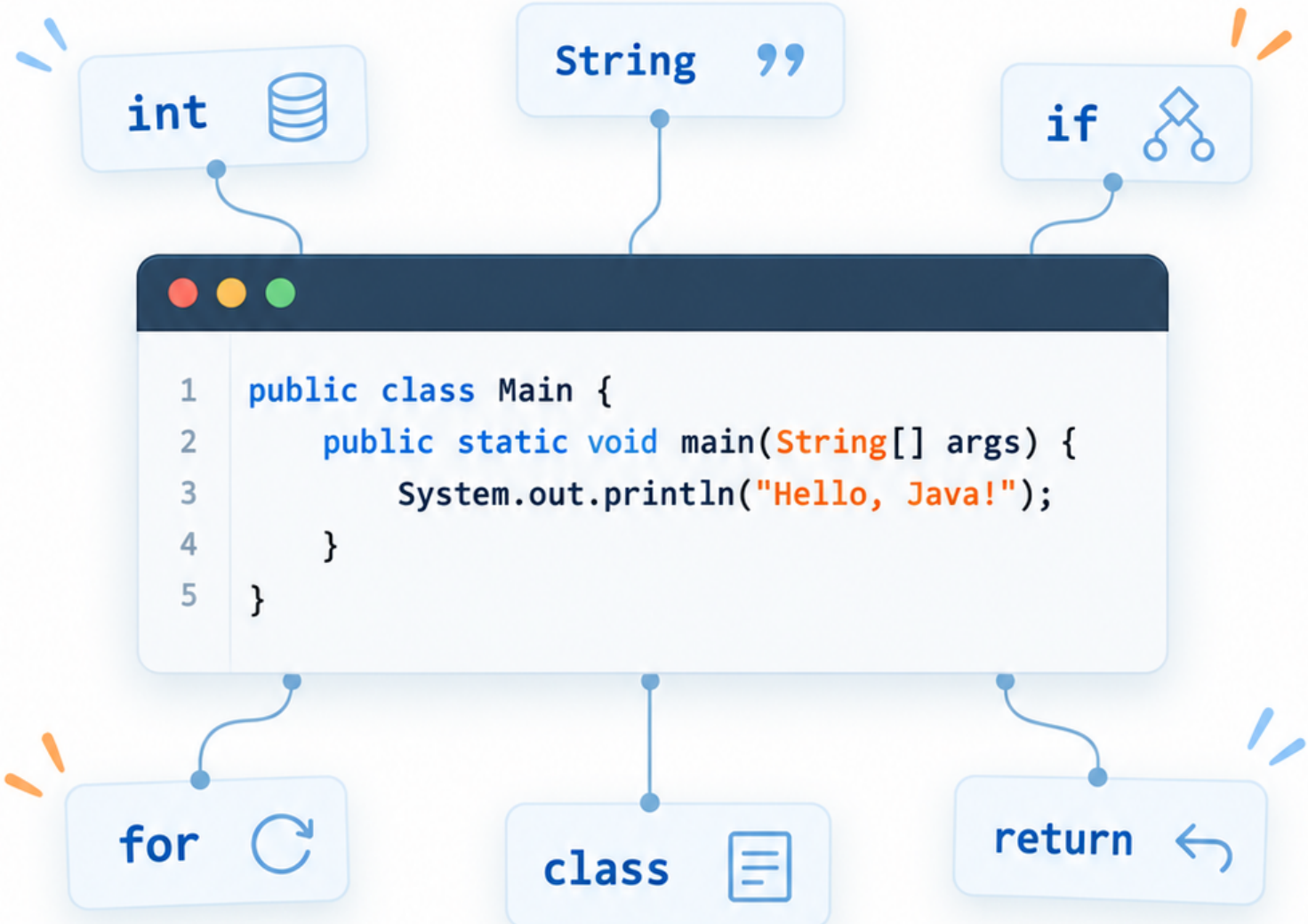
BONUS #3

SZYBKI PRZEWODNIK PO SKŁADNI JAVY

Najważniejsze zasady, konstrukcje i przykłady kodu w jednym miejscu.



Twoja podręczna pomoc w nauce Javy.

INFORMACJE O PUBLIKACJI

Szybki przewodnik po składni Javy

Bonus #3 — Materiał edukacyjny

O publikacji

Niniejszy przewodnik został przygotowany jako materiał pomocniczy do nauki języka programowania Java. Zawiera przykłady składni, podstawowe konstrukcje języka oraz krótkie wyjaśnienia ułatwiające samodzielną naukę.

Ważna informacja

Przykłady kodu mają charakter edukacyjny. Przed wykorzystaniem ich we własnych projektach należy sprawdzić działanie programu i dostosować go do konkretnych potrzeb.

Prawa autorskie

© 2026 — `[Właściciel praw autorskich do uzupełnienia]`
Wszelkie prawa zastrzeżone.

SPIS TREŚCI

Znajdź potrzebną konstrukcję Javy w kilka sekund.

NA POCZĄTEK

- Informacje o publikacji — **01**
- Spis treści — **02**
- Jak korzystać z przewodnika? — **03**

01 PODSTAWY SKŁADNI JAVY

Strony 04–12

Struktura programu, zasady zapisu, komentarze, zmienne, typy danych i konwersje.

02 OPERATORY I WYRAŻENIA

Strony 13–19

Działania matematyczne, porównania, logika i operatory przypisania.

03 INSTRUKCJE WARUNKOWE I PĘTLE

Strony 20–27

if, else, switch, for, while, do-while, break i continue.

04 TABLICE, TEKST I KOLEKCJE

Strony 28–35

Tablice, String, StringBuilder, ArrayList, HashMap i HashSet.

05 METODY, KLASY I OBIEKTY

Strony 36–43

Metody, parametry, klasy, konstruktory, obiekty, dziedziczenie i interfejsy.

06 ORGANIZACJA KODU, BŁĘDY I SZYBKA POWTÓRKA

Strony 44–49

Pakiety, importowanie, modyfikatory dostępu, wyjątki i końcowa ściąga.

BONUS #3

JAVA BEZ ZBĘDNEGO SZUKANIA

Szybko znajdź składnię, zrozum przykład i wróć do kodowania.

Witaj w swoim przewodniku! Podczas nauki programowania nie musisz pamiętać każdego polecenia i każdego znaku. Ważniejsze jest to, aby wiedzieć, gdzie znaleźć potrzebną informację i jak prawidłowo zastosować ją w kodzie. Ten przewodnik pomoże Ci szybko przypomnieć sobie najważniejsze konstrukcje języka Java.

Jak korzystać z przewodnika?

ZNAJDŹ

Wybierz temat, którego aktualnie potrzebujesz.

→

ZROZUM

Przeczytaj krótkie wyjaśnienie i przeanalizuj przykład.

→

ZASTOSUJ

Przepisz kod, uruchom go i sprawdź wynik.

```
1  int wynik = 5 + 3;
2  System.out.println(wynik);
```

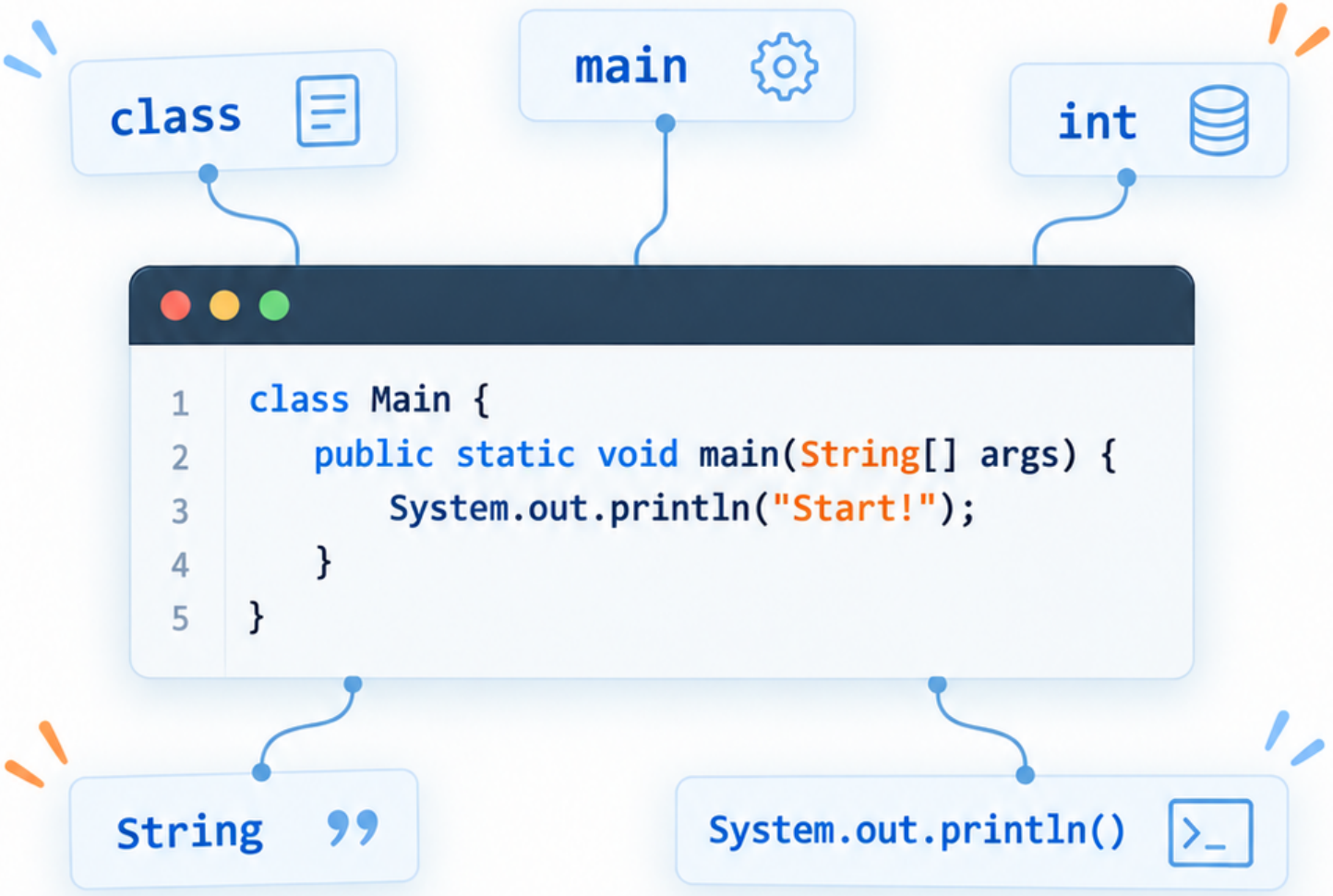
→ `Wynik: 8`

Nie musisz znać wszystkiego na pamięć. Ucz się krok po kroku i wracaj do przewodnika zawsze wtedy, gdy potrzebujesz szybkiej podpowiedzi.

MODUŁ 01

PODSTAWY SKŁADNI JAVY

Poznaj zasady zapisu kodu, typy danych i pierwsze instrukcje.



Każdy program zaczyna się od podstaw. W tym module poznasz najważniejsze elementy zapisu kodu w Javie.

PODSTAWOWY SZABLON PROGRAMU

JAK WYGLĄDA PROGRAM W JAVIE?

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Cześć, Javo!");
    }
}
```

1 **`public class Main`**
Definiuje klasę o nazwie `Main`.

2 **`public static void main(String[] args)`**
Deklaruje tradycyjną metodę startową programu.

3 **`System.out.println()`**
Wyświetla tekst lub wartość w konsoli.

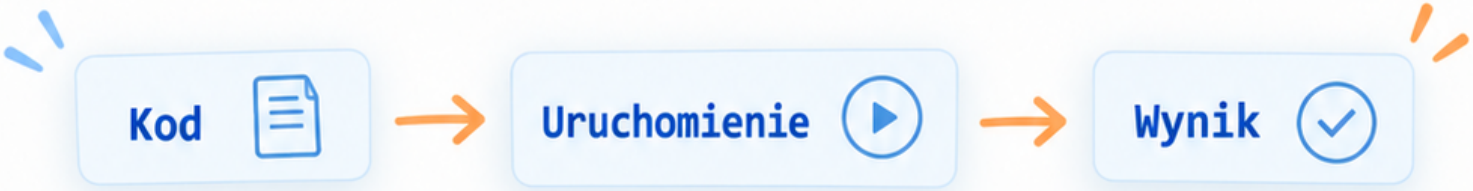
4 **`{ }`**
Wyznaczają początek i koniec bloku kodu.

WYNIK PROGRAMU

```
Cześć, Javo!
```

WAŻNE

Jeżeli publiczna klasa nazywa się `Main`, plik źródłowy powinien mieć nazwę `Main.java`.



BONUS #3

ZNAKI, KTÓRE MUSISZ ZNAĆ

;

ŚREDNIK – ;

Zwykle kończy pojedynczą instrukcję.

```
int liczba = 10;
```

{ }

NAWIASY KLAMROWE – { }

Wyznaczają blok kodu.

```
if (liczba > 5) {
    System.out.println("OK");
}
```

( )

NAWIASY OKRĄGŁE – ( )

Występują między innymi przy warunkach i wywołaniach metod.

```
System.out.println(liczba);
```

" "

CUDZYSŁÓW – " "

Otacza literał tekstowy.

```
String tekst = "Java";
```

Aa

WIELKOŚĆ LITER

Java rozróżnia wielkie i małe litery. Main i main to różne identyfikatory.

