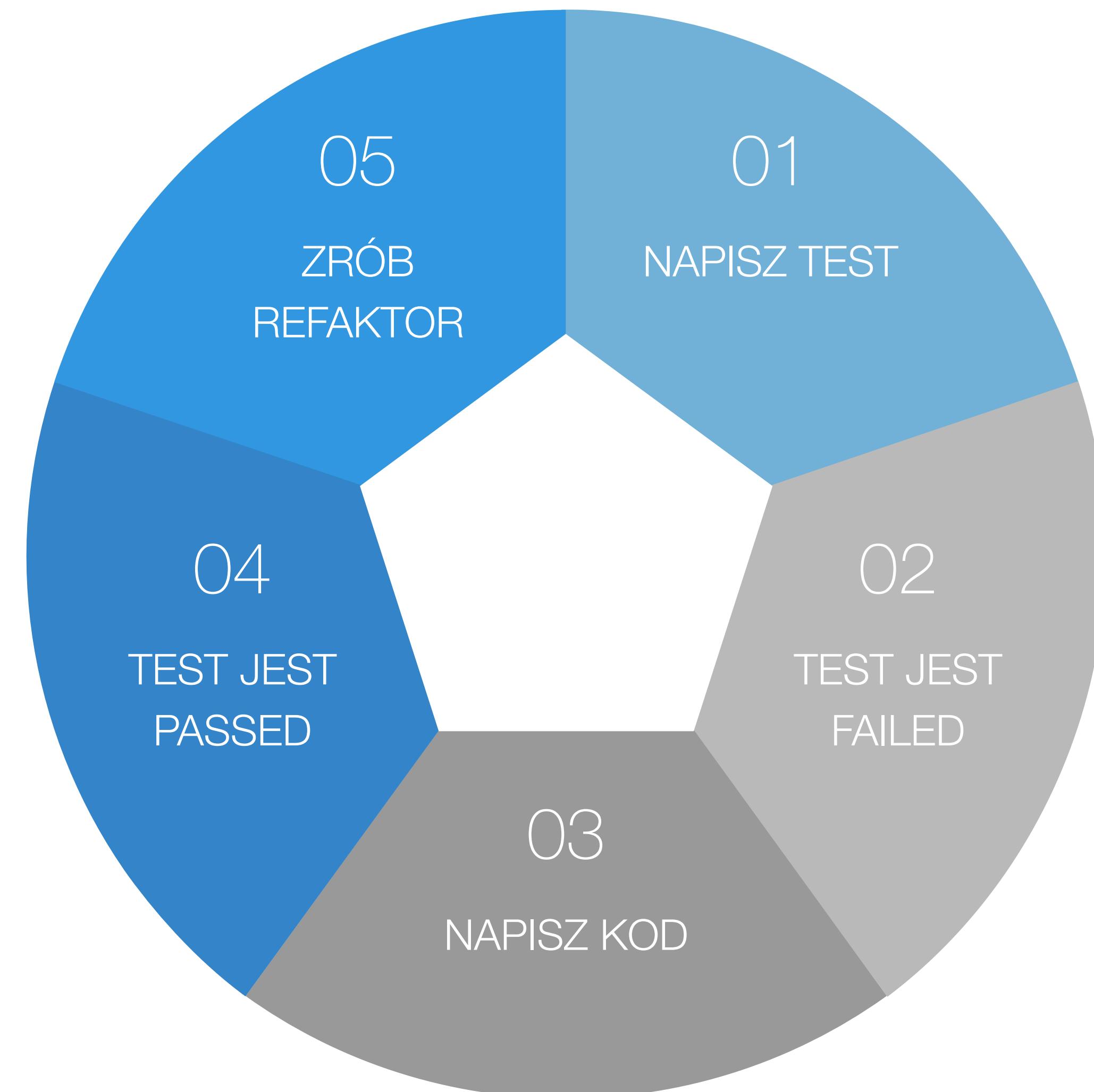


Switch będzie testował każdy przypadek (case) po kolej w przypadku braku instrukcji przerwania **break**.

Czytelność decyduje o tym czy wybrać instrukcję **if-else**, czy **switch**.

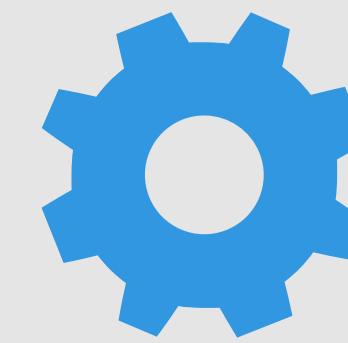
Instrukcja **if-else** pozwala testować zakres liczbowy,
Switch tylko pojedyncze wartości.

RZECZY DO ZAPAMIĘTANIA



RZECZY DO ZAPAMIĘTANIA

Vargs



To skrót składniowy do deklaracji parametrów o typie tablicowym.

Parametr **Vararg** musi być zawsze ostatnim parametrem metody.

Parametr **Vararg** jest opcjonalny, przyjmuje wtedy wartość **null**.

TABLICE I ŁAŃCUCHY - JAK ZAPISANE SĄ STRINGI W PAMIĘCI?

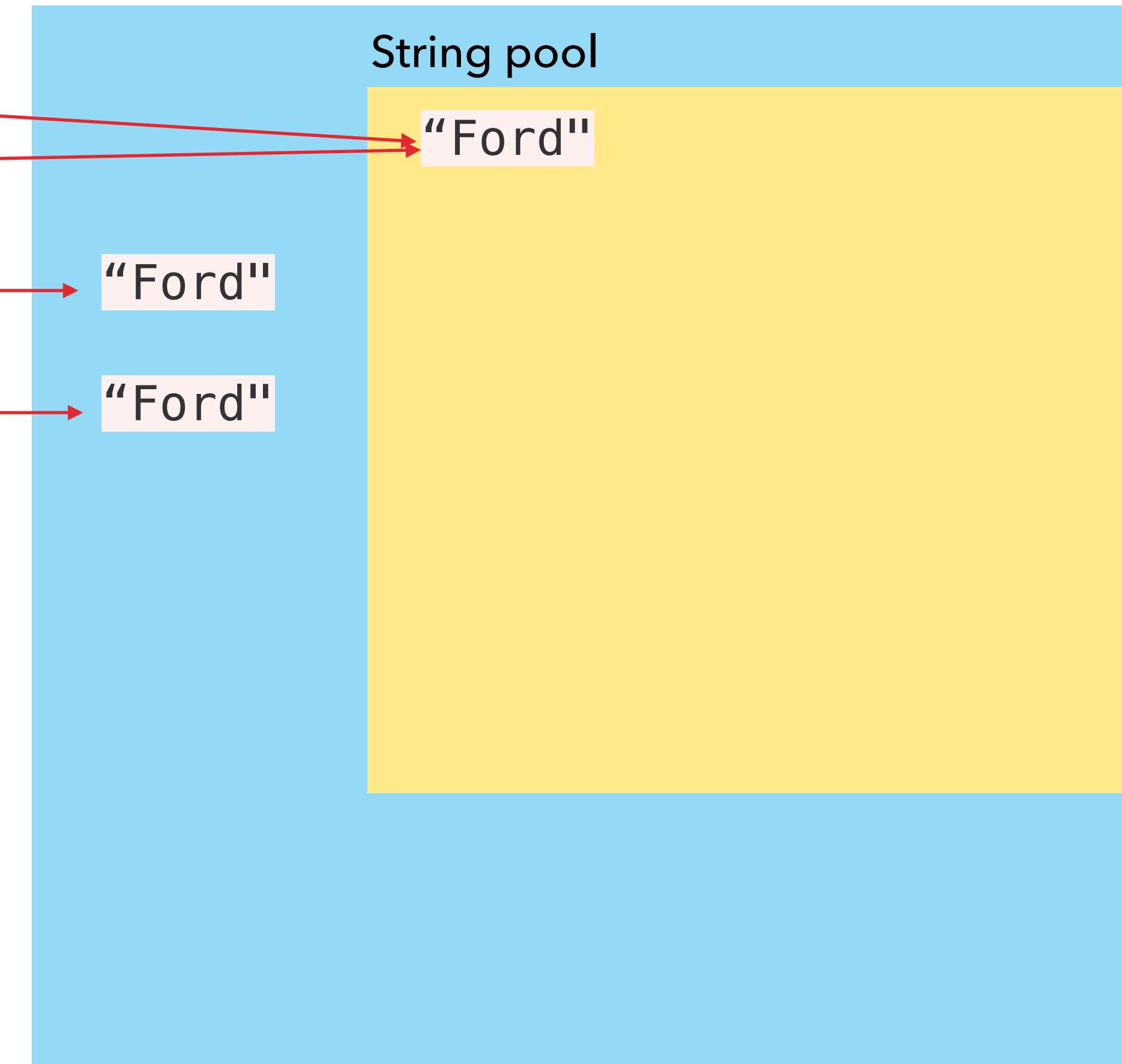
```
String car1 = "Ford";
```

```
String car2 = "Ford";
```

```
String car3 = new String("Ford");
```

```
String car4 = new String("Ford");
```

Memory



TABLICE I ŁAŃCUCHY - JAK ZAPISANE SĄ STRINGI W PAMIĘCI?

