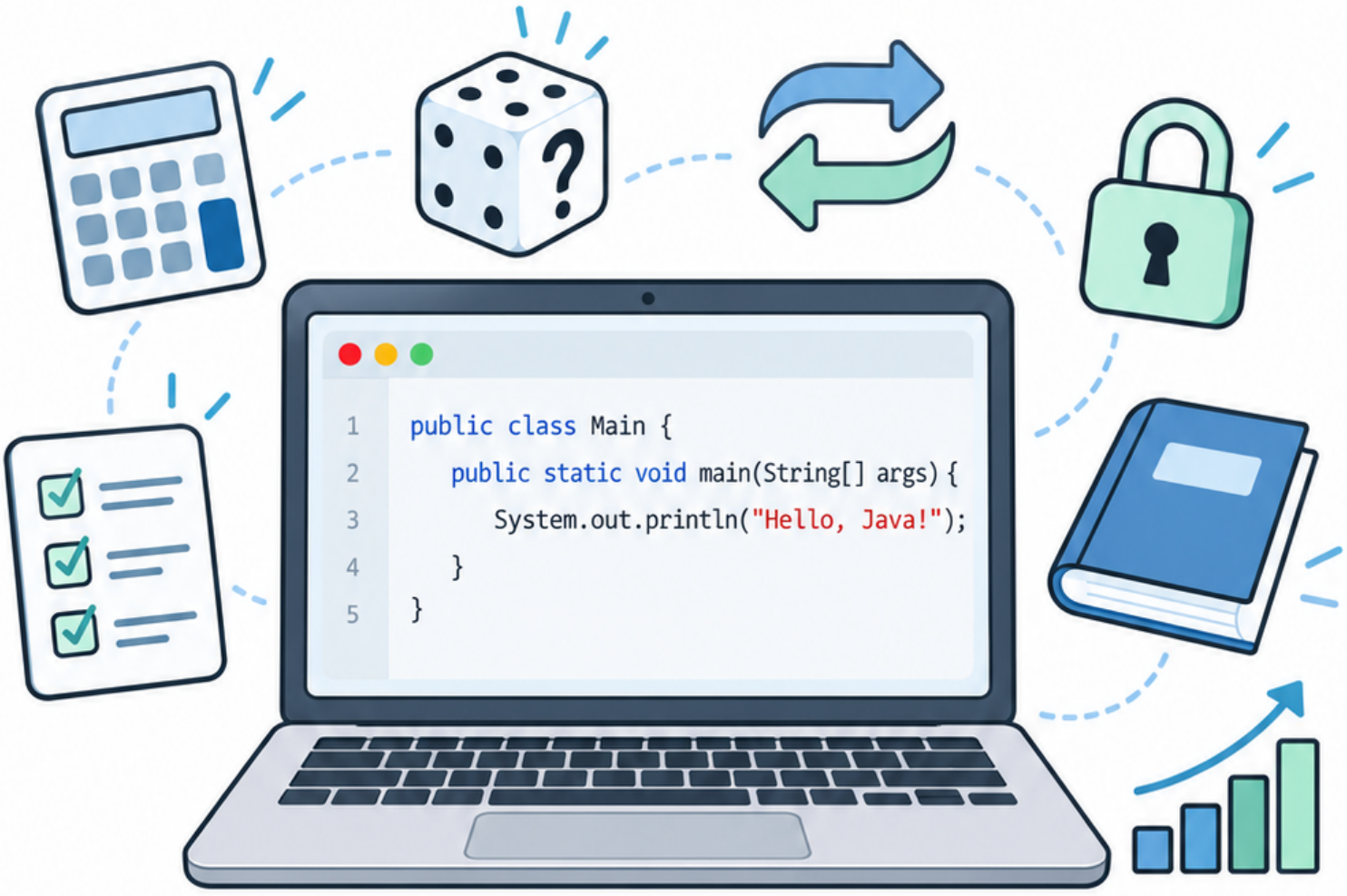
BONUS #4

# 10 PROJEKTÓW W JAVIE OD ZERA

**Od pierwszej linijki kodu do własnych działających programów.**

Praktyczny przewodnik krok po kroku dla początkujących.



# INFORMACJE O MATERIALE

## Bonus #4 · 10 projektów w Javie od zera

© 2026. Wszelkie prawa zastrzeżone.

Niniejszy materiał ma charakter edukacyjny. Został przygotowany z myślą o osobach, które chcą uczyć się programowania w Javie poprzez samodzielne tworzenie praktycznych projektów.

Przedstawione przykłady kodu służą do nauki, eksperymentowania i rozwijania własnych umiejętności programistycznych.

## Ważna informacja

Efekty nauki zależą od regularnej praktyki, samodzielnego wykonywania ćwiczeń i analizy popełnianych błędów.

## Jak korzystać z przykładów?

Możesz przepisywać kod, uruchamiać programy oraz modyfikować ćwiczenia na potrzeby własnej nauki.

# SPIS TREŚCI

Twoja droga od podstaw do 10 gotowych projektów.


**NA DOBRY POCZĄTEK**
Przygotowanie do pracy — str. 04

## 10 PRAKTYCZNYCH PROJEKTÓW

1. Kalkulator rachunku i napiwku — str. 10
2. Gra „Zgadnij liczbę" — str. 18
3. Konwerter jednostek — str. 26
4. Generator haseł — str. 34
5. Quiz wiedzy — str. 42
6. Lista zadań — str. 50
7. Książka kontaktów — str. 58
8. Menedżer budżetu — str. 66
9. Dziennik nawyków — str. 74
10. Mini biblioteka — str. 82

**NA ZAKOŃCZENIE**
Testowanie i portfolio — str. 90


**Każdy projekt to nowe umiejętności i kolejny krok do samodzielnego programowania.**

# ZACZNIJ PROGRAMOWAĆ NAPRAWDĘ!

**Nie musisz wiedzieć wszystkiego. Wystarczy, że zaczniesz od pierwszego projektu.**



## Witaj!

Jeśli dopiero zaczynasz naukę Javy, ten przewodnik jest dla Ciebie. Nie musisz znać zaawansowanych algorytmów ani rozumieć wszystkich pojęć programistycznych.

W kolejnych rozdziałach nauczysz się tworzyć prawdziwe, działające programy — krok po kroku.

Zaczniesz od prostego kalkulatora, a zakończysz na aplikacjach wykorzystujących obiekty, kolekcje i zapisywanie danych.

## JAK NAJLEPIEJ SIĘ UCZYĆ?

### 1. PRZEPISZ KOD

Samodzielnie wpisuj przykłady w edytorze.

### 2. URUCHOM PROGRAM

Sprawdź, co pojawia się na ekranie.

### 3. ZMIEŃ JEDNĄ RZECZ

Eksperymentuj z wartościami i instrukcjami.

### 4. NAPRAW BŁĘDY

Traktuj komunikaty o błędach jak wskazówki.

**CZYTAJ** → **PISZ** → **URUCHAMIAJ** → **POPRAWIAJ**



**Nie uczysz się programowania przez samo czytanie kodu. Uczysz się, kiedy go tworzysz!**

MODUŁ 00

# PRZYGOTOWANIE DO PRACY

Skonfiguruj środowisko i uruchom swój pierwszy program w Javie.




Zanim zbudujesz pierwszy projekt, przygotuj narzędzia, które będą Ci towarzyszyć podczas nauki.

**01** **Zainstaluj JDK**

**02** **Przygotuj edytor kodu**

**03** **Uruchom program**

# CZEGO POTRZEBUJESZ DO PROGRAMOWANIA?

## 1. KOMPUTER



Komputer z systemem Windows, macOS lub Linux.

To na nim będziesz pisać i uruchamiać swoje programy.

## 2. JDK — JAVA DEVELOPMENT KIT

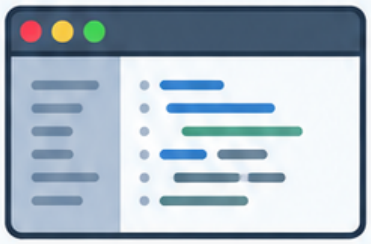



Zestaw narzędzi potrzebnych do tworzenia programów w Javie.

Zawiera między innymi kompilator javac oraz narzędzia do uruchamiania programów.

**TWOJE STANOWISKO DO PROGRAMOWANIA**

## 3. EDYTOR KODU



Program, w którym piszesz i poprawiasz kod.

Przykład: IntelliJ IDEA lub edytor obsługujący Javę.




**DZIAŁAJĄCY PROGRAM**

**JDK pozwala tworzyć i uruchaamiać programy.**
**Edytor pomaga wygodnie pisać kod.**

**LISTA KONTROLNA**

- [ ] Mam komputer.
- [ ] Zainstaluję JDK.
- [ ] Przygotuję edytor.
- [ ] Sprawdzę działanie Javy.

# ZAINSTALUJ JAVĘ I SPRAWDŹ, CZY DZIAŁA

1



## KROK 1 — POBIERZ JDK

Pobierz JDK z oficjalnego źródła, na przykład Eclipse Temurin, wybierając aktualną wersję LTS odpowiednią dla Twojego systemu.

2



## KROK 2 — ZAINSTALUJ

Uruchom instalator i postępuj zgodnie z instrukcjami.

3



## KROK 3 — OTWÓRZ TERMINAL

Uruchom terminal systemowy lub wbudowany terminal edytora.

4



## KROK 4 — WPISZ POLECENIA

```
java -version
javac -version
```



## WYNIK

Jeżeli oba polecenia pokazują wersję zainstalowanych narzędzi, środowisko jest gotowe.



## JEŚLI COŚ NIE DZIAŁA

Jeśli widzisz komunikat, że polecenie nie zostało rozpoznane, sprawdź instalację JDK i konfigurację zmiennej PATH.
Po zmianach otwórz terminal ponownie.



# TWÓJ PIERWSZY PROGRAM: HELLO, JAVA!

1

## KROK 1 — UTWÓRZ PLIK

Utwórz folder **PierwszyProgram**, a następnie plik **Main.java**.



2

## KROK 2 — WPISZ KOD

Main.java

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

3

## KROK 3 — ZAPISZ PLIK

Zapisz kod w pliku Main.java.



4

## KROK 4 — URUCHOM PROGRAM

W terminalu, w folderze zawierającym plik, wpisz:



```
javac Main.java
java Main
```

## WYNIK NA EKRANIE:

```
Witaj, Javo!
```



## Co właśnie się stało?

Kompilator **javac** przekształcił plik **Main.java** w plik **Main.class** zawierający kod bajtowy. Następnie polecenie **java Main** uruchomiło program w wirtualnej maszynie Javy (JVM).



## Małe wyzwanie

Zmień tekst powitania na własną wiadomość i uruchom program ponownie.

# JAK CZYTAĆ KOD W JAVIE?

Kod w Javie ma swoją strukturę i zasady. Warto wiedzieć, co oznaczają poszczególne elementy, aby łatwiej go czytać i rozumieć.

### 1. KLASA

`class Main`

Definiuje klasę, czyli podstawową jednostkę organizującą kod w Javie.

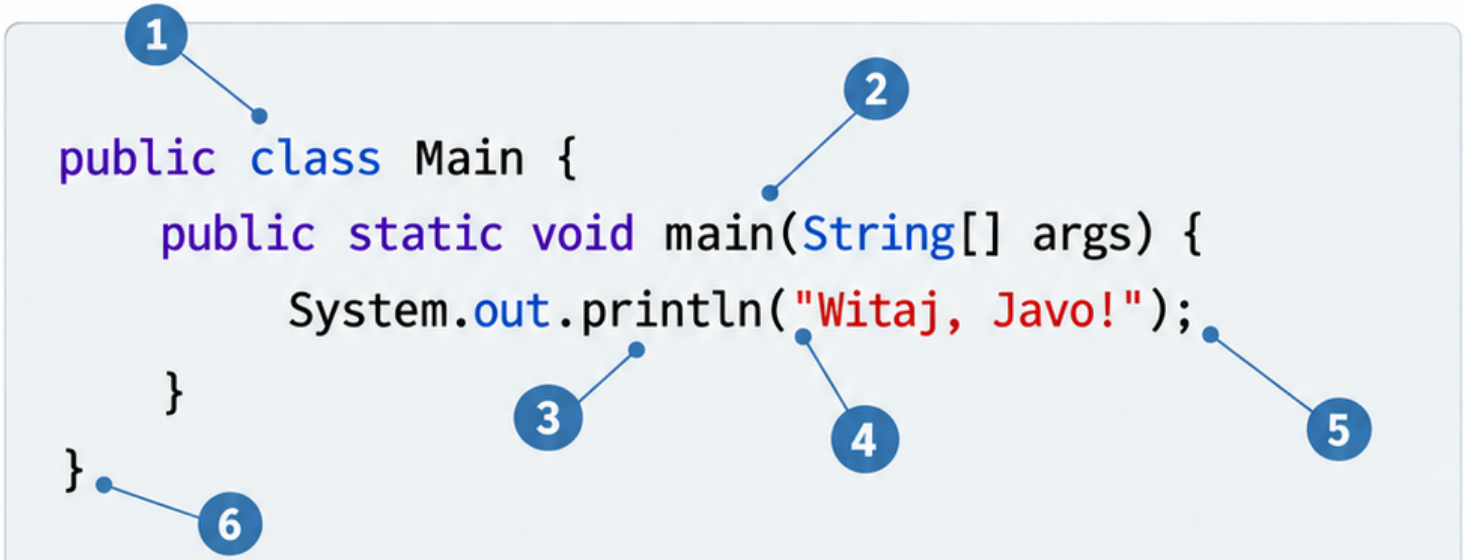
### 2. METODA main

`main(String[] args)`

Metoda, od której rozpoczyna się wykonywanie tego programu.

### 3. System.out.println

Wyświetla tekst w konsoli i przechodzi do nowego wiersza.




### 4. CUDZYSŁÓW

Tekst zapisany między cudzysłowami jest łańcuchem znaków, czyli napisem.

### 5. ŚREDNIK

`;`

Kończy wiele instrukcji w Javie.

### 6. NAWIASY KLAMROWE

`{ }`

Wyznaczają początek i koniec bloków kodu.

**Wskazówka**

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

**Małe ćwiczenie**

Znajdź w kodzie metodę main, średnik i dwa bloki wyznaczone nawiasami klamrowymi.

# JAK BUDOWAĆ KAŻDY PROJEKT KROK PO KROKU?

## 1. CEL PROJEKTU

Co program powinien robić? Zanim zaczniesz pisać kod, określ oczekiwany rezultat.

## 2. PLAN DZIAŁANIA

Jakie dane program otrzyma? Jakie operacje wykona? Co wyświetli użytkownikowi?

## 3. PISANIE KODU

Twórz rozwiązanie małymi fragmentami. Zapisuj zmiany.

## 4. URUCHAMIANIE

Uruchom program i sprawdź jego zachowanie.

## 5. TESTOWANIE

Wypróbuj poprawne, niepoprawne i nietypowe dane.

## 6. ROZBUDOWA

Dodaj własną funkcję lub zmień działanie programu.

### PRZYKŁAD

| | |
|---|---|
| **CEL:** | Obliczyć napiwek. |
| **DANE:** | Kwota rachunku i procent napiwku. |
| **DZIAŁANIE:** | Pomnożyć kwotę przez procent i podzielić przez 100. |
| **WYNIK:** | Wyświetlić kwotę napiwku oraz sumę do zapłaty. |



## ZANIM PRZEJDZIESZ DALEJ

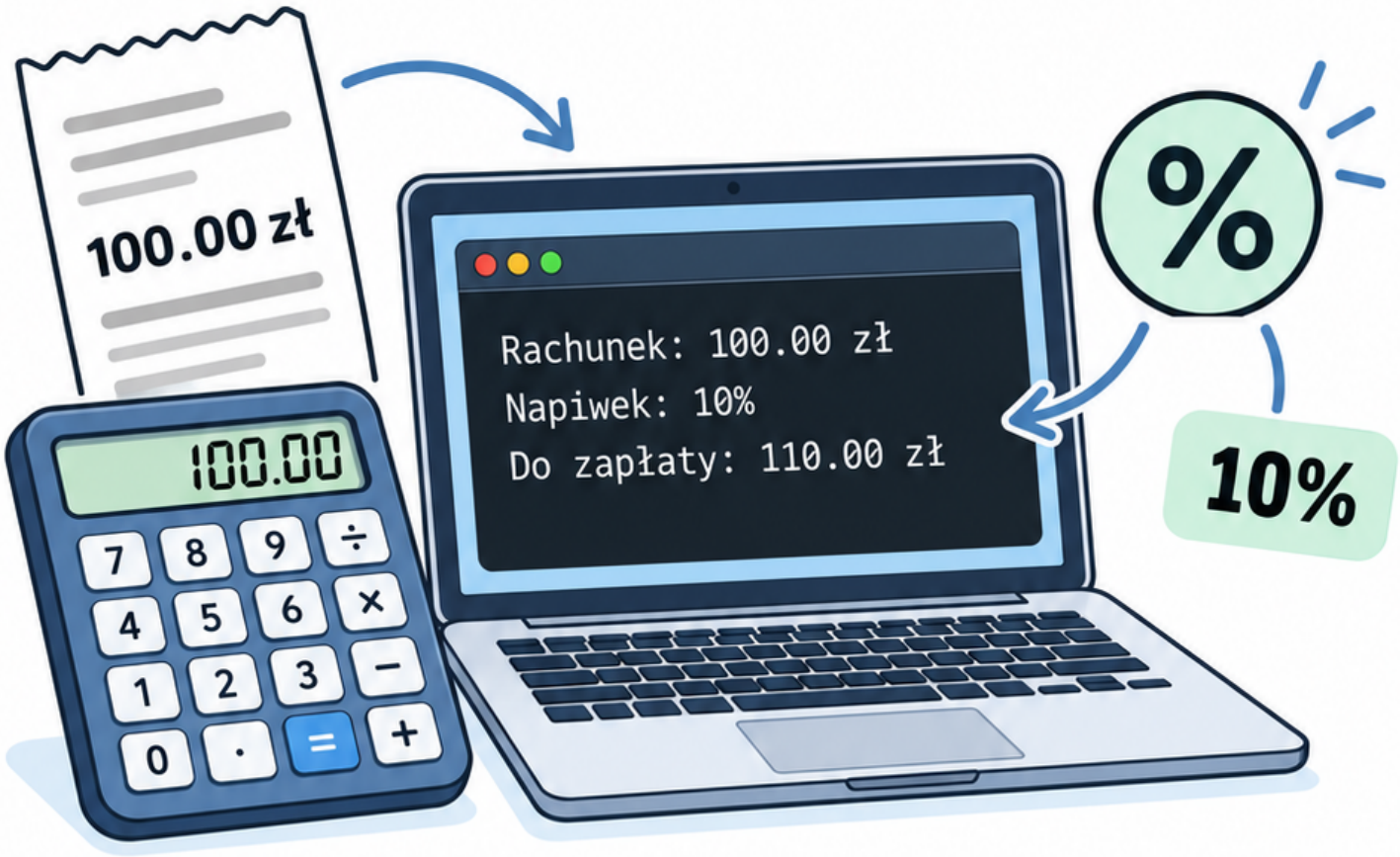
- [ ] Java działa na moim komputerze.
- [ ] Potrafię utworzyć plik .java.
- [ ] Umiem uruchomić prosty program.
- [ ] Rozumiem, że błędy są częścią nauki.

**Wszystko gotowe? Czas na pierwszy projekt!**

PROJEKT 01

# KALKULATOR RACHUNKU I NAnapiwku

**Stwórz swój pierwszy praktyczny program w Javie!**




## CZEGO SIĘ NAUCZYSZ?

- Pobierania danych od użytkownika.
- Przechowywania liczb w zmiennych.
- Wykonywania obliczeń matematycznych.
- Wyświetlania wyników.
- Sprawdzania poprawności danych.

**Jeden prosty projekt — wiele nowych umiejętności!**

# JAK DZIAŁA KALKULATOR NAPIWKU?

1



## KROK 1 — PODAJ KWOTĘ RACHUNKU

Użytkownik wpisuje kwotę rachunku.
Przykład: 100.00 zł

2



## KROK 2 — WYBIERZ PROCENT NAPIWKU

Użytkownik określa wysokość napiwku.
Przykład: 10%

3



## KROK 3 — OBLICZ NAPIWEK

Program wykonuje działanie:
100 × 10 / 100 = 10
Napiwek wynosi 10 zł.

4




## KROK 4 — WYŚWIETL SUMĘ

Program dodaje napiwek do rachunku.
100 + 10 = 110



## WYNIK KOŃCOWY

Do zapłaty: 110.00 zł



## CO TO JEST ALGORYTM?

Algorytm to uporządkowana lista kroków prowadzących do rozwiązania określonego problemu.



## SPRÓBUJ SAMODZIELNIE

Spróbuj samodzielnie obliczyć napiwek 20% od rachunku 50 zł.
Odpowiedź kontrolna: napiwek 10 zł, razem 60 zł.

# JAK POBIERAĆ LICZBY OD UŻYTKOWNIKA?

## 1 IMPORTUJ SCANNER

Na początku pliku wpisz:

```
import java.util.Scanner;
import java.util.Locale;
```



Scanner umożliwia odczytywanie danych wpisywanych przez użytkownika.

## 2 UTWÓRZ OBIEKT SCANNER

W metodzie main wpisz:

```
Scanner scanner = new Scanner(System.in);
scanner.useLocale(Locale.US);
```



Ten przykład używa kropki jako separatora dziesiętnego.

## 3 POBIERZ KWOTĘ RACHUNKU

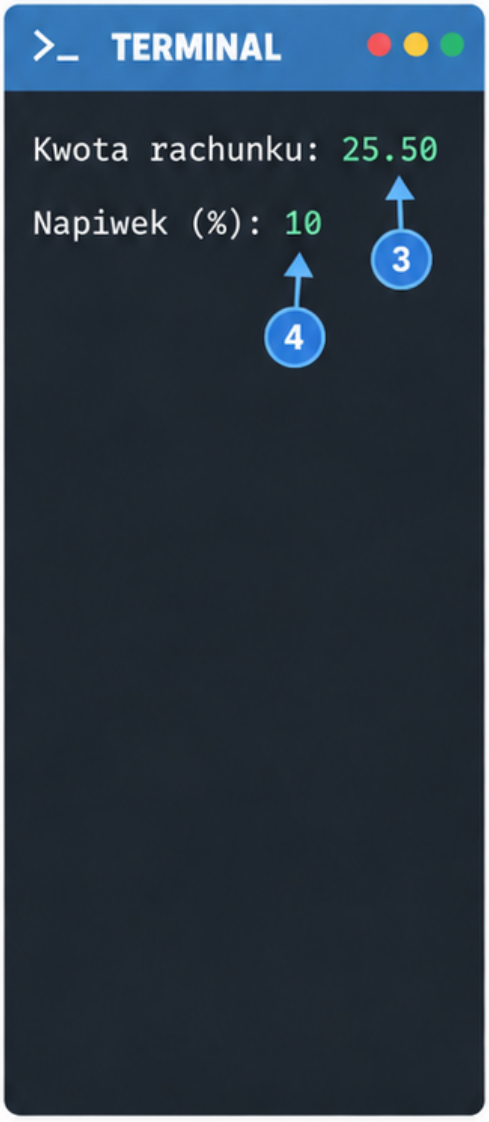
```
System.out.print("Kwota rachunku: ");
double rachunek = scanner.nextDouble();
```



## 4 POBIERZ PROCENT NAPIWKU

```
System.out.print("Napiwek (%): ");
double procent = scanner.nextDouble();
```






# CO OZNACZAJĄ TE ELEMENTY?

**double**



Typ danych przechowujący liczby, także z częścią dziesiętną.

**rachunek**



Zmienna przechowująca kwotę rachunku.

**nextDouble()**



Metoda odczytująca liczbę dziesiętną.

**System.out.print()**



Wyświetla komunikat bez przechodzenia do nowego wiersza.



**W tym projekcie wpisuj liczby dziesiętne z kropką, np. 25.50, a nie 25,50.**

# OBLICZAMY NAPIWEK I CAŁKOWITĄ KWOTĘ

## WZÓR NA NAPIWEK

**NAPIWEK = RACHUNEK × PROCENT / 100**




## PRZYKŁAD




Rachunek: 80 zł
Napiwek: 15%
Obliczenie: 80 × 15 / 100 = 12
Napiwek wynosi 12 zł.

## CO OZNACZAJĄ SYMBOLE?

- `*` — mnożenie
- `/` — dzielenie
- `=` — przypisanie wartości do zmiennej

## ZAPIS W JAVIE

```
double napiwek = rachunek * procent / 100;
```

## OBLICZENIE ŁĄCZNEJ KWOTY

```
double suma = rachunek + napiwek;
```

## WYNIK PRZYKŁADU

```
Rachunek: 80.00 zł
Napiwek: 12.00 zł
Do zapłaty: 92.00 zł
```



## WAŻNE

Operator = w tym przykładzie przypisuje wynik obliczenia do zmiennej. Nie służy do porównywania wartości.

## MINIWYZWANIE

Ile wynosi 5% napiwku od rachunku 200 zł?
Odpowiedź: 10 zł.

# NAPISZ I URUCHOM SWÓJ KALKULATOR

## KROK 1 — UTWÓRZ PLIK

Utwórz plik `Main.java`.

## KROK 2 — WPISZ CAŁY KOD

```java
import java.util.Locale;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        scanner.useLocale(Locale.US);

        System.out.print("Kwota rachunku: ");
        double rachunek = scanner.nextDouble();
        System.out.print("Napiwek (%): ");
        double procent = scanner.nextDouble();

        double napiwek = rachunek * procent / 100;
        double suma = rachunek + napiwek;

        System.out.printf(Locale.US, "Napiwek: %.2f zl%n", napiwek);
        System.out.printf(Locale.US, "Do zaplaty: %.2f zl%n", suma);
    }
}
```

## KROK 3 — SKOMPILUJ I URUCHOM

```
javac Main.java
java Main
```

## PRZYKŁADOWY WYNIK

```
Kwota rachunku: 100
Napiwek (%): 10
Napiwek: 10.00 zl
Do zaplaty: 110.00 zl
```



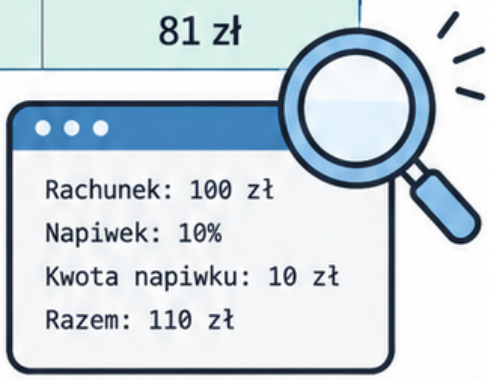
`%.2f` wyświetla liczbę z dwoma miejscami dziesiętnymi, a `%n` przechodzi do nowego wiersza.

# SPRAWDŹ, CZY TWÓJ PROGRAM DZIAŁA POPRAWNIE

## DLACZEGO TESTUJEMY PROGRAMY?

Program może się uruchamiać, ale nadal podawać niepoprawne wyniki. Testowanie polega na porównywaniu otrzymanych rezultatów z oczekiwanymi wartościami.

| Rachunek | Napiwek % | Kwota napiwku | Razem |
|---|---|---|---|
| 100 zł | 10% | 10 zł | 110 zł |
| 80 zł | 15% | 12 zł | 92 zł |
| 50 zł | 0% | 0 zł | 50 zł |
| 120 zł | 20% | 24 zł | 144 zł |
| 75 zł | 8% | 6 zł | 81 zł |




## INSTRUKCJA

1. Uruchom kalkulator.
2. Wprowadź dane z pierwszego wiersza.
3. Porównaj wynik programu z tabelą.
4. Powtórz dla pozostałych przykładów.

## CO SPRAWDZIĆ, JEŚLI WYNIK JEST INNY?

- [x] Czy wzór używa dzielenia przez 100?
- [x] Czy zmienne przechowują właściwe dane?
- [x] Czy nie pomylono operatorów?
- [x] Czy wpisano poprawne wartości wejściowe?



## WAŻNA UWAGA

Na tym etapie zakładamy, że użytkownik podaje poprawne liczby. W następnym kroku nauczysz program rozpoznawać błędne dane.

# CO ZROBIĆ, GDY UŻYTKOWNIK WPISZE BŁĘDNE DANE?

### PROBLEM 1 – UJEMNA KWOTA

**Przykład:** Kwota rachunku: -50

Rachunek nie powinien mieć ujemnej wartości.

### PROBLEM 2 – NIEPRAWIDŁOWY PROCENT

**Przykład:** Napiwek (%): 150

W tym projekcie przyjmujemy zakres od 0% do 100%.

### PROBLEM 3 – TEKST ZAMIAST LICZBY

**Przykład:** Kwota rachunku: abc

Program powinien wyświetlić komunikat zamiast zgłaszać nieobsłużony wyjątek.

## ROZWIĄZANIE – SPRAWDŹ DANE

### A. Zastąp odczyt rachunku:

```
if (!scanner.hasNextDouble()) {
    System.out.println("Blad: wpisz liczbe.");
    return;
}
double rachunek = scanner.nextDouble();
```

### B. Zastąp odczyt procentu:

```
if (!scanner.hasNextDouble()) {
    System.out.println("Blad: wpisz liczbe.");
    return;
}
double procent = scanner.nextDouble();
```

### C. Po odczycie obu liczb, przed obliczeniami, dodaj:

```
if (rachunek < 0 || procent < 0 || procent > 100) {
    System.out.println("Blad: niepoprawne wartosci.");
    return;
}
```

## WYJAŚNIENIE

| | |
|---|---|
| `if` | – wykonuje instrukcję, jeśli warunek jest spełniony. |
| `\|\|` | – oznacza logiczne „lub”. |
| `return` | – kończy wykonywanie metody main. |
| `hasNextDouble()` | – sprawdza, czy kolejne dane można odczytać jako liczbę typu double. |

Te fragmenty zastępują wcześniejsze linie odczytu danych i dodają kontrolę wartości. Pozostała część programu pozostaje bez zmian.