```
class Main { }
class main { }
```

BŁĄD I POPRAWNIE

BŁĄD

```
int liczba = 10
```

Brakuje średnika.

POPRAWNIE

```
int liczba = 10;
```

Instrukcja zakończona.

BONUS #3

JAK PISAĆ KOMENTARZE W KODZIE?

CO TO JEST KOMENTARZ?

Komentarz to fragment tekstu w kodzie źródłowym, który służy do wyjaśniania programu. Nie jest wykonywany jak zwykła instrukcja.

//

KOMENTARZ JEDNOLINIOWY

Stosuj `//`, aby opisać jedną linię.

Opis jednej linii kodu.

```
1  // Wiek użytkownika
2  int wiek = 20;
```

/* */

KOMENTARZ WIELOLINIOWY

Stosuj `/*...*/`, aby zapisać dłuższy komentarz.

Opis większego fragmentu kodu.

```
1  /* Obliczamy cenę
      po zastosowaniu
      rabatu. */
2  double cena = 100.0;
```

KOMENTARZ DOKUMENTACYJNY

Stosuj `/** ... */`, aby dokumentować klasy i metody.

Opis przeznaczony dla innych programistów.

```
1  /** Zwraca sumę dwóch
2      liczb. */
3  static int dodaj(int a, int b) {
4      return a + b;
5  }
```

Ostatni przykład reprezentuje metodę do umieszczenia wewnątrz klasy.

ZAPAMIĘTAJ

Dobry komentarz wyjaśnia cel kodu. Nie powinien jedynie powtarzać tego, co już widać w instrukcji.

BONUS #3

ZMIENNE — JAK PRZECHOWYWAĆ DANE?

CO TO JEST ZMIENNA?

Zmienna to nazwana przestrzeń przeznaczona do przechowywania wartości określonego typu.

Zmienna jak pudełko z etykietą!



DEKLARACJA ZMIENNEJ

```
int wiek = 18;
```

- `int` — Typ danych
- `wiek` — Nazwa zmiennej
- `18` — Wartość

PRZYKŁADY

```
1  int wiek = 18;
2  double cena = 29.99;
3  boolean aktywny = true;
```

ZMIANA WARTOŚCI

```
1  int punkty = 10;
2  punkty = 15;
3  System.out.println(punkty);
```

Deklaracja i pierwsza wartość

Aktualizacja wartości

▶ WYNIK: 15

WAŻNE ZASADY

- Nazwa zmiennej nie może zaczynać się od cyfry.
- Nie może zawierać spacji.
- Wielkość liter ma znaczenie.
- Przed odczytem zmienna lokalna musi mieć przypisaną wartość.

NAJCZĘSTSZY BŁĄD

✗ `int 2punkty = 10;`
Niepoprawna nazwa zmiennej.

✓ Poprawnie: `int punkty2 = 10;`

BONUS #3

8 TYPÓW PROSTYCH W JAVIE

CO TO SĄ TYPY PROSTE?

Typy proste określają podstawowe rodzaje danych, takie jak liczby, znaki i wartości logiczne.

Typ	Zastosowanie	Przykład wartości
byte	Małe liczby całkowite	100
short	Liczby całkowite	2000
int	Liczby całkowite	42
long	Duże liczby całkowite	123456789L
float	Liczby zmiennoprzecinkowe	3.14f
double	Liczby zmiennoprzecinkowe	3.14
char	Pojedynczy znak UTF-16	'A'
boolean	Prawda lub fałsz	true

Liczby całkowite · Liczby zmiennoprzecinkowe · Znaki · Wartości logiczne

PRZYKŁAD W KODZIE

```
1  int wiek = 25;
2  double wzrost = 1.75;
3  char ocena = 'A';
4  boolean gotowy = true;
```

TRZY WAŻNE ZASADY

1. `int` przechowuje liczby całkowite.
2. `double` jest często używany do liczb z częścią ułamkową.
3. `boolean` przyjmuje tylko `true` albo `false`.

NIE POMYL!

`char` — pojedynczy znak, np. `'A'`.

`String` — tekst, np. `"Anna"`. Nie jest typem prostym.

BONUS #3

KONWERSJA TYPÓW – JAK ZMIENIAĆ RODZAJ DANYCH?

CO TO JEST KONWERSJA TYPU?

Konwersja typu pozwala przedstawić wartość jednego typu jako wartość innego typu. W Javie niektóre konwersje odbywają się automatycznie, a inne wymagają jawnego wskazania typu.

AUTOMATYCZNA

int → double

```
1  int liczba = 25;
2  double wynik = liczba;
3  System.out.println(wynik);
```

```
Wynik: 25.0
```

Java automatycznie przekształca wartość `int` na `double`. Dla tej konkretnej liczby wartość zostaje zachowana.

Bez jawnego rzutowania

JAWNA

double → int

```
1  double cena = 19.99;
2  int wynik = (int) cena;
3  System.out.println(wynik);
```

```
Wynik: 19
```

Zapis `(int)` wymusza konwersję. Część ułamkowa zostaje odrzucona, a nie zaokrąglona.

Wymaga rzutowania

SCHEMAT SKŁADNI

```
typDocelowy zmienna = (typDocelowy) wartosc;
```

KIEDY UWAŻAĆ?

1. `int` → `double`: konwersja automatyczna; nie każda bardzo duża liczba zachowuje idealną precyzję w podobnych konwersjach.
2. `double` → `int`: wymagane rzutowanie; możliwa utrata części informacji.
3. `String` → `int`: wymaga parsowania tekstu, np. `Integer.parseInt("42")`, a nie zwykłego rzutowania.

ZAPAMIĘTAJ

Rzutowanie nie jest tym samym co matematyczne zaokrąglanie.

BONUS #3

Scanner – jak pobierać dane od użytkownika?

CO TO JEST SCANNER?

`Scanner` to klasa, która pozwala odczytywać dane, na przykład wpisane z klawiatury.

UŻYTKOWNIK WPISUJE DANE



`20`

Na przykład wpisuje z klawiatury liczbę 20.

SCANNER



`scanner.nextInt()`

Scanner odczytuje wpisane dane i zamienia je na odpowiedni typ.

ZMIENNA



`wiek = 20`

Odczytana wartość trafia do zmiennej.

1 IMPORTOWANIE KLASY

```
1  import java.util.Scanner;
```

2 UTWORZENIE OBIEKTU

```
1  Scanner scanner = new Scanner(System.in);
```

`System.in` oznacza standardowy strumień wejściowy.

3 ODCZYT LICZBY

```
1  System.out.print("Podaj wiek: ");
2  int wiek = scanner.nextInt();
```

4 KOMPLETNY PRZYKŁAD

```
1  import java.util.Scanner;
2  public class Main {
3    public static void main(String[] args) {
4      Scanner scanner = new Scanner(System.in);
5      System.out.print("Podaj wiek: ");
6      int wiek = scanner.nextInt();
7      System.out.println("Masz " + wiek + " lat.");
8      scanner.close();
9    }
10 }
```

5 PRZYKŁADOWE DZIAŁANIE

```
Podaj wiek: 20
Masz 20 lat.
```

6 NAJWAŻNIEJSZE METODY

Metoda	Co odczytuje?
`nextInt()`	Liczbę całkowitą
`nextDouble()`	Liczbę typu double
`next()`	Jeden wyraz
`nextLine()`	Całą linię tekstu
`nextBoolean()`	Wartość `true` lub `false`

WAŻNE Metoda `nextLine()` może odczytać pozostały znak końca linii po wcześniejszym użyciu `nextInt()`. Przy łączeniu tych metod zwróć uwagę na obsługę końca linii.

BONUS #3

PRINT, PRINTLN I PRINTF – POZNAJ RÓŻNICE

JAK WYŚWIETLAĆ INFORMACJE?

Java udostępnia różne metody do wyświetlania tekstu i wartości w konsoli.

System.out.print()

Wyświetla dane bez automatycznego przejścia do nowej linii.

```
1  System.out.print("Witaj ");
2  System.out.print("Javo!");
```

Wynik

```
Witaj Javo!
```

ta sama linia

System.out.println()

Wyświetla dane i przechodzi do nowej linii.

```
1  System.out.println("Pierwsza linia");
2  System.out.println("Druga linia");
```

Wynik

```
Pierwsza linia
Druga linia
```

nowa linia

System.out.printf()

Pozwala tworzyć sformatowany tekst.

```
1  int wiek = 21;
2  System.out.printf("Wiek: %d lat%n", wiek);
```

Wynik

```
Wiek: 21 lat
```

formatowany tekst

NAJCZĘŚCIEJ UŻYWANE ZNACZNIKI

Zapis	Znaczenie
%d	Liczba całkowita
%f	Liczba zmiennoprzecinkowa
%s	Tekst lub reprezentacja obiektu
%n	Nowa linia

ŁĄCZENIE TEKSTU Z WARTOŚCIĄ

```
1  String imie = "Anna";
2  int wiek = 18;
3  System.out.println(imie + " ma " + wiek + " lat.");
```

Wynik

```
Anna ma 18 lat.
```

SZYBKA ŚCIĄGA

`print()` → ta sama linia | `println()` → nowa linia | `printf()` → formatowany tekst

MODUŁ 02

OPERATORY I WYRAŻENIA

Wykonuj obliczenia, porównuj wartości i buduj wyrażenia logiczne.



Operatory pozwalają wykonywać działania na wartościach. Dzięki nim możesz liczyć, porównywać dane i tworzyć warunki sterujące działaniem programu.

W TYM MODUŁE POZNASZ:

- Operatory arytmetyczne.
- Operatory przypisania.
- Inkrementację i dekrementację.
- Operatory porównania.
- Operatory logiczne.
- Operator warunkowy i kolejność działań.

02

BONUS #3

OPERATORY ARYTMETYCZNE – LICZ W JAVIE

CO TO SĄ OPERATORY ARYTMETYCZNE?

Służą do wykonywania podstawowych obliczeń matematycznych.

+	-	*	/	%
Dodawanie	Odejmowanie	Mnożenie	Dzielenie	Reszta z dzielenia
5 + 3 → 8	5 - 3 → 2	5 * 3 → 15	6 / 3 → 2	7 % 3 → 1

PRZYKŁAD W KODZIE

```
1  int a = 10;
2  int b = 3;
3  System.out.println(a + b);
4  System.out.println(a - b);
5  System.out.println(a * b);
6  System.out.println(a / b);
7  System.out.println(a % b);
```

WYNIK PROGRAMU

```
13
7
30
3
1
```

DZIELENIE CAŁKOWITE

`7 / 2` → **3**

Ponieważ oba argumenty są typu `int`, Java wykonuje dzielenie całkowite.

DZIELENIE Z UŁAMKIEM

`7.0 / 2` → **3.5**

JAK DZIAŁA OPERATOR %?

`int reszta = 7 % 2;` → `Wynik: 1`

Reszta z dzielenia 7 przez 2 wynosi 1.

ZAPAMIĘTAJ

Dzielenie przez zero dla całkowitych typów liczbowych powoduje błąd podczas działania programu (`ArithmeticException`).

BONUS #3

OPERATORY PRZYPISANIA – ZAPISUJ KRÓCEJ

CO TO JEST OPERATOR PRZYPISANIA?

Operator przypisania pozwala zapisać wartość w zmiennej. Podstawowy operator to `=`.

1 ZWYKŁE PRZYPISANIE

```
1  int punkty = 10;
2  punkty = 20;
```

Po wykonaniu kodu zmienna **punkty** ma wartość 20.

2 PRZYPISANIE Z DZIAŁANIEM

```
punkty = punkty + 5;
```

→

```
punkty += 5;
```

3 TABELA OPERATORÓW

Operator	Przykład	Odpowiednik
=	x = 5	Przypisanie wartości 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

4 PRZYKŁAD PRAKTYCZNY

```
1  int saldo = 100;
2  saldo += 50;
3  saldo -= 20;
4  System.out.println(saldo);
```

Wynik: **130**

5 JAK ZMIENIA SIĘ WARTOŚĆ?

START 100 → +50 → 150 → −20 → 130

WAŻNE

Operatory `+=`, `-=`, `*=`, `/=` i `%=` aktualizują wartość zmiennej.
W przypadku liczb całkowitych operator `/=` również stosuje zasady dzielenia całkowitego.

BONUS #3

++ i -- — ZMIENIAJ WARTOŚĆ O JEDEN

CO OZNACZAJĄ TE OPERATORY?

`++` zwiększa wartość zmiennej liczbowej o 1.
`--` zmniejsza wartość zmiennej liczbowej o 1.

1 INKREMENTACJA

```
1  int licznik = 5;
2  licznik++;
3  System.out.println(licznik);
```

Wynik: 6

2 DEKREMENTACJA

```
1  int licznik = 5;
2  licznik--;
3  System.out.println(licznik);
```

Wynik: 4

PREFIKSOWO — ++x

Najpierw zwiększa wartość, potem zwraca nową wartość.

```
1  int x = 5;
2  int a = ++x;
3  System.out.println(a);
```

Najpierw zwiększ → 5 → 6 → Zwróć 6

Wynik: 6

POSTFIKSOWO — x++

Najpierw zwraca dotychczasową wartość, potem ją zwiększa.