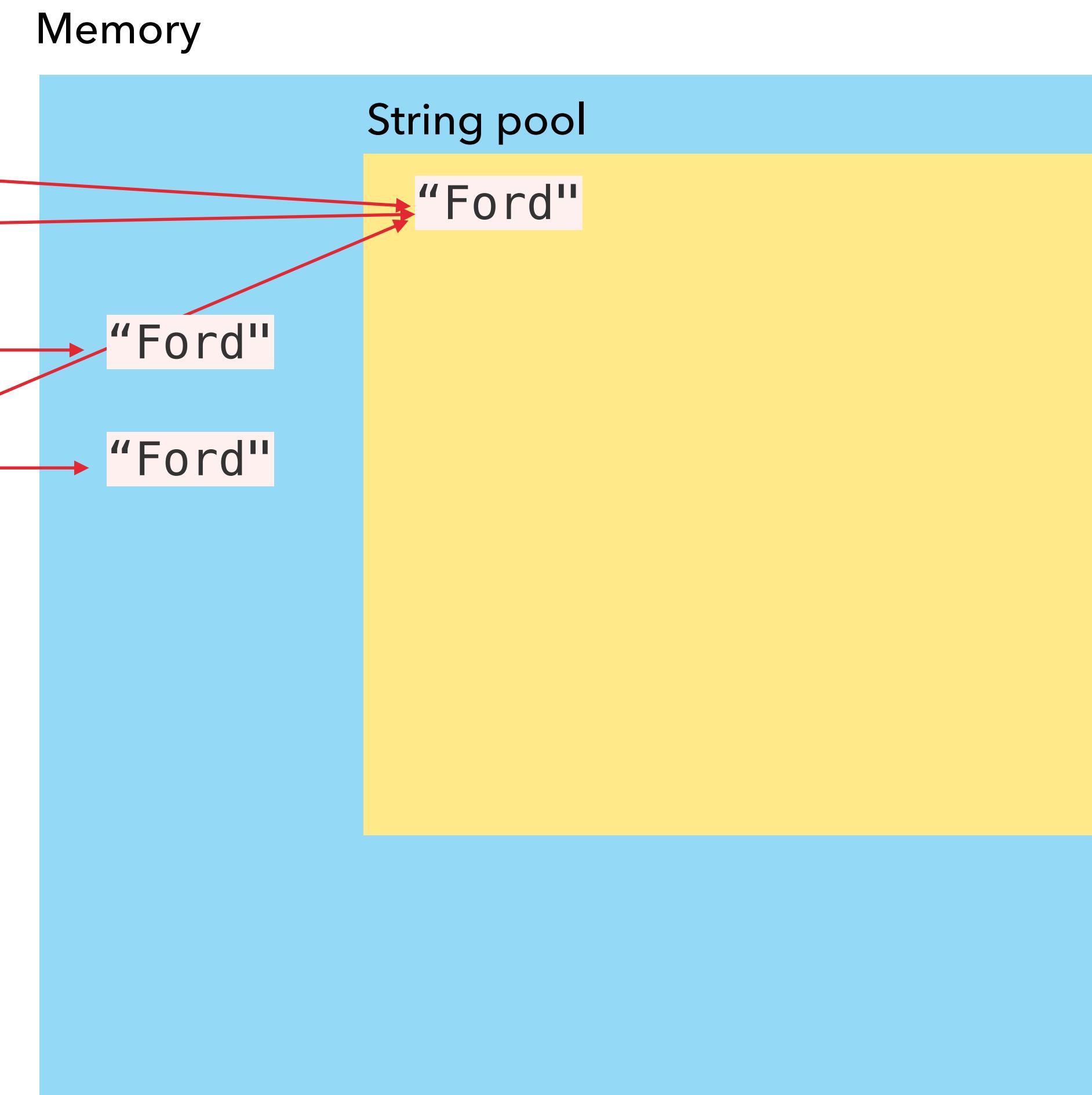
```
String car1 = "Ford";
```

```
String car2 = "Ford";
```

```
String car3 = new String("Ford");
```

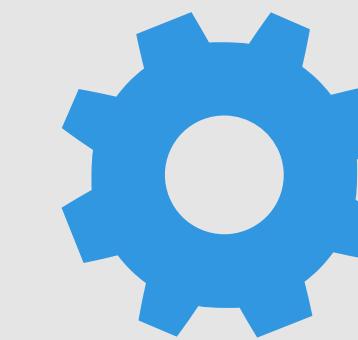
```
String car4 = new String("Ford");
```

```
String car4Interned = car4.intern();
```



RZECZY DO ZAPAMIĘTANIA

String pool



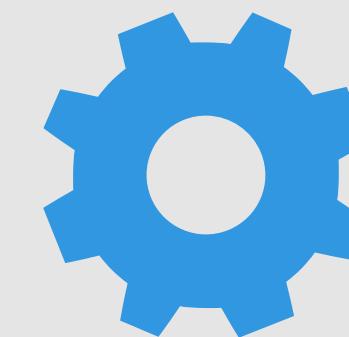
String pool to specjalne miejsce w pamięci, które przechowuje identyczne stringi.

Stringi zainicjalizowane za pomocą operatora **new** nie trafiają do **String pool**.

String pool został wymyślony w celu optymalizacji.

RZECZY DO ZAPAMIĘTANIA

String



Operator równość `==` sprawdza, czy zmienne wskazują (referują) na ten sam adres w pamięci.

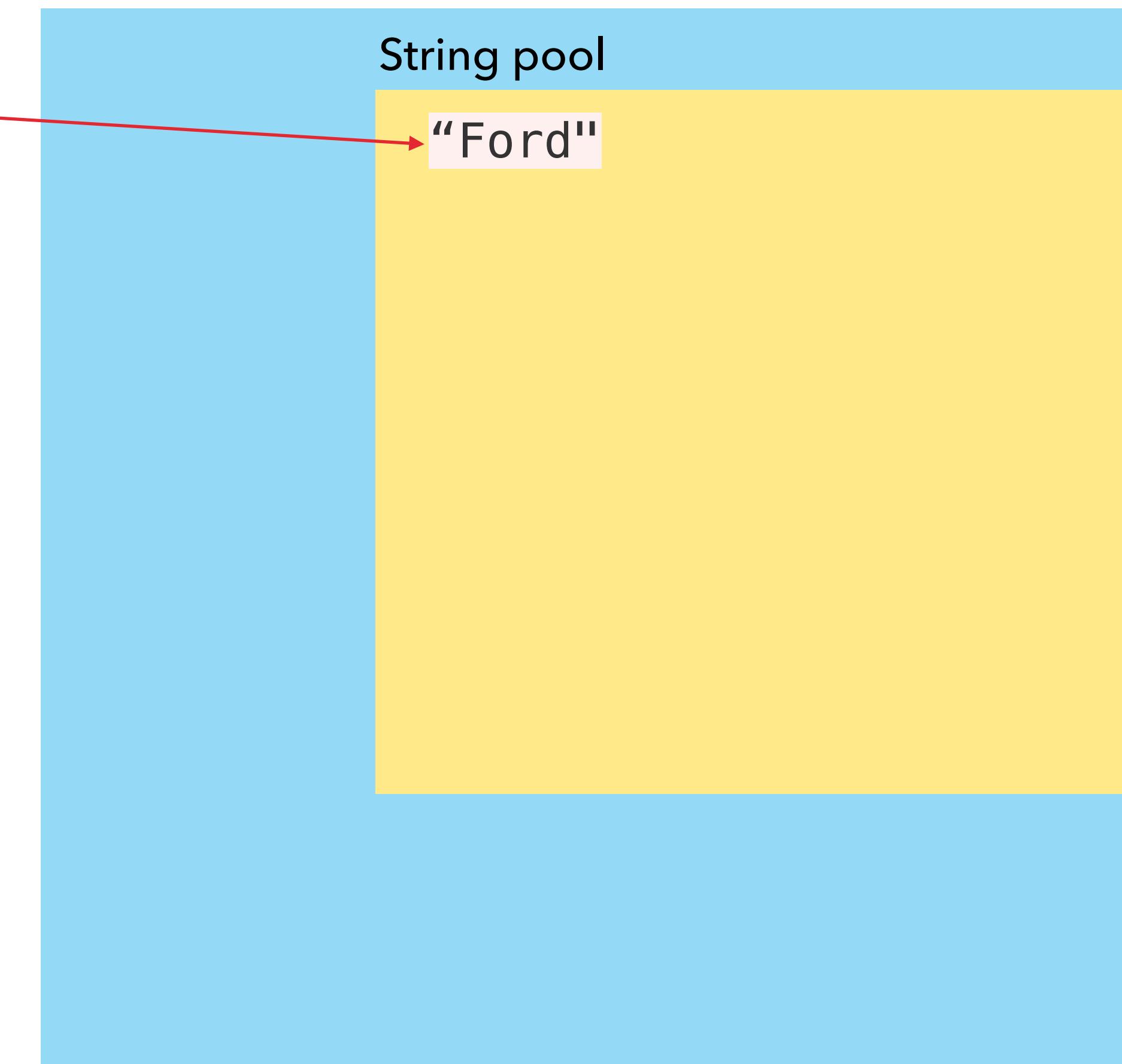
Do porównania czy 2 stringi mają taką samą wartość, używa się metody `“Ford”.equals(“Ford”)`.

Aby pobrać adres obiektu **String**, ze **String Pool**, używa się metody `.intern()`.