# ROZBUDUJ KALKULATOR — TWOJE PIERWSZE WYZWANIE!

## MISJA: PODZIEL RACHUNEK MIĘDZY KILKA OSÓB

Rozbuduj kalkulator tak, aby obliczał również kwotę do zapłaty przez jedną osobę.

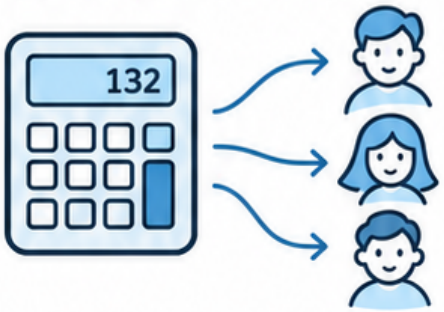
**PRZYKŁAD**

Rachunek: 120 zł
Napiwek: 10%
Liczba osób: 3

→

**OBLICZENIA**

Napiwek: 12 zł
Razem: 132 zł
Na osobę: 44 zł




## KROK 1 — POBIERZ LICZBĘ OSÓB

Po wczytaniu rachunku i procentu napiwku dodaj:

```
System.out.print("Liczba osob: ");
if (!scanner.hasNextInt()) {
    System.out.println("Blad: wpisz liczbe calkowita.");
    return;
}
int osoby = scanner.nextInt();
```

## KROK 2 — SPRAWDŹ POPRAWNOŚĆ

Przed obliczeniem podziału:

```
if (osoby <= 0) {
    System.out.println("Blad: liczba osob musi byc dodatnia.");
    return;
}
```

## KROK 3 — OBLICZ KWOTĘ NA OSOBĘ

Po obliczeniu zmiennej suma:

```
double naOsobe = suma / osoby;
```

## KROK 4 — WYŚWIETL WYNIK

```
System.out.printf(Locale.US, "Na osobe: %.2f zl%n", naOsobe);
```

**Dodatkowe zabezpieczenie**

Przed `scanner.nextInt()` użyj `scanner.hasNextInt()`, aby upewnić się, że użytkownik wpisał liczbę całkowitą. Jeśli nie, pokaż komunikat błędu i zakończ metodę.

**Uwaga**

Przy dzieleniu niektórych kwot mogą pojawić się różnice wynikające z zaokrąglania do groszy. To prosty kalkulator edukacyjny.

## CHECKLISTA UKOŃCZENIA PROJEKTU

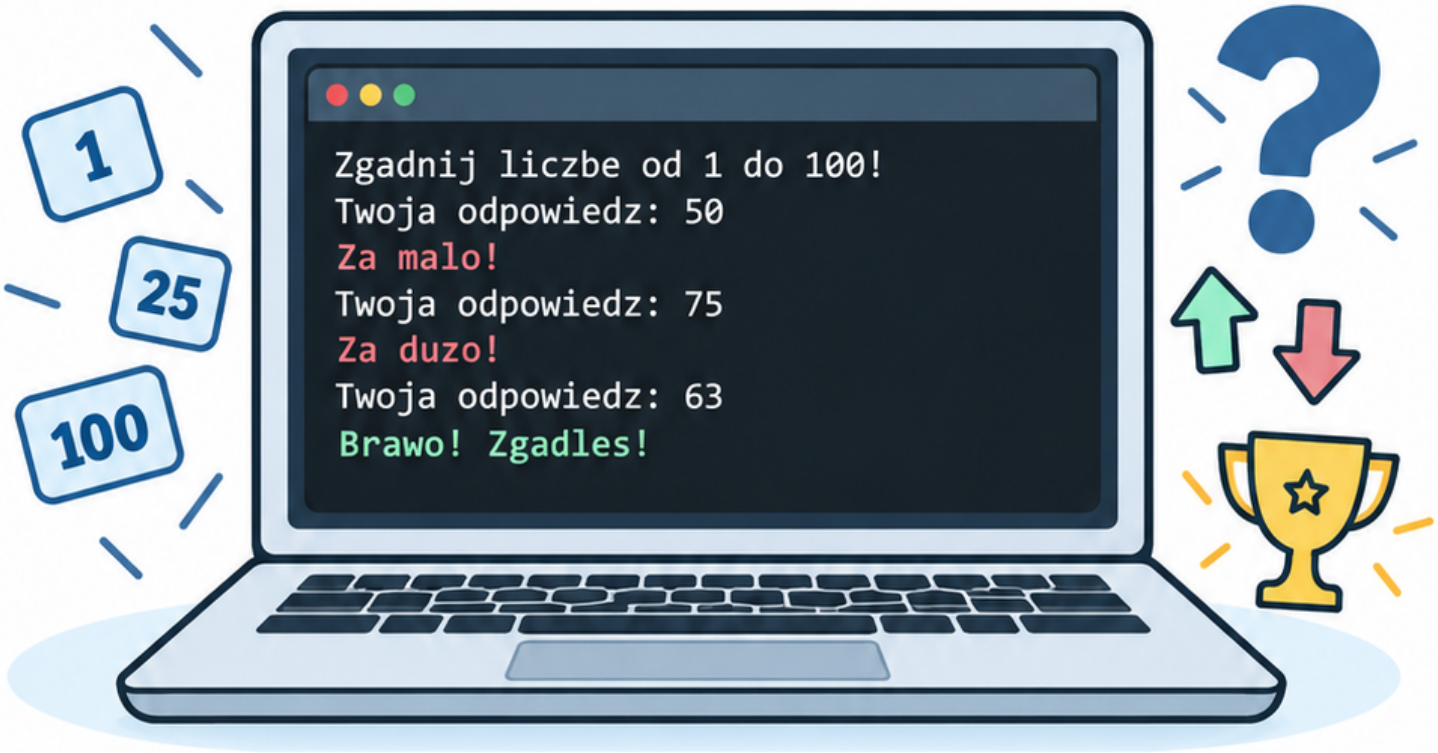
- [ ] Potrafię pobierać liczby od użytkownika.
- [ ] Umiem wykonywać obliczenia.
- [ ] Rozumiem działanie zmiennych.
- [ ] Potrafię używać instrukcji if.
- [ ] Umiem testować wyniki.
- [ ] Rozbudowałem działający program.

**Gratulacje! Twój pierwszy projekt w Javie jest gotowy.**

PROJEKT

# 02 GRA „ZGADNIJ LICZBĘ”

**Zbuduj prostą grę, która losuje tajną liczbę i sprawdza odpowiedzi gracza.**

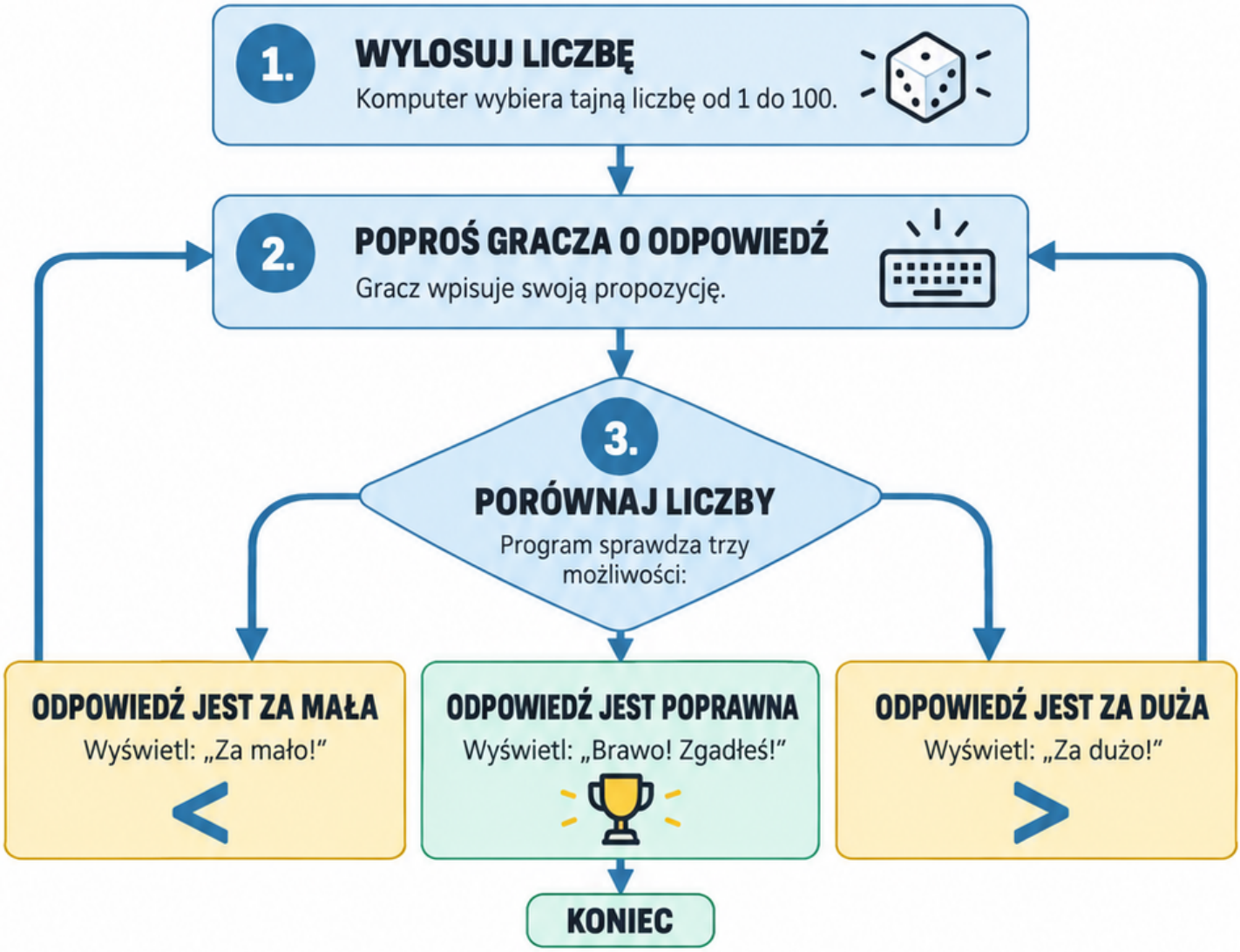



## CZEGO SIĘ NAUCZYSZ?

- Losowania liczb.
- Korzystania z klasy Random.
- Porównywania wartości.
- Tworzenia pętli.
- Liczenia prób gracza.
- Zakończenia gry po zwycięstwie.

**Tym razem Twój program będzie zadawać pytania i reagować na decyzje użytkownika!**

# JAK DZIAŁA GRA „ZGADNIJ LICZBĘ”?




## 4. POWTARZAJ LUB ZAKOŃCZ

Jeśli odpowiedź jest błędna, gracz próbuje ponownie. Jeśli liczba została odgadnięta, gra się kończy.

## KLUCZOWE POJĘCIE: PĘTLA

Pętla pozwala wielokrotnie wykonywać fragment kodu. W tym projekcie będzie powtarzać pytanie o liczbę, dopóki gracz nie wygra.

## PRZYKŁADOWA ROZGRYWKA

```
Tajna liczba: 42
Gracz wpisuje: 20 → Za mało!
Gracz wpisuje: 60 → Za dużo!
Gracz wpisuje: 42 → Brawo!
```

## MINIWYZWANIE

Jeśli szukana liczba wynosi 70, jakie podpowiedzi powinien wyświetlić program dla odpowiedzi 30, 90 i 70?

**Odpowiedź:**
30 — Za mało!
90 — Za dużo!
70 — Brawo!

# JAK WYLOSOWAĆ TAJNĄ LICZBĘ W JAVIE?

## CO TO JEST LICZBA LOSOWA?

W naszym projekcie komputer wybiera liczbę, której gracz nie zna. Każda nowa rozgrywka może rozpocząć się od innej wartości.

### 1 KROK 1 — IMPORTUJ KLASĘ RANDOM

```
import java.util.Random;
```

### 2 KROK 2 — UTWÓRZ GENERATOR

```
Random random = new Random();
```

### 3 KROK 3 — WYLOSUJ LICZBĘ

```
int tajnaLiczba = random.nextInt(100) + 1;
```

## JAK DZIAŁA TEN ZAPIS?

**random.nextInt(100)**
Generuje liczbę całkowitą od 0 do 99.

→

**+ 1**
Przesuwa zakres o jeden.

→

**WYNIK**
Otrzymujemy liczbę od 1 do 100.




0–99 → + 1 → 1–100

1 — 100

## PRZYKŁADOWE WYNIKI

7 — 42 — 86 — 13 — 99

## WAŻNE POJĘCIE

`int` to typ danych przechowujący liczby całkowite.

## MINIWYZWANIE

Jak wylosować liczbę od 1 do 10?
Rozwiązanie:

```
int liczba = random.nextInt(10) + 1;
```

# ZA MAŁO, ZA DUŻO CZY DOBRZE?




## TRZY MOŻLIWE SYTUACJE

Program porównuje odpowiedź gracza z wylosowaną liczbą.

```
if (proba < tajnaLiczba) {
    System.out.println("Za malo!");
} else if (proba > tajnaLiczba) {
    System.out.println("Za duzo!");
} else {
    System.out.println("Brawo! Zgadles!");
}
```

1. LICZBA JEST ZA MAŁA
2. LICZBA JEST ZA DUŻA
3. GRACZ ZGADŁ

## JAK TO DZIAŁA?

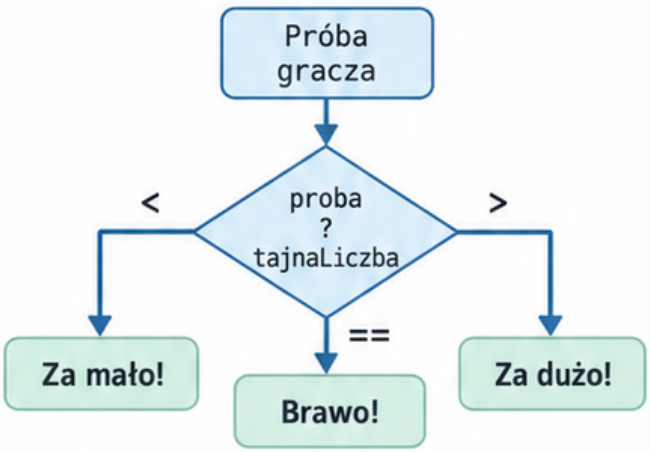
Jeśli próba jest mniejsza od tajnej liczby, program wyświetla komunikat „Za mało!”, Jeśli jest większa, wyświetla „Za dużo!”. W przeciwnym razie liczby są równe.

## PRZYKŁAD

**Tajna liczba: 65**

| Próba | Porównanie | Komunikat |
|---|---|---|
| 30 | 30 < 65 | Za mało! |
| 80 | 80 > 65 | Za dużo! |
| 65 | 65 == 65 | Brawo! |

## SCHEMAT DECYZJI




## CO OZNACZAJĄ OPERATORY?

< — mniejsze niż
> — większe niż
== — równe

## WARTO PAMIĘTAĆ!

W Javie operator == służy do porównywania wartości, a pojedynczy = do przypisywania wartości zmiennej.

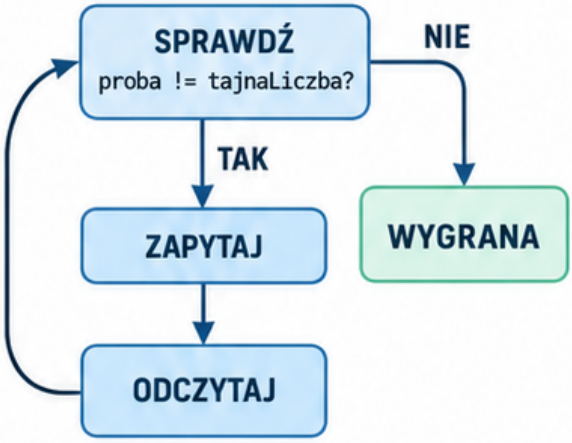
# JAK POWTARZAĆ PRÓBY AŻ DO WYGRANEJ?

## PROBLEM

Gracz może nie odgadnąć liczby za pierwszym razem. Program powinien pozwalać na kolejne próby, aż użytkownik znajdzie poprawną odpowiedź.

## ROZWIĄZANIE – PĘTLA WHILE

```
int proba = 0;
while (proba != tajnaLiczba) {
    System.out.print("Podaj liczbe: ");
    proba = scanner.nextInt();
}
```




## JAK DZIAŁA PĘTLA?

**1 SPRAWDŹ WARUNEK**

Czy wpisana liczba różni się od tajnej liczby?

**2 WYKONAJ INSTRUKCJE**

Jeśli tak, poproś o nową odpowiedź.

**3 POWTÓRZ**

Sprawdź warunek ponownie.

**4 ZAKOŃCZ**

Gdy liczby są równe, pętla przestaje działać.

## WERSJA Z LICZNIKIEM PRÓB

Wersja poniżej zastępuje poprzedni przykład.

```
int liczbaProb = 0;
int proba = 0;
while (proba != tajnaLiczba) {
    System.out.print("Podaj liczbe: ");
    proba = scanner.nextInt();
    liczbaProb++;
}
```

### WAŻNE

`!=` oznacza „różne od”.

### WYJAŚNIENIE

`liczbaProb++` zwiększa licznik o jeden po każdej próbie.

Ten przykład pokazuje zasadę działania pętli. W pełnej wersji gry dodamy wskazówki i sprawdzanie poprawności danych.

# ZBUDUJ KOMPLETNĄ GRĘ „ZGADNIJ LICZBĘ”

## KROK 1 — UTWÓRZ PLIK Main.java

W prostym edytorze tekstu utwórz plik o nazwie **Main.java**.

## KROK 2 — WPISZ KOD

Skopiuj poniższy kod do pliku **Main.java**.

```java
import java.util.Random;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Random random = new Random();
    int tajnaLiczba = random.nextInt(100) + 1;
    int liczbaProb = 0;
    int proba = 0;
    System.out.println("Zgadnij liczbe od 1 do 100!");
    while (proba != tajnaLiczba) {
      System.out.print("Twoja odpowiedz: ");
      if (!scanner.hasNextInt()) {
        System.out.println("Wpisz liczbe calkowita!");
        scanner.next();
        continue;
      }
      proba = scanner.nextInt();
      if (proba < 1 || proba > 100) {
        System.out.println("Podaj liczbe od 1 do 100!");
        continue;
      }
      liczbaProb++;
      if (proba < tajnaLiczba) {
        System.out.println("Za malo!");
      } else if (proba > tajnaLiczba) {
        System.out.println("Za duzo!");
      } else {
        System.out.println("Brawo! Zgadles!");
      }
    }
    System.out.println("Liczba prob: " + liczbaProb);
  }
}
```

## KROK 3 — URUCHOM

Skompiluj i uruchom program w konsoli.

```
javac Main.java
java Main
```



**Objaśnienie**

**continue** pomija pozostałą część bieżącej iteracji pętli i przechodzi do następnego sprawdzenia warunku.

## PRZYKŁADOWY WYNIK

Poniżej znajduje się przykładowy wynik działania programu.

```
Zgadnij liczbe od 1 do 100!
Twoja odpowiedz: 50
Za malo!
Twoja odpowiedz: 75
Za duzo!
Twoja odpowiedz: 63
Brawo! Zgadles!
Liczba prob: 3
```

# PRZETESTUJ GRĘ JAK PRAWDZIWY PROGRAMISTA



### TEST 01 — POPRAWNA ODPOWIEDŹ

Gracz wpisuje tajną liczbę.

**Oczekiwany wynik:**
Program wyświetla komunikat o zwycięstwie i kończy grę.



### TEST 02 — LICZBA ZA MAŁA

Gracz wpisuje liczbę mniejszą od tajnej.

**Oczekiwany wynik:**
Pojawia się komunikat „Za mało!" i kolejna próba.



### TEST 03 — LICZBA ZA DUŻA

Gracz wpisuje liczbę większą od tajnej.

**Oczekiwany wynik:**
Pojawia się komunikat „Za dużo!" i kolejna próba.



### TEST 04 — NIEPOPRAWNY ZAKRES

Gracz wpisuje 150 lub -5.

**Oczekiwany wynik:**
Program prosi o liczbę od 1 do 100, nie zwiększając licznika poprawnie przyjętych prób.



### TEST 05 — NIEPOPRAWNY TYP

Gracz wpisuje `abc`.

**Oczekiwany wynik:**
Program wyświetla komunikat o błędnych danych i umożliwia kolejną próbę.



## NAJCZĘSTSZE BŁĘDY

| Problem | Co sprawdzić? |
|---|---|
| Program nie kończy gry | Warunek pętli `while` |
| Błędna wskazówka | Operatory `<` i `>` |
| Zły zakres losowania | `nextInt(100) + 1` |
| Błędna liczba prób | Położenie `liczbaProb++` |
| Program nie kompiluje się | Nawiasy, średniki i nazwy |



### WAŻNA WSKAZÓWKA

Ponieważ liczba jest losowana, poszczególne testy mogą wymagać wielu rozgrywek. Podczas testowania możesz tymczasowo ustawić `int tajnaLiczba = 42;`, aby uzyskać przewidywalny wynik.

# ROZBUDUJ GRĘ I DODAJ POZIOMY TRUDNOŚCI



## MISJA — TRZY POZIOMY TRUDNOŚCI

Pozwól graczowi wybrać zakres losowania.

| ŁATWY | ŚREDNI | TRUDNY |
|---|---|---|
| Liczby od 1 do 10. | Liczby od 1 do 50. | Liczby od 1 do 100. |

Wybór poziomu i ustalenie zakresu dodaj po utworzeniu obiektów `scanner` i `random`, ale przed losowaniem tajnej liczby.

### KROK 1 — WYBIERZ POZIOM

```
System.out.println("1 - Latwy");
System.out.println("2 - Sredni");
System.out.println("3 - Trudny");
System.out.print("Wybierz poziom: ");
if (!scanner.hasNextInt()) {
   System.out.println("Niepoprawny wybor!");
   return;
}
int poziom = scanner.nextInt();
```

### KROK 2 — USTAL ZAKRES

```
int maksimum;
if (poziom == 1) {
    maksimum = 10;
} else if (poziom == 2) {
    maksimum = 50;
} else if (poziom == 3) {
    maksimum = 100;
} else {
   System.out.println("Niepoprawny wybor!");
   return;
}
```

### KROK 3 — WYLOSUJ LICZBĘ

Zastąp wcześniejszą instrukcję losowania:

```
int tajnaLiczba = random.nextInt(maksimum) + 1;
```

### KROK 4 — DOSTOSUJ GRĘ

Zastąp stałą 100 zmienną `maksimum` w komunikacie początkowym, komunikacie błędu i sprawdzaniu zakresu odpowiedzi.

### DODATKOWE WYZWANIE

Dodaj informację o wyniku:

- 1–5 prób: „Świetny wynik!"
- 6–10 prób: „Dobra robota!"
- Powyżej 10 prób: „Nie poddawaj się!"

### CHECKLISTA UMIEJĘTNOŚCI

- [ ] Potrafię losować liczby.
- [ ] Rozumiem instrukcje warunkowe.
- [ ] Potrafię tworzyć pętle while.
- [ ] Umiem liczyć próby.
- [ ] Potrafię sprawdzać dane użytkownika.
- [ ] Umiem rozbudować działający program.

**Gratulacje! Ukończyłeś drugi projekt w Javie.**

2 z 10 projektów

PROJEKT

# 03 KONWERTER JEDNOSTEK

**Stwórz program, który szybko przelicza długość, temperaturę i masę.**



**DŁUGOŚĆ**

Metry → Kilometry



**TEMPERATURA**

Stopnie Celsjusza →
Stopnie Fahrenheita



**MASA**

Kilogramy → Gramy






## CZEGO SIĘ NAUCZYSZ?

- Tworzenia menu tekstowego.
- Stosowania wzorów matematycznych.
- Korzystania z instrukcji switch.
- Dzielenia kodu na metody.
- Organizowania programów.
- Sprawdzania wyników przeliczeń.



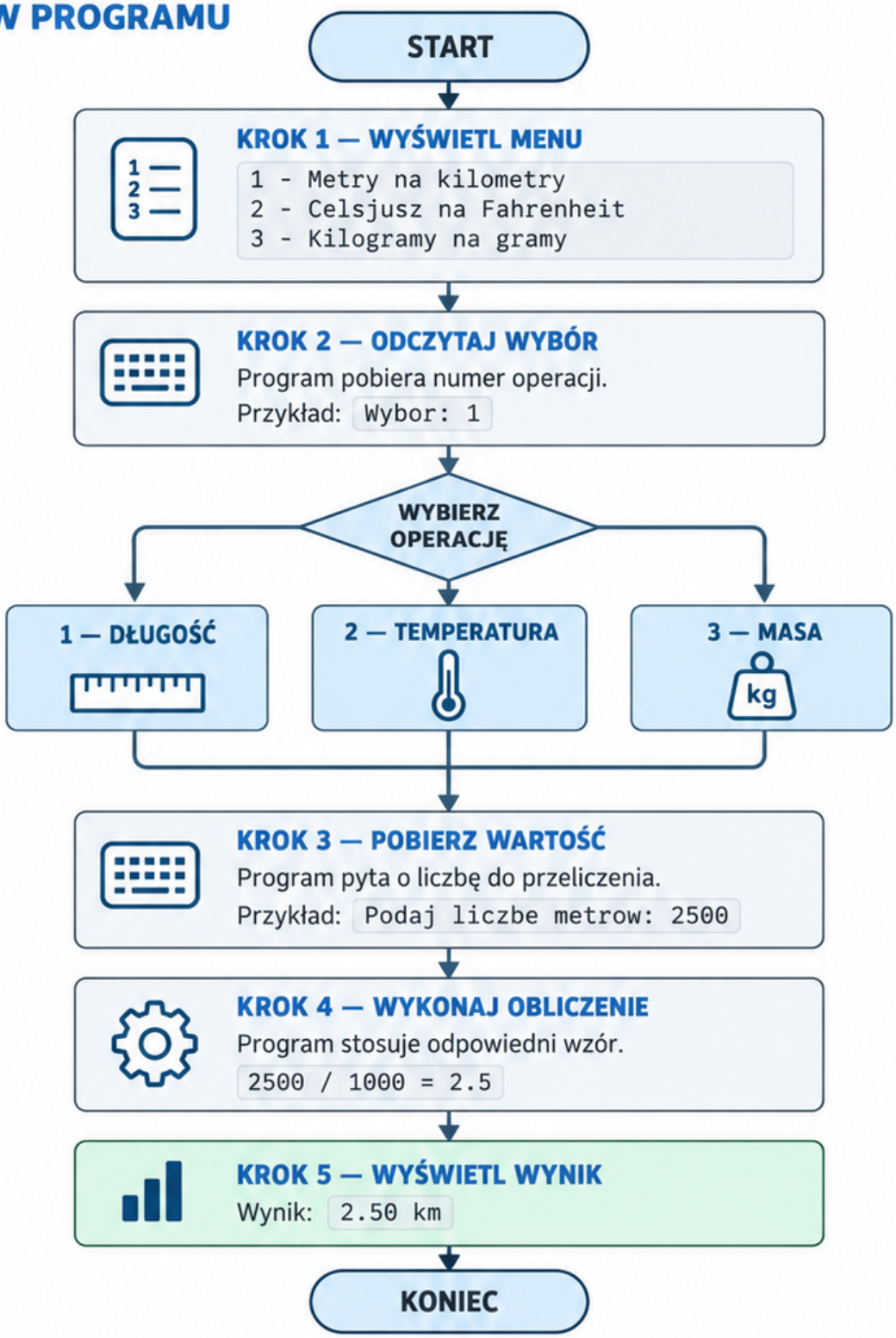
**Trzy rodzaje jednostek — jeden praktyczny program!**

# JAK ZAPROJEKTOWAĆ KONWERTER JEDNOSTEK?

## CEL PROGRAMU

Użytkownik wybiera rodzaj przeliczenia, wpisuje wartość i otrzymuje wynik.

## PRZEPŁYW PROGRAMU




### WAŻNE POJĘCIE

Menu tekstowe to lista dostępnych działań, z których użytkownik wybiera jedną opcję.

### MINIWYZWANIE

Co powinien zrobić program, jeśli użytkownik wpisze numer 5, chociaż menu ma tylko trzy opcje?

**Odpowiedź:** Powinien poinformować użytkownika, że wybrano nieprawidłową opcję.

# POZNAJ WZORY, KTÓRE WYKORZYSTA TWÓJ PROGRAM



## BLOK 1 — DŁUGOŚĆ

**METRY → KILOMETRY**

1 kilometr = 1000 metrów.

**Wzór:**

**kilometry = metry / 1000**

**Przykład:**

3500 metrów = 3,5 kilometra.

**Kod Java:**

```
double kilometry = metry / 1000.0;
```



## BLOK 2 — TEMPERATURA

**CELSJUSZ → FAHRENHEIT**

**Przykład:**

20°C = 68°F

**Wzór:**

**°F = (°C × 9 / 5) + 32**

**Kod Java:**

```
double fahrenheit = celsius * 9.0 / 5.0 + 32;
```



## BLOK 3 — MASA

**KILOGRAMY → GRAMY**

1 kilogram = 1000 gramów.

**Wzór:**

**gramy = kilogramy × 1000**

**Przykład:**

2,5 kg = 2500 g.