```
1  int x = 5;
2  int a = x++;
3  System.out.println(a);
4  System.out.println(x);
```

Najpierw zwróć 5 → 5 → 6 → Nowa wartość: 6

Wynik:
5
6

SZYBKIE PORÓWNANIE

Zapis	Kiedy następuje zmiana?
`++x`	Przed uzyskaniem wartości wyrażenia
`x++`	Po uzyskaniu wartości wyrażenia
`--x`	Przed uzyskaniem wartości wyrażenia
`x--`	Po uzyskaniu wartości wyrażenia

ZAPAMIĘTAJ

Jeśli zapisujesz samo `x++;`, bez wykorzystywania wyniku wyrażenia, efekt końcowy jest taki sam jak przy `++x;`.

BONUS #3

OPERATORY PORÓWNANIA – PRAWDA CZY FAŁSZ?

CO TO SĄ OPERATORY PORÓWNANIA?

Pozwalają porównywać wartości i sprawdzać określone warunki. Wynikiem porównania jest wartość `true` albo `false`.

PODSTAWOWE OPERATORY

`==` `!=` `>`

`<` `>=` `<=`

OPERATORY W PRAKTYCE

Operator	Znaczenie	Przykład
==	Równe	5 == 5 → true
!=	Różne	5 != 3 → true
>	Większe	8 > 4 → true
<	Mniejsze	2 < 7 → true
>=	Większe lub równe	5 >= 5 → true
<=	Mniejsze lub równe	3 <= 2 → false

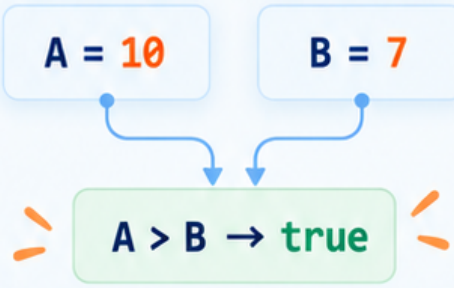
PRZYKŁAD W KODZIE

```
1  int a = 10;
2  int b = 7;
3  System.out.println(a > b);
4  System.out.println(a == b);
5  System.out.println(a != b);
```

WYNIK PROGRAMU

```
true
false
true
```

PRZYKŁAD W PRAKTYCE



PRZECHOWYWANIE WYNIKU PORÓWNANIA

```
1  int wiek = 20;
2  boolean pelnoletni = wiek >= 18;
3  System.out.println(pelnoletni);
```

WYNIK PROGRAMU

```
Wynik: true
```

= A == TO NIE TO SAMO!

`=` — przypisuje wartość.

`==` — sprawdza równość wartości lub referencji, zależnie od typów argumentów.

WAŻNE PRZY TEKSTACH

```
1  String a = "Java";
2  String b = "Java";
3  System.out.println(a.equals(b));
```

Wynik programu:

```
Wynik: true
```

Do porównywania zawartości obiektów `String` zazwyczaj używaj `.equals()`, a nie `==`.

BONUS #3

&&, || i ! – ŁĄCZ WARUNKI LOGICZNE

DO CZEGO SŁUŻĄ OPERATORY LOGICZNE?

Pozwalają połączyć kilka warunków w jedno wyrażenie logiczne lub odwrócić wynik warunku.

OPERATOR AND – &&

Wynik jest true tylko wtedy, gdy oba warunki są prawdziwe.

```
1  int wiek = 20;
2  boolean maBilet = true;
3  boolean wejscie = wiek >= 18 && maBilet;
```

Oba warunki

```
Wynik zmiennej wejscie:
true
```

OPERATOR OR – ||

Wynik jest true, gdy przynajmniej jeden warunek jest prawdziwy.

```
1  boolean maKarte = false;
2  boolean maGotowke = true;
3  boolean mozeZaplacic = maKarte || maGotowke;
```

Co najmniej jeden

```
Wynik zmiennej mozeZaplacic:
true
```

OPERATOR NOT – !

Odwraca wartość logiczną.

```
1  boolean aktywny = false;
2  System.out.println(!aktywny);
```

Zaprzeczenie

```
Wynik: true
```

TABELA PRAWDY

A	B	A && B	A || B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

PRAKTYCZNE ZASTOSOWANIE

```
1  int temperatura = 25;
2  boolean cieplo = temperatura > 20;
3  boolean nieGoraco = temperatura < 30;
4  boolean komfort = cieplo && nieGoraco;
```

```
Wynik zmiennej komfort: true
```

ZAPAMIĘTAJ

1. && — wszystkie warunki muszą być prawdziwe.
2. || — wystarczy jeden prawdziwy warunek.
3. ! — odwraca wynik logiczny.

Operatory && i || stosują ocenę skróconą: drugi argument nie jest sprawdzany, jeżeli pierwszy wystarcza do ustalenia wyniku.

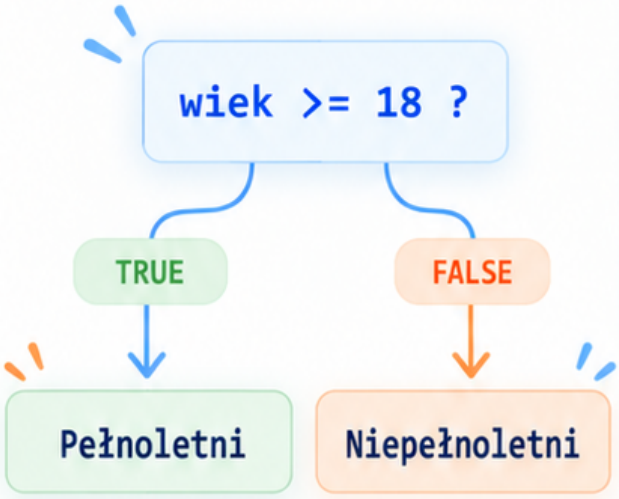
BONUS #3

OPERATOR ?: I KOLEJNOŚĆ WYKONYWANIA DZIAŁAŃ

CO TO JEST OPERATOR WARUNKOWY?

Operator `?` `:` pozwala wybrać jedną z dwóch wartości na podstawie warunku.

```
warunek ? wartoscGdyTrue : wartoscGdyFalse
```



PRZYKŁAD W KODZIE

```
1  int wiek = 20;
2  String status = wiek >= 18 ? "Pełnoletni" :
3                  "Niepełnoletni";
   System.out.println(status);
```

WYNIK PROGRAMU

```
Wynik: Pełnoletni
```

JAK TO DZIAŁA?

1. Java sprawdza warunek `wiek >= 18`.
2. Jeśli jest prawdziwy, wybiera `"Pełnoletni"`.
3. Jeśli jest fałszywy, wybiera `"Niepełnoletni"`.

KOLEJNOŚĆ WYKONYWANIA DZIAŁAŃ

Podobnie jak w matematyce, w Javie obowiązują zasady pierwszeństwa operatorów.

PRZYKŁAD A

```
2 + 3 * 4
```

↓

14

Mnożenie wykonuje się przed dodawaniem.

PRZYKŁAD B

```
(2 + 3) * 4
```

↓

20

Nawiasy zmieniają kolejność działań.

UPROSZCZONA TABELA PRIORYTETÓW

Priorytet	Operatory
1	`()` — grupowanie
2	`++`, `--`, `!`, znaki jednoargumentowe
3	`*`, `/`, `%`
4	`+`, `-`
5	`<`, `>`, `<=`, `>=`
6	`==`, `!=`
7	`&&`
8	`||`
9	`? :`
10	`=`, `+=`, `-=`, inne przypisania

Tabela uproszczona: obejmuje najważniejsze operatory poznane w tym module. Java ma również inne poziomy priorytetu, np. dla operatorów bitowych.

NAJWAŻNIEJSZA ZASADA

Gdy wyrażenie jest skomplikowane, użyj nawiasów, aby jasno pokazać, które działania mają być wykonane razem.

MODUŁ 03

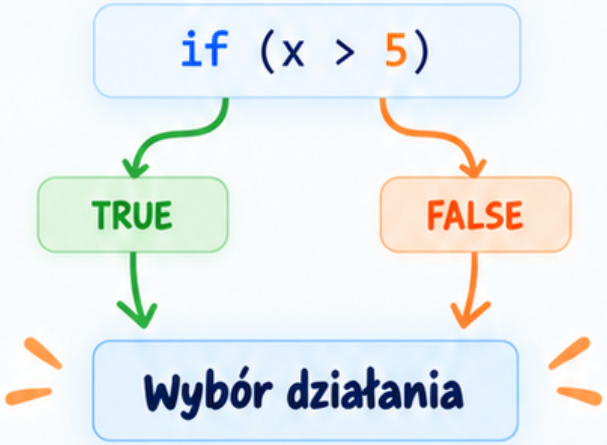
INSTRUKCJE WARUNKOWE I PĘTLE

Podejmuj decyzje w kodzie i automatyzuj powtarzalne działania.

JAK PROGRAM PODEJMUJE DECYZJE?

Instrukcje warunkowe pozwalają wykonywać różne fragmenty kodu w zależności od spełnienia określonych warunków. Pętle umożliwiają wielokrotne wykonywanie instrukcji bez konieczności powtarzania ich zapisu.

WARUNEK



PĘTLA



Warunek → Decyzja | Pętla → Powtarzanie

W TYM MODUŁE POZNASZ:

- `if` — sprawdzanie warunku.
- `else` i `else if` — wybieranie jednej z możliwości.
- `switch` — wybór spośród wielu przypadków.
- `for` — powtarzanie instrukcji z licznikiem.
- `while` — powtarzanie, dopóki warunek jest prawdziwy.
- `do-while` — wykonanie instrukcji przynajmniej raz.
- `break` i `continue` — sterowanie przebiegiem pętli.

03

BONUS #3

if – WYKONAJ KOD, GDY WARUNEK JEST SPEŁNIONY

CO TO JEST IF?

Instrukcja `if` sprawdza warunek logiczny. Jeżeli warunek ma wartość `true`, program wykonuje instrukcje znajdujące się w bloku `if`. Jeżeli warunek jest fałszywy, pomija ten blok.

PODSTAWOWA SKŁADNIA

```
1  if (warunek) {
2    // Kod wykonywany, gdy warunek jest prawdziwy
3  }
```

Warunek w nawiasach

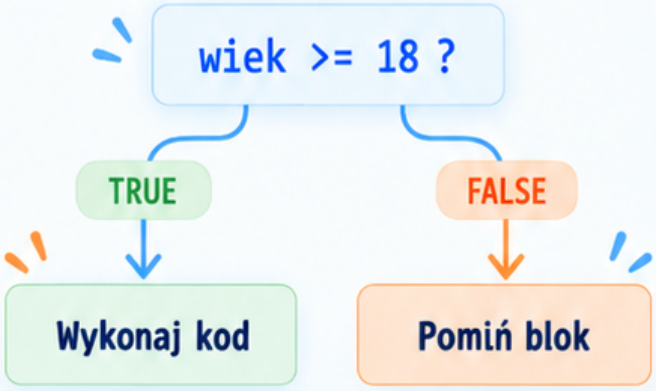
Blok kodu w klamrach

PRZYKŁAD PRAKTYCZNY

```
1  int wiek = 20;
2  if (wiek >= 18) {
3    System.out.println("Jesteś pełnoletni.");
4  }
```

```
Wynik: Jesteś pełnoletni.
```

SCHEMAT DZIAŁANIA



CO SIĘ DZIEJE KROK PO KROKU?

1. Program odczytuje wartość zmiennej `wiek`.
2. Sprawdza warunek `wiek >= 18`.
3. Otrzymuje wynik `true`.
4. Wykonuje instrukcję wewnątrz bloku.

GDY WARUNEK JEST FAŁSZYWY

```
1  int wiek = 15;
2  if (wiek >= 18) {
3    System.out.println("Dostęp dozwolony.");
4  }
```

Program nie wyświetla komunikatu, ponieważ warunek jest fałszywy.

KILKA WARUNKÓW JEDNOCZEŚNIE

```
1  int wiek = 25;
2  boolean maBilet = true;
3  if (wiek >= 18 && maBilet) {
4    System.out.println("Możesz wejść.");
5  }
```

```
Wynik: Możesz wejść.
```

ZAPAMIĘTAJ

Warunek w `if` musi być wyrażeniem typu `boolean`. Nie można użyć samej liczby zamiast warunku logicznego.

BONUS #3

if, else if i else – WYBIERAJ RÓŻNE ŚCIEŻKI

KIEDY UŻYWAĆ ELSE?

Jeżeli chcesz wykonać jeden fragment kodu, gdy warunek jest prawdziwy, a inny, gdy jest fałszywy, zastosuj `if` i `else`.

IF + ELSE

```java
int wiek = 16;
if (wiek >= 18) {
    System.out.println("Dorosły");
} else {
    System.out.println("Niepełnoletni");
}
```

```
Wynik: Niepełnoletni
```

KILKA MOŻLIWOŚCI – ELSE IF

```java
int wynik = 75;
if (wynik >= 90) {
    System.out.println("Świetnie!");
} else if (wynik >= 60) {
    System.out.println("Dobrze!");
} else {
    System.out.println("Ćwicz dalej!");
}
```