TABLICE I ŁAŃCUCHY - PRACA Z KLASĄ STRING

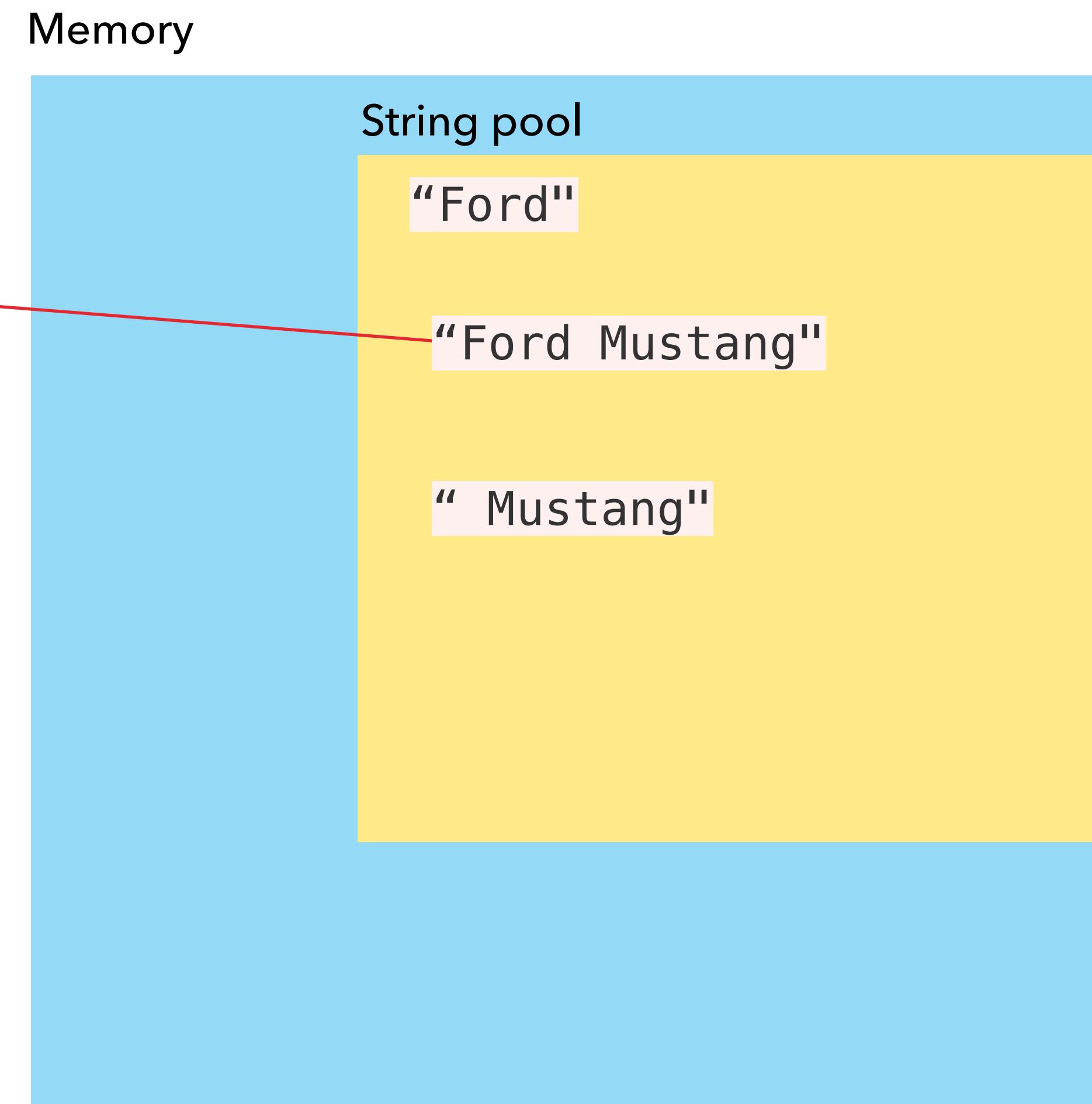
```
String car = "Ford";
```

Memory



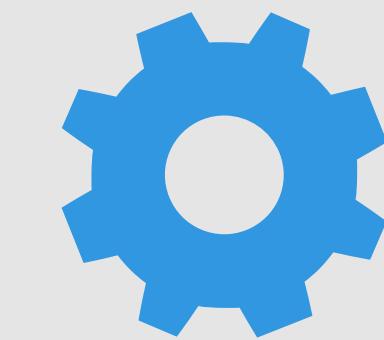
TABLICE I ŁAŃCUCHY - PRACA Z KLASĄ STRING

```
String car = "Ford";  
car = "Ford" + " Mustang";
```



RZECZY DO ZAPAMIĘTANIA

String

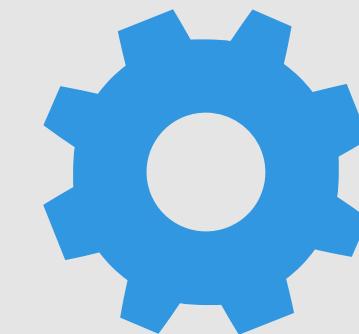


Każdy **String** jest niezmienny (**immutable**). Oznacza to, że nie można zmienić wartości raz zainicjalizowanej zmiennej.

Konkatenacja to łączenie Stringów. Do łączenia **Stringów** służy metoda **.concat()**, albo operator **+**.

RZECZY DO ZAPAMIĘTANIA

String

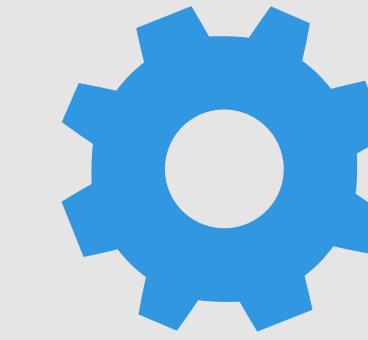


Ponieważ każdy **String** jest niezmienny, to przy konkatenacji **Stringów** każdy napis jest zapisywany w pamięci (**String pool**).

Do łączenia dużej liczby napisów używa się klasy **StringBuilder**.

RZECZY DO ZAPAMIĘTANIA

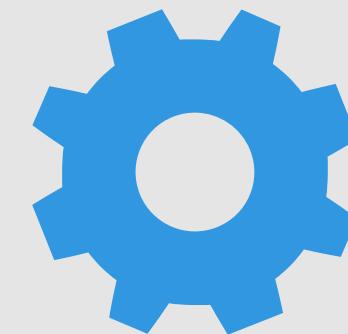
Enkapsulacja/Hermetyzacja



Polega na ukrywaniu pewnych danych składowych lub metod obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub klasom zaprzyjaźnionym.

RZECZY DO ZAPAMIĘTANIA

Modyfikatory dostępu



Private - widoczność tylko w środku klasy.

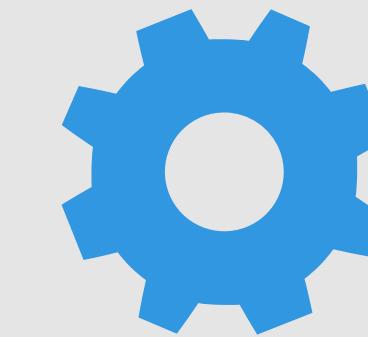
Default - widoczność tylko w pakiecie.

Protected - widoczność w pakiecie i przez dziedziczenie.

Public - widoczność nieograniczona.

RZECZY DO ZAPAMIĘTANIA

Klasa zagnieżdżona i wewnętrzna



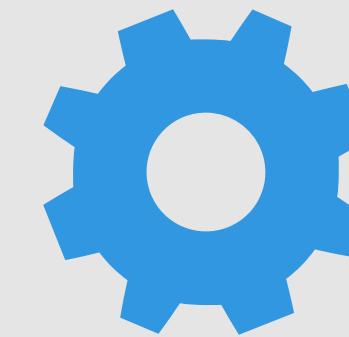
To **klasa**, której definicja znajduje się wewnątrz innej **klasy**.

Klasa zagnieżdżona może być **statyczna** albo **niestatyczna**.

Niestatyczne klasy zagnieżdżone nazywamy **klasami wewnętrznymi**.

RZECZY DO ZAPAMIĘTANIA

Klasa wewnętrzna i statyczna zagnieżdżona



Klasa wewnętrzna “widzi” wszystkie elementy **klasy zewnętrznej** (również prywatne).

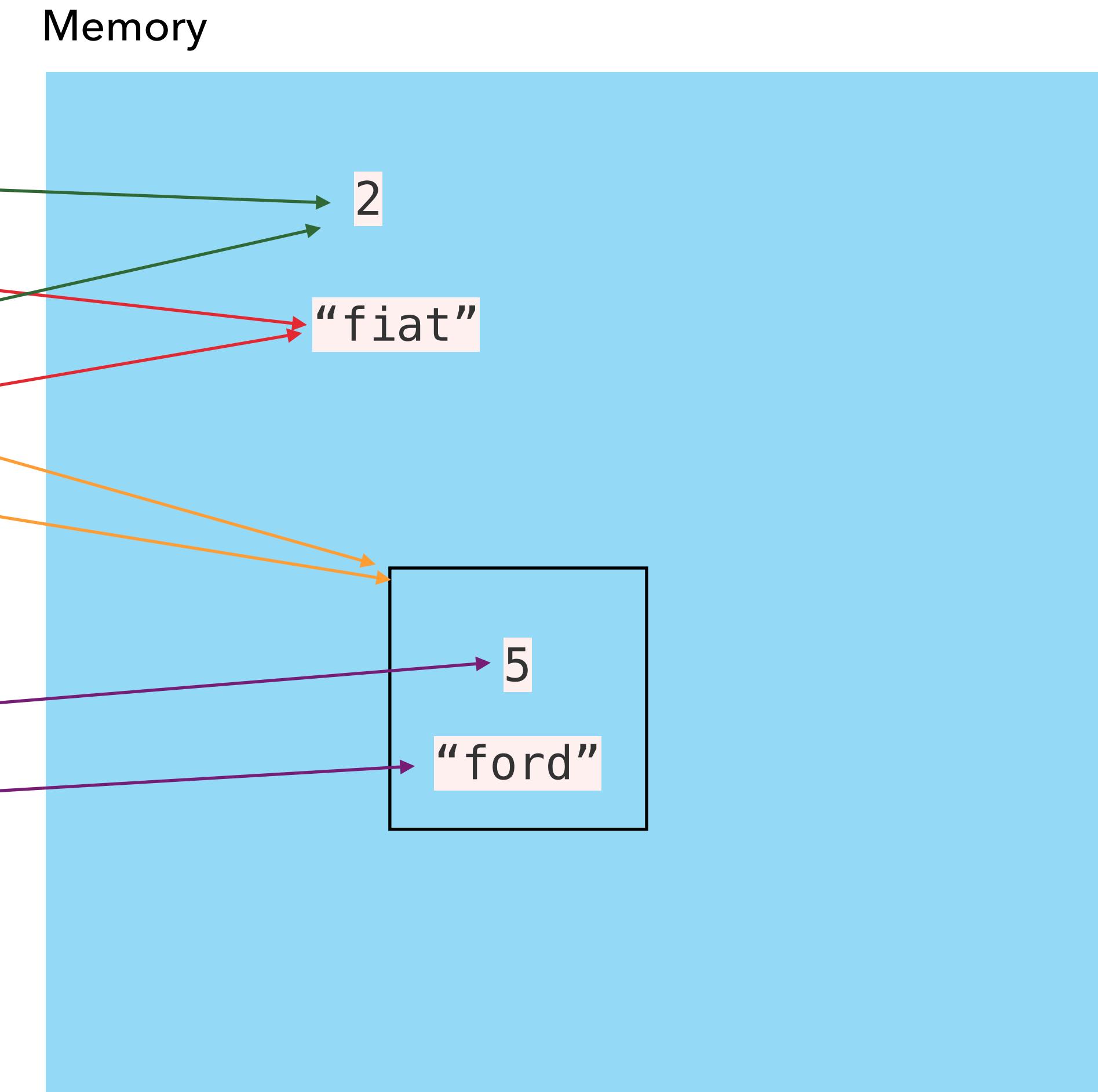
Klasa statyczna zagnieżdżona “widzi” wszystkie elementy **klasy zewnętrznej**, które są **statyczne**.