**Kod Java:**

```
double gramy = kilogramy * 1000;
```



### DLACZEGO W KODZIE PISZEMY 1000.0 I 9.0?

Zapis z częścią dziesiętną jasno wskazuje, że wykonujemy obliczenia na liczbach zmiennoprzecinkowych.



## MINIWYZWANIE

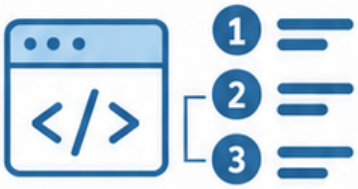
**Oblicz:**

1. 4500 m = ? km
2. 0°C = ? °F
3. 1,8 kg = ? g

**Odpowiedzi:**

1. 4,5 km
2. 32°F
3. 1800 g

# JAK OBSŁUGIWAĆ MENU ZA POMOCĄ SWITCH?



## CO TO JEST SWITCH?

Instrukcja **switch** pozwala wykonać różne fragmenty kodu w zależności od wartości wyrażenia. Jest przydatna, gdy użytkownik wybiera jedną z kilku dostępnych opcji.

## KROK 1 — UTWÓRZ MENU

```
System.out.println("KONWERTER JEDNOSTEK");
System.out.println("1 - Metry na kilometry");
System.out.println("2 - Celsjusz na Fahreheit");
System.out.println("3 - Kilogramy na gramy");
```

## KROK 2 — POBIERZ WYBÓR

```
Scanner scanner = new Scanner(System.in);
System.out.print("Wybor: ");
int wybor = scanner.nextInt();
```

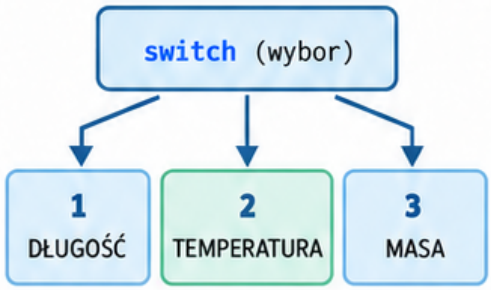
## KROK 3 — WYKORZYSTAJ SWITCH

```
01 switch (wybor) {
02     case 1:
03         System.out.println("Wybrano dlugosc");
04         break;
05     case 2:
06         System.out.println("Wybrano temperature");
07         break;
08     case 3:
09         System.out.println("Wybrano mase");
10         break;
11     default:
12         System.out.println("Niepoprawny wybor");
13 }
```

## PRZYKŁADOWY WYNIK

```
KONWERTER JEDNOSTEK
1 - Metry na kilometry
2 - Celsjusz na Fahrenheit
3 - Kilogramy na gramy
Wybor: 2
Wybrano temperature
```

## SCHEMAT DZIAŁANIA SWITCH




**default**
Wykonuje się, gdy żadna z wymienionych wartości nie pasuje.

## JAK DZIAŁA TEN KOD?

**switch (wybor)**
Sprawdza wartość zmiennej wybor.

**case**
Określa działania dla konkretnej wartości.

**break**
Kończy wykonywanie instrukcji switch.

**default**
Wykonuje się, gdy żadna z wymienionych wartości nie pasuje.

## WAŻNE

To fragment przygotowujący obsługę menu, a nie gotowy konwerter. Umieść go wewnątrz metody main w pliku Main.java.

Na początku pliku dodaj instrukcję: `import java.util.Scanner;`

W następnym etapie zastąpimy komunikaty rzeczywistymi obliczeniami i dodamy sprawdzanie poprawności wpisanych danych.

# ZBUDUJ SWÓJ PIERWSZY KONWERTER JEDNOSTEK

## KROK 1 — UTWÓRZ PLIK

Utwórz plik Main.java.

## KROK 2 — WPISZ KOD

```
import java.util.Locale;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    scanner.useLocale(Locale.US);
    System.out.println("KONWERTER JEDNOSTEK");
    System.out.println("1 - Metry na kilometry");
    System.out.println("2 - Celsjusz na Fahrenheit");
    System.out.println("3 - Kilogramy na gramy");
    System.out.print("Wybor: ");
    if (!scanner.hasNextInt()) {
      System.out.println("Niepoprawny wybor!");
      return;
    }
    int wybor = scanner.nextInt();
    if (wybor < 1 || wybor > 3) {
      System.out.println("Niepoprawna opcja!");
      return;
    }
    System.out.print("Podaj wartosc: ");
    if (!scanner.hasNextDouble()) {
      System.out.println("Wpisz poprawna liczbe!");
      return;
    }
    double wartosc = scanner.nextDouble();
    switch (wybor) {
      case 1:
        System.out.printf(Locale.US,
          "Wynik: %.2f km%n", wartosc / 1000.0);
        break;
      case 2:
        System.out.printf(Locale.US,
          "Wynik: %.2f F%n", wartosc * 9.0 / 5.0 + 32);
        break;
      case 3:
        System.out.printf(Locale.US,
          "Wynik: %.2f g%n", wartosc * 1000);
        break;
    }
  }
}
```

## KROK 3 — URUCHOM PROGRAM

```
javac Main.java
java Main
```

## PRZYKŁADOWY WYNIK

```
KONWERTER JEDNOSTEK
1 - Metry na kilometry
2 - Celsjusz na Fahrenheit
3 - Kilogramy na gramy
Wybor: 1
Podaj wartosc: 2500
Wynik: 2.50 km
```

## CO WŁAŚNIE ZROBIŁEŚ?

- Utworzyłeś menu programu.
- Odczytałeś wybór użytkownika.
- Pobrałeś wartość liczbową.
- Wybrałeś odpowiedni wzór.
- Wyświetliłeś wynik obliczenia.

### WSKAZÓWKA

Wpisuj liczby dziesiętne z kropką, np. **2.5**. Ten program korzysta z `Locale.US`.

# UPROŚĆ SWÓJ PROGRAM ZA POMOCĄ METOD

## CO TO JEST METODA?

Metoda to nazwany fragment kodu wykonujący określone zadanie. Możesz ją wywołać w różnych miejscach programu bez ponownego pisania tej samej operacji.

### PRZYKŁAD — PRZELICZANIE METRÓW

```
static double metryNaKilometry(double metry) {
  return metry / 1000.0;
}
```

### WYJAŚNIENIE

`static` — pozwala wywołać tę metodę bez tworzenia obiektu klasy.

`double` — typ zwracanej wartości.

`metryNaKilometry` — nazwa metody.

`double metry` — parametr przyjmujący liczbę.

`return` — zwraca wynik obliczenia.

## UTWÓRZ TRZY METODY

Poniższy zestaw zastępuje wcześniejszy przykład metody `metryNaKilometry`.

```
static double metryNaKilometry(double m) {
  return m / 1000.0;
}
static double celsiusNaFahrenheit(double c) {
  return c * 9.0 / 5.0 + 32;
}
static double kilogramyNaGramy(double kg) {
  return kg * 1000.0;
}
```

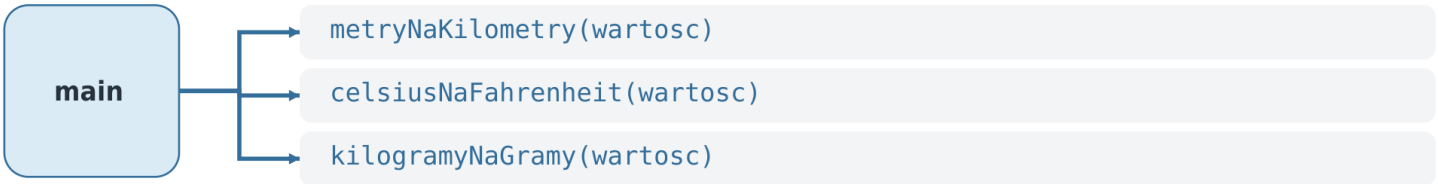
## JAK UŻYĆ METODY?

```
double wynik = metryNaKilometry(3500);
System.out.println(wynik);
```

**WYNIK** **3.5**

## JAK ULEPSZYĆ KONWERTER?

Umieść te trzy metody wewnątrz klasy `Main`, ale poza metodą `main`. W odpowiednich gałęziach `switch` zastąp bezpośrednie obliczenia wywołaniami:




### PRZED — DŁUGIE OBLICZENIA

```
wartosc / 1000.0
```

### PO — CZYTELNE METODY

```
metryNaKilometry(wartosc)
```

Kod główny staje się prostszy, ponieważ nazwa metody wyjaśnia, co robi obliczenie.

# PRZETESTUJ KONWERTER I ZABEZPIECZ DANE

### TEST 01

#### DŁUGOŚĆ

| Dane wejściowe | Oczekiwany wynik |
|---|---|
| 1000 m | 1 km |
| 2500 m | 2,5 km |
| 500 m | 0,5 km |

### TEST 02

#### TEMPERATURA

| Dane wejściowe | Oczekiwany wynik |
|---|---|
| 0°C | 32°F |
| 20°C | 68°F |
| 100°C | 212°F |
| -40°C | -40°F |

### TEST 03

#### MASA

| Dane wejściowe | Oczekiwany wynik |
|---|---|
| 1 kg | 1000 g |
| 2,5 kg | 2500 g |
| 0,5 kg | 500 g |

W terminalu wpisuj same liczby, bez jednostek i z kropką dziesiętną, np. **2.5**.

## DLACZEGO SPRAWDZAMY DANE?

Nie każda wpisana wartość powinna zostać zaakceptowana. W naszym programie długość i masa nie mogą być ujemne. Temperatura w stopniach Celsjusza może być ujemna.

## DODAJ WALIDACJĘ

Po odczytaniu zmiennej `wartosc`, a przed instrukcją `switch`, dodaj:

```
if (!Double.isFinite(wartosc)) {
  System.out.println("Niepoprawna wartosc!");
  return;
}
if ((wybor == 1 || wybor == 3) && wartosc < 0) {
  System.out.println("Wartosc nie moze byc ujemna!");
  return;
}
```

**WYJAŚNIENIE:** `Double.isFinite()` sprawdza, czy liczba jest skończona. Warunek `wybor == 1 || wybor == 3` dotyczy długości i masy.

## SPRAWDŹ SWÓJ PROGRAM

- [ ] Wpisz wartość dodatnią.
- [ ] Wpisz zero.
- [ ] Wpisz liczbę ujemną.
- [ ] Wybierz nieistniejącą opcję.
- [ ] Wpisz tekst zamiast liczby.

**Dobry program nie tylko oblicza poprawne wyniki. Potrafi również rozpoznawać niepoprawne dane.**

# ROZBUDUJ KONWERTER O NOWĄ FUNKCJĘ

## MISJA — DODAJ PRZELICZANIE KILOMETRÓW NA METRY

Twój konwerter obsługuje trzy rodzaje obliczeń. Teraz samodzielnie dodasz kolejną opcję.

### KROK 1 — ROZSZERZ MENU

```
System.out.println(
    "4 - Kilometry na metry");
```



### KROK 2 — ZMIEŃ SPRAWDZANIE WYBORU

Zaktualizuj warunek sprawdzający numer opcji:

```
if (wybor < 1 || wybor > 4) {
    System.out.println(
        "Niepoprawna opcja!");
    return;
}
```

### KROK 3 — UTWÓRZ NOWĄ METODĘ

W klasie `Main`, poza metodą `main`, dodaj:

```
static double kilometryNaMetry(
    double km) {
    return km * 1000.0;
}
```

### KROK 4 — ROZSZERZ SWITCH

Dodaj nowy przypadek:

```
case 4:
    System.out.printf(Locale.US,
        "Wynik: %.2f m%n",
        kilometryNaMetry(wartosc));
    break;
```

### KROK 5 — ZAKTUALIZUJ WALIDACJĘ

W warunku odrzucającym ujemne długości i masy uwzględnij również opcję 4:

```
if ((wybor == 1 || wybor == 3 || wybor == 4) && wartosc < 0) {
    System.out.println("Wartosc nie moze byc ujemna!");
    return;
}
```

Zastąp poprzedni warunek. Zachowaj wcześniejsze sprawdzenie `Double.isFinite(wartosc)`.

### PRZYKŁADOWY WYNIK

```
Wybor: 4
Podaj wartosc: 3.5
Wynik: 3500.00 m
```

### DODATKOWE WYZWANIE

Dodaj kolejne przeliczenia:

- Godziny na minuty.
- Centymetry na metry.
- Litry na mililitry.

### CHECKLISTA UMIEJĘTNOŚCI

- [ ] Rozumiem instrukcję switch.
- [ ] Potrafię tworzyć metody.
- [ ] Umiem korzystać z parametrów.
- [ ] Rozumiem działanie return.
- [ ] Potrafię testować obliczenia.
- [ ] Umiem rozbudowywać menu.

**Gratulacje! Twój trzeci projekt jest gotowy.**

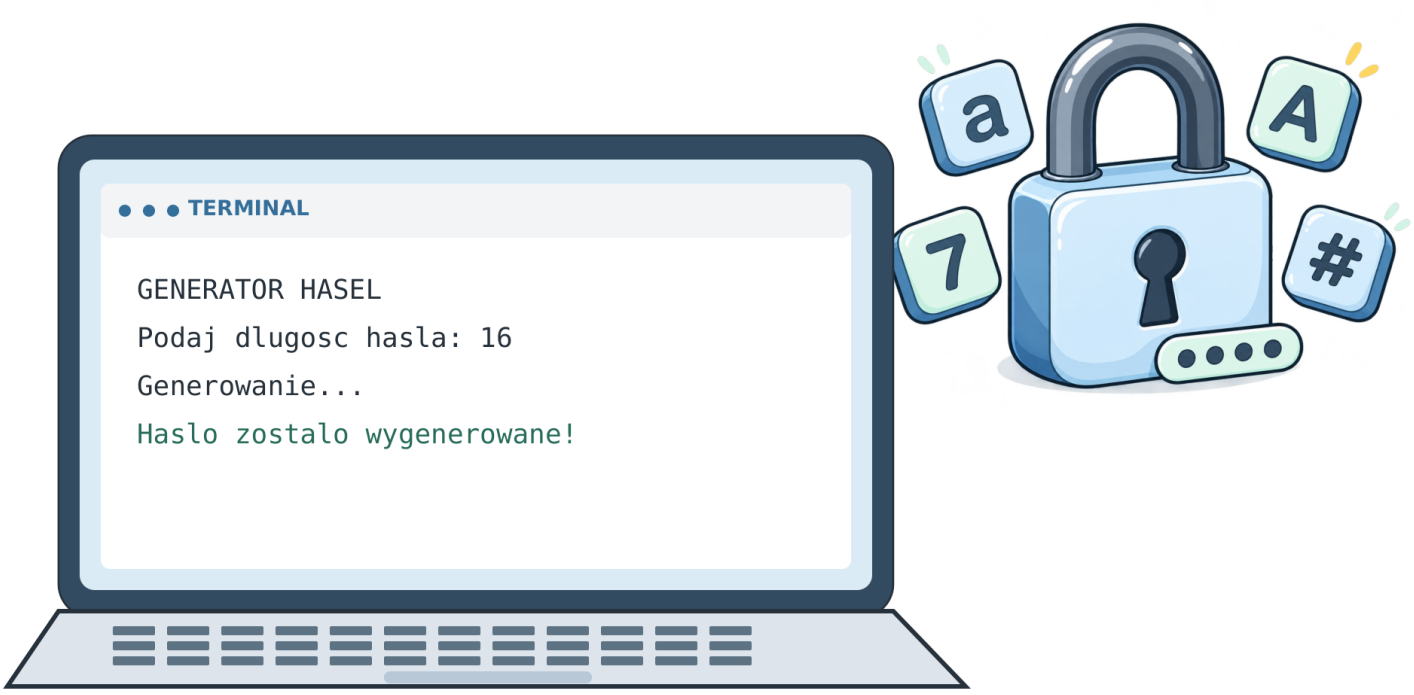
**3 z 10 projektów**

PROJEKT

# 04

# GENERATOR HASEŁ

**Naucz się tworzyć program, który generuje losowe hasła.**




## CZEGO SIĘ NAUCZYSZ?

- Pracować z napisami typu String.
- Tworzyć tekst za pomocą StringBuilder.
- Losować znaki z określonego zbioru.
- Korzystać z SecureRandom.
- Tworzyć pętle for.
- Sprawdzać długość hasła.
- Testować działanie generatora.

## WAŻNA INFORMACJA

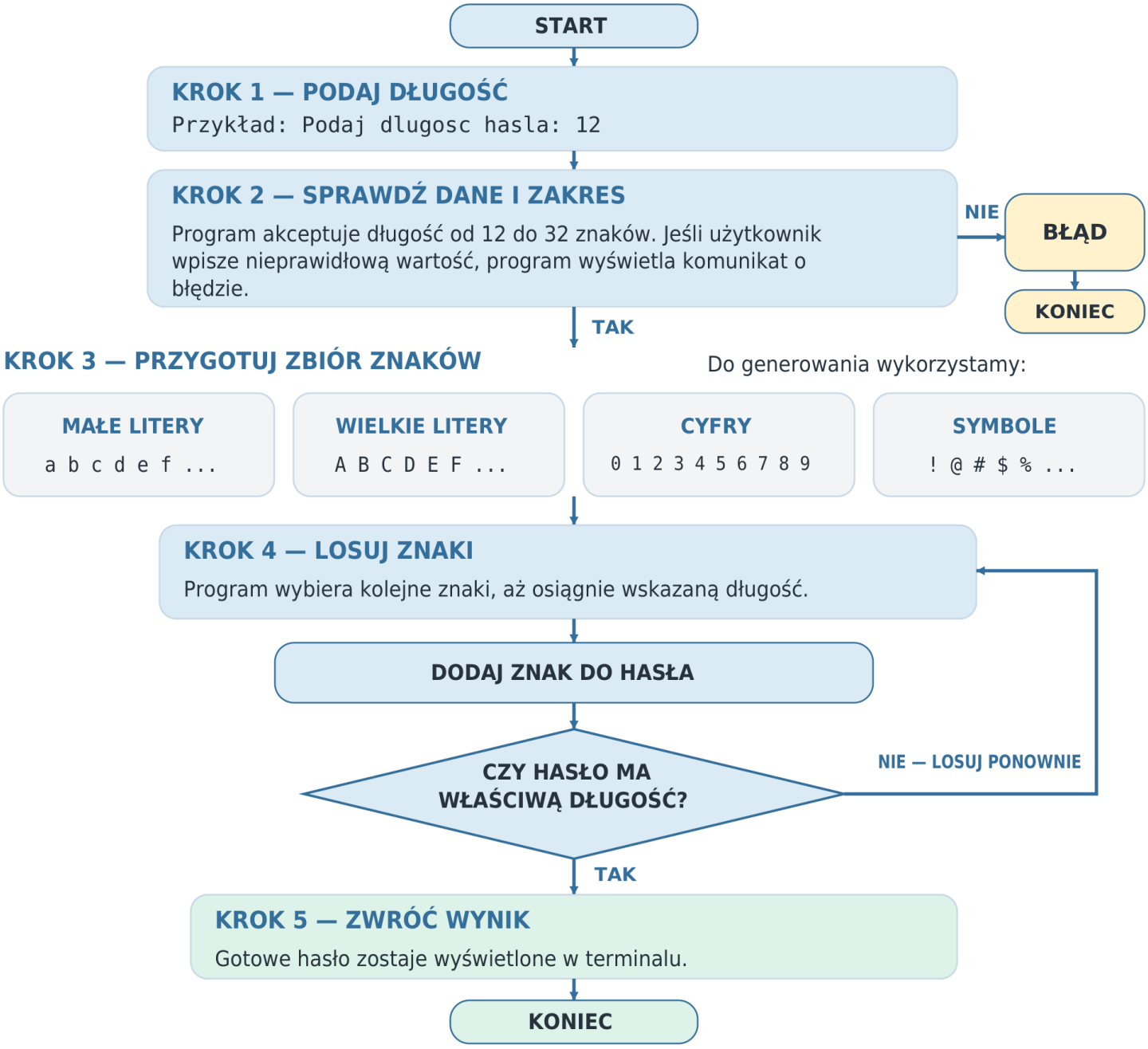
Będziemy korzystać z klasy `SecureRandom`, zaprojektowanej do generowania losowości odpowiedniej do zastosowań związanych z bezpieczeństwem.

**Zbuduj generator krok po kroku i odkryj, jak Java pracuje z tekstem.**

# JAK DZIAŁA GENERATOR LOSOWYCH HASEŁ?

## CEL PROGRAMU

Użytkownik podaje długość hasła. Program losuje znaki z przygotowanego zbioru i tworzy hasło o wybranej długości.




## CO SPRAWIA, ŻE HASŁO JEST TRUDNE DO ODGADNIĘCIA?

Znaczenie mają między innymi długość, odpowiedni sposób losowania oraz wielkość zbioru możliwych znaków.

## UWAGA

Losowanie z całego zbioru nie gwarantuje, że każda wygenerowana sekwencja będzie zawierać wszystkie cztery rodzaje znaków. Generator można później rozbudować o takie wymaganie.

# STRING I STRINGBUILDER — JAK TWORZYĆ TEKST?

## 1. CO TO JEST STRING?

String to typ używany do przechowywania tekstu.

**PRZYKŁAD**

```
String nazwa = "Java";
```

## 2. JAK SPRAWDZIĆ DŁUGOŚĆ TEKSTU?

```
String tekst = "Java";
System.out.println(tekst.length());
```

**WYNIK:** 4

## 3. JAK POBRAĆ JEDEN ZNAK?

```
String tekst = "Java";
char znak = tekst.charAt(0);
System.out.println(znak);
```

**WYNIK:** J

### WYJAŚNIENIE

`charAt(0)` pobiera pierwszy znak napisu. W Javie indeksowanie znaków rozpoczyna się od zera.

## 4. CO TO JEST STRINGBUILDER?

StringBuilder umożliwia wygodne tworzenie tekstu poprzez dodawanie kolejnych znaków.

**PRZYKŁAD**

```
StringBuilder haslo =
    new StringBuilder();
haslo.append("Ab");
haslo.append("12");
haslo.append("!");
System.out.println(haslo.toString());
```

**WYNIK:** Ab12!




### MINIWYZWANIE

Jaki wynik zwróci:

```
StringBuilder napis =
    new StringBuilder();
napis.append("Kod");
napis.append("123");
System.out.println(napis.toString());
```

**ODPOWIEDŹ:** Kod123

## PORÓWNANIE

### STRING

Przechowuje tekst, którego zawartość nie może być zmieniona bez utworzenia nowego napisu.

### STRINGBUILDER

Pozwala stopniowo budować i modyfikować sekwencję znaków.

## DLACZEGO UŻYJEMY STRINGBUILDER?

Ponieważ generator będzie dodawał kolejne wylosowane znaki aż do uzyskania odpowiedniej długości.

To niezależne przykłady. Przy łączeniu kodu zadeklaruj zmienną `tekst` tylko raz.

# LOSUJ ZNAKI ZA POMOCĄ SECURERANDOM

## DLACZEGO SECURERANDOM?

Wcześniej używaliśmy klasy Random do stworzenia prostej gry. W generatorze haseł zastosujemy SecureRandom, ponieważ jest przeznaczony do zastosowań wymagających silniejszej, trudniejszej do przewidzenia losowości.

### KROK 1 — IMPORTUJ KLASĘ

```
import java.security.SecureRandom;
```

### KROK 2 — UTWÓRZ GENERATOR

```
SecureRandom random = new SecureRandom();
```

### KROK 3 — PRZYGOTUJ ZBIÓR ZNAKÓW

```
String znaki = "abcdefghijklmnopqrstuvwxyz"
    + "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    + "0123456789!@#$%";
```

### KROK 4 — WYLOSUJ POZYCJĘ ZNAKU

```
int indeks = random.nextInt(znaki.length());
```

### KROK 5 — POBIERZ WYLOSOWANY ZNAK

```
char znak = znaki.charAt(indeks);
```

### CO OZNACZA TEN KOD?

`znaki.length()` — liczba dostępnych znaków.

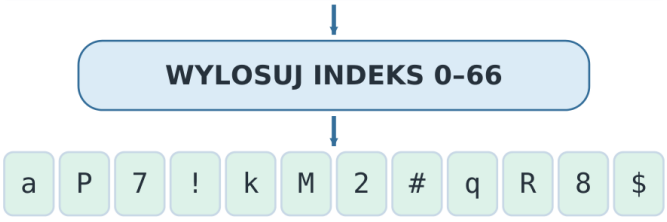
`random.nextInt(...)` — losuje indeks od zera do liczby znaków minus jeden.

`znaki.charAt(indeks)` — pobiera znak znajdujący się pod wybranym indeksem.

### ZBIÓR → INDEKS → ZNAK

**67 DOSTĘPNYCH ZNAKÓW**

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!@#$%
```




Przykładowa sekwencja: **12 znaków**.

## POŁĄCZ WSZYSTKO W PĘTLI

Zachowaj kroki 1–3. Poniższa pętla zastępuje kroki 4 i 5 — usuń wcześniejsze deklaracje `indeks` i `znak`.

```
StringBuilder haslo = new StringBuilder();
for (int i = 0; i < 12; i++) {
    int indeks = random.nextInt(znaki.length());
    haslo.append(znaki.charAt(indeks));
}
System.out.println(haslo.toString());
```

## WYJAŚNIENIE PĘTLI FOR

`int i = 0` — wartość początkowa licznika.
`i < 12` — warunek wykonywania kolejnych powtórzeń.
`i++` — zwiększenie licznika o jeden.

**WYNIK:** Program tworzy napis zawierający 12 losowo wybranych znaków.

**Ważne:** Każde uruchomienie może dać inny wynik. Pokazany zbiór znaków nie gwarantuje obecności każdego rodzaju znaku w pojedynczym haśle.

# STWÓRZ WŁASNY GENERATOR HASEŁ W JAVIE

## KROK 1 — UTWÓRZ PLIK

`Main.java`

## KROK 2 — WPISZ KOMPLETNY KOD

```java
import java.security.SecureRandom;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    SecureRandom random = new SecureRandom();
    String znaki = "abcdefghijklmnopqrstuvwxyz"
      + "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      + "0123456789!@#$%";
    System.out.println("GENERATOR HASEL");
    System.out.print("Podaj dlugosc hasla (12-32): ");
    if (!scanner.hasNextInt()) {
      System.out.println("Wpisz liczbe calkowita!");
      return;
    }
    int dlugosc = scanner.nextInt();
    if (dlugosc < 12 || dlugosc > 32) {
      System.out.println("Wybierz od 12 do 32 znakow!");
      return;
    }
    StringBuilder haslo = new StringBuilder();
    for (int i = 0; i < dlugosc; i++) {
      int indeks = random.nextInt(znaki.length());
      haslo.append(znaki.charAt(indeks));
    }
    System.out.println("Wygenerowane haslo:");
    System.out.println(haslo.toString());
  }
}
```

## KROK 3 — URUCHOM PROGRAM

```
javac Main.java
java Main
```

## PRZYKŁADOWY WYNIK

```
GENERATOR HASEL
Podaj dlugosc hasla (12-32): 16
Wygenerowane haslo:
aP7!kM2#qR8$zT4@
```

## CO ROBI PROGRAM?

1. Sprawdza, czy użytkownik podał liczbę całkowitą.
2. Akceptuje długość od 12 do 32 znaków.
3. Tworzy generator SecureRandom.
4. Wielokrotnie losuje znaki z przygotowanego zbioru.
5. Buduje tekst za pomocą StringBuilder.
6. Wyświetla wygenerowane hasło.

## UWAGA DOTYCZĄCA BEZPIECZEŃSTWA

Hasło wyświetlone w terminalu może zostać zauważone przez inne osoby lub zapisane w historii sesji. Nie udostępniaj rzeczywistych haseł i nie zapisuj ich w kodzie programu.

Ten program jest edukacyjnym generatorem znaków. Nie zapewnia automatycznie wszystkich możliwych wymagań polityki haseł.

# SPRAWDŹ, CZY GENERATOR DZIAŁA POPRAWNIE

### TEST 01 — MINIMALNA DŁUGOŚĆ

Użytkownik wpisuje: **12**

**Oczekiwany wynik:** Program generuje hasło składające się dokładnie z 12 znaków.

### TEST 02 — WIĘKSZA DŁUGOŚĆ

Użytkownik wpisuje: **24**

**Oczekiwany wynik:** Program generuje hasło składające się dokładnie z 24 znaków.

### TEST 03 — ZBYT KRÓTKIE HASŁO

Użytkownik wpisuje: **5**

**Oczekiwany wynik:** Program informuje o dozwolonym zakresie i nie generuje hasła.

### TEST 04 — ZBYT DŁUGIE HASŁO

Użytkownik wpisuje: **50**

**Oczekiwany wynik:** Program odrzuca nieprawidłową długość.

### TEST 05 — NIEPRAWIDŁOWY TYP DANYCH

Użytkownik wpisuje: **abc**

**Oczekiwany wynik:** Program wyświetla komunikat: `Wpisz liczbe calkowita!`

## TABELA TESTÓW

| Dane wejściowe | Oczekiwane zachowanie |
|---|---|
| 12 | Hasło 12 znaków |
| 16 | Hasło 16 znaków |
| 32 | Hasło 32 znaków |
| 5 | Komunikat błędu |
| 50 | Komunikat błędu |
| abc | Komunikat błędu |

## CO JESZCZE WARTO SPRAWDZIĆ?

- [ ] Czy hasło ma dokładnie wybraną długość?
- [ ] Czy wszystkie znaki należą do dozwolonego zbioru?
- [ ] Czy program odrzuca nieprawidłowe wartości?
- [ ] Czy wielokrotne uruchomienia zwykle dają różne wyniki?

### WAŻNE OGRANICZENIE

Program losuje każdy znak z całego zbioru. Nie gwarantuje więc, że pojedyncze hasło zawiera jednocześnie małe litery, wielkie litery, cyfry i symbole.

Różne wyniki są oczekiwane, ale sam fakt otrzymania dwóch identycznych haseł nie dowodzi błędu generatora.

## DODATKOWA ANALIZA

Przyjrzyj się pętli:

```
for (int i = 0; i < dlugosc; i++) {
  int indeks = random.nextInt(znaki.length());
  haslo.append(znaki.charAt(indeks));
}
```

## ODPOWIEDZ

1. Dlaczego pętla rozpoczyna się od zera?
2. Co zmieni się, jeśli wartość `dlugosc` wyniesie 20?
3. Dlaczego używamy StringBuilder?
4. Czy każda wygenerowana sekwencja musi mieć wszystkie rodzaje znaków?

## ODPOWIEDZI KONTROLNE

1. Przy warunku `i < dlugosc` licznik przyjmuje wartości od 0 do `dlugosc - 1`, co daje dokładnie `dlugosc` powtórzeń.
2. Program wygeneruje 20 znaków.
3. Aby wygodnie dodawać kolejne znaki do napisu.
4. Nie, ponieważ każdy znak jest losowany niezależnie z całego zbioru.

# JAK ZAGWARANTOWAĆ RÓŻNE RODZAJE ZNAKÓW W HAŚLE?

## PROBLEM

Nasz generator losuje znaki z dużego zbioru. Nie mamy jednak gwarancji, że wygenerowane hasło zawiera wszystkie rodzaje znaków.

## ROZWIĄZANIE

Najpierw wybierzemy po jednym znaku z każdej grupy, a następnie uzupełnimy hasło pozostałymi znakami. Na koniec wymieszamy ich kolejność.

### KROK 1 — PRZYGOTUJ GRUPY ZNAKÓW

```
String male = "abcdefghijklmnopqrstuvwxyz";
String duze = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
String cyfry = "0123456789";
String symbole = "!@#$%";
String wszystkie = male + duze + cyfry + symbole;
```

### KROK 2 — DODAJ PO JEDNYM ZNAKU Z KAŻDEJ GRUPY

```
StringBuilder haslo = new StringBuilder();
haslo.append(male.charAt(random.nextInt(male.length())));
haslo.append(duze.charAt(random.nextInt(duze.length())));
haslo.append(cyfry.charAt(random.nextInt(cyfry.length())));
haslo.append(symbole.charAt(random.nextInt(symbole.length())));
```

### KROK 3 — UZUPEŁNIJ HASŁO

```
while (haslo.length() < dlugosc) {
  int indeks =
      random.nextInt(wszystkie.length());
  haslo.append(wszystkie.charAt(indeks));
}
```

### KROK 4 — WYMIESZAJ KOLEJNOŚĆ ZNAKÓW

```
for (int i = haslo.length() - 1;
     i > 0; i--) {
  int j = random.nextInt(i + 1);
  char temp = haslo.charAt(i);
  haslo.setCharAt(i, haslo.charAt(j));
  haslo.setCharAt(j, temp);
}
```

### JAK TO DZIAŁA?

- Program losuje co najmniej jeden znak z każdej grupy.
- Uzupełnia hasło do wybranej długości.
- Zamienia losowo pozycje znaków.
- Zachowuje wymagane rodzaje znaków.

## WAŻNE

Wstaw ten kod w miejsce wcześniejszego bloku budowania hasła. Zachowaj obiekt `SecureRandom random`, odczyt i sprawdzanie długości **12–32** oraz wyświetlanie wyniku. Zostaw tylko jedną deklarację `haslo`.

Do mieszania stosujemy algorytm Fishera–Yatesa, wykorzystując ten sam generator `SecureRandom`.

| małe litery | wielkie litery | cyfry | symbole | → PRZYKŁADOWY REZULTAT |
|---|---|---|---|---|
| a–z | A–Z | 0–9 | !@#$% | **m8#Pq2!Zk5Rt** |

**MIESZANIE — FRAGMENT:** a A 7 ! → 7 ! a A

# ROZBUDUJ GENERATOR I ZAKOŃCZ CZWARTY PROJEKT

## MISJA — DODAJ MOŻLIWOŚĆ WYBORU LICZBY HASEŁ

Użytkownik powinien móc wygenerować kilka losowo wygenerowanych haseł podczas jednego uruchomienia programu.

## KROK 1 — ZAPYTAJ O LICZBĘ HASEŁ

```
System.out.print(
    "Ile hasel wygenerowac (1-5)? ");
if (!scanner.hasNextInt()) {
  System.out.println("Wpisz poprawna liczbe!");
  return;
}
int ile = scanner.nextInt();
```

## KROK 2 — SPRAWDŹ ZAKRES

```
if (ile < 1 || ile > 5) {
  System.out.println(
      "Wybierz od 1 do 5 hasel!");
  return;
}
```

## KROK 3 — POWTÓRZ GENEROWANIE

**Szablon pętli — uzupełnij wskazane miejsca.**

```
for (int numer = 1; numer <= ile; numer++) {
  // Tutaj umiesc kod generowania
  // jednego hasla.
  // Tworz nowy StringBuilder
  // w kazdym powtorzeniu.
  System.out.print("Haslo nr " + numer + ": ");
  // Wyswietl haslo metoda println.
}
```

## INSTRUKCJA

Przenieś do wnętrza pętli kod tworzenia hasła z poprzedniej strony, wraz z losowaniem, uzupełnianiem, mieszaniem znaków i wyświetlaniem hasła. Usuń dawne wyświetlanie hasła poza pętlą. Nie przenoś sprawdzania długości ani odczytu danych do wnętrza pętli — wystarczy wykonać je raz.

## PRZYKŁADOWY EFEKT

```
Ile hasel wygenerowac (1-5)? 3
Haslo nr 1: [wygenerowane haslo]
Haslo nr 2: [wygenerowane haslo]
Haslo nr 3: [wygenerowane haslo]
```

## DODATKOWE WYZWANIA

### WYZWANIE A — ZMIEŃ DOZWOLONĄ DŁUGOŚĆ

Pozwól generować hasła o długości od **16 do 64 znaków**. Zmień jednocześnie warunek zakresu, pytanie o długość i komunikat błędu.

### WYZWANIE B — DODAJ NOWE SYMBOLE

Rozszerz zbiór o: & * ? _ -

### WYZWANIE C — UTWÓRZ METODĘ

Przenieś algorytm do metody w klasie Main, poza metodą main.

### SZABLON DO UZUPEŁNIENIA

```
static String generujHaslo(
    int dlugosc, SecureRandom random) {
  StringBuilder haslo = new StringBuilder();
  // Tutaj dodaj algorytm generowania.
  return haslo.toString();
}
```

W tej metodzie zmienna haslo powinna zostać utworzona i wypełniona wewnątrz jej ciała. Skopiuj ze strony 40 grupy znaków, losowanie, uzupełnianie i mieszanie. Nie powtarzaj deklaracji haslo. **Bez tych instrukcji metoda zwraca pusty napis.**

## CHECKLISTA UMIEJĘTNOŚCI

- [ ] Rozumiem różnicę między String i StringBuilder.
- [ ] Potrafię pobierać pojedyncze znaki.
- [ ] Umiem korzystać z SecureRandom.
- [ ] Potrafię tworzyć pętle for i while.
- [ ] Rozumiem walidację długości.
- [ ] Potrafię zmieniać i rozbudowywać istniejący program.

### WAŻNA UWAGA

W rzeczywistych zastosowaniach nie należy wyświetlać ani przechowywać haseł bez uwzględnienia bezpieczeństwa środowiska. Ten program jest ćwiczeniem edukacyjnym.

**Gratulacje! Ukończyłeś czwarty projekt w Javie.**

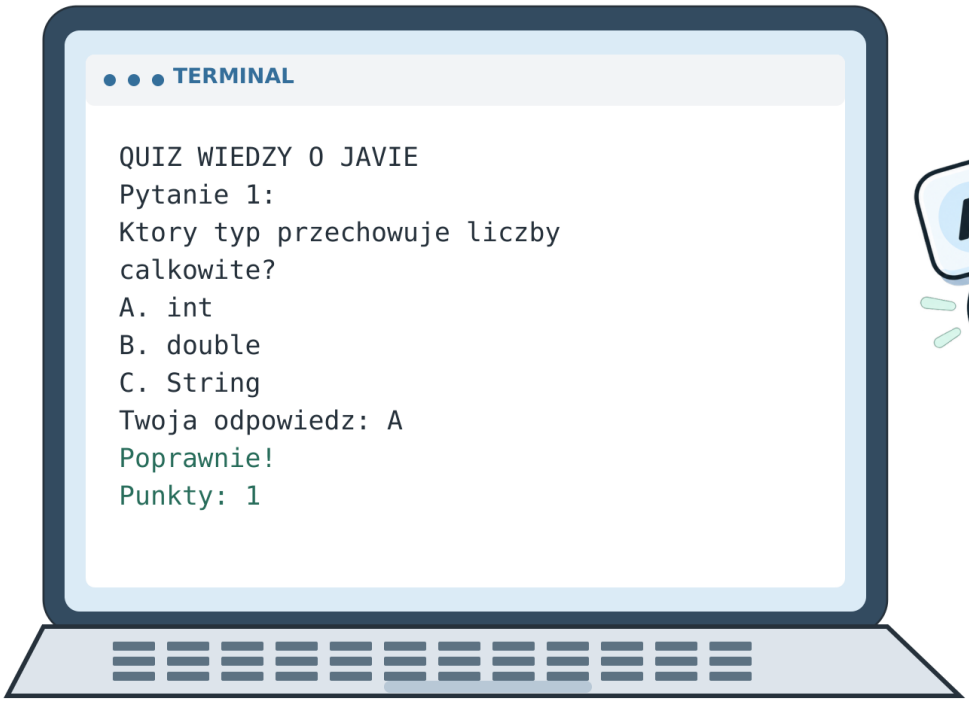
**4 z 10 projektów**

PROJEKT

05

# QUIZ WIEDZY

**Zbuduj interaktywną grę, która zadaje pytania, sprawdza odpowiedzi i liczy punkty!**







## CZEGO SIĘ NAUCZYSZ?

- Przechowywania wielu pytań.
- Korzystania z tablic.
- Wyświetlania odpowiedzi.
- Porównywania tekstów.
- Tworzenia systemu punktacji.
- Obliczania wyniku końcowego.
- Organizowania interaktywnego programu.

**Teraz Twój program będzie nie tylko wykonywał polecenia, ale również sprawdzał wiedzę użytkownika!**

# JAK ZAPLANOWAĆ WŁASNY QUIZ W JAVIE?

## CEL PROGRAMU

Quiz wyświetla pytania o podstawach Javy. Użytkownik wybiera odpowiedź A, B lub C. Za każdą poprawną odpowiedź otrzymuje jeden punkt. Po zakończeniu wszystkich pytań program pokazuje wynik końcowy.

### KROK 1 — PRZYGOTUJ PYTANIA

Każde pytanie powinno mieć:

- Treść pytania.
- Trzy możliwe odpowiedzi.
- Jedną poprawną odpowiedź.

### KROK 2 — WYŚWIETL PYTANIE

Program pokazuje pierwsze pytanie oraz odpowiedzi A, B i C.

### KROK 3 — ODCZYTAJ ODPOWIEDŹ

Użytkownik wpisuje wybraną literę.

### KROK 4 — SPRAWDŹ POPRAWNOŚĆ

Program porównuje odpowiedź użytkownika z poprawną odpowiedzią.

### KROK 5 — DODAJ PUNKT

Jeśli odpowiedź jest prawidłowa, zwiększa wynik o jeden.

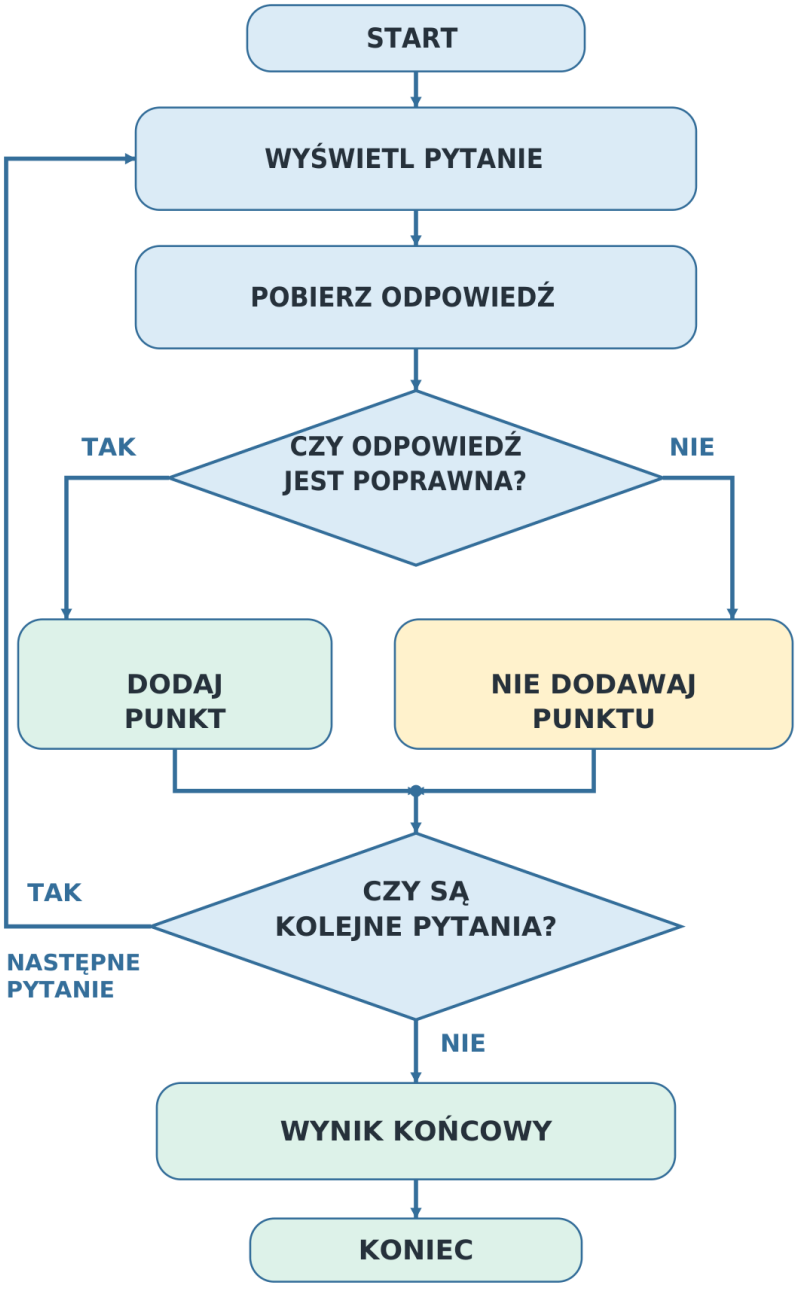
### KROK 6 — PRZEJDŹ DO KOLEJNEGO PYTANIA

Program powtarza proces, aż wyświetli wszystkie pytania.

### KROK 7 — POKAŻ WYNIK

Na koniec pokazuje liczbę zdobytych punktów i procent poprawnych odpowiedzi.

## PRZEPŁYW PROGRAMU




### WAŻNE POJĘCIE

Punktacja to mechanizm, który mierzy liczbę poprawnych odpowiedzi użytkownika.

### PRZYKŁAD

Liczba pytań: **5**
Poprawne odpowiedzi: **4**
Wynik: **4/5**
Procent: **80%**

# JAK TWORZYĆ PYTANIA I SPRAWDZAĆ ODPOWIEDZI?

## KROK 1 — UTWÓRZ PYTANIE

```
String pytanie =
    "Ktory typ przechowuje liczby calkowite?";
```

## KROK 2 — PRZYGOTUJ ODPOWIEDZI

```
String odpowiedzA = "int";
String odpowiedzB = "double";
String odpowiedzC = "String";
```

## KROK 3 — USTAL POPRAWNĄ ODPOWIEDŹ

```
String poprawna = "A";
```

## KROK 4 — WYŚWIETL PYTANIE

```
System.out.println(pytanie);
System.out.println("A. " + odpowiedzA);
System.out.println("B. " + odpowiedzB);
System.out.println("C. " + odpowiedzC);
```

## KROK 5 — ODCZYTAJ ODPOWIEDŹ

Na początku metody `main` utwórz obiekt `Scanner`:

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

Następnie:

```
System.out.print("Twoja odpowiedz: ");
String wybor = scanner.nextLine().trim();
```

## KROK 6 — SPRAWDŹ WYNIK

```
if (wybor.equalsIgnoreCase(poprawna)) {
    System.out.println("Poprawnie!");
} else {
    System.out.println("Niepoprawna odpowiedz.");
}
```

## UMIEJSCOWIENIE KODU

Na początku pliku `Main.java`, ponad klasą `Main`, dodaj:

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

Pozostałe fragmenty tej strony umieść wewnątrz metody `main` w klasie `Main`.

### PYTANIE

**Ktory typ przechowuje liczby calkowite?**

| A | B | C |
|---|---|---|
| int | double | String |

## WAŻNE POJĘCIA

**`equalsIgnoreCase()`**

Porównuje dwa napisy bez rozróżniania wielkich i małych liter.

**`trim()`**

Usuwa zbędne spacje na początku i końcu tekstu.

**`Scanner`**

Pozwala odczytywać odpowiedzi wpisywane przez użytkownika.

## PRZYKŁAD DZIAŁANIA

Odpowiedź użytkownika: **a**
Poprawna odpowiedź: **A**
Wynik: **Poprawnie!**

## DLACZEGO NIE UŻYWAMY == DO PORÓWNYWANIA NAPISÓW?

Operator == porównuje referencje obiektów typu `String`, a nie ich zawartość tekstową. Do sprawdzania treści odpowiedzi używamy `equals()` lub `equalsIgnoreCase()`.

# TABLICE W JAVIE — PRZECHOWUJ WIELE PYTAŃ

## CO TO JEST TABLICA?

Tablica to struktura przechowująca określoną liczbę elementów tego samego typu. W Javie indeksy tablic rozpoczynają się od zera.

## PRZYKŁAD PROSTEJ TABLICY

```
String[] kolory = {"czerwony", "zielony", "niebieski"};
System.out.println(kolory[0]);
```

**WYNIK**

```
czerwony
```

## INDEKSY ŁĄCZĄ PYTANIA, OPCJE I ODPOWIEDZI

| Indeks | Pytanie | Odpowiedzi | Poprawna |
|---|---|---|---|
| 0 | Jaki typ przechowuje liczby calkowite? | A. int<br>B. double<br>C. String | A |
| 1 | Ktora instrukcja tworzy petle? | A. if<br>B. while<br>C. class | B |
| 2 | Ktory operator porownuje rownosc? | A. =<br>B. ==<br>C. ! | B |

## PRZYGOTUJ TABLICĘ PYTAŃ

```
String[] pytania = {
    "Jaki typ przechowuje liczby calkowite?",
    "Ktora instrukcja tworzy petle?",
    "Ktory operator porownuje rownosc?"
};
```

## PRZYGOTUJ ODPOWIEDZI DO WYBORU

```
String[][] opcje = {
    {"A. int", "B. double", "C. String"},
    {"A. if", "B. while", "C. class"},
    {"A. =", "B. ==", "C. !"}
};
```

## PRZYGOTUJ KLUCZ ODPOWIEDZI

```
char[] poprawne = {'A', 'B', 'B'};
```

## DLACZEGO TO DZIAŁA?

Elementy o tym samym indeksie dotyczą tego samego pytania. Na przykład:

- `pytania[1]` — drugie pytanie.
- `opcje[1]` — odpowiedzi do drugiego pytania.
- `poprawne[1]` — prawidłowa odpowiedź na drugie pytanie.

## JAK ODCZYTAĆ DANE?

### PIERWSZE PYTANIE

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

### PIERWSZA OPCJA PIERWSZEGO PYTANIA

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

### POPRAWNA ODPOWIEDŹ NA PIERWSZE PYTANIE

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

## WYJAŚNIENIE

`String[]` — tablica napisów.

`String[][]` — tablica dwuwymiarowa, która tutaj przechowuje listę trzech opcji dla każdego pytania.

`char[]` — tablica pojedynczych znaków.

## MINIWYZWANIE

Jaka wartość znajduje się w `poprawne[2]`?
**Odpowiedź: B**

# LICZ PUNKTY I OBLICZAJ WYNIK KOŃCOWY

Rozwijaj jedną pętlę: kroki 3 i 4 umieść wewnątrz pętli z kroku 2. Kroki 5 i 6 wykonaj dopiero po jej zakończeniu.

## KROK 1 — UTWÓRZ LICZNIK

```
int punkty = 0;
```

Na początku gracz ma zero punktów.

## KROK 2 — POWTARZAJ PYTANIA

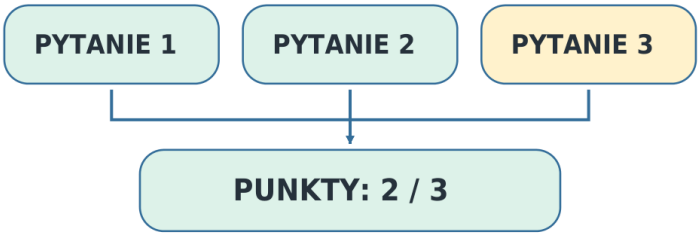
```
for (int i = 0; i < pytania.length; i++) {
  System.out.println(pytania[i]);
}
```

## KROK 3 — WYŚWIETL OPCJE

Wewnątrz pętli pytań:

```
for (String opcja : opcje[i]) {
  System.out.println(opcja);
}
```

## TRZY PYTANIA → WSPÓLNY LICZNIK




## KROK 4 — SPRAWDŹ ODPOWIEDŹ

Po wczytaniu i normalizacji odpowiedzi do zmiennej `wybor`:

```
String wybor = scanner.nextLine()
    .trim().toUpperCase(Locale.ROOT);
if (wybor.equals(
    String.valueOf(poprawne[i]))) {
  punkty++;
  System.out.println("Poprawnie!");
} else {
  System.out.println("Niepoprawnie!");
}
```

## KROK 5 — OBLICZ PROCENT

Po zakończeniu wszystkich pytań:

```
double procent = (double) punkty
    / pytania.length * 100;
```

## KROK 6 — WYŚWIETL WYNIK

```
System.out.println("Punkty: " + punkty);
System.out.printf("Wynik: %.0f%%%n", procent);
```

## WYJAŚNIENIE

`punkty++` zwiększa wynik o jeden.

`pytania.length` podaje liczbę elementów w tablicy.

`(double)` umożliwia wykonanie dzielenia zmiennoprzecinkowego.

`%%` w formacie `printf` oznacza dosłowny znak procenta.

## PRZYKŁAD

Liczba pytań: **3** · Poprawne odpowiedzi: **2**

**≈ 67%**

**Ważne:** Przed tymi fragmentami utwórz tablice z poprzedniej strony i odczytuj wybór użytkownika. Dodaj `import java.util.Locale;` na początku pliku, aby użyć normalizacji `toUpperCase(Locale.ROOT)`. Pełna, połączona wersja kodu znajduje się na następnej stronie.

# NAPISZ I URUCHOM KOMPLETNY QUIZ W JAVIE

## KROK 1 — UTWÓRZ PLIK

`Main.java`

## KROK 2 — WPISZ KOMPLETNY KOD

```java
import java.util.Locale;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String[] pytania = {
      "Jaki typ przechowuje liczby calkowite?",
      "Ktora instrukcja tworzy petle?",
      "Ktory operator porownuje rownosc?"
    };
    String[][] opcje = {
      {"A. int", "B. double", "C. String"},
      {"A. if", "B. while", "C. class"},
      {"A. =", "B. ==", "C. !"}
    };
    char[] poprawne = {'A', 'B', 'B'};
    int punkty = 0;
    System.out.println("QUIZ WIEDZY O JAVIE");
    for (int i = 0; i < pytania.length; i++) {
      System.out.println("\nPytanie " + (i + 1));
      System.out.println(pytania[i]);
      for (String opcja : opcje[i]) {
        System.out.println(opcja);
      }
      System.out.print("Odpowiedz (A/B/C): ");
      String wybor = scanner.nextLine()
        .trim().toUpperCase(Locale.ROOT);
      if (!wybor.equals("A") && !wybor.equals("B")
        && !wybor.equals("C")) {
        System.out.println("Niepoprawny wybor!");
        continue;
      }
      if (wybor.equals(String.valueOf(poprawne[i]))) {
        punkty++;
        System.out.println("Poprawnie!");
      } else {
        System.out.println("Niepoprawnie!");
      }
    }
    double procent = (double) punkty / pytania.length * 100;
    System.out.println("\nKONIEC QUIZU");
    System.out.println("Punkty: " + punkty + "/" + pytania.length);
    System.out.printf(Locale.US, "Wynik: %.0f%%%n", procent);
  }
}
```

## KROK 3 — URUCHOM

```
javac Main.java
java Main
```

## PRZYKŁADOWY WYNIK KOŃCOWY

```
KONIEC QUIZU
Punkty: 2/3
Wynik: 67%
```

## JAK DZIAŁA TEN PROGRAM?

1. Przechowuje pytania i odpowiedzi w tablicach.
2. Wyświetla kolejne pytania.
3. Odczytuje odpowiedź użytkownika.
4. Akceptuje litery A, B lub C.
5. Dodaje punkt za poprawną odpowiedź.
6. Oblicza procent uzyskanych punktów.
7. Wyświetla wynik końcowy.

**WAŻNA INFORMACJA:** Nieprawidłowa odpowiedź, np. **D** albo pusty tekst, powoduje przejście do następnego pytania bez przyznania punktu. W kolejnych rozszerzeniach można dodać ponawianie pytania po błędnym wpisie.

# SPRAWDŹ, CZY QUIZ POPRAWNIE LICZY PUNKTY

## TABELA TESTÓW

| Odpowiedzi | Punkty | Wynik |
|---|---|---|
| A, B, B | 3/3 | 100% |
| A, B, C | 2/3 | 67% |
| C, A, A | 0/3 | 0% |
| a, b, b | 3/3 | 100% |

### TEST 01 — WSZYSTKIE ODPOWIEDZI POPRAWNE

Wpisz kolejno: **A, B, B**
**Oczekiwany wynik: 3/3 punkty · 100%**.

### TEST 02 — DWIE POPRAWNE ODPOWIEDZI

Wpisz kolejno: **A, B, C**
**Oczekiwany wynik: 2/3 punkty · 67%** po zaokrągleniu.

### TEST 03 — BRAK POPRAWNYCH ODPOWIEDZI

Wpisz kolejno: **C, A, A**
**Oczekiwany wynik: 0/3 punkty · 0%**.

### TEST 04 — MAŁE LITERY

Wpisz kolejno: **a, b, b**
**Oczekiwany wynik: 3/3 punkty · 100%**.
Program nie powinien rozróżniać wielkości liter.

### TEST 05 — BŁĘDNA OPCJA

Wpisz **D** przy pierwszym pytaniu.
**Oczekiwany wynik:** Program informuje o nieprawidłowym wyborze, nie przyznaje punktu i przechodzi do kolejnego pytania.

## NAJCZĘSTSZE PROBLEMY

### PROBLEM 1 — PROGRAM NIE DODAJE PUNKTU

Sprawdź, czy `punkty++` znajduje się w gałęzi poprawnej odpowiedzi.

### PROBLEM 2 — BŁĘDNE PYTANIE

Sprawdź indeks `i` i powiązanie tablic.

### PROBLEM 3 — NIEPOPRAWNY PROCENT

Sprawdź, czy przed dzieleniem używasz `(double)`.

### PROBLEM 4 — PROGRAM NIE ROZPOZNAJE MAŁYCH LITER

Sprawdź wywołanie `toUpperCase(Locale.ROOT)`.

### PROBLEM 5 — NIEPOPRAWNY INDEKS

Upewnij się, że tablice pytań, opcji i odpowiedzi mają zgodną liczbę elementów.

## MINIWYZWANIE

Zmień jedno pytanie i jedną poprawną odpowiedź, a następnie sprawdź, czy program nadal poprawnie liczy punkty.

# ROZBUDUJ QUIZ I STWÓRZ WŁASNĄ GRĘ EDUKACYJNĄ

## MISJA 1 — DODAJ NOWE PYTANIA

Rozbuduj quiz do pięciu pytań. Dodaj przykładowo:

### PYTANIE 4 · INDEKS 3

Które słowo kluczowe służy do definiowania klasy w Javie?
A. class
B. method
C. repeat
**Poprawna odpowiedź: A.**

### PYTANIE 5 · INDEKS 4

Który typ danych przechowuje wartość true lub false?
A. double
B. boolean
C. char
**Poprawna odpowiedź: B.**

**CO MUSISZ ZMIENIĆ?** Dodaj nowe elementy do trzech tablic: `pytania`, `opcje` i `poprawne`. Zachowaj zgodność indeksów. Zastąp dotychczasowy klucz odpowiedzi:

```
char[] poprawne = {'A', 'B', 'B', 'A', 'B'};
```

## MISJA 2 — DODAJ OCENĘ WYNIKU

Po zakończeniu quizu dopisz kod:

```
if (procent >= 80) {
  System.out.println("Swietny wynik!");
} else if (procent >= 50) {
  System.out.println("Dobra robota!");
} else {
  System.out.println("Cwicz dalej!");
}
```

### JAK TO DZIAŁA?

| | |
|---|---|
| **80–100%** | Świetny wynik! |
| 50% ≤ wynik < 80% | Dobra robota! |
| Poniżej 50% | Ćwicz dalej! |

## MISJA 3 — ZMIEŃ TEMAT QUIZU

Przygotuj własną wersję gry. Możliwe tematy:

- Podstawy programowania.
- Matematyka.
- Geografia.
- Wiedza ogólna.
- Historia.

### DODATKOWE WYZWANIE

Dodaj możliwość ponownego rozpoczęcia quizu po zakończeniu rozgrywki. Zastanów się, w jaki sposób wykorzystać pętlę `while`, aby uruchamiać kolejne rundy.

## CHECKLISTA UMIEJĘTNOŚCI

- [ ] Rozumiem tablice `String[]`.
- [ ] Potrafię korzystać z tablic dwuwymiarowych.
- [ ] Umiem przechowywać poprawne odpowiedzi.
- [ ] Potrafię porównywać napisy.
- [ ] Rozumiem pętlę `for`.
- [ ] Umiem tworzyć licznik punktów.
- [ ] Potrafię obliczać wynik procentowy.
- [ ] Umiem rozbudować istniejący program.

**Gratulacje! Ukończyłeś już 5 z 10 projektów w Javie.**

**5 z 10 projektów**

### POŁOWA DROGI ZA TOBĄ!

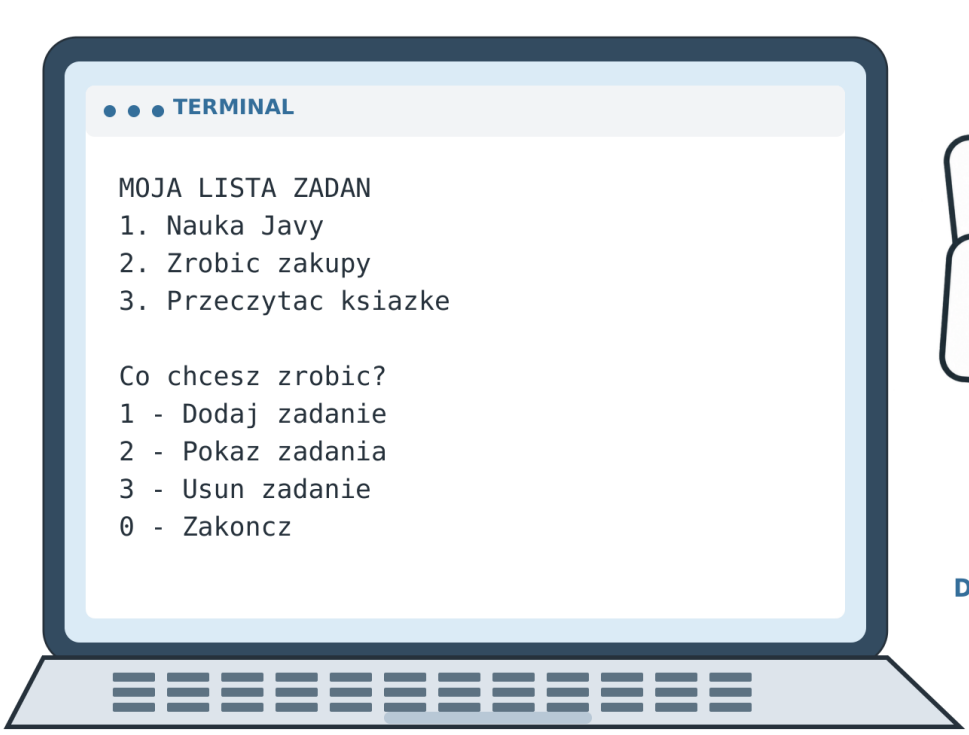
Przed Tobą kolejne projekty, w których nauczysz się tworzyć bardziej rozbudowane programy i zarządzać danymi.

PROJEKT

# 06

# LISTA ZADAŃ

**Stwórz własną aplikację do organizowania codziennych obowiązków!**







## CZEGO SIĘ NAUCZYSZ?

- Korzystania z klasy ArrayList.
- Przechowywania wielu napisów.
- Dodawania nowych elementów.
- Wyświetlania zawartości listy.
- Usuwania wybranych zadań.
- Tworzenia interaktywnego menu.
- Obsługi niepoprawnych danych.

**Twój pierwszy program do zarządzania danymi!**

# JAK DZIAŁA APLIKACJA DO ZARZĄDZANIA ZADANIAMI?

## CEL PROGRAMU

Użytkownik powinien mieć możliwość dodawania nowych zadań, przeglądania listy oraz usuwania niepotrzebnych pozycji. Program będzie działał w terminalu i wyświetlał menu po każdej operacji.

## KROK 1 — UTWÓRZ PUSTĄ LISTĘ

Na początku program tworzy miejsce do przechowywania zadań.

## KROK 2 — WYŚWIETL MENU

```
1 - Dodaj zadanie
2 - Pokaz zadania
3 - Usun zadanie
0 - Zakoncz
```

## KROK 3 — ODCZYTAJ WYBÓR

Użytkownik wpisuje numer wybranej operacji.

## KROK 4 — WYKONAJ DZIAŁANIE

### OPCJA 1 — DODAJ

Program prosi o treść nowego zadania i zapisuje ją na liście.

### OPCJA 2 — POKAŻ

Program wyświetla wszystkie zapisane zadania.

### OPCJA 3 — USUŃ

Program prosi o numer zadania i usuwa wybraną pozycję.

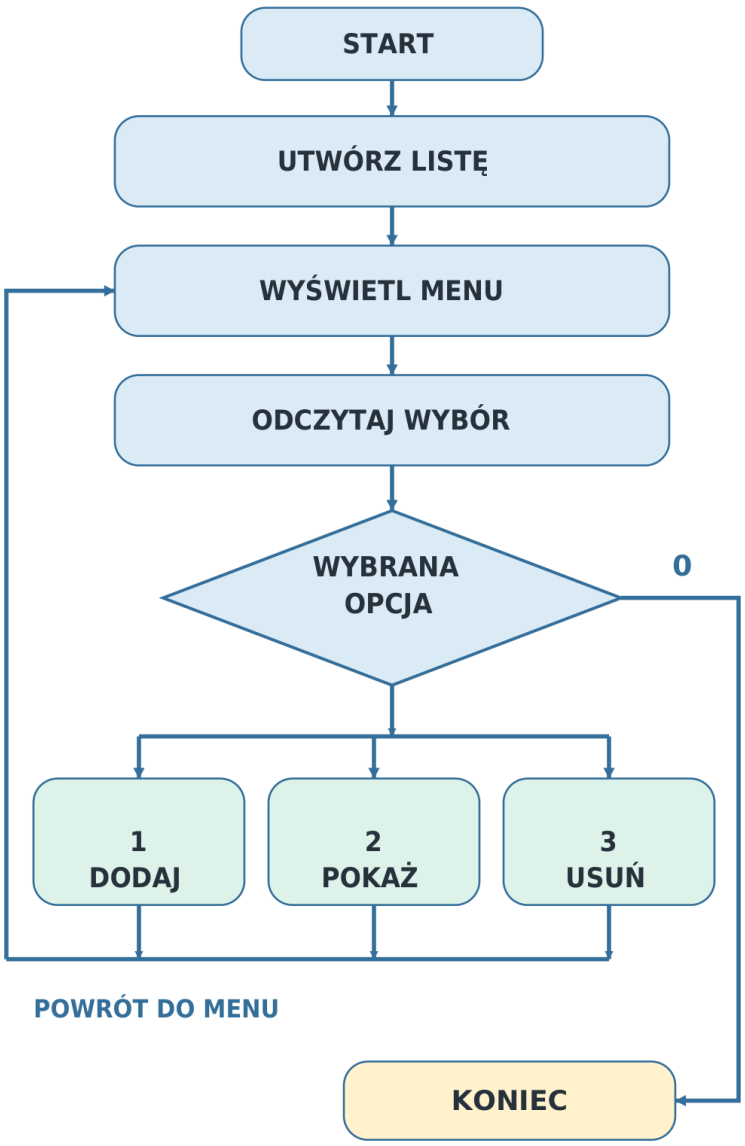
### OPCJA 0 — ZAKOŃCZ

Program kończy działanie.

## KROK 5 — POWRÓĆ DO MENU

Po zakończeniu operacji program ponownie pokazuje menu, chyba że użytkownik wybrał wyjście.

## GŁÓWNY SCHEMAT




## WAŻNE POJĘCIE

Dane programu będą przechowywane w pamięci operacyjnej. Po zakończeniu aplikacji lista zniknie. Zapisywania danych do pliku nauczysz się w późniejszych projektach.

## MINIWYZWANIE

Co powinien zrobić program, jeśli użytkownik wybierze usuwanie zadania, ale lista jest pusta?

**Odpowiedź:** Powinien wyświetlić odpowiedni komunikat i wrócić do menu bez próby usunięcia elementu.

# ARRAYLIST — LISTA, KTÓRA MOŻE ROSNĄĆ

## CO TO JEST ARRAYLIST?

ArrayList to klasa w Javie, która umożliwia przechowywanie wielu elementów w uporządkowanej liście. W przeciwieństwie do zwykłej tablicy lista może zmieniać swoją wielkość podczas działania programu.

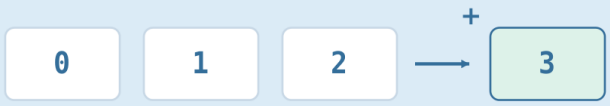
### TABLICA — ARRAY

- Ma ustaloną długość.
- Pozwala odczytywać elementy przez indeks.
- Nie zmienia automatycznie swojego rozmiaru.




### LISTA — ARRAYLIST

- Może się powiększać.
- Można dodawać i usuwać elementy.
- Udostępnia gotowe metody do zarządzania zawartością.




## KROK 1 — IMPORTUJ ARRAYLIST

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

## KROK 2 — UTWÓRZ LISTĘ

```
ArrayList<String> zadania = new ArrayList<>();
```

## KROK 3 — DODAJ ELEMENTY

```
zadania.add("Nauka Javy");
zadania.add("Zrobic zakupy");
zadania.add("Przeczytac ksiazke");
```

## KROK 4 — ODCZYTAJ ELEMENT

```
System.out.println(zadania.get(0));
```

**WYNIK:** Nauka Javy

## KROK 5 — SPRAWDŹ WIELKOŚĆ LISTY

```
System.out.println(zadania.size());
```

**WYNIK:** 3

## WYJAŚNIENIE

`ArrayList<String>` oznacza listę przechowującą napisy.

`zadania` to nazwa naszej zmiennej.

`new ArrayList<>()` tworzy nową, początkowo pustą listę.

## NAJWAŻNIEJSZE METODY

| Metoda | Działanie |
|---|---|
| add() | Dodaje element |
| get() | Odczytuje element |
| size() | Zwraca liczbę elementów |
| remove() | Usuwa element |
| isEmpty() | Sprawdza, czy lista jest pusta |

Import umieść na początku pliku. Pozostałe instrukcje wpisz w metodzie `main`.

## WAŻNE

Indeksy listy rozpoczynają się od zera. Pierwszy element ma indeks **0**, drugi **1**, a trzeci **2**.

## MINIWYZWANIE

Jaki będzie wynik `zadania.size()` po dodaniu czwartego zadania?
**Odpowiedź: 4**

# DODAWAJ NOWE ZADANIA DO SWOJEJ LISTY

## KROK 1 — PRZYGOTUJ NARZĘDZIA

Na początku pliku, przed klasą `Main`:

```
import java.util.ArrayList;
import java.util.Scanner;
```

W metodzie `main` utwórz:

```
Scanner scanner = new Scanner(System.in);
ArrayList<String> zadania = new ArrayList<>();
```

## KROK 2 — ZAPYTAJ O TREŚĆ ZADANIA

```
System.out.print("Podaj nowe zadanie: ");
```

## KROK 3 — ODCZYTAJ TEKST

```
String noweZadanie = scanner.nextLine().trim();
```

## KROK 4 — SPRAWDŹ, CZY TEKST NIE JEST PUSTY

```
if (noweZadanie.isEmpty()) {
    System.out.println("Zadanie nie moze byc puste!");
} else {
    zadania.add(noweZadanie);
    System.out.println("Dodano zadanie!");
}
```

## KROK 5 — SPRAWDŹ REZULTAT

```
System.out.println(
    "Liczba zadan: " + zadania.size());
```

## PRZYKŁADOWY WYNIK

```
Podaj nowe zadanie: Nauka Javy
Dodano zadanie!
Liczba zadan: 1
```

### TYLKO RAZ NA START

Obiekt `Scanner` i pustą listę utwórz tylko raz, przed dodawaniem zadań. Pozostałe instrukcje wykonuj w metodzie `main`.

### DLACZEGO nextLine()?

Metoda `nextLine()` umożliwia odczytanie całego wiersza tekstu, również zawierającego spacje.

Na przykład:
`Przeczytac rozdzial o metodach`

### CO ROBI TEN KOD?

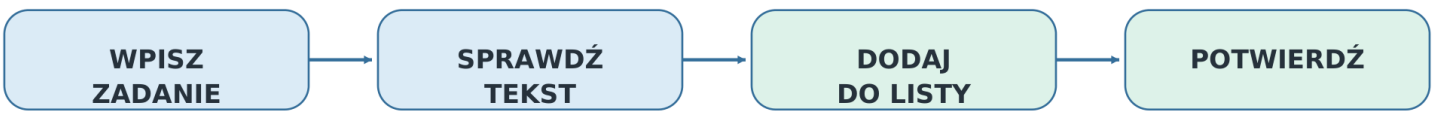
1. Pyta użytkownika o zadanie.
2. Odczytuje pełny tekst.
3. Usuwa zbędne spacje na początku i końcu.
4. Sprawdza, czy tekst nie jest pusty.
5. Dodaje zadanie do listy.
6. Wyświetla potwierdzenie.

### WAŻNE POJĘCIE

`isEmpty()` zwraca `true`, jeśli napis jest pusty. Dzięki `trim()` również wpis składający się wyłącznie ze spacji zostanie uznany za pusty.

### TWOJA LISTA

**1.** Nauka Javy
**2.** Zrobic zakupy
**3.** Przeczytac ksiazke




Pokazana ścieżka dotyczy niepustego tekstu. Pusty wpis powoduje komunikat o błędzie.

### MINIWYZWANIE

Dodaj trzy zadania i sprawdź, czy `size()` zwraca wartość **3**. Wykonaj kroki 2–5 w pętli `for` z trzema powtórzeniami. Obiekt `Scanner` i listę utwórz tylko raz, przed pętlą.

# WYŚWIETLAJ I USUWAJ ZADANIA Z LISTY

## CZĘŚĆ 1 — WYŚWIETLANIE ZADAŃ

Aby pokazać wszystkie elementy, wykorzystaj pętlę `for`.

```
for (int i = 0; i < zadania.size(); i++) {
  System.out.println(
      (i + 1) + ". " + zadania.get(i)
  );
}
```

## PRZYKŁADOWY WYNIK

```
1. Nauka Javy
2. Zrobic zakupy
3. Przeczytac ksiazke
```

### DLACZEGO i + 1?

W Javie indeksy zaczynają się od zera, ale użytkownikowi wygodniej pokazywać numery od jednego.

## CZĘŚĆ 2 — SPRAWDZANIE PUSTEJ LISTY

```
if (zadania.isEmpty()) {
  System.out.println("Lista jest pusta!");
}
```

Ten warunek pozwala uniknąć wyświetlania pustego zestawienia. Gdy lista nie jest pusta, wykonaj pętlę z części 1.

| Numer na ekranie | Indeks w Javie |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 3 | 2 |

## CZĘŚĆ 3 — USUWANIE ZADANIA

Najpierw użytkownik podaje numer:

```
System.out.print(
    "Numer zadania do usuniecia: ");
String tekst = scanner.nextLine().trim();
```

Następnie program sprawdza wartość:

```
try {
  int numer = Integer.parseInt(tekst);
  if (numer >= 1 &&
      numer <= zadania.size()) {
    String usuniete =
        zadania.remove(numer - 1);
    System.out.println(
        "Usunieto: " + usuniete);
  } else {
    System.out.println("Niepoprawny numer!");
  }
} catch (NumberFormatException e) {
  System.out.println(
      "Wpisz liczbe calkowita!");
}
```

## WYJAŚNIENIE

`Integer.parseInt()` próbuje zamienić tekst na liczbę całkowitą.

`try` obejmuje kod, w którym może wystąpić wyjątek.

`catch` pozwala obsłużyć błąd konwersji.

`remove(numer - 1)` usuwa element odpowiadający numerowi wybranemu przez użytkownika.

## PRZYKŁAD — UŻYTKOWNIK WPISUJE: 2

### PRZED USUNIĘCIEM

```
1. Nauka Javy
2. Zrobic zakupy
3. Przeczytac ksiazke
```

### PO USUNIĘCIU

```
1. Nauka Javy
2. Przeczytac ksiazke
```

**Ważne:** Po usunięciu elementu numery pozostałych zadań mogą się zmienić. Fragmenty korzystają z listy `zadania` i obiektu `scanner` utworzonych wcześniej w metodzie `main`.

# POŁĄCZ WSZYSTKIE FUNKCJE W JEDNYM MENU

## PROBLEM

Nie chcemy uruchamiać programu ponownie po każdym dodanym lub usuniętym zadaniu. Potrzebujemy menu, które będzie wyświetlane wielokrotnie.

**Jak połączyć fragmenty?** Kroki 3–5 umieść wewnątrz `while` z kroku 2. Obiekt `scanner` i listę `zadania` utwórz tylko raz, przed pętlą.

## KROK 1 — ZMIENNA STERUJĄCA

```
boolean dziala = true;
```

**Co to jest boolean?** Typ przyjmujący jedną z dwóch wartości: `true` — prawda, `false` — fałsz.

## KROK 2 — PĘTLA GŁÓWNA

```
while (dziala) {
  // Tutaj umiesc menu
  // i obsluge wyboru.
}
```

Pętla wykonuje się, dopóki `dziala` ma wartość `true`.

## KROK 3 — WYŚWIETL MENU

```
System.out.println("\nLISTA ZADAN");
System.out.println("1 - Dodaj zadanie");
System.out.println("2 - Pokaz zadania");
System.out.println("3 - Usun zadanie");
System.out.println("0 - Koniec");
```

## KROK 4 — ODCZYTAJ WYBÓR

```
System.out.print("Wybor: ");
String wybor =
    scanner.nextLine().trim();
```

## KROK 5 — OBSŁUŻ WYBRANĄ OPCJĘ

```
switch (wybor) {
  case "1":
    System.out.println("Dodawanie zadania");
    break;
  case "2":
    System.out.println("Wyswietlanie listy");
    break;
  case "3":
    System.out.println("Usuwanie zadania");
    break;
  case "0":
    dziala = false;
    break;
  default:
    System.out.println("Nieznana opcja!");
}
```

## KROK 6 — ZAKOŃCZ PROGRAM

Po wybraniu **0** zmienna `dziala` przyjmuje wartość `false`. Pętla zatrzymuje się, a program kończy działanie.

### WAŻNE

Samo `break` w instrukcji `switch` kończy daną gałąź instrukcji, a nie zewnętrzną pętlę `while`.

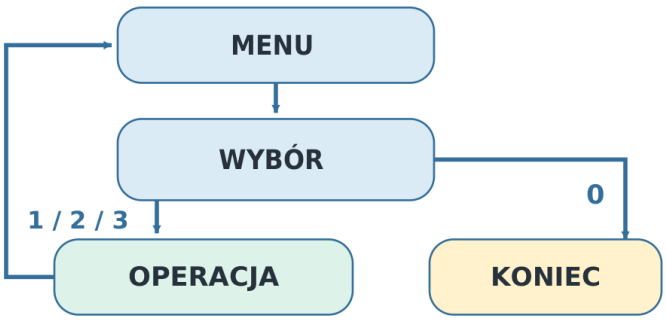
## CO DALEJ?

Zastąp komunikaty w poszczególnych przypadkach kodem dodawania, wyświetlania i usuwania zadań z poprzednich stron.

## PRZYKŁADOWA SESJA

```
LISTA ZADAN
1 - Dodaj zadanie
2 - Pokaz zadania
3 - Usun zadanie
0 - Koniec
Wybor: 1
Dodawanie zadania
LISTA ZADAN
...
```

## MENU POWRACA PO OPERACJI



# STWÓRZ KOMPLETNĄ APLIKACJĘ LISTA ZADAŃ

## KROK 1 — UTWÓRZ PLIK

`Main.java`

## KROK 2 — WPISZ KOMPLETNY KOD

### 1. POCZĄTEK KODU · 01–38

```
import java.util.ArrayList;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ArrayList<String> zadania = new ArrayList<>();
    boolean dziala = true;
    while (dziala) {
      System.out.println("\nLISTA ZADAN");
      System.out.println("1 - Dodaj");
      System.out.println("2 - Pokaz");
      System.out.println("3 - Usun");
      System.out.println("0 - Koniec");
      System.out.print("Wybor: ");
      String wybor = sc.nextLine().trim();
      switch (wybor) {
        case "1":
          System.out.print("Nowe zadanie: ");
          String tekst = sc.nextLine().trim();
          if (tekst.isEmpty()) {
            System.out.println("Puste zadanie!");
          } else {
            zadania.add(tekst);
            System.out.println("Dodano!");
          }
          break;
        case "2":
          if (zadania.isEmpty()) {
            System.out.println(
              "Lista jest pusta.");
          } else {
            for (int i = 0; i < zadania.size();
              i++) {
              System.out.println(
                (i + 1) + ". " + zadania.get(i));
            }
          }
          break;
```

### 2. DALSZY CIĄG · 39–70

```
        case "3":
          if (zadania.isEmpty()) {
            System.out.println("Brak zadan.");
            break;
          }
          System.out.print("Numer zadania: ");
          String numerTekst =
            sc.nextLine().trim();
          try {
            int numer =
              Integer.parseInt(numerTekst);
            if (numer >= 1 &&
              numer <= zadania.size()) {
              zadania.remove(numer - 1);
              System.out.println("Usunieto!");
            } else {
              System.out.println("Zly numer!");
            }
          } catch (NumberFormatException e) {
            System.out.println("Wpisz liczbe!");
          }
          break;
        case "0":
          dziala = false;
          System.out.println("Do zobaczenia!");
          break;
        default:
          System.out.println("Nieznana opcja!");
      }
    }
  }
}
```

## KROK 3 URUCHOM

```
javac Main.java
java Main
```

## PRZYKŁADOWE DZIAŁANIE — SKRÓT SESJI

```
LISTA ZADAN
1 - Dodaj
2 - Pokaz
3 - Usun
0 - Koniec
Wybor: 1
Nowe zadanie: Nauka Javy
Dodano!
Wybor: 2
1. Nauka Javy
Wybor: 3
Numer zadania: 1
Usunieto!
```

## TESTY PROGRAMU

| Test | Oczekiwany rezultat |
|---|---|
| Dodanie zadania | Nowy element na liście |
| Dodanie pustego tekstu | Komunikat o błędzie |
| Wyświetlenie pustej listy | Informacja o braku zadań |
| Usunięcie istniejącego zadania | Element znika |
| Niepoprawny numer | Komunikat o błędzie |
| Wybór 0 | Zakończenie programu |

**WAŻNA INFORMACJA:** Ten program przechowuje zadania wyłącznie podczas działania. Po zamknięciu aplikacji dane nie zostają zapisane na dysku.

# ROZBUDUJ LISTĘ ZADAŃ O NOWE FUNKCJE

## MISJA — DODAJ OPCJĘ WYCZYSZCZENIA LISTY

Rozbuduj menu o nową funkcję: `4 - Wyczysc liste`.

### KROK 1 — DODAJ OPCJĘ DO MENU

```
System.out.println("4 - Wyczysc liste");
```

### KROK 2 — NOWY PRZYPADEK SWITCH

Wstaw do `switch` z poprzedniej strony, przed `default`. Użyj tego samego obiektu `sc`.

```
case "4":
  System.out.print(
      "Usunac wszystkie zadania? (T/N): ");
  String decyzja = sc.nextLine().trim();
  if (decyzja.equalsIgnoreCase("T")) {
    zadania.clear();
    System.out.println("Lista wyczyszczona!");
  } else {
    System.out.println("Anulowano.");
  }
  break;
```

### JAK DZIAŁA CLEAR()?

Metoda `clear()` usuwa wszystkie elementy z listy. Po jej wykonaniu `zadania.size()` zwraca **0**.

### KROK 3 — SPRAWDŹ DZIAŁANIE

1. Dodaj trzy zadania.
2. Wybierz opcję 4.
3. Potwierdź literą T.
4. Wyświetl listę.
5. Sprawdź, czy jest pusta.



**LISTA ZADAŃ**

- Nauka Javy
- Zakupy
- Czytanie

## DODATKOWE WYZWANIA

### A — WYSZUKIWANIE

Dodaj opcję, która pozwoli znaleźć zadania zawierające określone słowo.
**Podpowiedź:** `contains()`

### B — LICZBA ZADAŃ

Wyświetl informację: `Liczba zadan: 5`
**Podpowiedź:** `zadania.size()`

### C — OZNACZANIE ZADAŃ

Zastanów się, jak oznaczać zadania jako ukończone bez usuwania. W przyszłości możesz utworzyć własną klasę reprezentującą zadanie.

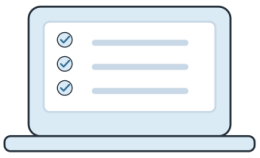
## CHECKLISTA UMIEJĘTNOŚCI

- [ ] Potrafię tworzyć `ArrayList`.
- [ ] Umiem dodawać nowe elementy.
- [ ] Rozumiem metodę `get()`.
- [ ] Potrafię usuwać wybrane dane.
- [ ] Umiem sprawdzać zawartość listy.
- [ ] Potrafię korzystać z `switch`.
- [ ] Rozumiem pętlę `while`.
- [ ] Potrafię obsługiwać niepoprawne dane.
- [ ] Umiem rozbudować działający program.



**Gratulacje! Ukończyłeś szósty projekt w Javie.**

**6 Z 10 PROJEKTÓW GOTOWYCH!**





**6 z 10 projektów**

PROJEKT

07

# KSIĄŻKA KONTAKTÓW

**Stwórz własną aplikację do zapisywania, wyszukiwania i organizowania kontaktów!**






**PRZYKŁADOWA KARTA KONTAKTU**

Imię: **Anna**
Telefon: 123456789
E-mail: anna@example.com

## CZEGO SIĘ NAUCZYSZ?

- Organizowania powiązanych danych.
- Tworzenia własnych klas.
- Korzystania z obiektów.
- Przechowywania obiektów w ArrayList.
- Wyszukiwania informacji.
- Wyświetlania danych kontaktowych.
- Usuwania wybranych kontaktów.
- Przygotowania aplikacji do późniejszego zapisu danych.

**WAŻNE POJĘCIE:** Programowanie obiektowe pozwala łączyć powiązane informacje i zachowania w logiczne struktury.

**Teraz nauczysz się tworzyć własne struktury danych zamiast przechowywać wszystko w pojedynczych napisach.**

# JAK ZAPROJEKTOWAĆ WŁASNĄ KSIĄŻKĘ KONTAKTÓW?

## CEL PROGRAMU

Użytkownik może tworzyć kontakty, przeglądać je, wyszukiwać po nazwie oraz usuwać niepotrzebne wpisy. Każdy kontakt będzie przechowywał trzy informacje.




## GŁÓWNE FUNKCJE PROGRAMU

### 1 — DODAWANIE KONTAKTU

Użytkownik podaje imię, numer telefonu i adres e-mail. Program tworzy nowy kontakt.

### 2 — WYŚWIETLANIE KONTAKTÓW

Program pokazuje zapisane kontakty w czytelnej formie.

### 3 — WYSZUKIWANIE

Użytkownik wpisuje imię lub jego fragment. Program znajduje pasujące kontakty.

### 4 — USUWANIE

Użytkownik wybiera kontakt, który chce usunąć.

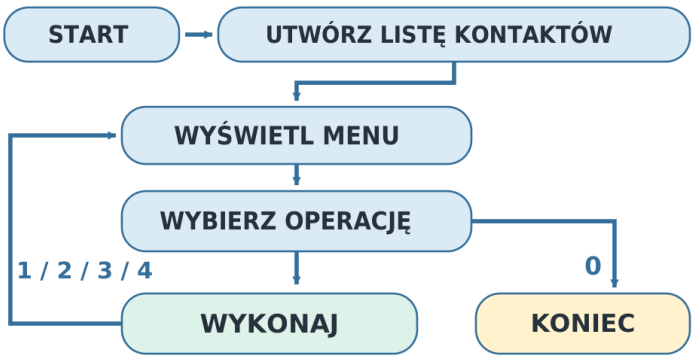
### 0 — ZAKOŃCZENIE

Program kończy działanie.

### DLACZEGO TELEFON JAKO STRING?

Numer telefonu nie służy do obliczeń matematycznych. Może zawierać znak **+**, zera na początku, spacje lub inne znaki formatujące. Dlatego wygodnie przechowywać go jako tekst.

## PRZEPŁYW PROGRAMU




## NOWE POJĘCIE — KLASA

Klasa to opis struktury, według której tworzymy obiekty. W tym projekcie utworzymy klasę `Kontakt`.

### PRZYKŁADOWA STRUKTURA