```
Wynik: Dobrze!
```

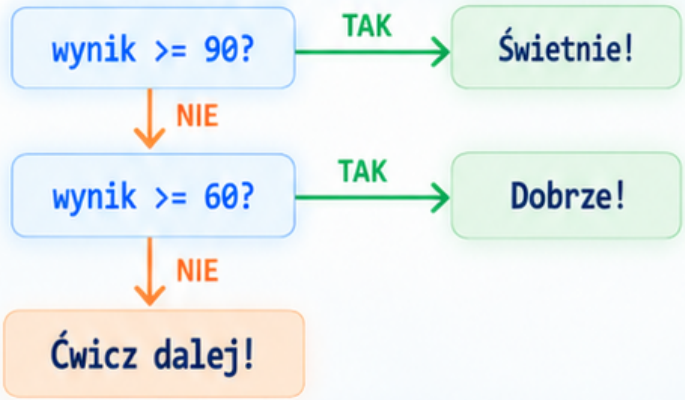
JAK DZIAŁA ŁAŃCUCH WARUNKÓW?

1. Najpierw sprawdzany jest warunek `if`.
2. Jeżeli jest fałszywy, program sprawdza kolejne `else if`.
3. Wykonywany jest pierwszy pasujący blok.
4. Jeżeli żaden warunek nie jest spełniony, wykonywany jest blok `else`, o ile istnieje.

WAŻNA KOLEJNOŚĆ

Warunki należy zapisywać we właściwej kolejności. W przykładzie z punktami warunek `wynik >= 90` musi znajdować się przed `wynik >= 60`. W przeciwnym razie program mógłby wybrać mniej odpowiedni komunikat.

SCHEMAT DZIAŁANIA



SZYBKA ŚCIĄGA

Instrukcja	Znaczenie
`if`	Sprawdza pierwszy warunek
`else if`	Sprawdza kolejny warunek
`else`	Obsługuje pozostałe przypadki

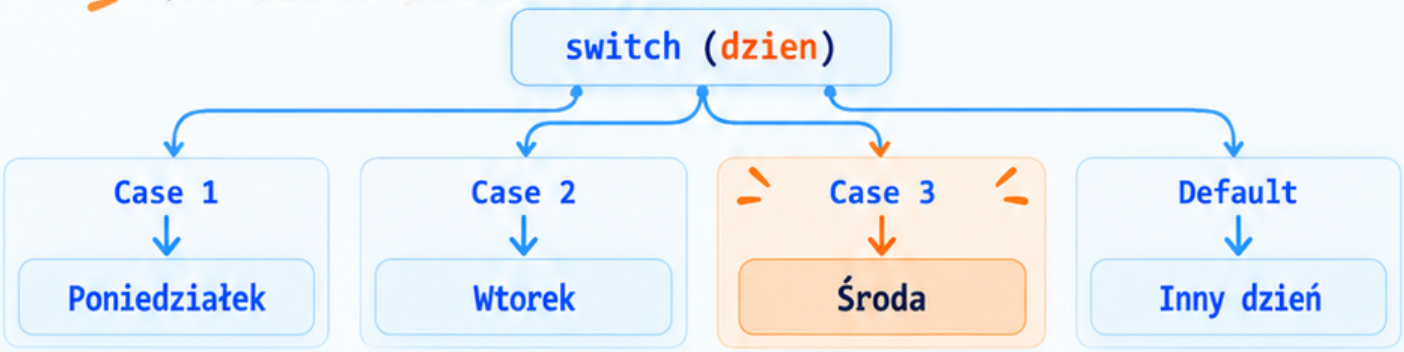
BONUS #3

switch – WYBIERAJ SPOŚRÓD WIELU PRZYPADKÓW

CO TO JEST SWITCH?

Instrukcja `switch` pozwala wybrać fragment kodu na podstawie wartości określonego wyrażenia. Jest szczególnie przydatna, gdy porównujesz jedną wartość z wieloma możliwymi przypadkami.

JAK DZIAŁA SWITCH?



PODSTAWOWA SKŁADNIA

```
switch (wartosc) {
    case 1:
        // Instrukcje
        break;
    default:
        // Przypadek domyślny
}
```

PRZYKŁAD – DZIEŃ TYGODNIA

```
int dzien = 3;
switch (dzien) {
   case 1:
       System.out.println("Poniedziałek");
       break;
   case 2:
       System.out.println("Wtorek");
       break;
   case 3:
       System.out.println("Środa");
       break;
   default:
       System.out.println("Inny dzień");
}
```

Wynik: Środa

NAJWAŻNIEJSZE ELEMENTY

switch – rozpoczyna wybór na podstawie wartości wyrażenia.

case – określa możliwy przypadek.

break – kończy wykonywanie instrukcji switch.

default – obsługuje sytuację, w której żaden przypadek nie pasuje.

NOWOCZESNY ZAPIS ZE STRZAŁKĄ

```
int dzien = 2;
switch (dzien) {
   case 1 -> System.out.println("Poniedziałek");
   case 1 -> System.out.println("Wtorek");
   default -> System.out.println("Inny dzień");
}
```

Wynik: Wtorek

Przy zapisie `case ... ->` nie stosuje się zwykłego przechodzenia do kolejnych przypadków.

CO SIĘ STANIE BEZ BREAK?

W tradycyjnej składni `switch` brak `break` może spowodować przejście do instrukcji kolejnych przypadków, bez ponownego sprawdzania ich etykiet.

BONUS #3

for — POWTARZAJ INSTRUKCJE Z LICZNIKIEM

CO TO JEST PĘTLA FOR?

Pętla for jest przydatna, gdy chcesz wielokrotnie wykonać kod i kontrolować liczbę powtórzeń.

PODSTAWOWA SKŁADNIA

```
for (inicjalizacja; warunek; aktualizacja) {
    // Powtarzany kod
}
```

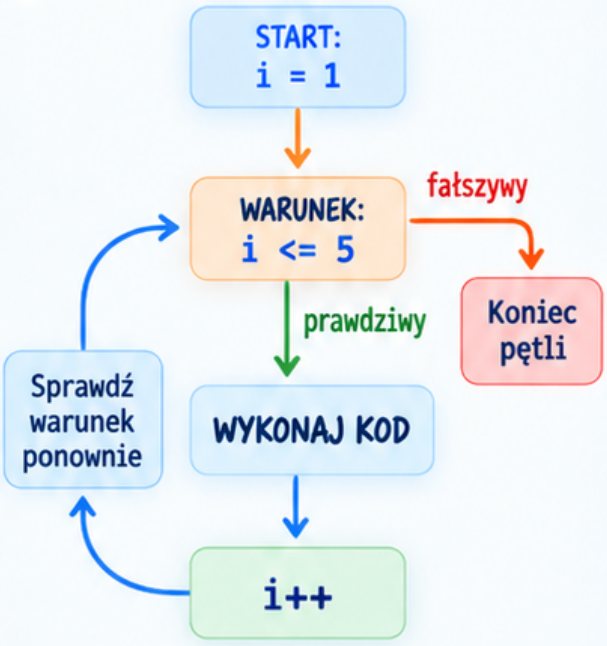
PRZYKŁAD — LICZENIE OD 1 DO 5

```
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

Wynik:

```
1
2
3
4
5
```

JAK DZIAŁA PĘTLA FOR?



TRZY CZĘŚCI PĘTLI

INICJALIZACJA

`int i = 1`

Ustawia początkową wartość licznika.

WARUNEK

`i <= 5`

Decyduje, czy pętla ma wykonać kolejną iterację.

AKTUALIZACJA

`i++`

Zwiększa licznik po wykonaniu bloku.

PRZYKŁAD — SUMOWANIE LICZB

```
int suma = 0;
for (int i = 1; i <= 5; i++) {
    suma += i;
}
System.out.println(suma);
```

Wynik:

15

ILE RAZY WYKONA SIĘ PĘTLA?

Iteracja	Wartość i
1	1
2	2
3	3
4	4
5	5

Po zwiększeniu licznika do 6 warunek `i <= 5` staje się fałszywy.

ZAPAMIĘTAJ

Pętla for składa się z inicjalizacji, warunku oraz aktualizacji licznika. Inicjalizacja odbywa się raz, a warunek jest sprawdzany przed każdą iteracją.

BONUS #3

while – POWTARZAJ KOD, DOPÓKI WARUNEK JEST PRAWDZIWY

CO TO JEST PĘTLA WHILE?

Pętla **while** wykonuje instrukcje tak długo, jak długo podany warunek ma wartość `true`. Warunek jest sprawdzany przed każdą iteracją.

PODSTAWOWA SKŁADNIA

```
1  while (warunek) {
2      // Powtarzany kod
3  }
```

JAK DZIAŁA WHILE?

1. Program sprawdza warunek `licznik <= 5`.
2. Jeśli jest prawdziwy, wykonuje kod.
3. Zwiększa wartość licznika.
4. Ponownie sprawdza warunek.
5. Kończy pętlę, gdy warunek staje się fałszywy.

PRZYKŁAD – LICZENIE DO 5

```
1  int licznik = 1;
2  while (licznik <= 5) {
3      System.out.println(licznik);
4      licznik++;
   }
```

Wynik programu:

```
1
2
3
4
5
```

SCHEMAT DZIAŁANIA PĘTLI WHILE



GDY WARUNEK JEST FAŁSZYWY OD POCZĄTKU

```
1  int licznik = 10;
2  while (licznik < 5) {
3      System.out.println(licznik);
   }
```

Wynik: Nic nie zostanie wyświetlone. Warunek jest fałszywy już przy pierwszym sprawdzeniu.

UWAGA NA PĘTLE NIESKOŃCZONE

```
1  int x = 1;
2  while (x <= 5) {
3      System.out.println(x);
   }
```

W tym przykładzie wartość `x` nie zmienia się, dlatego warunek pozostaje prawdziwy i program może wykonywać pętlę bez końca.

ZAPAMIĘTAJ

Pętla **while** może nie wykonać się ani razu. Zawsze upewnij się, że istnieje możliwość zakończenia pętli.

BONUS #3

do-while – WYKONAJ KOD PRZYNAJMNIEJ RAZ

CO TO JEST DO-WHILE?

Pętla **do-while** najpierw wykonuje blok kodu, a dopiero później sprawdza warunek. Dlatego wykona się co najmniej jeden raz.

PODSTAWOWA SKŁADNIA

```
do {
    // Wykonywany kod
} while (warunek);
```

PRZYKŁAD PRAKTYCZNY

```
int licznik = 1;
do {
  System.out.println(licznik);
  licznik++;
} while (licznik <= 3);
```

```
Wynik:
1
2
3
```

NAWET GDY WARUNEK JEST FAŁSZYWY

```
int liczba = 10;
do {
  System.out.println("Wykonano!");
} while (liczba < 5);
```

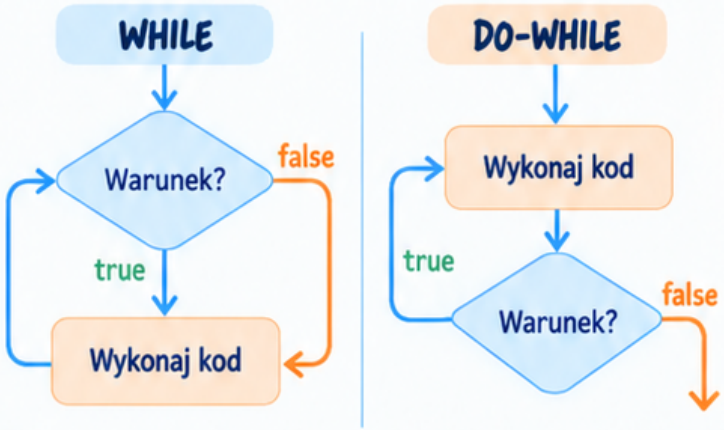
```
Wynik: Wykonano!
```

Warunek jest fałszywy, ale kod został wykonany raz.

WHILE VS DO-WHILE

while	do-while
Najpierw sprawdza warunek	Najpierw wykonuje kod
Może wykonać się 0 razy	Wykonuje się minimum 1 raz
Przydatna przy sprawdzaniu warunku przed działaniem	Przydatna przy menu i pytaniach powtarzanych użytkownikowi

PORÓWNANIE SCHEMATÓW DZIAŁANIA



TYPOWY PRZYKŁAD ZASTOSOWANIA

Menu programu, które powinno zostać wyświetlone przynajmniej raz, a następnie ponawiane do momentu wybrania opcji zakończenia.

WAŻNE

Po warunku `while` w konstrukcji **do-while** należy umieścić średnik:

```
while (warunek);
```

BONUS #3

BREAK i CONTINUE — KONTROLUJ PRZEBIEG PĘTLI

PO CO SĄ BREAK i CONTINUE?

Obie instrukcje pozwalają sterować wykonywaniem pętli. Działają jednak w różny sposób.

BREAK — ZATRZYMAJ PĘTLĘ

```
1  for (int i = 1; i <= 5; i++) {
2      if (i == 4) {
3          break;
4      }
5      System.out.println(i);
   }
```

```
Wynik: 1 2 3
```

Gdy i osiąga wartość 4, pętla zostaje zakończona.



CONTINUE — POMIŃ ITERACJĘ

```
1  for (int i = 1; i <= 5; i++) {
2      if (i == 3) {
3          continue;
4      }
5      System.out.println(i);
   }
```

```
Wynik: 1 2 4 5
```

Wartość 3 została pominięta, ale pętla działała dalej.



PORÓWNANIE

Instrukcja	Działanie
`break`	Kończy całą najbliższą pętlę
`continue`	Pomija resztę bieżącej iteracji
`return`	Kończy wykonywanie metody, w której występuje

PRZYKŁAD PRAKTYCZNY — POMIJANIE LICZB PARZYSTYCH

```
1  for (int i = 1; i <= 6; i++) {
2      if (i % 2 == 0) {
3          continue;
4      }
5      System.out.println(i);
   }
```

```
Wynik: 1 3 5
```

KIEDY STOSOWAĆ?

`break` — Gdy znaleziono szukaną wartość i nie trzeba wykonywać kolejnych iteracji.

`continue` — Gdy chcesz pominąć określony przypadek, ale kontynuować przetwarzanie pozostałych danych.

ZAPAMIĘTAJ

`break` kończy pętlę, a `continue` przechodzi do kolejnej iteracji. W pętli `for` po `continue` następuje jeszcze aktualizacja licznika przed kolejnym sprawdzeniem warunku.

MODUŁ 04

TABLICE, TEKST I KOLEKCJE

Przechowuj wiele danych, pracuj z tekstem i organizuj informacje.