KLASY I OBIEKTY - PRZEKAZYWANIE ZMIENNYCH DO METOD

```
int i = 2;           ← green arrow points to 2
String s = "fiat";   ← red arrow points to "fiat"
Car car = new Car(); ← orange arrow points to Car object
method(i,s,car);

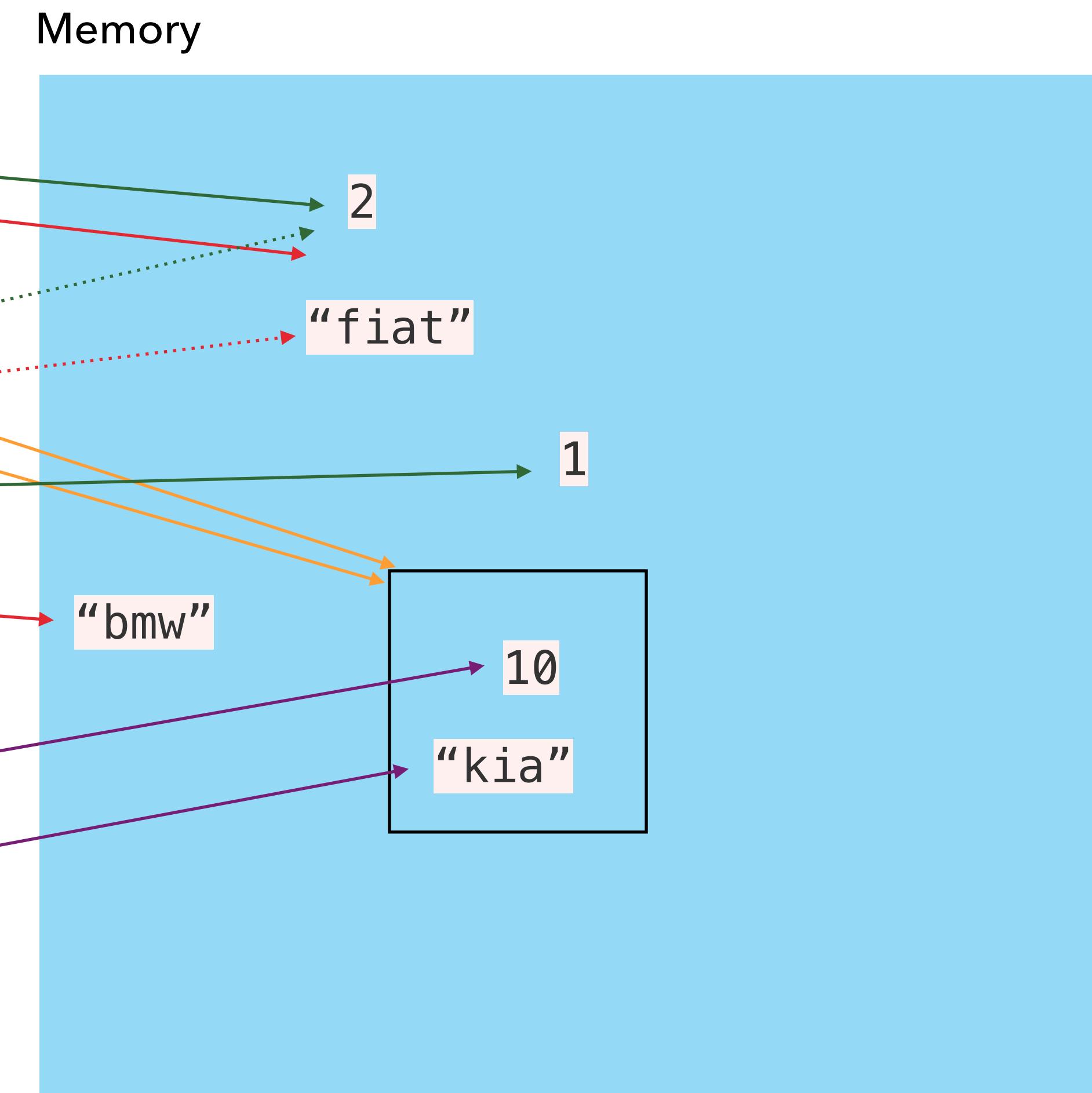
public void method(int x, String y, Car c){
}

class Car{
    private int a = 5;   ← purple arrow points to 5
    private String b ="ford"; ← purple arrow points to "ford"
}
```



KLASY I OBIEKTY - PRZEKAZYWANIE ZMIENNYCH DO METOD

```
int i = 2;  
String s = "fiat";  
Car car = new Car();  
  
method(i,s,car);  
  
public void method(int x, String y, Car c){  
    x = 1;  
    y = "bmw";  
    c.a = 10;  
    c.b = "kia";  
}  
  
class Car{  
    private int a = 5;  
    private String b = "ford";  
}
```

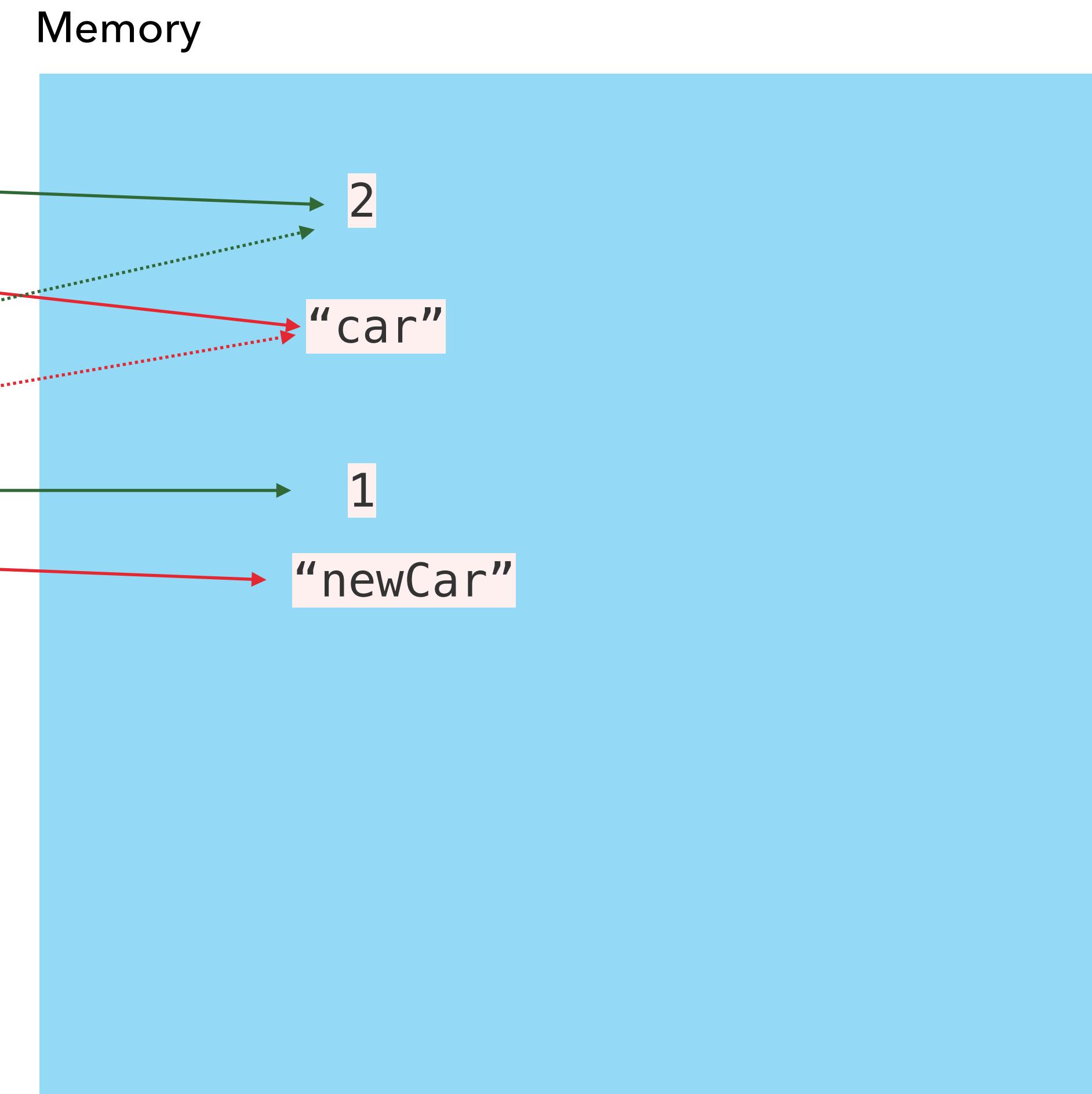


KLASY I OBIEKTY - PRZEKAZYWANIE ZMIENNYCH DO METOD

```
int i = 2;  
String s = "car";  
Car car = new Car();
```

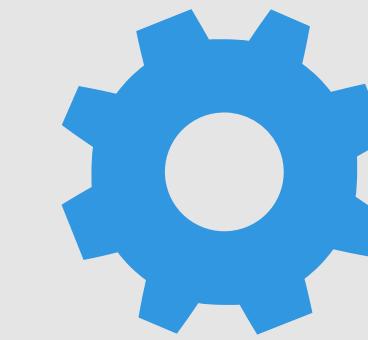
```
method(i,s);
```

```
public void method(int x, String y){  
    x = 1;  
    y = "newCar";  
}
```



RZECZY DO ZAPAMIĘTANIA

Przekazywanie zmiennych do metod

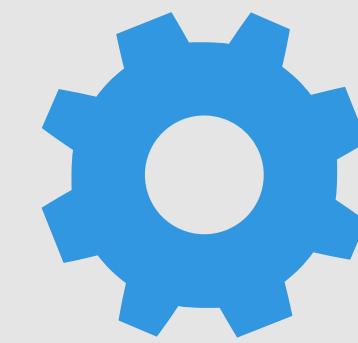


W Java zmienne przekazywane są do metod za pomocą mechanizmu **pass-by-value**.

Do metody nie jest przekazywana wartość zmiennej, tylko **kopia** referencji do adresu w pamięci, gdzie zapisana jest zmienna.

RZECZY DO ZAPAMIĘTANIA

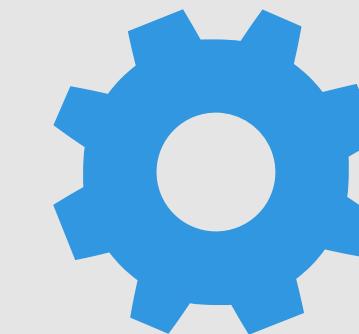
Inicjalizacja zmiennych instancji



- instrukcja deklaracji i inicjalizacji
- konstruktor
- blok inicjalizacyjny

RZECZY DO ZAPAMIĘTANIA

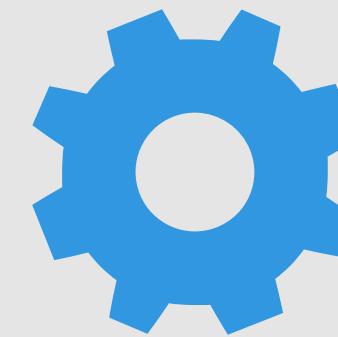
Inicjalizacja zmiennych statycznych



- instrukcja deklaracji i inicjalizacji
- statyczny blok inicjalizacyjny

RZECZY DO ZAPAMIĘTANIA

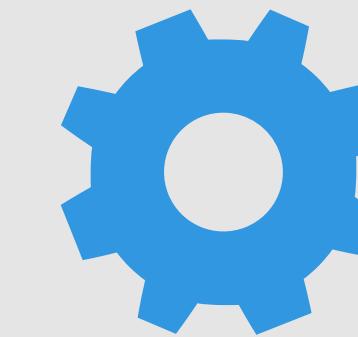
Kolejność wywoływania



- Statyczne bloki inicjalizacyjne (wg kolejności w pliku źródłowym)
- Bloki inicjalizacyjne (wg kolejności w pliku źródłowym)
- Konstruktory

RZECZY DO ZAPAMIĘTANIA

STOS - STACK



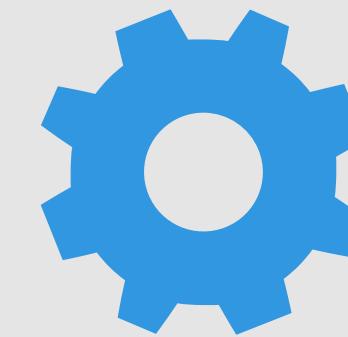
Rośnie i kurczy się wraz z wywoływaniem i wychodzeniem z metod. Ma stały rozmiar.