```
class Kontakt {
  String imie;
  String telefon;
  String email;
}
```

### CO OZNACZAJĄ POLA?

`imie` — przechowuje imię kontaktu.

`telefon` — przechowuje numer telefonu jako tekst.

`email` — przechowuje adres e-mail.

### CO TO JEST OBIEKT?

Obiekt to konkretny egzemplarz utworzony na podstawie klasy. Na przykład obiekt reprezentujący kontakt Anny może przechowywać wszystkie trzy informacje w jednym miejscu.

**Ważne:** Na tym etapie planujemy strukturę aplikacji. Na kolejnych stronach utworzymy klasę, konstruktor i listę obiektów oraz nauczymy się nimi zarządzać.

## MINIWYZWANIE

Jakie dodatkowe informacje można przechowywać w książce kontaktów? Przykłady:

**1.** Nazwisko. **2.** Nazwa firmy. **3.** Data urodzin. **4.** Dodatkowy numer telefonu. **5.** Notatka.

### PRZYKŁADOWE KONTAKTY

Anna | Jan | Ewa

# STWÓRZ WŁASNĄ KLASĘ KONTAKT W JAVIE

## CO TO JEST KLASA?

Klasa to opis struktury obiektów, które mogą przechowywać określone dane i wykonywać określone działania. W naszej aplikacji klasa `Kontakt` będzie przechowywała informacje o jednej osobie.

## JAKIE DANE BĘDZIEMY PRZECHOWYWAĆ?

Każdy kontakt otrzyma trzy pola.

**1. IMIĘ**

Anna

**2. TELEFON**

123456789

**3. E-MAIL**

anna@example.com

## KROK 1 — UTWÓRZ KLASĘ

```
class Kontakt {
    String imie;
    String telefon;
    String email;
}
```

## CO OZNACZA TEN KOD?

`class Kontakt` — definiuje nową klasę.

`String imie` — pole przechowujące imię.

`String telefon` — pole przechowujące numer telefonu.

`String email` — pole przechowujące adres e-mail.

## DLACZEGO TELEFON JEST NAPISEM?

Numer telefonu nie jest zwykłą liczbą służącą do obliczeń. Może zawierać:

- Znak plus.
- Zera na początku.
- Spacje.
- Separatory.

Dlatego użyjemy typu `String`.

## KLASA A OBIEKT

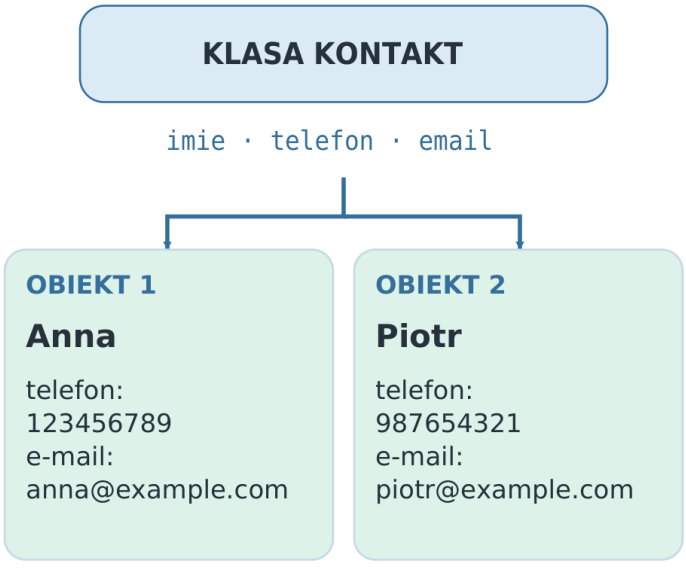
### KLASA

Opisuje, jakie informacje może zawierać kontakt.

### OBIEKT

Reprezentuje konkretną osobę, na przykład Annę.

## JEDEN MODEL — WIELE OBIEKTÓW




### WAŻNE

Jedna klasa może służyć do tworzenia wielu obiektów. Każdy obiekt przechowuje własne wartości pól.

### MINIWYZWANIE

Jakie pole można dodać do klasy, aby przechowywać nazwisko osoby?
**Odpowiedź:** `String nazwisko;`

# KONSTRUKTOR — TWÓRZ KONTAKTY Z WŁASNYMI DANYMI

## CO TO JEST KONSTRUKTOR?

Konstruktor to specjalny element klasy wywoływany podczas tworzenia obiektu. Pozwala przypisać początkowe wartości jego polom.

## KROK 1 — DODAJ KONSTRUKTOR DO KLASY

```
class Kontakt {
    String imie;
    String telefon;
    String email;
    Kontakt(String imie, String telefon,
            String email) {
        this.imie = imie;
        this.telefon = telefon;
        this.email = email;
    }
}
```

## KROK 2 — UTWÓRZ OBIEKT

```
Kontakt kontakt1 = new Kontakt(
    "Anna",
    "123456789",
    "anna@example.com"
);
```

## KROK 3 — ODCZYTAJ DANE

```
System.out.println(kontakt1.imie);
System.out.println(kontakt1.telefon);
System.out.println(kontakt1.email);
```

## KROK 4 — UTWÓRZ DRUGI OBIEKT

```
Kontakt kontakt2 = new Kontakt(
    "Piotr",
    "987654321",
    "piotr@example.com"
);
```

## JAK DZIAŁA KONSTRUKTOR?

`Kontakt(...)` — konstruktor klasy Kontakt.

`String imie` — pierwszy parametr.

`String telefon` — drugi parametr.

`String email` — trzeci parametr.

`this.imie` — pole należące do tworzonego obiektu.

## DLACZEGO UŻYWAMY this?

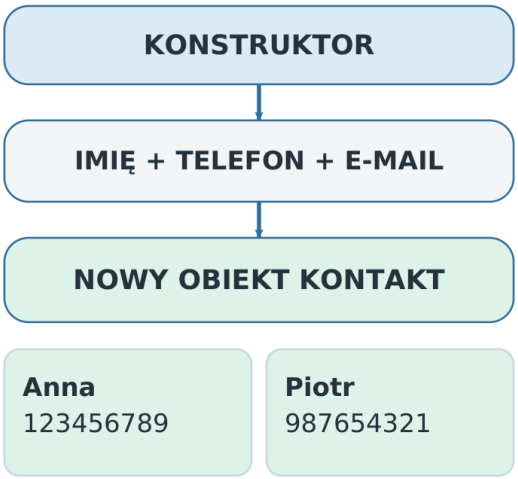
Słowo `this` pozwala odwoływać się do pól bieżącego obiektu. Na przykład:

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

Oznacza przypisanie wartości parametru `imie` do pola `imie` w obiekcie.

## WYNIK KROKU 3

```
Anna
123456789
anna@example.com
```




**WAŻNE.** Umieść klasę `Kontakt` w tym samym pliku `Main.java`, poza klasą `Main`, a kod tworzący obiekty wewnątrz metody `main`. W ten sposób program może tworzyć dowolną liczbę kontaktów na podstawie jednego modelu.

**MINIWYZWANIE**
Utwórz obiekt reprezentujący osobę o imieniu Maria.

# PRZECHOWUJ WIELE KONTAKTÓW W JEDNEJ LIŚCIE

## PROBLEM

Potrafimy utworzyć jeden kontakt. Ale co zrobić, gdy chcemy przechowywać dziesiątki różnych kontaktów?

**ROZWIĄZANIE — ARRAYLIST**
Użyjemy listy, która może przechowywać obiekty klasy `Kontakt`.

## KROK 1 — IMPORTUJ ARRAYLIST

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

## KROK 2 — UTWÓRZ LISTĘ KONTAKTÓW

```
ArrayList<Kontakt> kontakty = new ArrayList<>();
```

## KROK 3 — DODAJ OBIEKTY

```
kontakty.add(new Kontakt(
    "Anna", "123456789", "anna@example.com"
));
kontakty.add(new Kontakt(
    "Piotr", "987654321", "piotr@example.com"
));
```

## KROK 4 — SPRAWDŹ LICZBĘ KONTAKTÓW

```
System.out.println(kontakty.size());
```

**WYNIK:** 2

## KROK 5 — WYŚWIETL WSZYSTKIE KONTAKTY

```
for (Kontakt k : kontakty) {
    System.out.println("Imie: " + k.imie);
    System.out.println("Telefon: " + k.telefon);
    System.out.println("E-mail: " + k.email);
    System.out.println("---");
}
```

## CO OZNACZA TEN ZAPIS?

`ArrayList<Kontakt>` oznacza listę przechowującą obiekty typu `Kontakt`.

## PRZYKŁADOWY WYNIK

```
Imie: Anna
Telefon: 123456789
E-mail: anna@example.com
---
Imie: Piotr
Telefon: 987654321
E-mail: piotr@example.com
---
```

## NOWE POJĘCIE — FOR-EACH

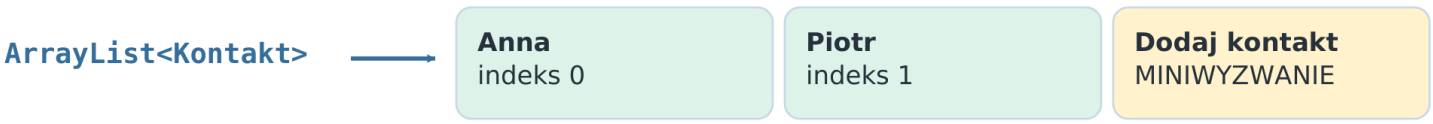
```
for (Kontakt k : kontakty)
```

Ta pętla przechodzi kolejno przez wszystkie obiekty znajdujące się na liście. Zmienna k reprezentuje aktualnie przetwarzany kontakt.

## PORÓWNANIE

**ZWYKŁA PĘTLA FOR**
Przydatna, gdy potrzebujemy indeksów elementów.

**PĘTLA FOR-EACH**
Wygodna, gdy chcemy przeglądać wszystkie obiekty bez ręcznego zarządzania indeksem.




## WAŻNE

Przykłady zakładają istnienie klasy `Kontakt` z konstruktorem przedstawionym na poprzedniej stronie. Import umieść na początku pliku, a instrukcje listy — w metodzie `main`.

## MINIWYZWANIE

Dodaj trzeci kontakt i sprawdź, czy lista zawiera trzy obiekty.

# WYSZUKUJ I USUWAJ KONTAKTY Z APLIKACJI

Fragmenty umieść w metodzie `main`. Korzystają z istniejącej listy `kontakty`, obiektu `scanner` oraz klasy `Kontakt`.

## CZĘŚĆ 1 — WYSZUKIWANIE KONTAKTU

Użytkownik wpisuje imię lub fragment imienia. Program przeszukuje wszystkie zapisane kontakty.

### KROK 1 — POBIERZ TEKST

```
System.out.print("Szukaj imienia: ");
String szukane =
    scanner.nextLine().trim();
```

### KROKI 2-3 — SPRAWDŹ I WYSZUKAJ

```
if (szukane.isEmpty()) {
  System.out.println("Podaj imie!");
} else {
  for (Kontakt k : kontakty) {
    if (k.imie
        .toLowerCase(java.util.Locale.ROOT)
        .contains(szukane.toLowerCase(
            java.util.Locale.ROOT))) {
      System.out.println(
          k.imie + " - " + k.telefon);
    }
  }
}
```

Pętla wykonuje się tylko wtedy, gdy zapytanie nie jest puste.

### JAK TO DZIAŁA?

`toLowerCase()` zamienia litery na małe. `contains()` sprawdza, czy napis zawiera wskazany fragment. Dzięki temu wyszukiwanie może znaleźć Annę również po wpisaniu ann.



ann → **Anna**

123456789

## CZĘŚĆ 2 — USUWANIE KONTAKTU

Najpierw wyświetl listę z numerami.

```
for (int i = 0;
     i < kontakty.size(); i++) {
  System.out.println(
      (i + 1) + ". " +
      kontakty.get(i).imie);
}
```

### KROK 1 — POBIERZ NUMER

```
System.out.print("Numer kontaktu: ");
String tekst =
    scanner.nextLine().trim();
```

### KROK 2 — SPRAWDŹ I USUŃ

```
try {
  int numer = Integer.parseInt(tekst);
  if (numer >= 1 &&
      numer <= kontakty.size()) {
    kontakty.remove(numer - 1);
    System.out.println("Kontakt usuniety!");
  } else {
    System.out.println("Niepoprawny numer!");
  }
} catch (NumberFormatException e) {
  System.out.println("Wpisz liczbe!");
}
```

### NUMER ≠ INDEKS

Numer widoczny dla użytkownika zaczyna się od **1**, a indeksy `ArrayList` od **0**. Dlatego przy usuwaniu używamy `numer - 1`.

## PRZYKŁAD — USUŃ KONTAKT NR 2

**PRZED**

1. Anna
2. Piotr
3. Maria

→

**PO**

1. Anna
2. Maria

### BEZPIECZEŃSTWO DANYCH

Nigdy nie usuwaj elementu bez sprawdzenia, czy wskazany numer istnieje. W przeciwnym razie program może zakończyć działanie z błędem.

# ZBUDUJ KOMPLETNĄ KSIĄŻKĘ KONTAKTÓW

## KROK 1 — NOWY PLIK Main.java

### 1. POCZĄTEK · 01–53

```
import java.util.ArrayList;
import java.util.Locale;
import java.util.Scanner;
public class Main {
  static class Kontakt {
    String imie, telefon, email;
    Kontakt(String i, String t, String e) {
      imie = i;
      telefon = t;
      email = e;
    }
  }
  static void pokaz(ArrayList<Kontakt> lista) {
    if (lista.isEmpty()) {
      System.out.println("Brak kontaktow.");
      return;
    }
    for (int i = 0; i < lista.size(); i++) {
      Kontakt k = lista.get(i);
      System.out.println((i + 1) + ". " +
        k.imie + " | " + k.telefon +
        " | " + k.email);
    }
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ArrayList<Kontakt> lista = new ArrayList<>();
    while (true) {
      System.out.println("\nKSIAZKA KONTAKTOW");
      System.out.println("1 - Dodaj");
      System.out.println("2 - Pokaz");
      System.out.println("3 - Szukaj");
      System.out.println("4 - Usun");
      System.out.println("0 - Koniec");
      System.out.print("Wybor: ");
      String wybor = sc.nextLine().trim();
      switch (wybor) {
        case "1":
          System.out.print("Imie: ");
          String imie = sc.nextLine().trim();
          System.out.print("Telefon: ");
          String tel = sc.nextLine().trim();
          System.out.print("E-mail: ");
          String email = sc.nextLine().trim();
          if (imie.isEmpty() || tel.isEmpty()
            || email.isEmpty()) {
            System.out.println("Puste dane!");
          } else {
            lista.add(
              new Kontakt(imie, tel, email));
            System.out.println("Dodano kontakt!");
          }
          break;
```

## KROK 2 — WPISZ CAŁY KOD

### 2. DALSZY CIĄG · 54–103

```
        case "2":
          pokaz(lista);
          break;
        case "3":
          System.out.print("Szukaj imienia: ");
          String szukane = sc.nextLine().trim()
            .toLowerCase(Locale.ROOT);
          boolean znaleziono = false;
          if (szukane.isEmpty()) {
            System.out.println("Podaj imie!");
            break;
          }
          for (Kontakt k : lista) {
            if (k.imie.toLowerCase(Locale.ROOT)
              .contains(szukane)) {
              System.out.println(k.imie + " | " +
                k.telefon + " | " + k.email);
              znaleziono = true;
            }
          }
          if (!znaleziono) {
            System.out.println("Nie znaleziono.");
          }
          break;
        case "4":
          pokaz(lista);
          if (lista.isEmpty()) break;
          System.out.print("Numer kontaktu: ");
          try {
            int nr = Integer.parseInt(
              sc.nextLine().trim());
            if (nr >= 1 && nr <= lista.size()) {
              lista.remove(nr - 1);
              System.out.println("Usunieto!");
            } else {
              System.out.println("Zly numer!");
            }
          } catch (NumberFormatException e) {
            System.out.println("Wpisz liczbe!");
          }
          break;
        case "0":
          System.out.println("Do zobaczenia!");
          return;
        default:
          System.out.println("Nieznana opcja!");
      }
    }
  }
}
```

## KROK 3 — URUCHOM

```
javac Main.java
java Main
```

## PRZYKŁADOWA SESJA — SKRÓT

```
KSIAZKA KONTAKTOW
1 - Dodaj
2 - Pokaz
3 - Szukaj
4 - Usun
0 - Koniec
Wybor: 1
Imie: Anna
Telefon: 123456789
E-mail: anna@example.com
Dodano kontakt!
Wybor: 2
1. Anna | 123456789 | anna@example.com
```

## CO POTRAFI PROGRAM?

- Tworzyć nowe obiekty Kontakt.
- Przechowywać obiekty w ArrayList.
- Wyświetlać wszystkie kontakty.
- Wyszukiwać kontakty po fragmencie imienia.
- Usuwać kontakty po numerze.
- Obsługiwać podstawowe błędy wpisywania danych.
- Działać aż do wybrania zakończenia.

**WAŻNA INFORMACJA:** Program sprawdza, czy pola są niepuste, ale nie weryfikuje pełnej poprawności numerów telefonów ani adresów e-mail. Dane pozostają w pamięci operacyjnej i znikają po zamknięciu programu. Tutaj `Kontakt` jest klasą `static` wewnątrz `Main`.

# PRZETESTUJ KSIĄŻKĘ KONTAKTÓW I ROZBUDUJ JEJ FUNKCJE

## CZĘŚĆ 1 — TESTY PROGRAMU

### 01 DODAWANIE

Dodaj: **Anna, 123456789, anna@example.com**
Kontakt pojawia się na liście.

### 02 WYSZUKIWANIE

Wpisz: **ann**
Program odnajduje kontakt Anny.

### 03 USUWANIE

Usuń pierwszy kontakt.
Kontakt znika z listy.

### 04 PUSTE DANE

Spróbuj dodać kontakt bez imienia.
Program wyświetla komunikat o błędzie.

### 05 BŁĘDNY NUMER

Wpisz numer nieistniejącego kontaktu.
Program nie usuwa żadnych danych.

## CZĘŚĆ 2 — DODAJ NAZWISKO

Rozbuduj klasę `Kontakt`.

### KROK 1 — DODAJ POLE

```
String nazwisko;
```

### KROK 2 — ROZSZERZ KONSTRUKTOR

Dodaj parametr nazwiska i przypisz go do nowego pola.

### KROK 3 — ZAPYTAJ O NAZWISKO

Dodaj pytanie przy tworzeniu kontaktu. Zaktualizuj wszystkie wywołania konstruktora, przekazując również nazwisko.

### KROK 4 — WYŚWIETLAJ RAZEM

Pokazuj imię i nazwisko razem.

## DODATKOWE WYZWANIA

### A — WYSZUKIWANIE PO TELEFONIE

Pozwól użytkownikowi wyszukiwać kontakty na podstawie numeru telefonu.

### B — LICZBA KONTAKTÓW

Dodaj komunikat: `Liczba kontaktow: 5`
Podpowiedź: `lista.size()`.

### C — POTWIERDZENIE USUNIĘCIA

Przed usunięciem zapytaj:
`Czy na pewno usunac kontakt? (T/N)`

## CHECKLISTA UMIEJĘTNOŚCI

- ☐ Rozumiem pojęcie klasy.
- ☐ Potrafię tworzyć obiekty.
- ☐ Umiem korzystać z konstruktora.
- ☐ Rozumiem działanie `this`.
- ☐ Potrafię przechowywać obiekty w `ArrayList`.
- ☐ Umiem przeszukiwać dane.
- ☐ Potrafię usuwać wybrane obiekty.
- ☐ Rozumiem podstawowe zasady programowania obiektowego.

**KONTAKT**
Anna

## 7 Z 10 PROJEKTÓW UKOŃCZONYCH!



**7 z 10 projektów**

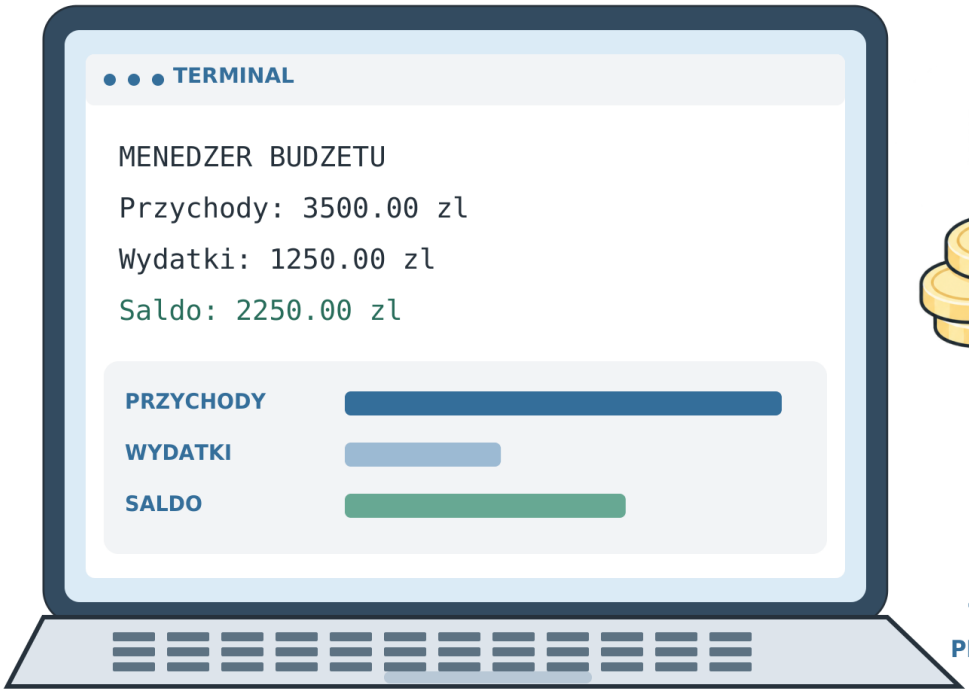
**Gratulacje! Siódmy projekt w Javie jest gotowy.**

PROJEKT

# 08

# MENEDŻER BUDŻETU

**Stwórz program, który pomaga zapisywać przychody, wydatki i obliczać saldo.**







## CZEGO SIĘ NAUCZYSZ?

- Tworzenia struktury transakcji.
- Rozróżniania przychodów i wydatków.
- Przechowywania operacji finansowych.
- Korzystania z obiektów i list.
- Obliczania sum i salda.
- Tworzenia prostych raportów.
- Organizowania programu za pomocą metod.
- Testowania obliczeń.

**WAŻNA INFORMACJA:** Projekt jest edukacyjnym menedżerem budżetu. Do nauki wykorzystamy typ `double`, ale w rzeczywistych aplikacjach finansowych należy stosować rozwiązania zapewniające dokładne obliczenia dziesiętne, np. `BigDecimal` albo kwoty zapisane w najmniejszych jednostkach waluty.

**Poznaj praktyczne zastosowanie Javy do organizowania codziennych informacji finansowych.**

# JAK DZIAŁA PROSTY MENEDŻER BUDŻETU?

## CEL PROGRAMU

Użytkownik zapisuje swoje przychody i wydatki. Program przechowuje transakcje, oblicza sumy i pokazuje aktualne saldo.

## TRZY PODSTAWOWE POJĘCIA

### 01 · PRZYCHÓD

Pieniądze, które otrzymujemy.

**PRZYKŁADY**

Wynagrodzenie.
Dodatkowa praca.
Sprzedaż przedmiotu.

### 02 · WYDATEK

Pieniądze, które wydajemy.

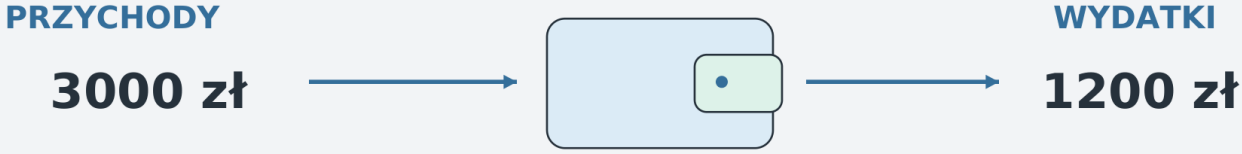
**PRZYKŁADY**

Zakupy.
Transport.
Rachunki.

### 03 · SALDO

Różnica między sumą przychodów i sumą wydatków.

Saldo = przychody − wydatki.




**SALDO = PRZYCHODY − WYDATKI** **1800 zł**

## GŁÓWNE FUNKCJE PROGRAMU

### 1 — DODAJ PRZYCHÓD

Wpisz opis i kwotę otrzymanych pieniędzy.

### 2 — DODAJ WYDATEK

Wpisz opis i kwotę wydanych pieniędzy.

### 3 — POKAŻ TRANSAKCJE

Program pokazuje historię zapisanych operacji.

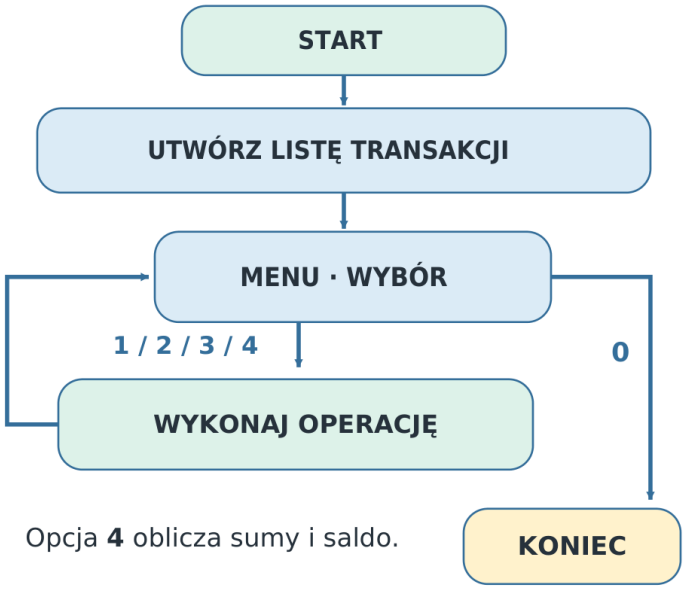
### 4 — POKAŻ PODSUMOWANIE

Program oblicza przychody, wydatki i saldo.

### 0 — ZAKOŃCZ

Program kończy działanie.

## SCHEMAT DZIAŁANIA




Opcja **4** oblicza sumy i saldo.

## WAŻNE

Każda transakcja powinna zawierać opis, kwotę i informację, czy jest przychodem, czy wydatkiem.

## MINIWYZWANIE

Oblicz saldo: przychody **2500 zł**, wydatki **1750 zł**. **Odpowiedź: 750 zł.**

# JAK PRZECHOWYWAĆ PRZYCHODY I WYDATKI W JAVIE?

**CO TO JEST TRANSAKCJA?** Transakcja to pojedynczy zapis operacji finansowej. W naszym programie każda transakcja będzie zawierała trzy informacje:

**OPIS:** Zakupy spożywcze. **KWOTA:** 150.00 zł. **RODZAJ:** Przychód albo wydatek.

## KROK 1 — UTWÓRZ KLASĘ TRANSAKCJA

```
class Transakcja {
    String opis;
    double kwota;
    boolean przychod;
    Transakcja(String opis, double kwota,
               boolean przychod) {
        this.opis = opis;
        this.kwota = kwota;
        this.przychod = przychod;
    }
}
```

## KROK 2 — UTWÓRZ LISTĘ TRANSAKCJI

```
ArrayList<Transakcja> transakcje =
    new ArrayList<>();
```

## KROK 3 — DODAJ PRZYCHÓD

```
transakcje.add(
    new Transakcja("Wynagrodzenie", 3500.00, true)
);
```

## KROK 4 — DODAJ WYDATEK

```
transakcje.add(
    new Transakcja("Zakupy", 250.00, false)
);
```

## KROK 5 — DODAJ KOLEJNY WYDATEK

```
transakcje.add(
    new Transakcja("Transport", 100.00, false)
);
```

## PRZYKŁADOWY WYNIK

```
Przychod: Wynagrodzenie - 3500.0
Wydatek: Zakupy - 250.0
Wydatek: Transport - 100.0
```

## JAK DZIAŁAJĄ POLA?

`opis` — nazwa transakcji.
`kwota` — wartość pieniężna.
`przychod` — informacja logiczna o rodzaju operacji.
**true** oznacza przychód.
**false** oznacza wydatek.

## KROK 6 — WYŚWIETL WSZYSTKIE OPERACJE

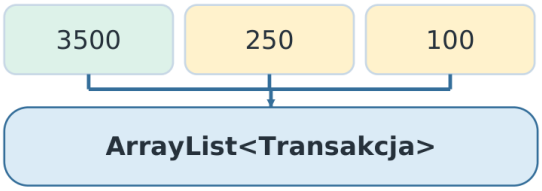
```
for (Transakcja t : transakcje) {
    String rodzaj = t.przychod
        ? "Przychod" : "Wydatek";
    System.out.println(
        rodzaj + ": " + t.opis
        + " - " + t.kwota
    );
}
```

## OPERATOR WARUNKOWY

```
warunek ? wartosc1 : wartosc2
```

To skrócony zapis wyboru jednej z dwóch wartości. Jeśli warunek jest prawdziwy, otrzymujemy pierwszą wartość. W przeciwnym razie drugą.

## 3 OBIEKTY → JEDNA LISTA




| Rodzaj | Opis | Kwota |
|---|---|---|
| + | Wynagrodzenie | 3500 |
| − | Zakupy | 250 |
| − | Transport | 100 |

**WAŻNE.** Wszystkie zapisywane kwoty powinny być dodatnie i skończone. Rodzaj transakcji określa, czy dana kwota zwiększa, czy zmniejsza saldo.

**NOTA TECHNICZNA.** Dodaj `import java.util.ArrayList;` na początku pliku. Klasę `Transakcja` umieść w tym samym pliku co `Main`, poza klasą `Main`. Tworzenie listy i operacji umieść w metodzie `main`.

# OBLICZAJ SALDO I TWÓRZ PODSUMOWANIE BUDŻETU

**CEL** · Nasza aplikacja przechowuje już przychody i wydatki. Teraz stworzymy metody, które obliczą **sumę przychodów, sumę wydatków i aktualne saldo**.

## KROK 1 — SUMA PRZYCHODÓW

```
static double sumaPrzychodow(
    ArrayList<Transakcja> lista) {
  double suma = 0;
  for (Transakcja t : lista) {
    if (t.przychod) {
      suma += t.kwota;
    }
  }
  return suma;
}
```

**Wyjaśnienie:** Metoda sprawdza każdą transakcję. Jeśli `przychod` ma wartość `true`, dodaje kwotę do sumy przychodów.

## KROK 2 — SUMA WYDATKÓW