DLACZEGO WARTO POZNAĆ TE STRUKTURY?

Zamiast tworzyć osobną zmienną dla każdej wartości, możesz przechowywać wiele danych w uporządkowanych strukturach. Java udostępnia również narzędzia do przetwarzania tekstu i wygodnego zarządzania zbiorami danych.

TABLICA

10 | 20 | 30

`int[] liczby`

STRING

"Java"

`String jezyk`

```
1  int[] liczby = {10, 20, 30};
2  String jezyk = "Java";
```

ARRAYLIST

A | B

`List<String> lista`

HASHMAP

klucz → wartość

`Map<String, Integer> mapa`

W TYM MODUŁE POZNASZ:

1. **TABLICE** `int[]`
   Przechowywanie wielu wartości tego samego typu.
2. **NAPISY** `String`
   Tworzenie i przetwarzanie tekstu.
3. **STRINGBUILDER**
   Efektywne budowanie i modyfikowanie tekstu.
4. **ARRAYLIST**
   Listy, których rozmiar może się zmieniać.
5. **HASHMAP**
   Przechowywanie danych w parach klucz–wartość.
6. **HASHSET**
   Przechowywanie unikalnych elementów.

PRZYKŁAD

```
1  int[] liczby = {10, 20, 30};
2  String jezyk = "Java";
```

Dobra organizacja danych ułatwia pisanie przejrzystych i praktycznych programów.

04

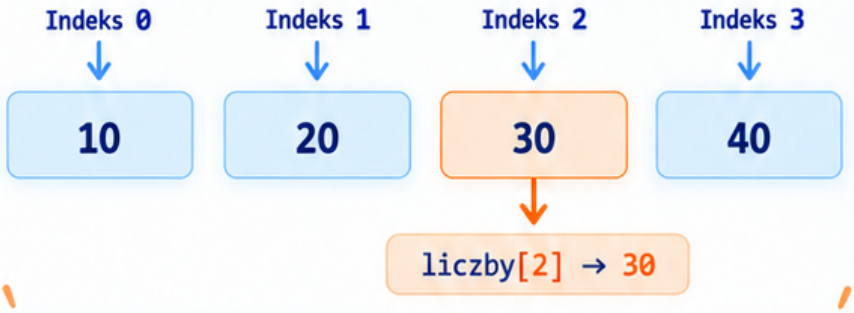
BONUS #3

TABLICE W JAVIE — WIELE WARTOŚCI W JEDNEJ STRUKTURZE

CO TO JEST TABLICA?

Tablica pozwala przechowywać wiele wartości określonego typu pod jedną nazwą.
Każdy element ma swój numer, nazywany indeksem.

```
int[] liczby = {10, 20, 30, 40};
```



`length` → Liczba elementów

`for-each` → Przeglądanie elementów

1 TWORZENIE TABLICY

```
1  int[] liczby = {10, 20, 30, 40};
```

Tablica zawiera cztery liczby całkowite.

2 INDEKSY ZACZYNAJĄ SIĘ OD ZERA

Indeks:	0	1	2	3
Wartość:	10	20	30	40

3 ODCZYTYWANIE ELEMENTU

```
1  int[] liczby = {10, 20, 30, 40};
2  System.out.println(liczby[0]);
3  System.out.println(liczby[2]);
```

```
Wynik:
10
30
```

4 ZMIANA ELEMENTU

```
1  int[] liczby = {10, 20, 30};
2  liczby[1] = 99;
3  System.out.println(liczby[1]);
```

```
Wynik:
99
```

5 SPRAWDZANIE DŁUGOŚCI TABLICY

```
1  int[] liczby = {10, 20, 30};
2  System.out.println(liczby.length);
```

```
Wynik:
3
```

6 TWORZENIE TABLICY O OKREŚLONYM ROZMIARZE

```
1  int[] wyniki = new int[5];
```

Powstaje tablica pięciu elementów typu int.
Początkowo każdy element ma wartość 0.

7 PRZECHODZENIE PRZEZ ELEMENTY — FOR-EACH

```
1  int[] liczby = {10, 20, 30};
2  for (int liczba : liczby) {
3      System.out.println(liczba);
4  }
```

```
Wynik:
10
20
30
```

8 NAJCZĘSTSZE BŁĘDY

Błąd 1 — nieprawidłowy indeks

```
1  int[] liczby = {10, 20, 30};
2  System.out.println(liczby[3]);
```

Ten kod spowoduje `ArrayIndexOutOfBoundsException`, ponieważ poprawne indeksy to `0`, `1` i `2`.

Błąd 2 — długość tablicy

- `liczby.length` — poprawnie.
- `liczby.length()` — niepoprawnie dla tablic.

ZAPAMIĘTAJ

Tablice mają stałą długość po utworzeniu. Jeżeli potrzebujesz struktury, której rozmiar można wygodnie zmieniać, poznaj `ArrayList`.

BONUS #3

STRING – JAK PRZECHOWYWAĆ TEKST W JAVIE?

CO TO JEST STRING?

String to klasa służąca do reprezentowania tekstu w Javie.
Obiekt typu String może zawierać litery, cyfry, spacje i inne znaki.

1 TWORZENIE NAPISU

```
1   String imie = "Anna";
```

Tekst zapisujemy w podwójnych cudzysłowach.

2 ŁĄCZENIE TEKSTÓW

"Cześć, " + "Anna" → "Cześć, Anna"

```
1   String tekst = "Cześć, " + "Anna";
```

```
Wynik:
Cześć, Anna
```

3 ŁĄCZENIE TEKSTU Z LICZBĄ

```
1   String komunikat = "Wiek: " + 20;
```

```
Wynik:
Wiek: 20
```

4 STRING A CHAR

Typ	Przykład	Opis
char	'A'	Pojedyncza jednostka kodowa UTF-16
String	"Anna"	Ciąg znaków

5 PORÓWNYWANIE TEKSTÓW

```
1   String a = "Java";
2   String b = "Java";
3   System.out.println(a.equals(b));
```

```
Wynik:
true
```

Do porównywania zawartości dwóch napisów używaj `.equals()`.

6 CZY STRING MOŻNA ZMIENIAĆ?

```
1   String tekst = "Java";
2   tekst = tekst + " 21";
3   System.out.println(tekst);
```

```
Wynik:
Java 21
```

Obiekty String są niemutowalne.
Oznacza to, że ich zawartość nie może zostać zmieniona po utworzeniu.
Operacje na napisach mogą tworzyć nowe obiekty. Zmienna tekst wskazuje teraz na nowy napis.

ZAPAMIĘTAJ

String jest klasą, a nie typem prostym. Zawartość napisów porównuj za pomocą `.equals()`, nie operatora `==`.

BONUS #3

STRING — NAJWAŻNIEJSZE METODY W PRAKTYCE

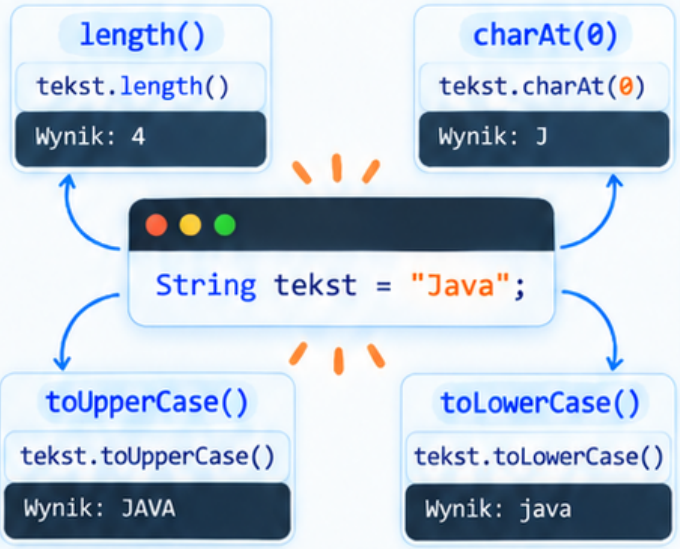
JAK PRACOWAĆ Z TEKSTEM?

Klasa `String` oferuje wiele metod, które pomagają analizować i przetwarzać tekst.

1 NAJWAŻNIEJSZE METODY

Metoda	Opis
`length()`	Zwraca długość napisu
`charAt(i)`	Pobiera jednostkę kodową z podanej pozycji
`toUpperCase()`	Zamienia litery na wielkie
`toLowerCase()`	Zamienia litery na małe
`contains()`	Sprawdza, czy tekst zawiera fragment
`startsWith()`	Sprawdza początek napisu
`endsWith()`	Sprawdza koniec napisu
`substring()`	Pobiera fragment napisu
`replace()`	Zastępuje wybrane fragmenty
`trim()`	Usuwa wybrane białe znaki z początku i końca

PODSTAWOWE PRZYKŁADY



2 DŁUGOŚĆ NAPISU

```
1  tekst.length();
```

```
Wynik: 4
```

3 ZAMIANA WIELKOŚCI LITER

```
1  tekst.toUpperCase();
2  tekst.toLowerCase();
```

```
Wynik:
JAVA
java
```

4 POBIERANIE ZNAKU

```
1  tekst.charAt(0);
```

```
Wynik: J
```

5 POBIERANIE FRAGMENTU TEKSTU

```
1  String napis = "Programowanie";
2  System.out.println(napis.substring(0, 7));
```

```
Wynik:
Program
```

Pierwszy indeks jest uwzględniany, a drugi wyznacza koniec fragmentu i nie jest uwzględniany.

6 SPRAWDZANIE ZAWARTOŚCI

```
1  String napis = "Programowanie w Javie";
2  System.out.println(napis.contains("Java"));
```

```
Wynik:
true
```

7 ZASTĘPOWANIE TEKSTU

```
1  String zdanie = "Ala ma kota";
2  System.out.println(zdanie.replace("kota", "psa"));
```

```
Wynik:
Ala ma psa
```

WAŻNE

Większość metod przekształcających `String` zwraca nowy napis. Oryginalny obiekt pozostaje niezmieniony.
W metodzie `substring()` indeksy odnoszą się do jednostek kodowych UTF-16.

BONUS #3

STRINGBUILDER – BUDUJ I MODYFIKUJ TEKST

CO TO JEST STRINGBUILDER?

StringBuilder to klasa, która pozwala wygodnie modyfikować zawartość tekstu bez konieczności tworzenia nowego obiektu przy każdej zmianie. Jest szczególnie przydatna, gdy wielokrotnie dodajesz lub zmieniasz fragmenty tekstu.

1 TWORZENIE OBIEKTU

```
1  StringBuilder sb =
   new StringBuilder("Java");
```

2 DODAWANIE TEKSTU – APPEND

```
1  sb.append(" jest");
```

```
Wynik:
Java jest
```

3 DODAWANIE W OKREŚLONYM MIEJSCU – INSERT

```
1  StringBuilder sb2 =
   new StringBuilder("Jva");
2  sb2.insert(1, "a");
```

```
Wynik:
Java
```

4 USUWANIE FRAGMENTU – DELETE

```
1  StringBuilder sb2 =
   new StringBuilder("Java123");
2  sb3.delete(4, 7);
```

```
Wynik:
Java
```

JAK ROŚNIE STRINGBUILDER?

"Java"

↓

append(" jest")

↓

"Java jest"

↓

append(" świetna!")

↓

"Java jest świetna!"

Ten sam obiekt StringBuilder rośnie – nie tworzymy nowych obiektów przy każdej zmianie!

5 ODWRACANIE KOLEJNOŚCI – REVERSE

```
1  StringBuilder sb4 =
   new StringBuilder("Java");
2  sb4.reverse();
```

```
Wynik:
avaJ
```

6 PORÓWNANIE STRING I STRINGBUILDER

String	StringBuilder
Niemutowalny	Mutowalny
Dobry do zwykłego przechowywania tekstu	Dobry do częstych zmian tekstu
Przekształcenia mogą tworzyć nowe obiekty	Metody modyfikują istniejący obiekt

7 KONWERSJA DO STRING

```
1  String tekst = sb.toString();
```

ZAPAMIĘTAJ

StringBuilder jest wygodnym narzędziem do łączenia tekstu w pętlach i budowania dłuższych komunikatów. Nie zapewnia jednak synchronizacji przy równoczesnej modyfikacji przez wiele wątków.

BONUS #3

ARRAYLIST — LISTA, KTÓRA MOŻE ROSNĄĆ

CO TO JEST ARRAYLIST?

ArrayList to klasa reprezentująca listę elementów. W odróżnieniu od zwykłej tablicy jej rozmiar może zmieniać się podczas działania programu.

1 IMPORTOWANIE KLASY

```
1  import java.util.ArrayList;
```

2 TWORZENIE LISTY

```
1  ArrayList<String> imiona =
   new ArrayList<>();
```

Lista będzie przechowywać napisy typu String.

3 DODAWANIE ELEMENTÓW — ADD

```
1  imiona.add("Anna");
2  imiona.add("Jan");
3  imiona.add("Ola");
```

```
Wynik:
[Anna, Jan, Ola]
```

4 ODCZYTYWANIE ELEMENTU — GET

```
1  System.out.println(imiona.get(1));
```

```
Wynik:
Jan
```

Indeksy listy zaczynają się od zera.