Zapisane są tutaj **zmienné prymitywne lokalne** oraz **referencje** do obiektów.

Zmienne istnieją do czasu działania metody, w której są zdefiniowane.

RZECZY DO ZAPAMIĘTANIA

STOS - STACK

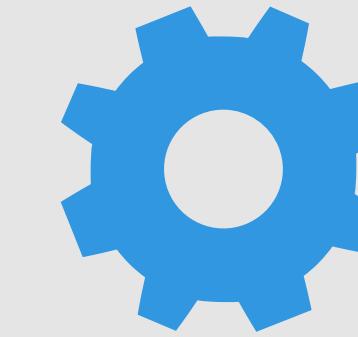


Każdy wątek ma swój **stos**.

Jeżeli brakuje pamięci Java wyrzuci wyjątek
java.lang.StackOverFlowError.

RZECZY DO ZAPAMIĘTANIA

STERTA - HEAP

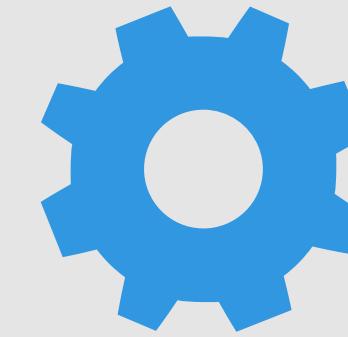


Jest dynamiczna. Nie ma ograniczeń w rozmiarze.

Zapisane są tutaj wartości **typów obiektowych i zmienne prymitywne** należące do obiektów.

RZECZY DO ZAPAMIĘTANIA

STERTA - HEAP



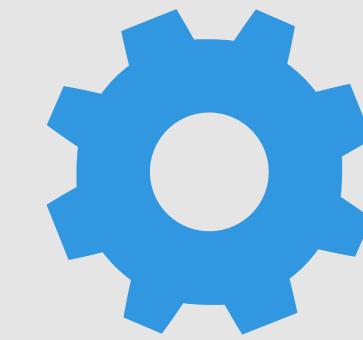
Sterta jest wspólna dla wszystkich wątków.

Jeżeli brakuje pamięci Java wyrzuci wyjątek
java.lang.OutOfMemoryError.

String pool jest częścią **sterty**.

RZECZY DO ZAPAMIĘTANIA

Dziedziczenie

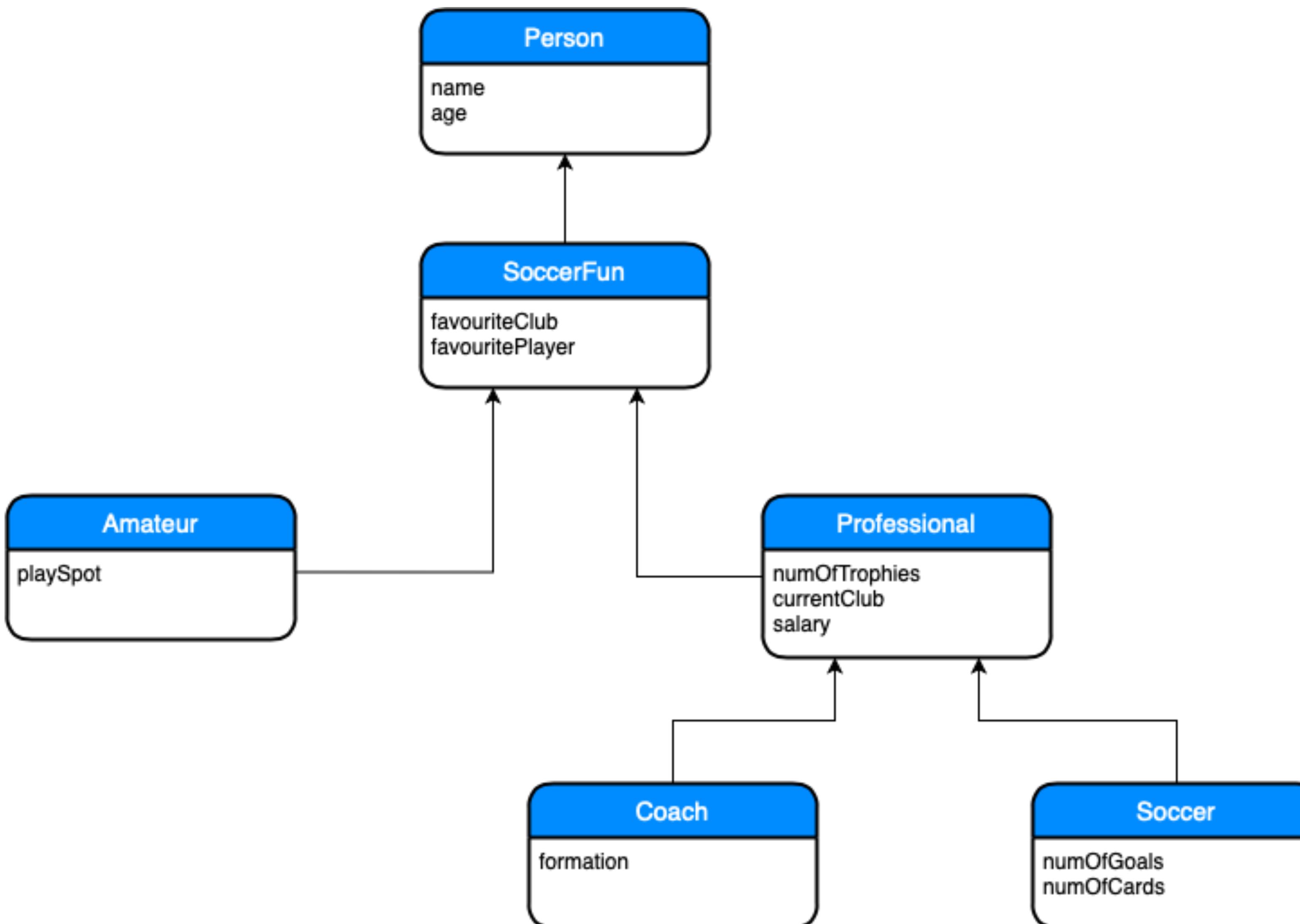


Dziedziczenie umożliwia przekazywanie pewnych cech między klasą nadzcznąą (**parent class/ super class**), a klasami podlegającymi (**child class/subclass**).

DZIEDZICZENIE, ABSTRAKCJA I POLIMORFIZM - DZIEDZICZENIE KLAS

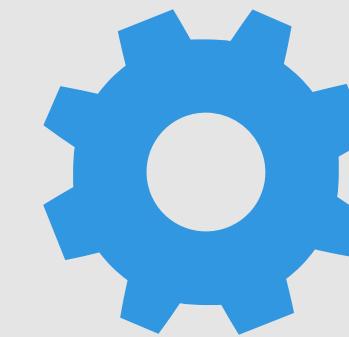


DZIEDZICZENIE, ABSTRAKCJA I POLIMORFIZM - DZIEDZICZENIE KLAS



RZECZY DO ZAPAMIĘTANIA

Dziedziczenie



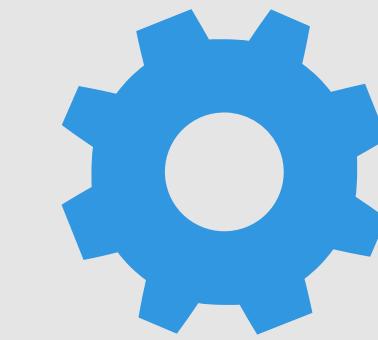
Java pozwala na dziedziczenie tylko 1 klasy. Wielodziedziczenie jest nieosiągalne.

Do dziedziczenia używa się słowa kluczowe **extends** w deklaracji klasy.

Wszystkie elementy instancji klasy oznacza **protected** zostaną odziedziczone.

RZECZY DO ZAPAMIĘTANIA

Modyfikatory dostępu



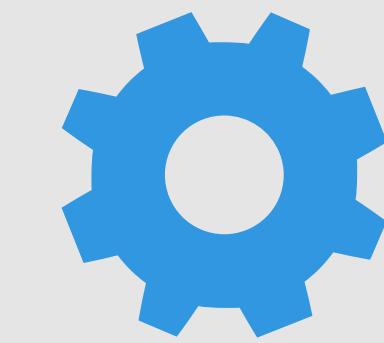
Nadpisywana metoda nie może mieć “mniejszego” modyfikatora dostępu niż zapisywana metoda.

public > protected > default > private

Prywatnych metod nie da się nadpisać. Prywatne metody nie są dziedziczone!

RZECZY DO ZAPAMIĘTANIA

Zwracany typ



Typ zwracany z nadpisywanej metody musi być albo taki sam albo musi być jego subklasą.

RZECZY DO ZAPAMIĘTANIA

Lista parametrów



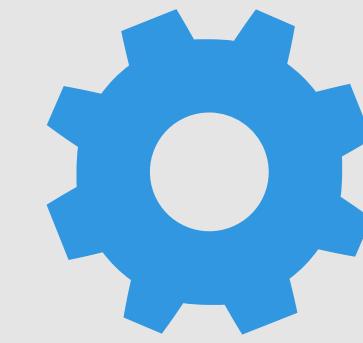
Lista parametrów nadpisywanej metody, musi być zgodna pod względem typu i kolejności.

Jeżeli parametry nie są takie same, to wówczas jest to przeładowanie metody.