```
static double sumaWydatkow(
    ArrayList<Transakcja> lista) {
  double suma = 0;
  for (Transakcja t : lista) {
    if (!t.przychod) {
      suma += t.kwota;
    }
  }
  return suma;
}
```

**Wyjaśnienie:** Znak `!` oznacza logiczne zaprzeczenie. Warunek `!t.przychod` jest prawdziwy dla wydatków.

## KROK 3 — OBLICZ SALDO

```
static double obliczSaldo(
    ArrayList<Transakcja> lista) {
  return sumaPrzychodow(lista)
      - sumaWydatkow(lista);
}
```

## KROK 4 — POKAŻ PODSUMOWANIE

```
System.out.printf(
    java.util.Locale.US,
    "Przychody: %.2f zl%n" +
    "Wydatki: %.2f zl%n" +
    "Saldo: %.2f zl%n",
    sumaPrzychodow(transakcje),
    sumaWydatkow(transakcje),
    obliczSaldo(transakcje)
);
```

## NOWE POJĘCIE — OPERATOR +=

```
suma += t.kwota;
```

Jest skrótem zapisu:

```
suma = suma + t.kwota;
```

Pozwala zwiększać aktualną wartość zmiennej.

## PRZYKŁADOWY WYNIK

```
Przychody: 3500.00 zl
Wydatki: 350.00 zl
Saldo: 3150.00 zl
```

| PRZYCHODY | | WYDATKI | | SALDO |
|---|---|---|---|---|
| **3500 zł** | − | **350 zł** | = | **3150 zł** |

**Przykład obliczenia:** wydatki 250 zł + 100 zł = 350 zł; saldo 3500 zł − 350 zł = 3150 zł.

## WAŻNE INFORMACJE

Metody umieść wewnątrz klasy `Main`, ale poza metodą `main`. Potrzebna jest klasa `Transakcja` oraz `import java.util.ArrayList;`
Przed dodaniem transakcji sprawdzaj, czy kwota jest dodatnia i skończona. Proste obliczenia typu `double` służą celom edukacyjnym i mogą powodować drobne błędy zaokrągleń.

## MINIWYZWANIE

Oblicz saldo: przychody **4200 zł**, wydatki **1750 zł** i dodatkowy wydatek **300 zł**. **Odpowiedź: 2150 zł.**

# STWÓRZ MENU SWOJEGO MENEDŻERA BUDŻETU

## DLACZEGO POTRZEBUJEMY MENU?

Nasz menedżer budżetu będzie wykonywał kilka różnych operacji. Użytkownik powinien sam decydować, co chce zrobić. Dzięki menu program stanie się wygodniejszy i bardziej praktyczny.

**Jak połączyć fragmenty?** Obiekt Scanner o nazwie `scanner` i listę transakcji utwórz raz w metodzie `main`. Wewnątrz pętli z kroku 2 wyświetl menu i umieść kroki 3–4.

## KROK 1 — ZAPLANUJ OPCJE

```
MENEDZER BUDZETU
1 - Dodaj przychod
2 - Dodaj wydatek
3 - Pokaz transakcje
4 - Pokaz podsumowanie
0 - Zakoncz
```

## KROK 2 — UTWÓRZ PĘTLĘ

```
while (true) {
  // Tutaj bedzie menu programu.
}
```

**Wyjaśnienie:** `while (true)` tworzy pętlę działającą do wykonania instrukcji kończącej jej działanie. Opcja **0** zakończy metodę `main`.

## KROK 3 — ODCZYTAJ WYBÓR

```
System.out.print("Wybor: ");
String wybor =
    scanner.nextLine().trim();
```

## KROK 4 — OBSŁUŻ MENU

Szkielet obsługi opcji:

```
switch (wybor) {
  case "1":
    System.out.println("Dodawanie przychodu");
    break;
  case "2":
    System.out.println("Dodawanie wydatku");
    break;
  case "3":
    System.out.println("Historia transakcji");
    break;
  case "4":
    System.out.println("Podsumowanie budzetu");
    break;
  case "0":
    System.out.println("Do zobaczenia!");
    return;
  default:
    System.out.println("Nieznana opcja!");
}
```

## KROK 5 — POŁĄCZ FUNKCJE

Na następnej stronie zastąpimy komunikaty właściwymi operacjami:

**OPCJA 1** · Utwórz obiekt reprezentujący przychód.

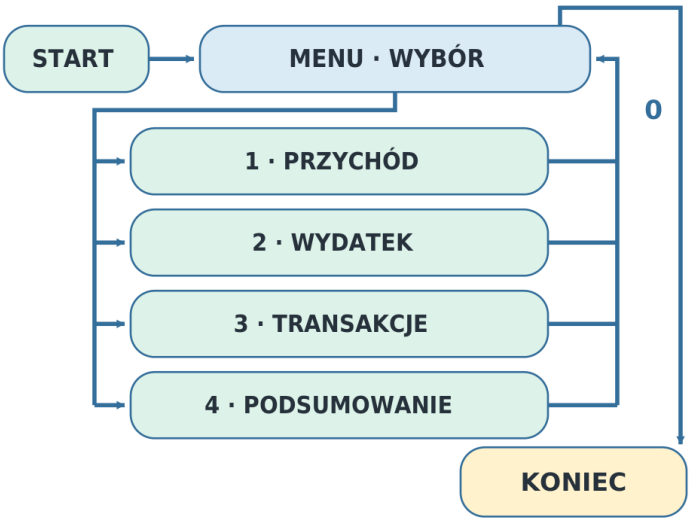
**OPCJA 2** · Utwórz obiekt reprezentujący wydatek.

**OPCJA 3** · Wyświetl wszystkie zapisane transakcje.

**OPCJA 4** · Oblicz i wyświetl przychody, wydatki oraz saldo.

**OPCJA 0** · Zakończ program.

## JEDNO MENU — WIELE DZIAŁAŃ




## MINIWYZWANIE

Dlaczego lepiej korzystać z jednej pętli menu zamiast uruchamiać aplikację ponownie po każdej operacji?
**Odpowiedź:** Możemy wykonywać wiele działań podczas jednej sesji i zachować dane przechowywane w pamięci programu.

# ZBUDUJ KOMPLETNY MENEDŻER BUDŻETU W JAVIE

## KROK 1 — UTWÓRZ PLIK Main.java

### 1 — KLASY I METODY · 01–63

```
import java.util.ArrayList;
import java.util.Locale;
import java.util.Scanner;
public class Main {
  static class Transakcja {
    String opis;
    double kwota;
    boolean przychod;
    Transakcja(String opis, double kwota,
      boolean przychod) {
      this.opis = opis;
      this.kwota = kwota;
      this.przychod = przychod;
    }
  }
  static void dodaj(ArrayList<Transakcja> lista,
    Scanner sc, boolean przychod) {
    System.out.print("Opis: ");
    String opis = sc.nextLine().trim();
    if (opis.isEmpty()) {
      System.out.println(
        "Opis nie moze byc pusty!");
      return;
    }
    System.out.print("Kwota: ");
    try {
      double kwota = Double.parseDouble(
        sc.nextLine().trim());
      if (!Double.isFinite(kwota) || kwota <= 0) {
        System.out.println("Niepoprawna kwota!");
        return;
      }
      lista.add(new Transakcja(
        opis, kwota, przychod));
      System.out.println("Zapisano transakcje!");
    } catch (NumberFormatException e) {
      System.out.println(
        "Wpisz poprawna liczbe!");
    }
  }
  static double suma(ArrayList<Transakcja> lista,
    boolean przychod) {
    double wynik = 0;
    for (Transakcja t : lista) {
      if (t.przychod == przychod) {
        wynik += t.kwota;
      }
    }
    return wynik;
  }
  static void pokaz(ArrayList<Transakcja> lista) {
    if (lista.isEmpty()) {
      System.out.println("Brak transakcji.");
      return;
    }
    for (Transakcja t : lista) {
      String rodzaj = t.przychod
        ? "Przychod" : "Wydatek";
      System.out.printf(Locale.US,
        "%s | %s | %.2f zl%n",
        rodzaj, t.opis, t.kwota);
    }
  }
```

## KROK 2 — WPISZ CAŁY KOD

### 2 — MENU I URUCHOMIENIE · 64–105

```
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ArrayList<Transakcja> lista =
      new ArrayList<>();
    while (true) {
      System.out.println("\nMENEDZER BUDZETU");
      System.out.println("1 - Dodaj przychod");
      System.out.println("2 - Dodaj wydatek");
      System.out.println("3 - Pokaz transakcje");
      System.out.println("4 - Podsumowanie");
      System.out.println("0 - Zakoncz");
      System.out.print("Wybor: ");
      String wybor = sc.nextLine().trim();
      switch (wybor) {
        case "1":
          dodaj(lista, sc, true);
          break;
        case "2":
          dodaj(lista, sc, false);
          break;
        case "3":
          pokaz(lista);
          break;
        case "4":
          double p = suma(lista, true);
          double w = suma(lista, false);
          System.out.printf(Locale.US,
            "Przychody: %.2f zl%n", p);
          System.out.printf(Locale.US,
            "Wydatki: %.2f zl%n", w);
          System.out.printf(Locale.US,
            "Saldo: %.2f zl%n", p - w);
          break;
        case "0":
          System.out.println("Do zobaczenia!");
          return;
        default:
          System.out.println("Nieznana opcja!");
      }
    }
  }
}
```

## KROK 3 — URUCHOM

```
javac Main.java
java Main
```

## PRZYKŁADOWA SESJA — SKRÓT

```
MENEDZER BUDZETU
Wybor: 1
Opis: Wynagrodzenie
Kwota: 3500
Zapisano transakcje!
Wybor: 2
Opis: Zakupy
Kwota: 250
Zapisano transakcje!
Wybor: 4
Przychody: 3500.00 zl
Wydatki: 250.00 zl
Saldo: 3250.00 zl
```

## JAK DZIAŁA PROGRAM?

**1.** Tworzy listę transakcji.
**2.** Wyświetla menu.
**3.** Pobiera decyzję użytkownika.
**4.** Dodaje przychody lub wydatki.
**5.** Sprawdza poprawność wpisanych kwot.
**6.** Przechowuje wszystkie transakcje w pamięci.
**7.** Oblicza podsumowanie.
**8.** Powtarza menu do zakończenia programu.

**WAŻNE INFORMACJE:** Wpisuj kwoty dziesiętne z kropką, np. 125.50. Program wykorzystuje typ double wyłącznie jako uproszczenie edukacyjne. Rzeczywiste systemy finansowe wymagają dokładnej reprezentacji kwot i odpowiednich zasad zaokrąglania. Dane nie są zapisywane na dysku.

# PRZETESTUJ MENEDŻER BUDŻETU I WYKRYJ BŁĘDY

### 01 DODAWANIE PRZYCHODÓW

Dodaj: Wynagrodzenie **3000 zł**
i Dodatkowa praca **500 zł**.
Oczekiwane przychody: **3500 zł**.

### 02 DODAWANIE WYDATKÓW

Dodaj: Zakupy **250 zł**
i Transport **100 zł**.
Oczekiwane wydatki: **350 zł**.

### 03 OBLICZANIE SALDA

Dla poprzednich transakcji: przychody **3500 zł**, wydatki **350 zł**.
Oczekiwane saldo: **3150 zł**.

### 04 UJEMNA KWOTA

Spróbuj dodać wydatek: `-100`
Program odrzuca wartość.

### 05 ZERO

Wpisz: `0`
Program nie dodaje transakcji.

### 06 TEKST ZAMIAST LICZBY

Wpisz: abc
Pojawia się komunikat o niepoprawnej liczbie.

### 07 NIEPOPRAWNA OPCJA MENU

Wybierz: 9
Program wyświetla „Nieznana opcja!” i wraca do menu.

### 08 WARTOŚĆ NIESKOŃCZONA

Wpisz: `Infinity`
Program odrzuca kwotę.

## TABELA KONTROLNA

| Test | Oczekiwany wynik |
|---|---|
| Przychód 3000 + 500 | 3500 zł |
| Wydatki 250 + 100 | 350 zł |
| Saldo | 3150 zł |
| Kwota -100 | Błąd |
| Kwota 0 | Błąd |
| Tekst abc | Błąd |
| Nieznana opcja 9 | Powrót do menu |
| Kwota Infinity | Błąd |

## NAJCZĘSTSZE PROBLEMY

### BŁĘDNE SALDO

Sprawdź, czy program odejmuje wydatki od przychodów.

### TRANSAKCJA MA ZŁY RODZAJ

Sprawdź wartość pola `przychod`.

### KWOTA Z PRZECINKIEM ODRZUCONA

W tej wersji wpisuj liczby z kropką.

### DANE ZNIKAJĄ PO ZAMKNIĘCIU

To oczekiwane zachowanie. Program przechowuje informacje wyłącznie w pamięci operacyjnej.



## WAŻNE

Obliczenia na typie `double` mogą powodować niewielkie różnice zaokrągleń. Testy podstawowe nie zastępują dokładnej arytmetyki wymaganej w rzeczywistym oprogramowaniu finansowym.

# ROZBUDUJ MENEDŻER BUDŻETU O NOWE FUNKCJE

## MISJA — OSTRZEŻENIE O UJEMNYM SALDZIE

Twój program oblicza już przychody i wydatki. Teraz dodaj ostrzeżenie, gdy wydatki przekraczają przychody.

### KROK 1 — OBLICZ SALDO

W gałęzi case "4" po obliczeniu sum przychodów i wydatków dodaj:

```
double saldo = p - w;
```

### KROK 2 — SPRAWDŹ WYNIK

```
if (saldo < 0) {
  System.out.println(
      "Uwaga! Wydatki przekraczaja przychody!"
  );
}
```

### KROK 3 — WYŚWIETL WYNIK

Zastąp wcześniejsze wyświetlanie salda:

```
System.out.printf(Locale.US,
    "Saldo: %.2f zl%n", saldo);
```

### PRZYKŁAD

```
Przychody: 2000 zl
Wydatki: 2500 zl
Saldo: -500 zl
```

**KOMUNIKAT**

```
Uwaga! Wydatki przekraczaja przychody!
```

### DODATKOWE WYZWANIE A

**LICZBA TRANSAKCJI**
Wyświetl liczbę zapisanych operacji.

```
System.out.println(
    "Liczba transakcji: " + lista.size()
);
```

### DODATKOWE WYZWANIE B

**NAJWIĘKSZY WYDATEK**
Przejdź przez listę transakcji i znajdź wydatek o największej kwocie. Wykorzystaj:

- Pętlę `for`.
- Warunek `if`.
- Zmienną przechowującą największą kwotę.

### DODATKOWE WYZWANIE C

**KATEGORIE**
Rozbuduj klasę `Transakcja`, dodając pole:

```
String kategoria;
```

Przykładowe kategorie: **Jedzenie, Transport, Rachunki, Rozrywka, Inne**. Pamiętaj o aktualizacji konstruktora, tworzenia obiektów oraz wyświetlania danych.

### CHECKLISTA UMIEJĘTNOŚCI

- [ ] Potrafię tworzyć klasy reprezentujące transakcje.
- [ ] Umiem przechowywać obiekty w `ArrayList`.
- [ ] Rozumiem różnicę między przychodem i wydatkiem.
- [ ] Potrafię obliczać sumy.
- [ ] Umiem korzystać z metod.
- [ ] Potrafię walidować wartości.
- [ ] Umiem przygotować podsumowanie.
- [ ] Rozumiem ograniczenia obliczeń typu `double`.
- [ ] Potrafię rozbudowywać istniejące aplikacje.



### 8 Z 10 PROJEKTÓW GOTOWYCH!

**8 z 10 projektów**

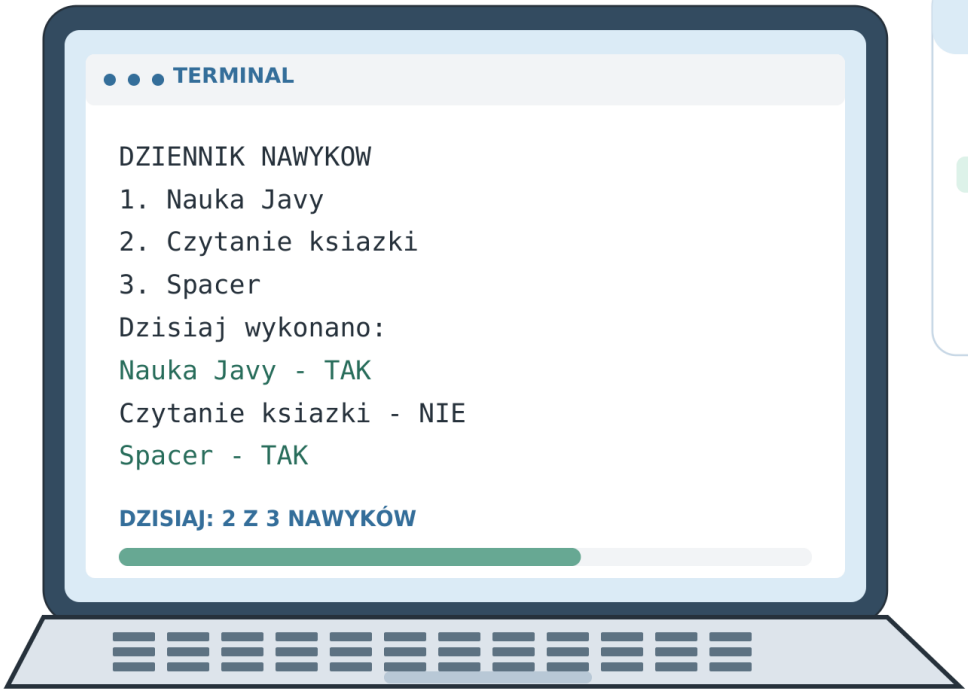
**Gratulacje! Ukończyłeś ósmy projekt w Javie.**

PROJEKT

# 09

# DZIENNIK NAWYKÓW

**Stwórz aplikację, która pomaga śledzić codzienne postępy i regularność działań!**




**PAŹDZIERNIK 2026**

| Pn | Wt | Śr | Cz | Pt | So | Nd |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | |



## CZEGO SIĘ NAUCZYSZ?

- Pracować z datami w Javie.
- Korzystać z klasy LocalDate.
- Tworzyć własną klasę Nawyk.
- Przechowywać wiele nawyków.
- Rejestrować wykonanie czynności.
- Unikać podwójnych zapisów tego samego dnia.
- Liczyć dni aktywności.
- Sprawdzać regularność.

## NOWE POJĘCIA

`LocalDate` — reprezentuje datę bez godziny.
`HashSet` — pozwala przechowywać unikalne elementy.

## WAŻNE

W pierwszej wersji aplikacji dane będą przechowywane tylko w pamięci, podczas jednej sesji działania programu.

**Małe działania wykonywane regularnie mogą prowadzić do wielkich postępów.**

# JAK ZAPROJEKTOWAĆ DZIENNIK CODZIENNYCH NAWYKÓW?

## CEL PROGRAMU

Użytkownik tworzy listę nawyków, które chce regularnie wykonywać. Każdego dnia może zaznaczyć, czy wykonał określony nawyk. Program zapisuje datę wykonania i umożliwia analizowanie regularności.

## PRZYKŁADOWE NAWYKI

### 1 · NAUKA JAVY

Cel: Ćwiczyć programowanie.

### 2 · CZYTANIE

Cel: Czytać codziennie książkę.

### 3 · SPACER

Cel: Poświęcić czas na aktywność fizyczną.

## GŁÓWNE FUNKCJE PROGRAMU

### 1 — DODAJ NAWYK

Użytkownik wpisuje nazwę nowego nawyku.

### 2 — POKAŻ NAWYKI

Program wyświetla wszystkie zapisane nawyki.

### 3 — OZNACZ WYKONANIE

Użytkownik wybiera nawyk i zaznacza, że wykonał go dzisiaj.

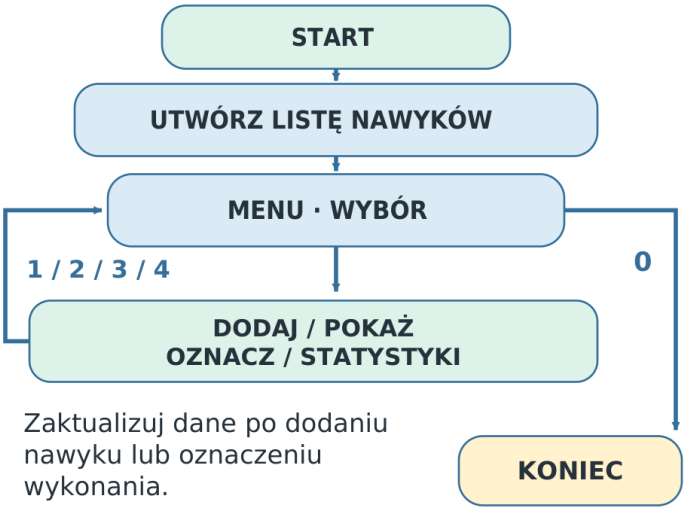
### 4 — POKAŻ STATYSTYKI

Program przedstawia liczbę dni wykonania oraz aktualną serię kolejnych dni.

### 0 — ZAKOŃCZ

Program kończy działanie.

## GŁÓWNY SCHEMAT PROGRAMU




Zaktualizuj dane po dodaniu nawyku lub oznaczeniu wykonania.

## JAKIE DANE PRZECHOWUJEMY?

Dla każdego nawyku potrzebujemy:
**NAZWA** — na przykład: Nauka Javy.
**DATY WYKONANIA** — na przykład: `2026-10-01`, `2026-10-02`, `2026-10-03`.

### PAŹDZIERNIK 2026 · NAUKA JAVY

| 01 | 02 | 03 | 04 |
|---|---|---|---|

**Przykład: dzisiaj jest 2026-10-03.**

## PRZYKŁADOWY RAPORT

```
Nawyk: Nauka Javy
Liczba dni wykonania: 3
Aktualna seria (do dzisiaj): 3 dni
```

## JEDEN ZAPIS NA DZIEŃ

Jeśli użytkownik zaznaczy ten sam nawyk kilka razy tego samego dnia, program powinien zapisać tylko jedną datę. Dzięki temu statystyki nie będą sztucznie zawyżane.

## DODATKOWA INFORMACJA

Data jest pobierana z zegara systemowego komputera. Program nie sprawdza, czy użytkownik rzeczywiście wykonał czynność.

## MINIWYZWANIE

Jeśli użytkownik wykonał nawyk w poniedziałek, wtorek i środę, ile różnych dni aktywności powinien policzyć program? **Odpowiedź: 3 dni.**

# UTWÓRZ KLASĘ NAWYK I ZAPISUJ DATY WYKONANIA

## CO TO JEST LOCALDATE?

LocalDate to klasa reprezentująca konkretną datę, bez informacji o godzinie. Pozwala pracować z dniem, miesiącem i rokiem.

### PRZYKŁAD

```
import java.time.LocalDate;
LocalDate dzisiaj = LocalDate.now();
System.out.println(dzisiaj);
```

**PRZYKŁADOWY FORMAT WYNIKU:**
`2026-10-07`
Wynik `LocalDate.now()` zależy od bieżącej daty. Pokazana data jest tylko przykładem formatu.

## KROK 1 — IMPORTUJ POTRZEBNE KLASY

```
import java.time.LocalDate;
import java.util.Set;
import java.util.HashSet;
```

## KROK 2 — UTWÓRZ KLASĘ NAWYK

```
class Nawyk {
    String nazwa;
    Set<LocalDate> dni = new HashSet<>();
    Nawyk(String nazwa) {
        this.nazwa = nazwa;
    }
}
```

## JAK DZIAŁAJĄ POLA?

`String nazwa` — przechowuje nazwę nawyku.
`Set<LocalDate> dni` — przechowuje unikalne daty wykonania.
`new HashSet<>()` — tworzy pusty zbiór.

**OBIEKT NAWYK: Nauka Javy**
dni: {2026-10-07} — zbiór jednej daty

## CO TO JEST HASHSET?

HashSet to kolekcja, która przechowuje unikalne elementy. Jeśli spróbujemy dodać element, który już istnieje, nie zostanie zapisany po raz drugi. Nie gwarantuje też kolejności przechowywanych elementów.

## DLACZEGO TO PRZYDATNE?

Ponieważ jeden nawyk powinien mieć najwyżej jeden zapis wykonania dla konkretnej daty.

## KROK 3 — UTWÓRZ OBIEKT

```
Nawyk nauka =
    new Nawyk("Nauka Javy");
```

## KROK 4 — DODAJ DZISIEJSZĄ DATĘ

```
nauka.dni.add(LocalDate.now());
```

## KROK 5 — SPRAWDŹ LICZBĘ DAT

```
System.out.println(nauka.dni.size());
```

**WYNIK:** 1

## PRÓBA PONOWNEGO DODANIA

```
nauka.dni.add(LocalDate.now());
System.out.println(nauka.dni.size());
```

**WYNIK:** 1

**DLACZEGO NADAL JEDEN?**
Powtórz dodanie **tego samego dnia**, dla tego samego obiektu. Zbiór nie przechowuje duplikatów tej samej daty.

### ARRAYLIST

Może przechowywać powtarzające się elementy.

| | |
|---|---|
| 2026-10-07 | 2026-10-07 |

### HASHSET

Przechowuje wyłącznie unikalne elementy.

| |
|---|
| 2026-10-07 |

**NOTA TECHNICZNA.** Importy umieść na początku pliku. Klasę `Nawyk` można umieścić w tym samym pliku co `Main`, poza klasą `Main`. Instrukcje przykładu z datą oraz kod tworzący obiekt wpisz w metodzie `main`.

# OZNACZAJ WYKONANIE NAWYKU KAŻDEGO DNIA

## PROBLEM

Użytkownik wykonał dzisiaj nawyk „Nauka Javy”. Chcemy zapisać wykonanie tej czynności i uniemożliwić podwójne naliczanie tego samego dnia.

## KROK 1 — UTWÓRZ NAWYK

```
Nawyk nawyk = new Nawyk("Nauka Javy");
```

## KROK 2 — POBIERZ DZISIEJSZĄ DATĘ

```
LocalDate dzisiaj = LocalDate.now();
```

## KROK 3 — SPRAWDŹ, CZY DATA JUŻ ISTNIEJE

```
if (nawyk.dni.contains(dzisiaj)) {
    System.out.println("Nawyk juz wykonany dzisiaj!");
}
```

## KROK 4 — DODAJ DATĘ

```
boolean dodano = nawyk.dni.add(dzisiaj);
```

## KROK 5 — WYŚWIETL KOMUNIKAT

```
if (dodano) {
    System.out.println("Zapisano wykonanie nawyku!");
} else {
    System.out.println("Nawyk juz oznaczony!");
}
```

## PRZYKŁADOWA SESJA

```
Nawyk: Nauka Javy
Oznacz wykonanie: TAK
Zapisano wykonanie nawyku!
Oznacz wykonanie ponownie: TAK
Nawyk juz oznaczony!
```

## CO ROBI contains()?

Sprawdza, czy wskazany element znajduje się w kolekcji. Zwraca `true` albo `false`.

## CO ZWRACA add()?

### TRUE

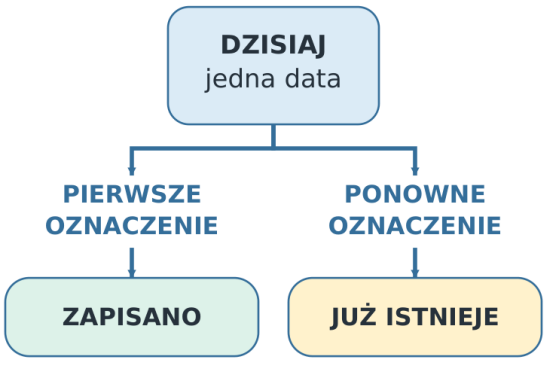
Data nie istniała w zbiorze i została dodana.

### FALSE

Data znajdowała się już w zbiorze.

## JAK TO DZIAŁA W PRAKTYCE?

1. Pobiera aktualną datę.
2. Sprawdza próbę dodania daty.
3. Zapisuje ją, jeśli nie istnieje.
4. Informuje użytkownika o rezultacie.
5. Nie zwiększa liczby dni przy ponownym oznaczeniu.




## WAŻNE

Do samego dodawania wystarczy sprawdzić wynik `add()`. Osobne wcześniejsze wywołanie `contains()` pokazuje stan nawyku przed zapisem. Ponowną próbę wykonaj na tym samym obiekcie i tego samego dnia.

## DODATKOWE WYZWANIE

Wyświetl komunikat, czy nawyk został już wykonany dzisiaj, bez zmieniania danych.
**Podpowiedź:**
`nawyk.dni.contains(LocalDate.now())`

# POLICZ DNI AKTYWNOŚCI I SERIĘ KOLEJNYCH DNI

## CO CHCEMY OBLICZYĆ?

- Łączną liczbę dni wykonania.
- Liczbę wykonań w ostatnich siedmiu dniach.
- Serię kolejnych dni zakończoną dzisiaj.

## KROK 1 — POLICZ WSZYSTKIE DNI

```
int liczbaDni = nawyk.dni.size();
System.out.println("Dni wykonania: " + liczbaDni);
```

Ponieważ HashSet nie przechowuje duplikatów, `size()` wskazuje liczbę różnych dni wykonania nawyku.

## KROK 2 — SPRAWDŹ OSTATNIE SIEDEM DNI

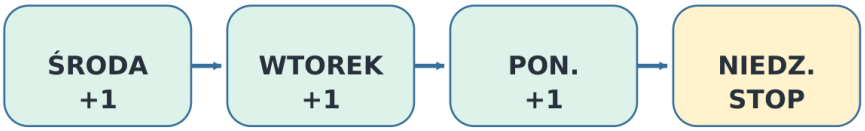
```
LocalDate dzisiaj = LocalDate.now();
LocalDate poczatek = dzisiaj.minusDays(6);
int ostatnie7 = 0;
for (LocalDate dzien : nawyk.dni) {
    if (!dzien.isBefore(poczatek)
            && !dzien.isAfter(dzisiaj)) {
        ostatnie7++;
    }
}
```

## WYNIK

```
System.out.println(
    "Wykonania w ostatnich 7 dniach: " + ostatnie7
);
```

## KROK 3 — POLICZ AKTUALNĄ SERIĘ

```
int seria = 0;
LocalDate dzien = LocalDate.now();
while (nawyk.dni.contains(dzien)) {
    seria++;
    dzien = dzien.minusDays(1);
}
System.out.println("Aktualna seria: " + seria);
```




## OPERACJE NA DATACH

**minusDays(6)** zwraca datę wcześniejszą o sześć dni. Wraz z dzisiejszym dniem otrzymujemy okres siedmiu dni kalendarzowych.

**isBefore()** sprawdza, czy wskazana data jest wcześniejsza od innej daty.

**isAfter()** sprawdza, czy data występuje później.

## JAK DZIAŁA ALGORYTM?

Program rozpoczyna sprawdzanie od dzisiejszej daty. Jeśli nawyk został wykonany dzisiaj, dodaje jeden dzień do serii. Następnie sprawdza wczoraj, przedwczoraj i kolejne wcześniejsze dni. Gdy napotka dzień bez wykonania, kończy liczenie.

## PRZYKŁAD: DZIŚ JEST ŚRODA

Zapisano tylko cztery daty. Pozostałe dni nie mają wykonania.

| | | | |
|---|---|---|---|
| **Czw.** NIE | **Pt.** NIE | **Sob.** TAK | **Niedz.** NIE |
| **Pon.** TAK | **Wt.** TAK | **Śr.** TAK | |

| | |
|---|---|
| **WSZYSTKIE DNI** | **4** |
| **OSTATNIE 7 DNI** | **4** |
| **AKTUALNA SERIA** | **3 dni** |

## WAŻNE

W tej wersji aktualna seria musi obejmować dzisiejszy dzień. Jeśli dzisiaj nie oznaczono wykonania, wynik wynosi **zero**, nawet jeśli nawyk był wykonywany przez wiele poprzednich dni.

## MINIWYZWANIE

Jeżeli nawyk został wykonany dzisiaj, wczoraj i przedwczoraj, ale **nie trzy dni temu**, ile wynosi aktualna seria?
**Odpowiedź: 3 dni.**

# POŁĄCZ NAWYKI, DATY I STATYSTYKI W JEDNEJ APLIKACJI

## KROKI 1-2 — LISTA NAWYKÓW I SCANNER

```
ArrayList<Nawyk> nawyki = new ArrayList<>();
Scanner scanner = new Scanner(System.in);
```

## KROK 4 — UTWÓRZ SZKIELET PROGRAMU

Działający szkielet menu. Poszczególne komunikaty zastąpimy właściwymi operacjami.