5 ZMIANA ELEMENTU — SET

```
1  imiona.set(1, "Piotr");
```

Lista zawiera teraz:
[Anna, Piotr, Ola]

6 USUWANIE ELEMENTU — REMOVE

```
1  imiona.remove(0);
```

Lista zawiera teraz:
[Piotr, Ola]

7 LICZBA ELEMENTÓW — SIZE

```
1  System.out.println(imiona.size());
```

```
Wynik:
2
```

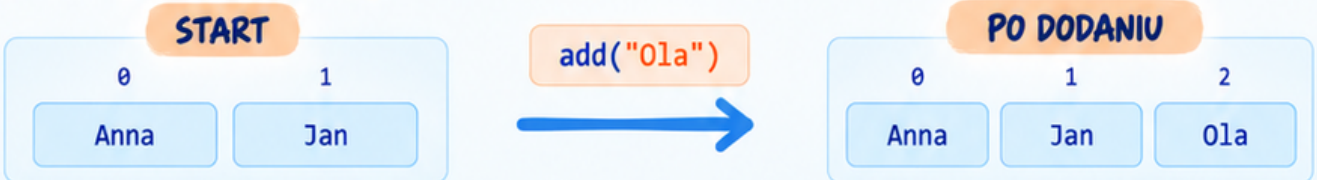
8 NAJWAŻNIEJSZE METODY

Metoda	Opis
`add(x)`	Dodaje element
`get(i)`	Odczytuje element
`set(i, x)`	Zmienia element
`remove(i)`	Usuwa element o indeksie i
`size()`	Zwraca liczbę elementów
`contains(x)`	Sprawdza obecność elementu
`clear()`	Usuwa wszystkie elementy

9 ARRAYLIST Z LICZBAMI

```
1  ArrayList<Integer> liczby =
   new ArrayList<>();
2  liczby.add(10);
3  liczby.add(20);
```

JAK ROŚNIE LISTA?



PORÓWNANIE

`int[]`
długość stała, odczytywana przez `.length`.

`ArrayList<Integer>`
rozmiar zmienny, odczytywany przez `.size()`.

PAMIĘTAJ

W typie generycznym używamy `Integer`, a nie `int`, ponieważ ArrayList przechowuje obiekty, a nie bezpośrednio wartości typów prostych.

HASHMAP – PRZECHOWUJ DANE W PARACH KLUCZ–WARTOŚĆ

CO TO JEST HASHMAP?

HashMap to klasa przechowująca dane w postaci par: KLUCZ → WARTOŚĆ.
Każdy klucz jest unikalny i pozwala odnaleźć przypisaną do niego wartość.



1 IMPORTOWANIE KLASY

```
1  import java.util.HashMap;
```

2 TWORZENIE MAPY

```
1  HashMap<String, Integer> wiek =
   new HashMap<>();
```

Kluczem będzie String, a wartością Integer.

3 DODAWANIE DANYCH – PUT

```
1  wiek.put("Anna", 20);
2  wiek.put("Jan", 25);
3  wiek.put("Ola", 18);
```

4 ODCZYTYWANIE WARTOŚCI – GET

```
1  System.out.println(wiek.get("Jan"));
```

```
Wynik:
25
```

5 AKTUALIZOWANIE WARTOŚCI

```
1  wiek.put("Anna", 21);
```

Gdy **klucz już istnieje**, put() zastępuje przypisaną do niego wartość.

6 SPRAWDZANIE ISTNIENIA KLUCZA

```
1  System.out.println(wiek.containsKey("Ola"));
```

```
Wynik:
true
```

7 USUWANIE ELEMENTU

```
1  wiek.remove("Ola");
```

Para z kluczem "Ola" zostaje usunięta.

9 BEZPIECZNE POBIERANIE DANYCH

```
1  System.out.println(wiek.getOrDefault
   ("Marek", 0));
```

```
Wynik:
0
```

8 NAJWAŻNIEJSZE METODY

Metoda	Opis
`put(k, v)`	Dodaje lub zastępuje wartość
`get(k)`	Pobiera wartość
`containsKey(k)`	Sprawdza obecność klucza
`remove(k)`	Usuwa wpis
`size()`	Zwraca liczbę wpisów
`keySet()`	Udostępnia zbiór kluczy
`getOrDefault(k, v)`	Pobiera wartość albo wartość domyślną

ZAPAMIĘTAJ

HashMap nie gwarantuje kolejności przechowywanych elementów.
Jeśli klucz nie istnieje, zwykłe `get()` zwraca `null`.

BONUS #3

HASHSET – PRZECHOWUJ UNIKALNE ELEMENTY

CO TO JEST HASHSET?

HashSet to klasa reprezentująca zbiór unikalnych elementów. Jeżeli spróbujesz dodać element, który już istnieje w zbiorze, druga kopia nie zostanie dodana.

ARRAYLIST



Duplikaty dozwolone

HASHSET



Tylko unikalne elementy

1 IMPORTOWANIE KLASY

```
1  import java.util.HashSet;
```

2 TWORZENIE ZBIORU

```
1  HashSet<String> zbior =
   new HashSet<>();
```

3 DODAWANIE ELEMENTÓW

```
1  zbior.add("A");
2  zbior.add("B");
3  zbior.add("A");
4  zbior.add("C");
```

Zbiór zawiera tylko trzy unikalne elementy.

4 SPRAWDZANIE LICZBY ELEMENTÓW

```
1  System.out.println(zbior.size());
```

```
Wynik:
3
```

5 SPRAWDZANIE OBECNOŚCI ELEMENTU

```
1  System.out.println(zbior.contains("B"));
```

```
Wynik:
true
```

6 USUWANIE ELEMENTU

```
1  zbior.remove("C");
```

Element zostaje usunięty ze zbioru.

7 PRZEGLĄDANIE ELEMENTÓW

```
1  for (String element : zbior) {
2      System.out.println(element);
3  }
```

Pętla wypisuje wszystkie elementy, ale nie należy zakładać konkretnej kolejności.

8 NAJWAŻNIEJSZE METODY

`add(x)`	Dodaje element, jeśli go nie ma
`remove(x)`	Usuwa element
`contains(x)`	Sprawdza obecność
`size()`	Zwraca liczbę elementów
`isEmpty()`	Sprawdza, czy zbiór jest pusty
`clear()`	Usuwa wszystkie elementy

9 ARRAYLIST VS HASHSET

ArrayList	HashSet
Może zawierać duplikaty	Nie przechowuje duplikatów
Zachowuje kolejność listy	Nie gwarantuje kolejności
Udostępnia elementy przez indeks	Nie udostępnia odczytu przez indeks

ZAPAMIĘTAJ

Wybierz HashSet, gdy najważniejsza jest unikalność elementów i sprawdzanie ich obecności, a nie pozycja na liście.

MODUŁ 05

METODY, KLASY I OBIEKTY

Organizuj kod w metody i poznaj podstawy programowania obiektowego.

DLACZEGO WARTO ORGANIZOWAĆ KOD?

Wraz z rozwojem programu liczba instrukcji rośnie. Metody pomagają dzielić kod na mniejsze części, a klasy pozwalają organizować dane i działania w logiczne struktury.

POLA

Dane obiektu

`np. imie, wiek`

KLASA

Definiuje dane i zachowania



METODY

Działania obiektu

`np. powitaj(), oblicz()`

OBIEKT

Instancja klasy



`np. nowa Osoba()`

W TYM MODUŁE POZNASZ:

1. **METODY**
   Tworzenie fragmentów kodu, które można wielokrotnie wywoływać.
2. **PARAMETRY I RETURN**
   Przekazywanie informacji do metod i zwracanie wyników.
3. **KLASY I OBIEKTY**
   Tworzenie własnych typów oraz ich instancji.
4. **KONSTRUKTORY**
   Przygotowywanie obiektów podczas ich tworzenia.
5. **ENKAPSULACJA**
   Kontrolowanie dostępu do danych.
6. **DZIEDZICZENIE**
   Tworzenie klas na podstawie innych klas.
7. **INTERFEJSY**
   Definiowanie wspólnych kontraktów zachowania.

PRZYKŁAD

```
1  class Osoba {
2      String imie;
3      void powitaj() {
4          System.out.println("Cześć!");
5      }
   }
```

Dobrze zorganizowany kod jest łatwiejszy do czytania, rozwijania i utrzymywania.

05

BONUS #3

METODY – TWÓRZ KOD, KTÓREGO MOŻESZ UŻYWAĆ PONOWNIE

CO TO JEST METODA?

Metoda to nazwany fragment kodu, który wykonuje określone zadanie. Zamiast wielokrotnie pisać te same instrukcje, możesz umieścić je w metodzie i wywoływać ją w różnych miejscach programu.

1 PODSTAWOWA SKŁADNIA

```
1  static void powitaj() {
2      System.out.println("Cześć!");
3  }
```

2 WYWOŁYWANIE METODY

```
1  powitaj();
```

Ta instrukcja uruchamia kod znajdujacy się w metodzie `powitaj()`.

3 KOMPLETNY PROGRAM

```
1  public class Main {
2      static void powitaj() {
3          System.out.println("Cześć!");
4      }
5      public static void main(String[] args) {
6          powitaj();
8      }
7  }
```

```
Wynik:
Cześć!
```

ZAPAMIĘTAJ

Metody w Javie deklaruje się wewnątrz klas lub innych dozwolonych struktur typów, a nie bezpośrednio wewnątrz innych metod.

CO OZNACZAJĄ ELEMENTY?

`static`	metoda należy do klasy i może być wywołana bez tworzenia obiektu
`void`	metoda nie zwraca wartości
`powitaj`	nazwa metody
`()`	miejsce na parametry metody
`{ }`	blok instrukcji wykonywanych przez metodę

4 METODA MOŻE WYKONYWAĆ WIELE INSTRUKCJI

```
1  static void powitaj() {
2      System.out.println("Cześć!");
3      System.out.println("Miło Cię widzieć.");
4      System.out.println("Zaczynajmy!");
   }
```

Metoda wykonuje trzy instrukcje w kolejności ich zapisania.

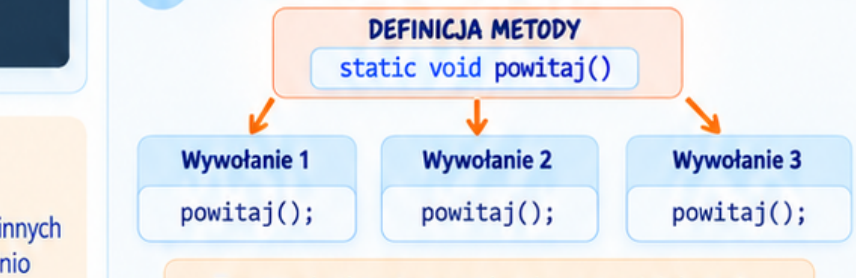
5 DLACZEGO WARTO STOSOWAĆ METODY?

MNIEJ POWTÓRZEŃ
Nie trzeba wielokrotnie zapisywać tego samego fragmentu kodu.

LEPSZA ORGANIZACJA
Program można podzielić na mniejsze, czytelne części.

ŁATWIEJSZE POPRAWKI
Zmianę można wprowadzić w jednym miejscu.

6 JEDNA DEFINICJA – WIELE UŻYĆ



Ten sam kod może zostać wykonany wielokrotnie.

BONUS #3

PARAMETRY I RETURN — PRZEKAZUJ DANE I ZWRACAJ WYNIKI

JAK METODA OTRZYMUJE DANE?

Parametry pozwalają przekazywać wartości do metody. Dzięki nim ta sama metoda może pracować z różnymi danymi.

1 METODA Z PARAMETREM

```
1  static void powitaj(String imie) {
2      System.out.println("Cześć, " + imie);
3  }
```

WYWOŁANIE

```
powitaj("Anna");
```

WYNIK:

```
Cześć, Anna
```

2 METODA Z DWOMA PARAMETRAMI

```
1  static int dodaj(int a, int b) {
2      return a + b;
3  }
```

WYWOŁANIE

```
System.out.println(dodaj(3, 5));
```

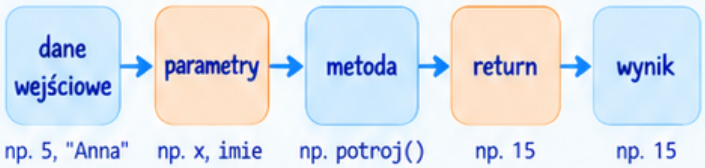
WYNIK:

```
8
```

3 CO TO JEST RETURN?

Instrukcja `return` pozwala zakończyć metodę i, jeśli metoda deklaruuje typ wyniku, zwrócić wartość do miejsca wywołania.

JAK PŁYNĄ DANE?



4 METODA ZWRACAJĄCA LICZBĘ

```
1  static int potroj(int x) {
2      return x * 3;
3  }
```

WYWOŁANIE

```
System.out.println(potroj(5));
```

WYNIK:

```
15
```

PORÓWNANIE

parametry	przekazują dane do metody
return	zwraca wynik z metody

5 KOMPLETNY PRZYKŁAD

```
1  public class Main {
2      static void powitaj(String imie) {
3          System.out.println("Cześć, " + imie);
4      }
5      static int dodaj(int a, int b) {
7          return a + b;
8      }
9      public static void main(String[] args) {
10         powitaj("Anna");
11         int wynik = dodaj(7, 8);
13         System.out.println(wynik);
14     }
15 }
```

WYNIK:

```
Cześć, Anna
15
```

ZAPAMIĘTAJ

Parametry rozszerzają możliwości metod, a **return** pozwala wykorzystać obliczony wynik w innym miejscu programu. Metoda typu **void** nie zwraca wartości.

BONUS #3

KONSTRUKTORY – PRZYGOTUJ OBIEKT PODCZAS TWORZENIA

CO TO JEST KONSTRUKTOR?

Konstruktor to specjalna konstrukcja, która jest wywoływana podczas tworzenia obiektu za pomocą operatora **new**. Służy do przypisania początkowych wartości i przygotowania obiektu do użycia.