RZECZY DO ZAPAMIĘTANIA

Wyjątki

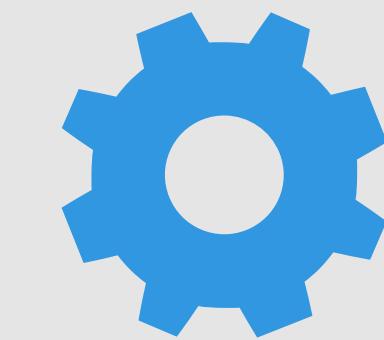
Dojdziemy do tego :)



4

RZECZY DO ZAPAMIĘTANIA

Nadpisywanie metod



Nie można nadpisywać metod, które są oznaczone jako **final**.

Nadpisując metodę można się odwołać do implementacji z klas nadrzędnych przy użyciu **super**.

Opcjonalnie można oznaczać nadpisywane metody anotacją **@override**.

DZIEDZICZENIE, ABSTRAKCJA I POLIMORFIZM - POLIMORFIZM

DZIEDZICZENIE

Przekazanie pewnych cech
innym klasom

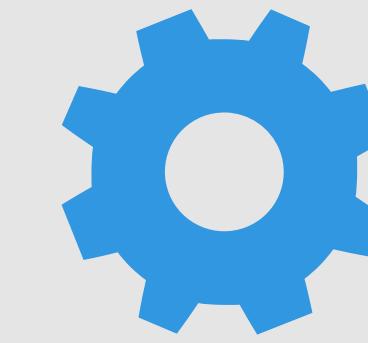


Enkapsulacja

Ukrywanie widoczności
elementów danej klasy dla
innych klas.

RZECZY DO ZAPAMIĘTANIA

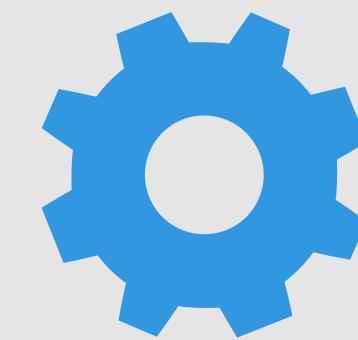
**Polimorfizm (gr. wiele form,
wielopostaciowość)**



To zdolność obiektu, żeby działać inaczej dla innych typów.

RZECZY DO ZAPAMIĘTANIA

Sygnatury metody



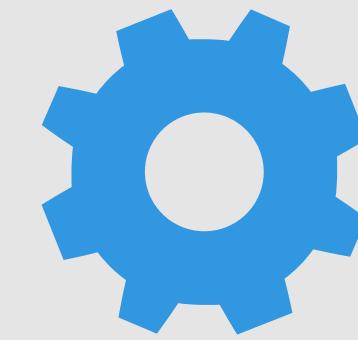
To unikalny identyfikator metody w klasie.

Składa się z:

- nazwy metody
- uporządkowanej listy parametrów

RZECZY DO ZAPAMIĘTANIA

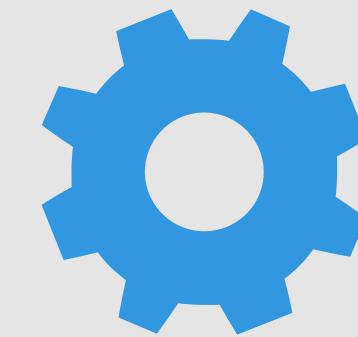
Przeciążanie metody



To definiowanie metod o tej samej nazwie, ale innych parametrów.

RZECZY DO ZAPAMIĘTANIA

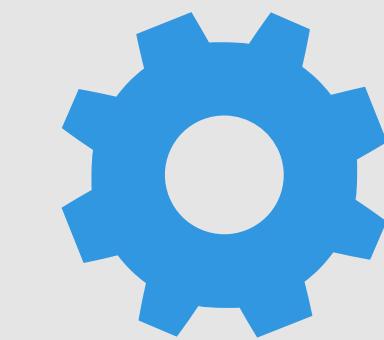
Zasady selekcji metody



- Dokładne dopasowanie
- Najbardziej konkretna metoda:
double > float > long > int > char
int > short > byte
- Widening przed autoboxing
- Autoboxing przed varargs

RZECZY DO ZAPAMIĘTANIA

Klasa abstrakcyjna

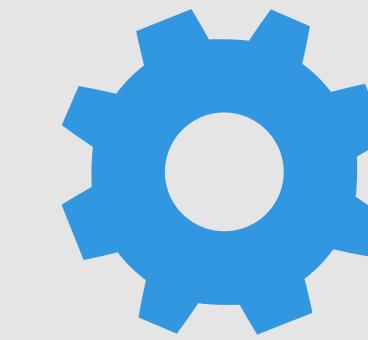


To klasa ma zablokowaną możliwość tworzenia instancji.

W klasach abstrakcyjnych, oprócz tworzenia elementów zwykłej klasy, można deklarować metody abstrakcyjne.

RZECZY DO ZAPAMIĘTANIA

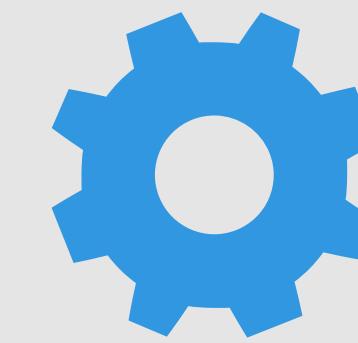
Metoda abstrakcyjna



To metoda, która musi być zaimplementowana w pierwszej konkretnej klasie, która dziedziczy z klasy abstrakcyjnej.

RZECZY DO ZAPAMIĘTANIA

Interfejs

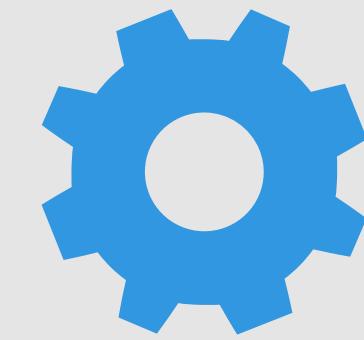


To to zestaw wymagań dla **klas**, które chcą być zgodne z **interfejsem**. **Interfejs** to nie jest **klasa**!

Do oznaczenia, że **klasa** implementuje **interfejs** używa się słowa kluczowego **implements** w deklaracji **klasy**.

RZECZY DO ZAPAMIĘTANIA

Interfejs - deklaracja



Każdy **interfejs** jest domyślnie oznaczony jako **abstract**.

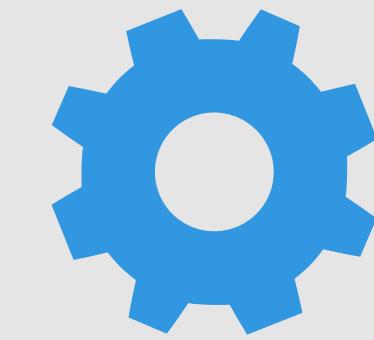
Wszystkie metody **interfejsu** są domyślnie **public abstract**.

(*nie do końca prawda, od Java 8)

Wszystkie zmienne są domyślnie **public static final**.

RZECZY DO ZAPAMIĘTANIA

Interfejs vs Klasa abstrakcyjna



Klasy mogą implementować wiele **interfejsów**, ale dziedziczyć tylko po jednej **klasie abstrakcyjnej**.

Interfejsy nie mogą mieć implementacji metod oznaczonych jako **public**, **protected** i **default**.

RZECZY DO ZAPAMIĘTANIA

Konwencje nazewnicze



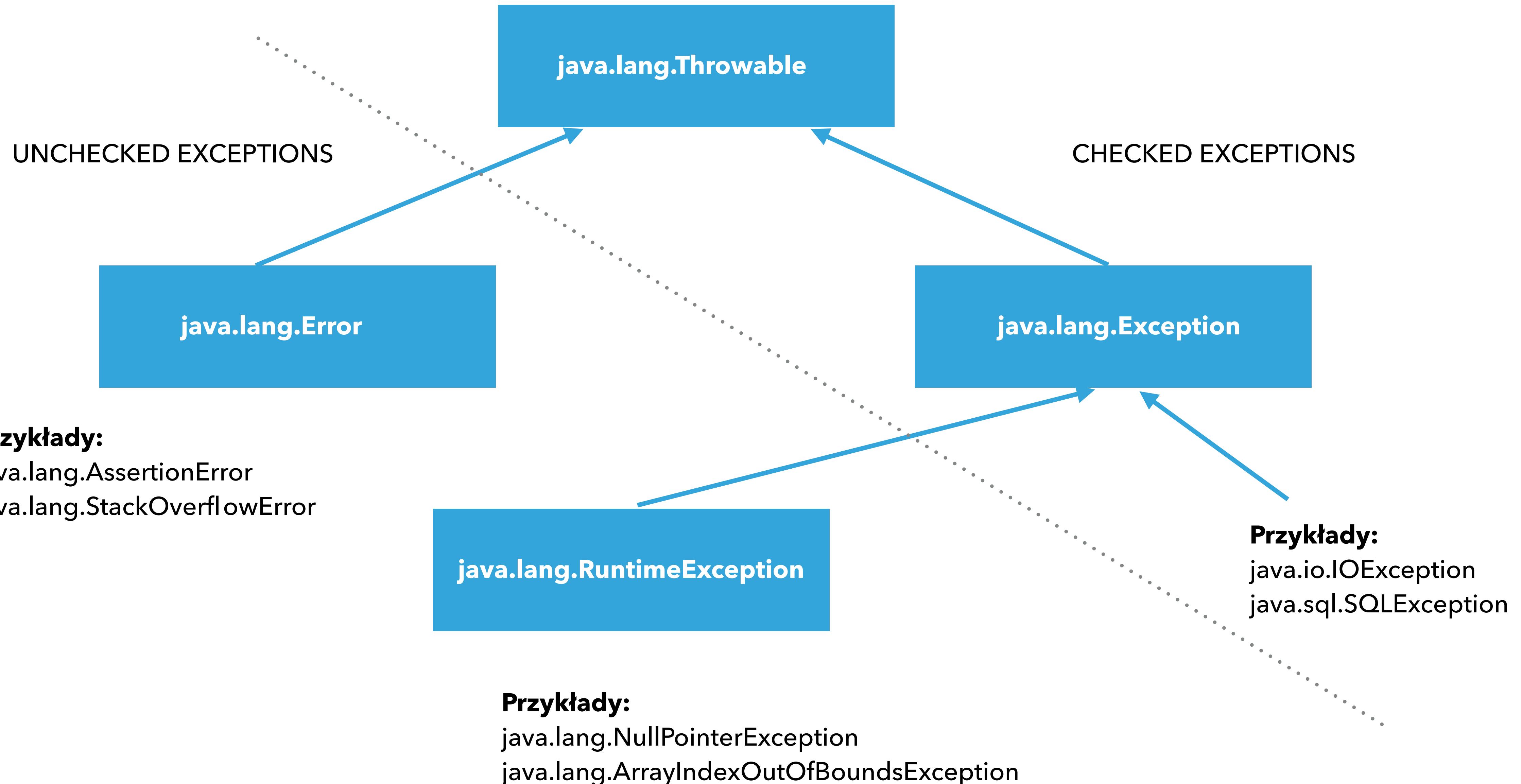
Nazwa **interfejsu** powinna być przymiotnikiem i wyrażać cechę, którą nadaje jej ten interfejs. Często stosuje się sposób tworzenia przymiotników od rzeczowników:

rzeczownik + **able**.

np. Walk + **able** = **walkable** (pol. chodzący)

Nazwa **interfejsu** może być też rzeczownikiem kiedy reprezentuje rodzinę klas. np. **List**, **Map**.