1 PODSTAWOWA SKŁADNIA

```
1  class Osoba {
2      String imie;
3      Osoba(String noweImie) {
4          imie = noweImie;
5      }
   }
```

2 TWORZENIE OBIEKTU

```
1  Osoba osoba = new Osoba("Anna");
```

Wartość "Anna" zostaje przekazana do konstruktora podczas tworzenia obiektu.

3 UŻYCIE SŁOWA THIS

Słowo this pozwala odróżnić pola obiektu od parametrów konstruktora, które mają te same nazwy.

```
1  class Osoba {
2      String imie;
3      int wiek;
4      Osoba(String imie, int wiek) {
5          this.imie = imie;
6          this.wiek = wiek;
6      }
   }
```

CO OZNACZA this.imie = imie?

- **this.imie** – pole obiektu
- **imie** – parametr konstruktora
- **=** – przypisuje przekazaną wartość do pola

JAK TO DZIAŁA?

```
new Osoba("Anna", 20)
```

↓

KONSTRUKTOR

Osoba(String imie, int wiek)

```
this.imie = imie;
this.wiek = wiek;
```

↓

UTWORZONY OBIEKT



imie	wiek
"Anna"	20

4 KOMPLETNY PRZYKŁAD

```
1  class Osoba {
2      String imie;
3      int wiek;
4      Osoba(String imie, int wiek) {
5          this.imie = imie;
6          this.wiek = wiek;
7      }
8      void pokazDane() {
9          System.out.println(imie + ", " + wiek);
10     }
11 }
12
13 public class Main {
14     public static void main(String[] args) {
15         Osoba osoba = new Osoba("Anna", 20);
16         osoba.pokazDane();
17     }
18 }
```

```
Wynik:
Anna, 20
```

5 WAŻNE ZASADY

Element	Znaczenie
Osoba(...)	Konstruktor klasy Osoba
new	Tworzy obiekt
this	Odwołuje się do bieżącego obiektu
Parametry	Dostarczają wartości początkowych

6 KONSTRUKTOR DOMYŚLNY

Jeśli nie zadeklarujesz żadnego konstruktora, kompilator automatycznie dostarczy konstruktor domyślny bez parametrów. Jeśli jednak zadeklarujesz własny konstruktor, Java nie doda już automatycznie konstruktora domyślnego.

ZAPAMIĘTAJ

Konstruktor ma taką samą nazwę jak klasa i nie deklaruje typu zwracanego – nawet void.

BONUS #3

ENKAPSULACJA – CHROŃ DANE SWOICH OBIEKTÓW

CO TO JEST ENKAPSULACJA?

Enkapsulacja to mechanizm, który pozwala kontrolować dostęp do danych obiektów. Zamiast pozwalać na dowolne zmiany pól, możesz ukryć je za pomocą modyfikatora `private` i udostępniać odpowiednie metody do bezpiecznej pracy z danymi.

1 POLE PRYWATNE – PRIVATE

```
class Konto {
    private int saldo = 0;
}
```

Pola `saldo` nie można bezpośrednio odczytać ani zmienić zwykłym kodem spoza klasy.

2 GETTER – ODCZYT WARTOŚCI

```
public int getSaldo() {
    return saldo;
}
```

3 METODA DO ZMIANY DANYCH

```
public void wplac(int kwota) {
    if (kwota > 0) {
        saldo += kwota;
    }
}
```

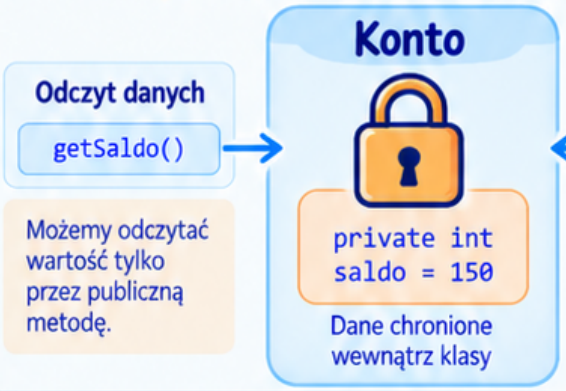
Akceptowane są tylko dodatnie kwoty. Dzięki temu unikamy nieprawidłowych zmian danych.

4 KOMPLETNY PRZYKŁAD

```
class Konto {
    private int saldo = 0;

    public void wplac(int kwota) {
        if (kwota > 0) {
            saldo += kwota;
        }
    }

    public int getSaldo() {
        return saldo;
    }
}

public class Main {
    public static void main(String[] args) {
        Konto konto = new Konto();
        konto.wplac(100);
        konto.wplac(50);
        System.out.println(konto.getSaldo());
    }
}
```

```
Wynik:
150
```

OBIEKT POD OCHRONĄ



5 DLACZEGO NIE ZMIENIAĆ POLA BEZPOŚREDNIO?

```
konto.saldo = -100;
```

Taka instrukcja nie jest dozwolona w klasie Main, ponieważ pole `saldo` jest prywatne (`private`). Dostęp do danych powinien odbywać się przez publiczne metody.

6 GETTER I SETTER

Element	Znaczenie
Getter	Odczytywanie wartości
Setter	Ustawianie lub aktualizacja wartości
private	Ograniczanie bezpośredniego dostępu do pól
public	Udostępnianie wybranych operacji

Nie każda klasa powinna udostępniać setter. Czasami lepiej jest zapewnić specjalną metodę (np. `wplac()`), która kontroluje poprawność zmian.

ZAPAMIĘTAJ

Enkapsulacja to nie tylko używanie modyfikatora `private`, ale także projektowanie bezpiecznych zasad dostępu do stanu obiektu.

BONUS #3

DZIEDZICZENIE — WYKORZYSTUJ WSPÓLNE CECHY KLAS

CO TO JEST DZIEDZICZENIE?

Dziedziczenie to tworzenie nowej klasy na podstawie już istniejącej, Klasa potomna może korzystać z odziedziczonych metod i pól, a także dodawać własne zachowanie.

1 SŁOWO KLUCZOWE EXTENDS

```
1  class Zwierze {
2    void jedz() {
3      System.out.println("Zwierzę je.");
4    }
5  }
6
7  class Pies extends Zwierze {
8  }
```

Klasa Pies dziedziczy po klasie Zwierze.

2 KORZYSTANIE Z METODY

```
1  Pies pies = new Pies();
2  pies.jedz();
```

```
Wynik:
Zwierzę je.
```

3 NADPISYWANIE METODY — OVERRIDE

```
1  class Zwierze {
2    void dajGlos() {
3      System.out.println("Dźwięk
               zwierzęcia");
4    }
5  }
6
7  class Pies extends Zwierze {
8    @Override
9    void dajGlos() {
10     System.out.println("Hau!");
11   }
12 }
```

HIERARCHIA KLAS

ZWIERZE

```
void dajGlos() {
  System.out.println(
   "Dźwięk zwierzęcia");
}
```

dziedziczy

PIES EXTENDS ZWIERZE

```
void dajGlos() {
  System.out.println(
   "Hau!");
}
```

@Override
Tutaj specjalizujemy zachowanie metody.

4 KOMPLETNY PRZYKŁAD

```
1  class Zwierze {
2    void dajGlos() {
3      System.out.println("Dźwięk
               zwierzęcia");
4    }
5  }
6
7  class Pies extends Zwierze {
8    @Override
9    void dajGlos() {
10     System.out.println("Hau!");
11   }
12 }
13
14 public class Main {
15   public static void main(String[] args) {
16     Pies pies = new Pies();
17     pies.dajGlos();
18   }
19 }
```

```
Wynik:
Hau!
```

5 NAJWAŻNIEJSZE ELEMENTY

Element	Opis
extends	Określa dziedziczenie klasy
Klasa bazowa	Klasa, po której dziedziczymy
Klasa potomna	Klasa rozszerzająca inną
@Override	Oznacza zamiar nadpisania metody
super	Pozwala odwoływać się do wybranych elementów klasy bazowej

6 WAŻNE OGRANICZENIE

- W Javie klasa może bezpośrednio dziedziczyć tylko po jednej klasie.
- Klasa może implementować wiele interfejsów.
- Prywatne członki klasy bazowej nie są bezpośrednio dostępne w kodzie klasy potomnej.

ZAPAMIĘTAJ

Dziedziczenie opisuje relację ***„jest-a"*** (is-a). Na przykład: `Pies jest rodzajem Zwierze.`

BONUS #3

INTERFEJSY – DEFINIUJ WSPÓLNE ZASADY DZIAŁANIA

CO TO JEST INTERFEJS?

Interfejs to zbiór **zasad**, które musi spełniać klasa. Może deklarować metody, które będą dostępne w klasach implementujących ten interfejs. Interfejs określa, co klasa ma potrafić, a nie jak to ma robić.

1 TWORZENIE INTERFEJSU

```
1   interface Powitanie {
2       void powitaj();
3   }
```

Metoda zadeklarowana w ten sposób jest domyślnie **publiczna** i **abstrakcyjna**.

2 IMPLEMENTOWANIE INTERFEJSU

```
1   class Uczen implements Powitanie {
2       @Override
3       public void powitaj() {
4           System.out.println("Cześć!");
5       }
6   }
```

Słowo **implements** oznacza, że klasa realizuje kontrakt określony przez interfejs.

3 KOMPLETNY PRZYKŁAD

```
1   interface Powitanie {
2       void powitaj();
3   }
4
5   class Uczen implements Powitanie {
6       @Override
7       public void powitaj() {
8           System.out.println("Cześć!");
9       }
10  }
11
12  public class Main {
13      public static void main(String[] args) {
14          Uczen uczen = new Uczen();
15          uczen.powitaj();
16      }
17  }
```

```
Wynik:
Cześć!
```

4 JEDEN INTERFEJS – RÓŻNE KLASY

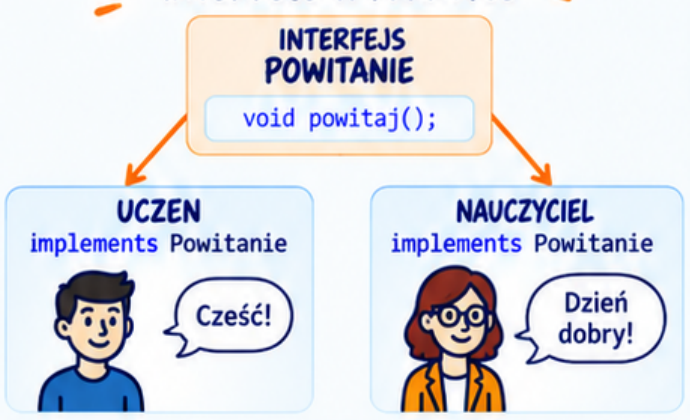
```
1   class Nauczyciel implements Powitanie {
2       @Override
3       public void powitaj() {
4           System.out.println("Dzień dobry!");
5       }
6   }
```

Zarówno klasa `Uczen`, jak i `Nauczyciel` mogą implementować ten sam interfejs, ale wykonywać metodę w różny sposób.

5 EXTENDS VS IMPLEMENTS

extends	implements
Klasa rozszerza inną klasę	Klasa implementuje interfejs
Pozwala korzystać z dziedziczenia	Zobowiązuje do realizacji kontraktu
Jedna bezpośrednia klasa bazowa	Możliwość implementacji wielu interfejsów

INTERFEJS W PRAKTYCE



6 WAŻNE ZASADY

- Interfejs definiujemy za pomocą słowa **interface**.
- Klasa implementuje interfejs za pomocą słowa **implements**.
- Metody implementujące publiczne metody interfejsu muszą być publiczne.
- Interfejsy mogą również zawierać metody domyślne (default) oraz statyczne (**static**).

ZAPAMIĘTAJ

Interfejsy pomagają projektować klasy, które dzielą wspólne zachowania, nawet gdy ich wewnętrzna struktura jest różna.

MODUŁ 06

ORGANIZACJA KODU, BŁĘDY I SZYBKA POWTÓRKA

Poznaj zasady organizowania projektów, obsługi wyjątków i szybkiego sprawdzania składni.

OD SKŁADNI DO DOBREGO KODU

Znajomość podstawowych instrukcji to pierwszy krok, ale równie ważne jest odpowiednie organizowanie klas, korzystanie z pakietów oraz rozpoznawanie najczęstszych błędów, które mogą pojawić się podczas pisania i uruchamiania kodu.

W TYM MODULE POZNASZ:

1. **PACKAGE I IMPORT**
   Organizowanie klas oraz korzystanie z klas dostępnych w innych pakietach.
2. **MODYFIKATORY DOSTĘPU**
   Kontrolowanie widoczności klas, pól i metod.
3. **TRY, CATCH I FINALLY**
   Obsługiwanie wyjątków podczas działania programu.
4. **NAJCZĘSTSZE BŁĘDY**
   Rozpoznawanie typowych problemów składniowych i logicznych.
5. **SZYBKA ŚCIĄGA JAVY**
   Najważniejsze konstrukcje w jednym miejscu.

NIEZBĘDNE NARZĘDZIA



Dobry programista nie musi pamiętać wszystkiego.
Ważne, aby potrafił szybko znaleźć rozwiązanie i zrozumieć działanie kodu.

06

BONUS #3

PACKAGE I IMPORT — ORGANIZUJ KLASY W PROJEKCIE

DO CZEGO SŁUŻĄ PAKIETY?

Pakiety grupują powiązane klasy i inne typy, dzięki czemu duże projekty pozostają czytelne i dobrze zorganizowane. Ułatwiają zarządzanie kodem i zapobiegają konfliktom nazw.

1 DEKLARACJA PAKIETU

```
1  package mojaaplikacja;
```

Deklaracja pakietu powinna znajdować się na początku pliku źródłowego, przed importami i deklaracjami typów.

2 PRZYKŁADOWA KLASA

Plik Osoba.java

```
1  package mojaaplikacja;
2  public class Osoba {
3      public String imie = "Anna";
   }
```

3 IMPORTOWANIE KLAS

```
1  import java.util.ArrayList;
```

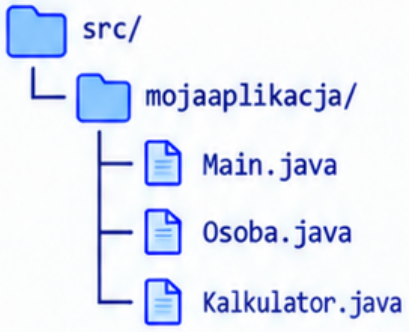
Import pozwala używać krótkich nazw typów z innych pakietów.

4 PRZYKŁAD Z ARRAYLIST

```
1  import java.util.ArrayList;
2
3  public class Main {
4      public static void main(String[] args) {
5          ArrayList<String> lista =
               new ArrayList<>();
6          lista.add("Java");
6          System.out.println(lista);
7      }
   }
```

```
Wynik:
[Java]
```

6 PRZYKŁADOWA STRUKTURA PROJEKTU



Te pliki mogą należeć do pakietu `mojaaplikacja`.

5 IMPORTOWANIE WSZYSTKICH TYPÓW Z PAKIETU

```
1  import java.util.*;
```

Ten import nie importuje automatycznie typów z podpakietów.

PACKAGE = ORGANIZACJA

Określa	Określa pakiet bieżącego pliku
Pomaga	Pomaga organizować projekt
Zwykle	Zwykle jedna deklaracja pakietu

IMPORT = UŻYCIE TYPÓW

Pozwala	Pozwala używać krótkich nazw typów
Ułatwia	Ułatwia korzystanie z innych pakietów
Można	Można używać wielu importów

ZAPAMIĘTAJ

Klasy z pakietu `java.lang`, takie jak `String` i `System`, są dostępne bez konieczności jawnego importowania.

BONUS #3

PUBLIC, PRIVATE I PROTECTED – KONTROLUJ DOSTĘP DO KODU

CO TO SĄ MODYFIKATORY DOSTĘPU?

Modyfikatory dostępu określają, z jakich miejsc można korzystać z klas oraz ich członków (pól, metod, konstruktorów). Pomagają chronić dane i definiować publiczną powierzchnię programu.

1 PUBLIC – DOSTĘP PUBLICZNY

Szeroki dostęp

```
1  public void pokazInfo() {
2      System.out.println(
           "Informacja");
3  }
```

Metoda publiczna może być wywołana z innych klas, gdy obiekt lub typ jest dostępny, a żadne dodatkowe ograniczenia modułów nie mają zastosowania.

3 PROTECTED – DOSTĘP CHRONIONY

Pakiet i podklasy

```
1  protected String nazwa;
```

Członek oznaczony jako protected jest dostępny w tej samej paczce oraz, zgodnie z zasadami dziedziczenia, w podklasach znajdujących się w innych paczkach.

4 BRAK MODYFIKATORA – PACKAGE-PRIVATE

Ten sam pakiet

```
1  int licznik;
```

Członek bez modyfikatora dostępu (package-private) jest dostępny tylko w tej samej paczce.

2 PRIVATE – DOSTĘP PRYWATNY

Dostęp w obrębie klasy

```
1  private int saldo;
```

Członek prywatny jest przeznaczony do użytku wyłącznie wewnątrz klasy, w której został zdefiniowany.

5 TABELA WIDOCZNOŚCI

Modyfikator	Ta sama klasa	Ten sam pakiet	Podklasa w innym pakiecie	Inna klasa w innym pakiecie
`public`	Tak	Tak	Tak	Tak
`protected`	Tak	Tak	Tak, według zasad dziedziczenia	Nie
`Brak`	Tak	Tak	Nie	Nie
`private`	Tak	Nie*	Nie	Nie

*Tabela uproszczona dla typowych, niezagnieżdżonych klas. Java dopuszcza również szczególne reguły dostępu między typami zagnieżdżonymi.

6 PRZYKŁAD PRAKTYCZNY

```
1   class Produkt {
2       private double cena;
3
4       public void setCena(double nowaCena) {
5           if (nowaCena >= 0) {
6               cena = nowaCena;
7           }
8       }
9
10      public double getCena() {
11          return cena;
12      }
13  }
```

7 WAŻNE ROZRÓŻNIENIE

`static` Nie jest modyfikatorem dostępu. Służy do powiązania pola lub metody z klasą, a nie z konkretną instancją obiektu.

`final` Nie jest modyfikatorem dostępu. Ogranicza możliwość ponownego przypisania wartości, przesłonięcia metody lub dziedziczenia klasy – w zależności od kontekstu.

ZAPAMIĘTAJ

Dobrą praktyką jest udostępnianie tylko tych elementów, które naprawdę powinny być dostępne z zewnątrz.

BONUS #3

TRY, CATCH I FINALLY – REAGUJ NA BŁĘDY PROGRAMU

CO TO JEST WYJĄTEK?

Wyjątek to sygnał informujący o problemie lub nietypowej sytuacji, która wystąpiła podczas wykonywania programu. Nieobsłużony wyjątek może przerwać bieżące zadanie programu.

1 BLOK TRY

```
try {
    int wynik = 10 / 0;
}
```

Dzielenie liczb całkowitych przez zero powoduje wyjątek `ArithmeticException`.

2 BLOK CATCH

```
try {
    int wynik = 10 / 0;
    System.out.println(wynik);
} catch (ArithmeticException e) {
    System.out.println("Nie można dzielić przez zero.");
}
```

```
Wynik:
Nie można dzielić przez zero.
```

3 BLOK FINALLY

Blok `finally` wykonuje kod do sprzątania lub zamykania zasobów, niezależnie od tego, czy wystąpił obsłużony wyjątek.
Istnieją jednak sytuacje (np. nagłe zakończenie programu), w których blok `finally` może się nie wykonać.

JAK DZIAŁA TRY-CATCH-FINALLY?



4 KOMPLETNY PRZYKŁAD

```
public class Main {
    public static void main(String[] args) {
        try {
            int wynik = 10 / 0;
            System.out.println(wynik);
        } catch (ArithmeticException e) {
            System.out.println("Błąd dzielenia!");
        } finally {
            System.out.println("Koniec operacji.");
        }
    }
}
```

```
Wynik:
Błąd dzielenia!
Koniec operacji.
```

5 NAJWAŻNIEJSZE ELEMENTY

`try`	Zawiera kod, w którym może wystąpić wyjątek
`catch`	Obsługuje określone typy wyjątków
`finally`	Zawiera działania końcowe
`throw`	Jawnie zgłasza wyjątek
`throws`	Deklaruje wyjątki, które metoda może przekazać dalej

6 INNE PRZYKŁADY WYJĄTKÓW

`NumberFormatException`	Niepoprawna próba konwersji tekstu na liczbę
`NullPointerException`	Próba wykonania określonej operacji przez referencję null
`ArrayIndexOutOfBoundsException`	Odwołanie do elementu tablicy o nieprawidłowym indeksie

7 DOBRA PRAKTYKA

Nie ignoruj wyjątków bez uzasadnienia.
Usuwaj przyczyny problemów, sprawdzaj poprawność danych wejściowych i obsługuj tylko te przypadki, na które program naprawdę może zareagować.

ZAPAMIĘTAJ

Nie każdy błąd w programie jest wyjątkiem. Błędy składniowe wykrywane przez kompilator należy poprawić w kodzie, a nie obsługiwać za pomocą `try-catch`.

BONUS #3

NAJCZĘSTSZE BŁĘDY W JAVIE – ZNAJDŹ I POPRAW

DLACZEGO WARTO ROZPOZNAWAĆ BŁĘDY?

Błędy są naturalną częścią nauki programowania. Czytanie komunikatów kompilatora i dokładne sprawdzanie składni pozwala szybciej znaleźć ich przyczyny i skuteczniej je naprawiać.

1 BRAKUJĄCY ŚREDNIK

BŁĄD

```
1  int wiek = 20
```

POPRAWNIE

```
1  int wiek = 20;
```

Większość prostych instrukcji w Javie kończy się średnikiem.

2 NIEZGODNY TYP DANYCH

BŁĄD

```
1  int liczba = "10";
```

POPRAWNIE

```
1  int liczba = 10;
```

Tekst "10" to nie to samo co wartość liczbowa 10. Należy używać odpowiednich typów danych.

3 NIEPOPRAWNA WIELKOŚĆ LITER

BŁĄD

```
1  system.out.println("Java");
```

POPRAWNIE

```
1  System.out.println("Java");
```

Java rozróżnia wielkość liter. Nazwy klas, metod i zmiennych muszą być zapisane dokładnie tak, jak w dokumentacji.

4 MYLENIE = I ==

BŁĄD

```
1  int x = 5;
2  if (x = 5) {
3      System.out.println("OK");
4  }
```

POPRAWNIE

```
1  int x = 5;
2  if (x == 5) {
3      System.out.println("OK");
4  }
```

Operator = przypisuje wartość, a == sprawdza, czy wartości są równe.

5 NIEPOPRAWNY INDEKS TABLICY

BŁĄD

```
1  int[] liczby = {1, 2, 3};
2  System.out.println(liczby[3]);
```

POPRAWNIE

```
1  int[] liczby = {1, 2, 3};
2  System.out.println(liczby[2]);
```

Indeksy tablic zaczynają się od zera. Użycie nieistniejącego indeksu powoduje wyjątek podczas wykonania programu.

6 PORÓWNYWANIE STRINGÓW

BŁĄD

```
1  String a = "Java";
2  String b = "Java";
3  boolean rowne = a == b;
```

POPRAWNIE

```
1  String a = "Java";
2  String b = "Java";
3  boolean rowne = a.equals(b);
```

Operator == porównuje referencje do obiektów, a metoda `.equals()` porównuje zawartość stringów.

7 TRZY RODZAJE PROBLEMÓW

Rodzaj błędu	Opis
Błąd kompilacji	Kod nie spełnia reguł języka.
Błąd wykonania	Problem pojawia się podczas działania programu.
Błąd logiczny	Program działa, ale daje niewłaściwy wynik.

8 LISTA KONTROLNA

- [x] Czy nie brakuje średników?
- [x] Czy nawiasy zostały zamknięte?
- [x] Czy nazwy zmiennych są poprawne?
- [x] Czy typy danych są zgodne?
- [x] Czy warunki logiczne mają typ boolean?
- [x] Czy nie odwołujesz się do nieistniejącego indeksu?
- [x] Czy nie pomyliłeś = z ==?

ZAPAMIĘTAJ

Komunikat o błędzie często wskazuje miejsce problemu, ale rzeczywista przyczyna może znajdować się także w poprzedniej linii kodu.

JAVA – NAJWAŻNIEJSZA SKŁADNIA W JEDNYM MIEJSCU

Twoja podręczna ściąga do szybkiej powtórki.

1 ZMIENNE I TYPY

```java
int liczba = 10;
double cena = 19.99;
boolean aktywny = true;
String tekst = "Java";
```

2 WYŚWIETLANIE DANYCH

```java
System.out.println("Hello!");
System.out.print("Java");
System.out.printf("Liczba: %d%n", 10);
```

3 WARUNKI

```java
if (x > 5) {
    // Kod
} else {
    // Inny kod
}
```

4 PĘTLA FOR

```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

5 PĘTLA WHILE

```java
while (x > 0) {
    x--;
}
```

6 TABLICE

```java
int[] liczby = {1, 2, 3};
System.out.println(liczby[0]);
```

7 ARRAYLIST

```java
ArrayList<String> lista = new ArrayList<>();
lista.add("Java");
```

Wymagany import: `java.util.ArrayList`.

8 METODY

```java
static int dodaj(int a, int b) {
    return a + b;
}
```

TWOJA SZYBKA ŚCIĄGA JAVY

9 KLASY I OBIEKTY

```java
class Osoba {
    String imie;
}
Osoba osoba = new Osoba();
```

Tworzenie obiektu (wiersz 4) to przykład użycia poza ciałem klasy.

10 OBSŁUGA WYJĄTKÓW

```java
try {
    // Kod
} catch (Exception e) {
    // Obsługa wyjątku
}
```

Fragmenty należy umieszczać w odpowiednich klasach i metodach, gdy jest to potrzebne.

11 NAJWAŻNIEJSZE SYMBOLE

Symbol	Znaczenie
;	Koniec instrukcji
{ }	Blok kodu
( )	Warunki i wywołania
[ ]	Tablice i indeksy
=	Przypisanie
==	Porównanie
&&	Logiczne AND
||	Logiczne OR
!	Zaprzeczenie
++	Zwiększenie o jeden

12 PIĘĆ ZŁOTYCH ZASAD JAVY

1. Pamiętaj o wielkości liter.
2. Sprawdzaj średniki i nawiasy.
3. Używaj odpowiednich typów danych.
4. Czytaj komunikaty błędów.
5. Pisz czytelny kod i testuj jego działanie.

NA ZAKOŃCZENIE

Nie musisz znać wszystkiego na pamięć. Najważniejsze, aby rozumieć podstawy, umieć czytać kod i wiedzieć, gdzie szukać potrzebnej składni. Wracaj do tego przewodnika podczas nauki i tworzenia własnych projektów. **Powodzenia w programowaniu!**