WYJĄTKI I ASERCJE - OBSŁUGA SYTUACJI WYJĄTKOWYCH



RZECZY DO ZAPAMIĘTANIA

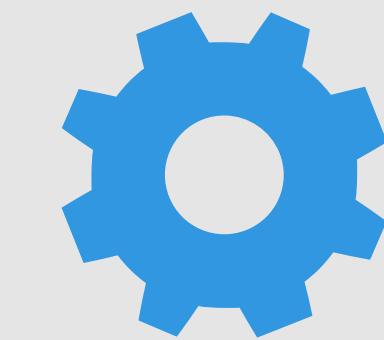
Checked vs Unchecked



Wyjątki **checked** muszą być obsłużone przez program.

RZECZY DO ZAPAMIĘTANIA

Modyfikatory dostępu



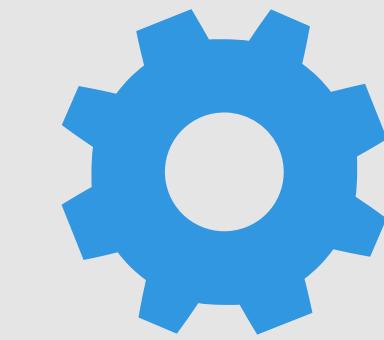
Nadpisywana metoda nie może mieć “mniejszego” modyfikatora dostępu niż zapisywana metoda.

public > protected > default > private

Prywatnych metod nie da się nadpisać. Prywatne metody nie są dziedziczone!

RZECZY DO ZAPAMIĘTANIA

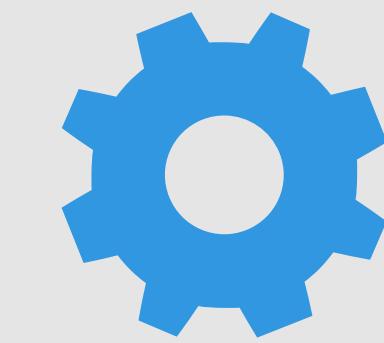
Zwracany typ



Typ zwracany z nadpisywanej metody musi być albo taki sam albo musi być jego subklasą.

RZECZY DO ZAPAMIĘTANIA

Lista parametrów

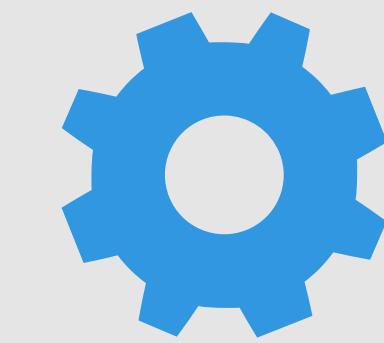


Lista parametrów nadpisywanej metody, musi być zgodna pod względem typu i kolejności.

Jeżeli parametry nie są takie same, to wówczas jest to przeładowanie metody.

RZECZY DO ZAPAMIĘTANIA

Wyjątki



Typ wyjątku wyrzucany z metody nie może być bardziej ogólny od wyrzucanego z nadpisywanej metody.

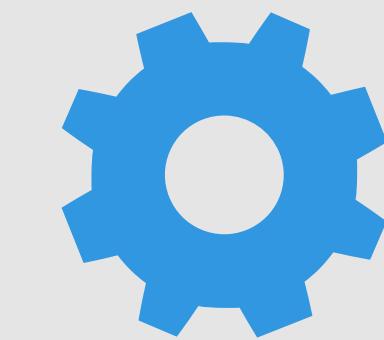
Metoda nie może wyrzucać nowych wyjątków.

Metoda może w ogóle nie wyrzucać wyjątków.

Zasady obowiązują tylko dla wyjątków checked.

RZECZY DO ZAPAMIĘTANIA

Nadpisywanie metod



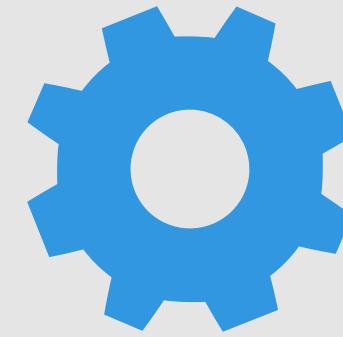
Nie można nadpisywać metod, które są oznaczone jako **final**.

Nadpisując metodę można się odwołać do implementacji z klas nadrzędnych przy użyciu **super**.

Opcjonalnie można oznaczać nadpisywane metody anotacją **@override**.

RZECZY DO ZAPAMIĘTANIA

Asercje



Używa się ich **tylko** w fazie developmentu.

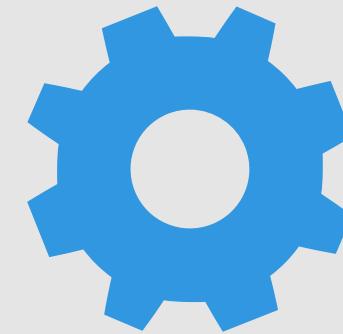
Służą do znalezienia błędów w kodzie.

Są domyślnie wyłączone i nie powinno ich się używać w aplikacji produkcyjnej.

Nie zmieniają kodu, są przezroczyste.

RZECZY DO ZAPAMIĘTANIA

Asercje



Używa się ich **tylko** w fazie developmentu.

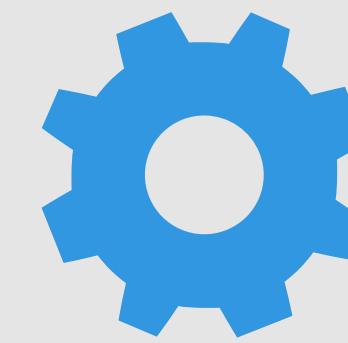
Służą do znalezienia błędów w kodzie.

Są domyślnie wyłączone i nie powinno ich się używać w aplikacji produkcyjnej.

Nie zmieniają kodu, są przezroczyste.

RZECZY DO ZAPAMIĘTANIA

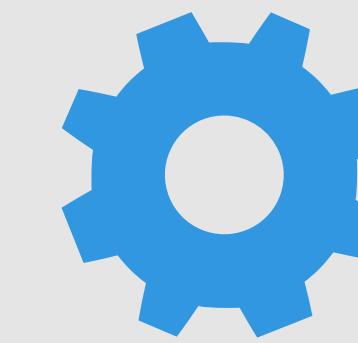
Cechy ArrayList



- przechowuje duplikaty
- można dodawać nulle
- elementy są uporządkowane
- wstawianie i wyciąganie elementów na dowolnym
miejscu (dostęp przez index)

RZECZY DO ZAPAMIĘTANIA

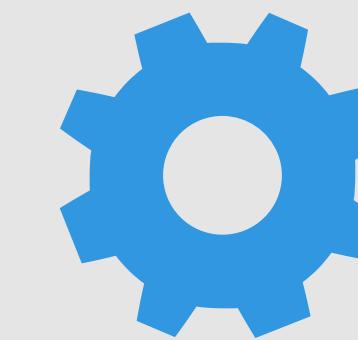
Cechy PriorityQueue



- przechowuje duplikaty
- **nie** można dodawać null
- elementy są uporządkowane
- wstawianie wg priorytetu, i wyciąganie głównego

RZECZY DO ZAPAMIĘTANIA

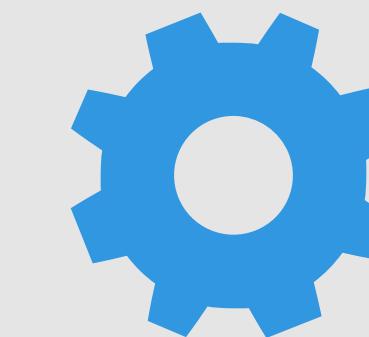
Cechy LinkedList



- przechowuje duplikaty
- **nie** można dodawać nulle
- elementy są uporządkowane (kolejność wstawiania, podwójne połączenie)
- wstawianie i wyciąganie elementów na początku i końcu jest szybkie
- usuwanie elementu jest wydajniejsze niż w ArrayList

RZECZY DO ZAPAMIĘTANIA

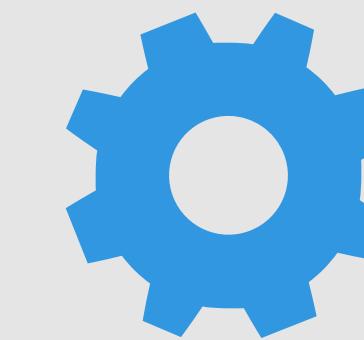
Cechy ArrayDeque



- przechowuje duplikaty
- **nie** można dodawać nulle
- elementy są uporządkowane (kolejność wstawiania, z przodu albo z tyłu)
- wstawianie i wyciąganie elementów tylko na początku i końcu
- usuwanie elementu na początku i końcu kolejki

RZECZY DO ZAPAMIĘTANIA

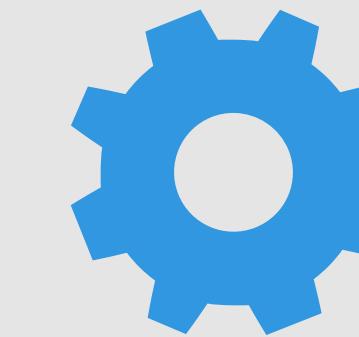
Kontrakt equals



- **zwrotna** - $x.equals(x) == true$
- **symetryczna** - jeżeli $x.equals(y) == true$ to $y.equals(x) == true$
- **przechodnia** - $x.equals(y) = true$ oraz $y.equals(z) = true$, to
 $x.equals(z) = true$
- **spójna** - wielokrotne wywoływanie $x.equals(y)$ musi zawsze dawać taką samą wartość boolean.
- zawsze $x.equals(null) = false$

RZECZY DO ZAPAMIĘTANIA

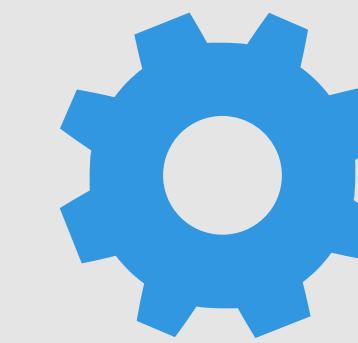
Kontrakt hashCode



- Zawsze wywołanie metody **hashCode** na tym samym obiekcie musi kończyć się zwróceniem tej samej liczby całkowitej
- Jeżeli dwa obiekty są sobie **równe** (wg metody **equals**), to ich hashCode również **musi** być równy.
- Jeżeli obiekty są **różne** (wg metody **equals**), to ich **hashCode może** być równy, jednak ze względów wydajnościowych powinno to być unikane.

RZECZY DO ZAPAMIĘTANIA

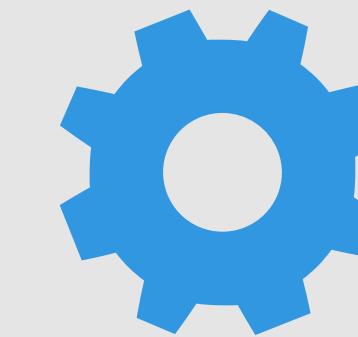
Dobre praktyki



- użycie tych samych pól w equals i Dashcode
- jednoznaczne identyfikatory, np. Pesel, nip, regon albo par pól np. waluta i ilość
- nie używać identyfikatorów bazodanowych
- null check

RZECZY DO ZAPAMIĘTANIA

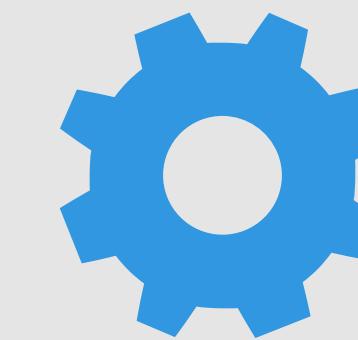
Cechy HashSet



- przechowuje unikalne elementy
- można dodawać nulle
- elementy są **nie** uporządkowane
- nie ma metody get, trzeba iterować
- wydajność wyciągania i usuwania elementów zależy od **hashcode**

RZECZY DO ZAPAMIĘTANIA

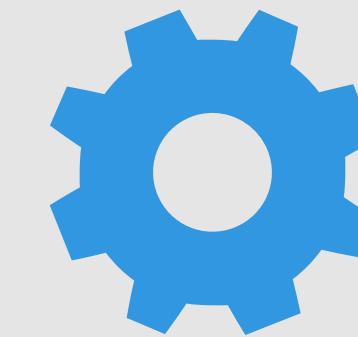
Cechy LinkedHashSet



- przechowuje unikalne elementy
- można dodawać nulle (tylko jeden)
- elementy **są** uporządkowane (kolejność dodawania)
- nie ma metody get, trzeba iterować
- wydajność usuwania elementów zależy od
hashcode

RZECZY DO ZAPAMIĘTANIA

Cechy TreeSet

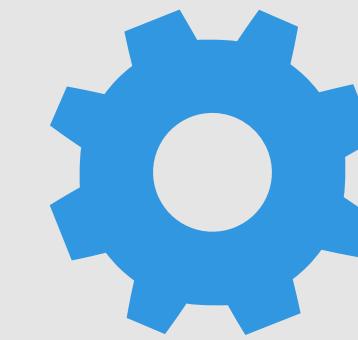


- przechowuje unikalne elementy
- Nie używa hashowania!
- **nie** można dodawać nulli

- elementy **są** uporządkowane (kolejność wg Comparable)
- nie ma metody get, trzeba iterować
- wydajność usuwania elementów **nie** zależy od
hashcode

RZECZY DO ZAPAMIĘTANIA

Cechy HashMap



- przechowuje kolekcja par klucz-wartość. Klucze są unikalne
- Używa hashowania dla kluczy
- **można** dodawać 1 null jako klucz
- elementy **nie** są uporządkowane
- jest metoda get, pobieranie po kluczu
- wydajność pobierania i usuwania po kluczu zależy od **hashcode**

RZECZY DO ZAPAMIĘTANIA

Cechy **LinkedHashMap**



- przechowuje kolekcja par klucz-wartość. Klucze są unikalne
- Używa hashowania dla kluczy
- **można** dodawać 1 null jako klucz
- elementy są uporządkowane (wg kolejności, albo wg najrzadziej używanego)
- wydajność pobierania i usuwania po kluczu zależy od **hashcode**

RZECZY DO ZAPAMIĘTANIA

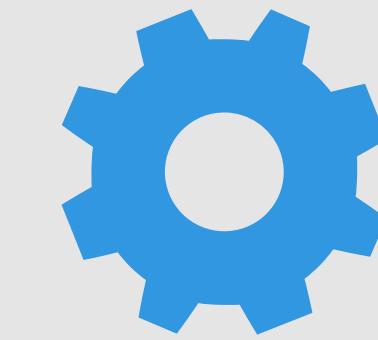
Cechy TreeMap



- przechowuje kolekcja par klucz-wartość. Klucze są unikalne
- **nie** używa hashowania dla kluczy
- **nie** można dodawać null jako klucz, ale jako wartości tak
- elementy są uporządkowane (domyślnie rosnąco, wg naturalnej kolejności)

RZECZY DO ZAPAMIĘTANIA

Anotacje

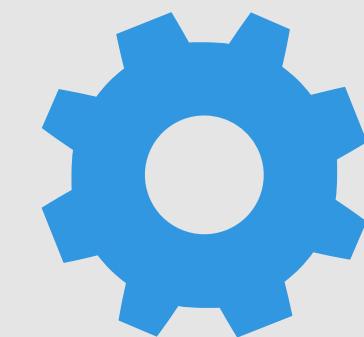


To mechanizm na dodawanie metadanych do kodu źródłowego.

Oznaczanie nie mają wpływu na działanie programu.

RZECZY DO ZAPAMIĘTANIA

Metody w anotacjach



Musza być bezparametrowe.

Nie mogą wyrzucać błędów.

Zwracany typ to: typ prymitywny, **String**, **Class**, **enum** albo tablica tych typów.

Wartość domyślna nie może być **null**.

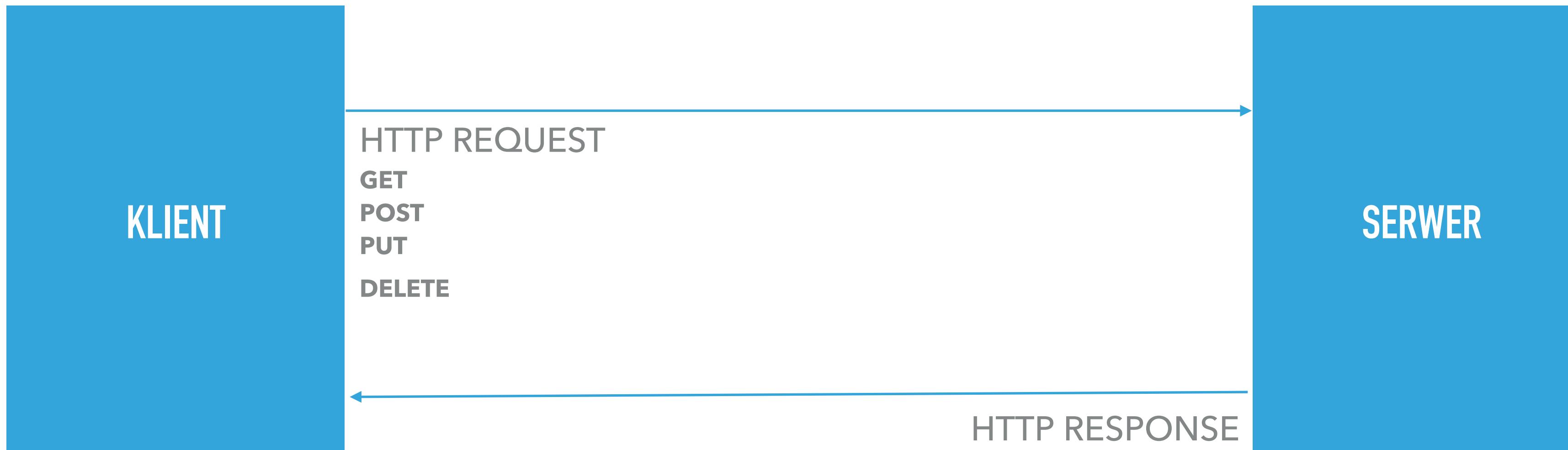
ANOTACJE I REFLEKSJA - DYNAMICZNE PROXY

<CLASS>
MAIN

<INTERFACE>
ICALCULATOR

<INTERFACE>
ICALCULATOR

KOMUNIKACJA Z SERWISAMI ZEWNĘTRZNYMI - REST API



PRAKTYCZNY KURS JAVA 11. POZIOM II

1. Wielowątkowość



PRAKTYCZNY KURS JAVA 11. POZIOM II

- 1. Wielowątkowość**
- 2. Typy generyczne**



PRAKTYCZNY KURS JAVA 11. POZIOM II

- 1. Wielowątkowość**
- 2. Typy generyczne**
- 3. Stream API**



PRAKTYCZNY KURS JAVA 11. POZIOM II

- 1. Wielowątkowość**
- 2. Typy generyczne**
- 3. Stream API**
- 4. Wyrażenia Lambda**



**JAVA
SKILLS.PL**



PRAKTYCZNY KURS JAVA 11. POZIOM II

- 1. Wielowątkowość**
- 2. Typy generyczne**
- 3. Stream API**
- 4. Wyrażenia Lambda**
- 5. Zarządzanie pamięcią**



**JAVA
SKILLS.PL**



PRAKTYCZNY KURS JAVA 11. POZIOM II

- 1. Wielowątkowość**
- 2. Typy generyczne**
- 3. Stream API**
- 4. Wyrażenia Lambda**
- 5. Zarządzanie pamięcią**
- 6. Moduły**



**JAVA
SKILLS.PL**



PRAKTYCZNY KURS JAVA 11. POZIOM II

- 1. Wielowątkowość**
- 2. Typy generyczne**
- 3. Stream API**
- 4. Wyrażenia Lambda**
- 5. Zarządzanie pamięcią**
- 6. Moduły**
- 7. SQL**



PRAKTYCZNY KURS JAVA 11. POZIOM II

- 1. Wielowątkowość**
- 2. Typy generyczne**
- 3. Stream API**
- 4. Wyrażenia Lambda**
- 5. Zarządzanie pamięcią**
- 6. Moduły**
- 7. SQL**
- 8. Nashron**