```
while (true) {
    System.out.println("\nDZIENNIK NAWYKOW");
    System.out.println("1 - Dodaj nawyk");
    System.out.println("2 - Pokaz nawyki");
    System.out.println("3 - Oznacz wykonanie");
    System.out.println("4 - Pokaz statystyki");
    System.out.println("0 - Zakoncz");
    System.out.print("Wybor: ");
    String wybor = scanner.nextLine().trim();
    switch (wybor) {
        case "1":
            System.out.println("Dodawanie nawyku");
            break;
        case "2":
            System.out.println("Lista nawykow");
            break;
        case "3":
            System.out.println("Oznaczanie wykonania");
            break;
        case "4":
            System.out.println("Statystyki");
            break;
        case "0":
            return;
        default:
            System.out.println("Nieznana opcja!");
    }
}
```

## DOCELOWA SESJA — PO ROZBUDOWIE

### 1 — POCZĄTEK

```
DZIENNIK NAWYKOW
Wybor: 1
Nazwa nawyku: Nauka Javy
Dodano nawyk!
Wybor: 2
1. Nauka Javy
Wybor: 3
```

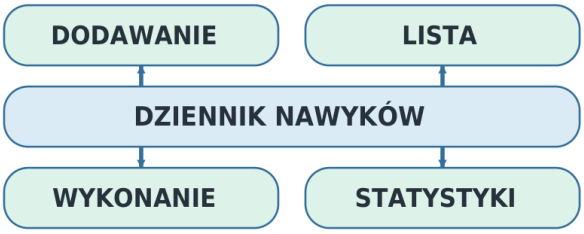
### 2 — CIĄG DALSZY

```
Numer nawyku: 1
Zapisano wykonanie!
Wybor: 4
Numer nawyku: 1
Dni wykonania: 1
Aktualna seria: 1
```

## CO JUŻ POTRAFIMY?

- Tworzyć obiekty Nawyk.
- Przechowywać je w ArrayList.
- Zapisywać daty w HashSet.
- Zapobiegać duplikatom.
- Obliczać liczbę dni.
- Sprawdzać kolejne dni aktywności.
- Organizować działania za pomocą menu.

## CEL — JEDNA APLIKACJA




0 — ZAKOŃCZ DZIAŁANIE

## KROK 3 — ZAPLANUJ MENU

```
DZIENNIK NAWYKOW
1 - Dodaj nawyk
2 - Pokaz nawyki
3 - Oznacz wykonanie
4 - Pokaz statystyki
0 - Zakoncz
```

## KROK 5 — ZAPLANUJ FUNKCJE

**1 — DODAJ NAWYK:** Pobierz nazwę, sprawdź, czy nie jest pusta, i utwórz nowy obiekt Nawyk.

**2 — POKAŻ NAWYKI:** Przejdź przez listę i wyświetl numer oraz nazwę każdego nawyku.

**3 — OZNACZ WYKONANIE:** Pobierz numer nawyku. Sprawdź, czy obiekt istnieje, i dodaj dzisiejszą datę do jego zbioru.

**4 — POKAŻ STATYSTYKI:** Oblicz liczbę dni wykonania i aktualną serię dla wybranego nawyku.

**0 — ZAKOŃCZ:** Zakończ działanie programu.

## KROK 6 — PRZEWIDUJ BŁĘDY

**PUSTA NAZWA:** Nie dodawaj nawyku bez nazwy.

**BŁĘDNY NUMER:** Nie odczytuj obiektu, który nie istnieje na liście.

**POWTÓRNE OZNACZENIE:** Nie zapisuj tej samej daty ponownie.

**PUSTA LISTA:** Poinformuj użytkownika, że nie utworzył jeszcze żadnych nawyków.

**URUCHOMIENIE.** Dodaj `import java.util.ArrayList;` i `import java.util.Scanner;`. Użyj klasy Nawyk i importów ze str. 76. Kroki 1, 2 i 4 wpisz w metodzie `main` klasy `Main`.

**WAŻNE:** Sesja pokazuje cel po rozbudowie; szkielet wyświetla tylko komunikaty. Na następnej stronie połączymy wszystkie elementy w kompletny program Java.

# ZBUDUJ KOMPLETNY DZIENNIK NAWYKÓW W JAVIE

### 1 — UTWÓRZ PLIK Main.java W FOLDERZE PROJEKTU

### CZĘŚĆ 1 — KLASA I METODY · 01–66

```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;
public class Main {
  static class Nawyk {
    String nazwa;
    Set<LocalDate> dni = new HashSet<>();
    Nawyk(String nazwa) {
      this.nazwa = nazwa;
    }
  }
  static void pokaz(ArrayList<Nawyk> lista) {
    if (lista.isEmpty()) {
      System.out.println("Brak nawykow.");
      return;
    }
    for (int i = 0; i < lista.size(); i++) {
      System.out.println(
        (i + 1) + ". " + lista.get(i).nazwa);
    }
  }
  static Nawyk wybierz(ArrayList<Nawyk> lista,
    Scanner sc) {
    if (lista.isEmpty()) {
      System.out.println("Brak nawykow.");
      return null;
    }
    pokaz(lista);
    System.out.print("Numer nawyku: ");
    try {
      int nr = Integer.parseInt(
        sc.nextLine().trim());
      if (nr >= 1 && nr <= lista.size()) {
        return lista.get(nr - 1);
      }
    } catch (NumberFormatException e) {
      // Niepoprawny numer.
    }
    System.out.println("Niepoprawny numer!");
    return null;
  }
  static void statystyki(Nawyk n) {
    LocalDate dzis = LocalDate.now();
    LocalDate poczatek = dzis.minusDays(6);
    int ostatnie7 = 0;
    for (LocalDate d : n.dni) {
      if (!d.isBefore(poczatek)
        && !d.isAfter(dzis)) {
        ostatnie7++;
      }
    }
    int seria = 0;
    LocalDate dzien = dzis;
    while (n.dni.contains(dzien)) {
      seria++;
      dzien = dzien.minusDays(1);
    }
    System.out.println("Dni wykonania: "
      + n.dni.size());
    System.out.println("Ostatnie 7 dni: "
      + ostatnie7);
    System.out.println("Aktualna seria: "
      + seria);
  }
```

### 2 — WPISZ KOMPLETNY KOD

### CZĘŚĆ 2 — MENU I DZIAŁANIE · 67–119

```
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ArrayList<Nawyk> lista = new ArrayList<>();
    while (true) {
      System.out.println("\nDZIENNIK NAWYKOW");
      System.out.println("1 - Dodaj nawyk");
      System.out.println("2 - Pokaz nawyki");
      System.out.println("3 - Oznacz wykonanie");
      System.out.println("4 - Statystyki");
      System.out.println("0 - Koniec");
      System.out.print("Wybor: ");
      String wybor = sc.nextLine().trim();
      switch (wybor) {
        case "1":
          System.out.print("Nazwa: ");
          String nazwa = sc.nextLine().trim();
          if (nazwa.isEmpty()) {
            System.out.println("Pusta nazwa!");
          } else {
            lista.add(new Nawyk(nazwa));
            System.out.println("Dodano nawyk!");
          }
          break;
        case "2":
          pokaz(lista);
          break;
        case "3":
          Nawyk n = wybierz(lista, sc);
          if (n != null) {
            if (n.dni.add(LocalDate.now())) {
              System.out.println(
                "Zapisano wykonanie!");
            } else {
              System.out.println(
                "Juz wykonano dzisiaj!");
            }
          }
          break;
        case "4":
          Nawyk s = wybierz(lista, sc);
          if (s != null) {
            statystyki(s);
          }
          break;
        case "0":
          System.out.println("Do zobaczenia!");
          return;
        default:
          System.out.println("Nieznana opcja!");
      }
    }
  }
}
```

### KROK 3 — URUCHOM

```
javac Main.java
java Main
```

### PRZYKŁADOWA SESJA · CZYTAJ 1 → 2

```
DZIENNIK NAWYKOW
Wybor: 1
Nazwa: Nauka Javy
Dodano nawyk!
Wybor: 3
1. Nauka Javy
Numer nawyku: 1
```

```
Zapisano wykonanie!
Wybor: 4
1. Nauka Javy
Numer nawyku: 1
Dni wykonania: 1
Ostatnie 7 dni: 1
Aktualna seria: 1
```

## JAK DZIAŁA PROGRAM?

**1.** Tworzy kolekcję obiektów Nawyk.
**2.** Pozwala dodawać nowe nawyki.
**3.** Wyświetla zapisane pozycje.
**4.** Rejestruje dzisiejsze wykonanie.
**5.** Zapobiega duplikatom tej samej daty.
**6.** Oblicza liczbę dni aktywności.
**7.** Analizuje wykonania z ostatnich 7 dni.
**8.** Oblicza serię kolejnych dni.
**9.** Powtarza menu aż do zakończenia.

**WAŻNE INFORMACJE:** Program wykorzystuje aktualną datę systemową. Aktualna seria jest liczona od dzisiaj wstecz. Jeżeli nawyk nie został oznaczony dzisiaj, seria wynosi zero. Dane są przechowywane wyłącznie w pamięci i nie przetrwają ponownego uruchomienia.

# PRZETESTUJ DZIENNIK I ROZBUDUJ SWOJE NAWYKI

## CZĘŚĆ 1 — TESTY PROGRAMU

### 01 — DODANIE NOWEGO NAWYKU

Wpisz: **Nauka Javy**
Program tworzy nowy nawyk i wyświetla go na liście.

### 02 — OZNACZENIE WYKONANIA

Wybierz utworzony nawyk i oznacz wykonanie.
Liczba dni wykonania zwiększa się do **1**.

### 03 — PODWÓJNE OZNACZENIE

Oznacz ten sam nawyk ponownie tego samego dnia. Program informuje, że został już wykonany. Liczba zapisanych dni nie zmienia się.

### 04 — PUSTA NAZWA

Spróbuj dodać nawyk bez nazwy. Program nie dodaje nowego obiektu.

### 05 — NIEPOPRAWNY NUMER

Wpisz numer nieistniejący na liście. Program wyświetla komunikat o błędzie i zachowuje dotychczasowe dane.

### 06 — SPRAWDZANIE SERII DNI

Utwórz tymczasowy nawyk i dodaj trzy kolejne daty:

```
Nawyk test = new Nawyk("Test");
LocalDate dzis = LocalDate.now();
test.dni.add(dzis);
test.dni.add(dzis.minusDays(1));
test.dni.add(dzis.minusDays(2));
statystyki(test);
```

**Oczekiwany wynik:**

```
Dni wykonania: 3
Ostatnie 7 dni: 3
Aktualna seria: 3
```

Fragment uruchom tymczasowo wewnątrz metody `main`, a po testowaniu usuń.

## CZĘŚĆ 2 — USUWANIE NAWYKÓW

### KROK 1 — ROZSZERZ MENU

Dodaj opcję:

```
5 - Usun nawyk
```

### KROK 2 — WYBIERZ OBIEKT

Wykorzystaj istniejącą metodę `wybierz()`, aby wskazać obiekt.

### KROK 3 — USUŃ NAWYK

Po poprawnym wyborze usuń go z listy.
Podpowiedź:

```
Nawyk wybrany = wybierz(lista, sc);
if (wybrany != null) {
  lista.remove(wybrany);
  System.out.println("Usunieto nawyk!");
}
```

### WYZWANIE A — PROCENT REGULARNOŚCI

Oblicz, przez ile z ostatnich siedmiu dni wykonano nawyk.

**WZÓR**

**REGULARNOŚĆ =**
**WYKONANE DNI / 7 × 100%**

### WYZWANIE B — NAJWYŻSZA SERIA

Spróbuj obliczyć najdłuższą serię kolejnych dni w całej historii nawyku, a nie tylko serię rozpoczynającą się dzisiaj.

### WYZWANIE C — EDYCJA NAZWY

Dodaj opcję zmiany nazwy istniejącego nawyku.

## CHECKLISTA UMIEJĘTNOŚCI

- ☐ Rozumiem klasę `LocalDate`.
- ☐ Potrafię korzystać z `HashSet`.
- ☐ Umiem zapobiegać duplikatom.
- ☐ Potrafię porównywać daty.
- ☐ Rozumiem `minusDays()`.
- ☐ Umiem tworzyć klasy i obiekty.
- ☐ Potrafię analizować kolejne dni aktywności.
- ☐ Umiem organizować interaktywne menu.



**9 Z 10 PROJEKTÓW GOTOWYCH!**



**9 z 10 projektów**



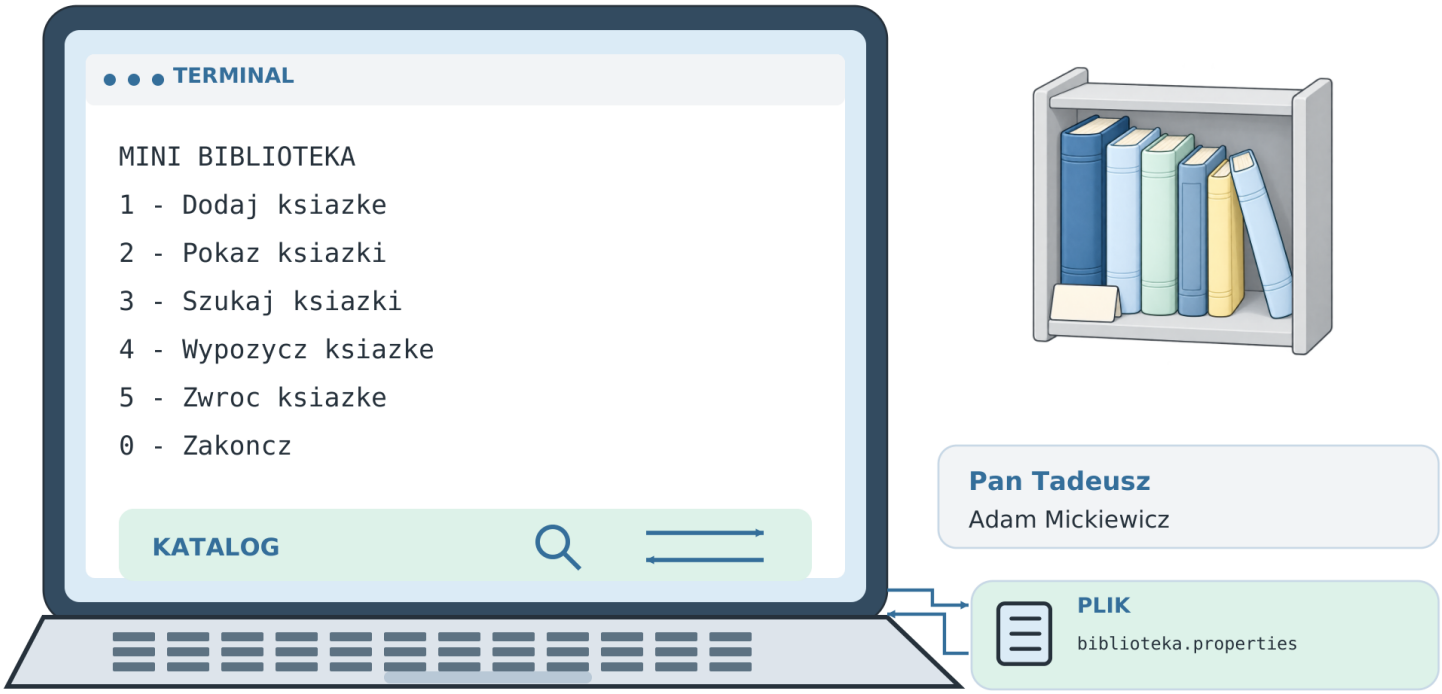
**Gratulacje! Ukończyłeś dziewiąty projekt w Javie.**

PROJEKT

10

# MINI BIBLIOTEKA

**Zbuduj własny system do zarządzania książkami i zapisywania danych na dysku!**




## CZEGO SIĘ NAUCZYSZ?

- Tworzenia klasy Ksiazka.
- Przechowywania obiektów.
- Organizowania katalogu.
- Wyszukiwania informacji.
- Zarządzania statusem wypożyczenia.
- Korzystania z metod.
- Zapisywania danych w pliku.
- Odczytywania zapisanych informacji.
- Obsługi podstawowych błędów zapisu i odczytu.

## NOWE POJĘCIE — TRWAŁOŚĆ DANYCH

Dotychczas większość programów przechowywała informacje tylko podczas działania. Tym razem zapiszemy dane do pliku, aby można było odtworzyć je po ponownym uruchomieniu programu.

**To Twój dziesiąty projekt. Połącz wszystkie zdobyte umiejętności w jednej aplikacji!**

# JAK ZAPROJEKTOWAĆ WŁASNĄ MINI BIBLIOTEKĘ?

## CEL PROGRAMU

Użytkownik powinien mieć możliwość przechowywania informacji o książkach, wyszukiwania wybranych pozycji oraz zmieniania ich dostępności. Program będzie zapisywał dane w pliku.

## JAKIE INFORMACJE PRZECHOWUJEMY?

Każda książka posiada trzy podstawowe pola.

### 1 · TYTUŁ

Pan Tadeusz

### 2 · AUTOR

Adam Mickiewicz

### 3 · STATUS

Dostępna albo wypożyczona

## GŁÓWNE FUNKCJE PROGRAMU

### 1 — DODAJ KSIĄŻKĘ

Użytkownik wpisuje tytuł i autora.

### 2 — POKAŻ KSIĄŻKI

Program wyświetla wszystkie pozycje wraz ze statusem.

### 3 — SZUKAJ KSIĄŻKI

Użytkownik wyszukuje książki po tytule lub autorze.

### 4 — WYPOŻYCZ KSIĄŻKĘ

Program zmienia status dostępnej książki na wypożyczoną.

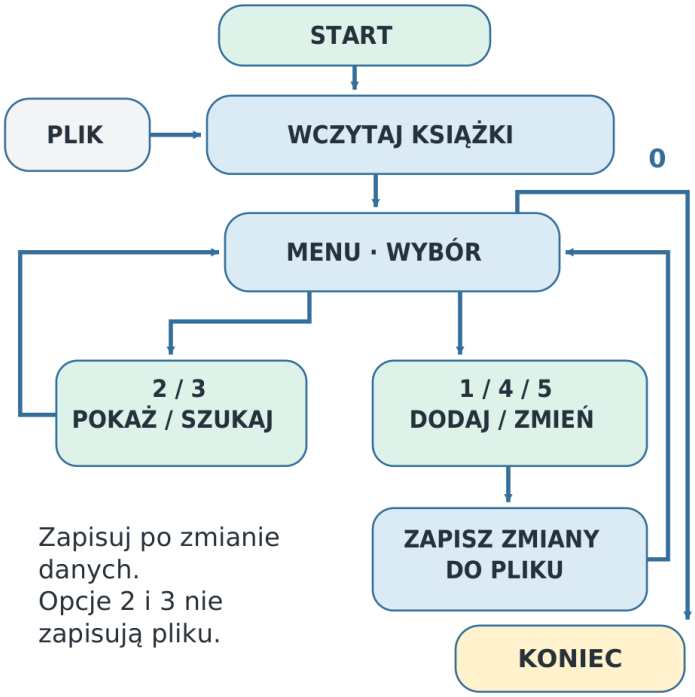
### 5 — ZWRÓĆ KSIĄŻKĘ

Program przywraca status dostępności.

### 0 — ZAKOŃCZ

Program kończy działanie.

## JAK DZIAŁA ZAPIS DANYCH?




Zapisuj po zmianie danych.
Opcje 2 i 3 nie zapisują pliku.




wypożycz / zwróć

## ZASADY WYPOŻYCZANIA

- Dostępną książkę można wypożyczyć.
- Wypożyczonej książki nie można wypożyczyć ponownie przed zwrotem.
- Zwrot zmienia status książki na dostępny.
- Próba zwrotu dostępnej książki nie zmienia jej statusu.

## WAŻNE OGRANICZENIE

W tym uproszczonym projekcie zapisujemy tylko stan wypożyczenia. Nie przechowujemy danych czytelników, terminów zwrotu ani historii wypożyczeń.

## MINIWYZWANIE

Co zrobić, gdy użytkownik próbuje wypożyczyć książkę ze statusem „Wypożyczona”? **Odpowiedź:** Wyświetlić odpowiedni komunikat i pozostawić dane bez zmian.

# STWÓRZ KLASĘ KSIĄŻKA I ZARZĄDZAJ JEJ STANEM

## CO POWINNA ZAWIERAĆ KLASA KSIĄŻKA?

Każdy obiekt będzie przechowywał:

**TYTUŁ**

Tytuł książki.

**AUTOR**

Imię i nazwisko autora.

**STAN**

Informacja o wypożyczeniu.

## KROK 1 — UTWÓRZ KLASĘ

```
class Ksiazka {
    String tytul;
    String autor;
    boolean wypozyczona;
    Ksiazka(String tytul, String autor) {
        this.tytul = tytul;
        this.autor = autor;
        this.wypozyczona = false;
    }
}
```

## KROK 2 — UTWÓRZ PIERWSZY OBIEKT

```
Ksiazka ksiazka1 = new Ksiazka(
    "Pan Tadeusz",
    "Adam Mickiewicz"
);
```

## KROK 3 — WYŚWIETL INFORMACJE

```
System.out.println(ksiazka1.tytul);
System.out.println(ksiazka1.autor);
System.out.println(ksiazka1.wypozyczona);
```

## KROK 4 — ZMIEŃ STATUS

```
ksiazka1.wypozyczona = true;
```

## KROK 5 — SPRAWDŹ REZULTAT

```
System.out.println(ksiazka1.wypozyczona);
```

**WYNIK:** true

## JAK DZIAŁAJĄ POLA?

`tytul` — przechowuje tytuł.
`autor` — przechowuje autora.
`wypozyczona` — przechowuje informację o stanie wypożyczenia.

**FALSE**

`false` oznacza, że książka jest dostępna.

**TRUE**

`true` oznacza, że książka została wypożyczona.

## WYNIK KROKU 3

```
Pan Tadeusz
Adam Mickiewicz
false
```

## JAK INTERPRETOWAĆ ZMIANĘ?

**DOSTĘPNA**

przed zmianą: false

↓

**WYPOŻYCZONA**

po zmianie: true

**WAŻNE.** Zmienna boolean pozwala wygodnie przechowywać informację, która może przyjmować tylko dwa stany. W bardziej zaawansowanej bibliotece można wykorzystać inne struktury do reprezentowania większej liczby statusów.

**MINIWYZWANIE.** Utwórz drugi obiekt reprezentujący książkę o dowolnym tytule i autorze. Następnie sprawdź, czy początkowo ma status dostępny.

# DODAWAJ KSIĄŻKI I TWÓRZ WŁASNY KATALOG

## KROK 1 — UTWÓRZ LISTĘ

```
Scanner scanner = new Scanner(System.in);
ArrayList<Ksiazka> ksiazki = new ArrayList<>();
```

Listę i Scanner utwórz tylko raz.

## KROK 2 — ZAPYTAJ O TYTUŁ

```
System.out.print("Tytul: ");
String tytul = scanner.nextLine().trim();
```

## KROK 3 — ZAPYTAJ O AUTORA

```
System.out.print("Autor: ");
String autor = scanner.nextLine().trim();
```

## KROK 4 — SPRAWDŹ DANE

```
if (tytul.isEmpty() || autor.isEmpty()) {
    System.out.println("Uzupelnij wszystkie pola!");
} else {
    ksiazki.add(new Ksiazka(tytul, autor));
    System.out.println("Dodano ksiazke!");
}
```

## KROK 5 — WYŚWIETL KATALOG

```
for (int i = 0; i < ksiazki.size(); i++) {
    Ksiazka k = ksiazki.get(i);
    String status = k.wypozyczona
        ? "Wypozyczona" : "Dostepna";
    System.out.println(
        (i + 1) + ". " + k.tytul +
        " - " + k.autor + " [" + status + "]"
    );
}
```

## KROK 6 — OBSŁUŻ PUSTY KATALOG

Przed wyświetlaniem:

```
if (ksiazki.isEmpty()) {
    System.out.println("Biblioteka jest pusta!");
}
```

## PRZYGOTUJ PROGRAM

```
import java.util.ArrayList;
import java.util.Scanner;
```

Importy umieść na początku pliku. Użyj klasy `Ksiazka` ze strony 84, poza klasą `Main`. Instrukcje wpisz w metodzie `main`.

## PRZYKŁADOWE DANE

**Tytuł:** Pan Tadeusz
**Autor:** Adam Mickiewicz

**WYNIK:** Dodano ksiazke!

## DLACZEGO UŻYWAMY NUMERÓW?

Numery pomagają użytkownikowi wybierać konkretne książki podczas wypożyczania i zwracania.

### WAŻNE

Numer wyświetlany w katalogu jest równy indeksowi listy powiększonemu o jeden.

## PRZYPOMNIENIE

ArrayList pozwala przechowywać wiele obiektów tego samego typu. W naszym projekcie kolekcja zawiera obiekty `Ksiazka`.

### MINIWYZWANIE

Dodaj trzy książki i sprawdź wynik metody `ksiazki.size()`.
**Odpowiedź: 3.**
Powtórz kroki 2–4, zachowując tę samą listę.

## PRZYKŁADOWY KATALOG — PO DODANIU TRZECH KSIĄŻEK

| | |
|---|---|
| **Pan Tadeusz** → | `1. Pan Tadeusz - Adam Mickiewicz [Dostepna]` |
| **Lalka** → | `2. Lalka - Boleslaw Prus [Dostepna]` |
| **Quo vadis** → | `3. Quo vadis - Henryk Sienkiewicz [Dostepna]` |

# WYSZUKUJ, WYPOŻYCZAJ I ZWRACAJ KSIĄŻKI

## CZĘŚĆ 1 — WYSZUKIWANIE KSIĄŻEK

Użytkownik wpisuje tytuł lub fragment nazwiska autora.

### KROK 1 — POBIERZ ZAPYTANIE

```
System.out.print("Szukaj: ");
String szukane = scanner.nextLine().trim()
    .toLowerCase(java.util.Locale.ROOT);
```

### KROKI 2–3 — SPRAWDŹ I WYSZUKAJ

```
if (szukane.isEmpty()) {
  System.out.println("Wpisz szukana fraze!");
} else {
  for (Ksiazka k : ksiazki) {
    if (k.tytul
        .toLowerCase(java.util.Locale.ROOT)
        .contains(szukane)
        || k.autor
        .toLowerCase(java.util.Locale.ROOT)
        .contains(szukane)) {
      System.out.println(
          k.tytul + " - " + k.autor);
    }
  }
}
```

### JAK SPRAWDZIĆ NUMER?

Najpierw pokaż katalog z numerami. Zamień wpis na liczbę i sprawdź zakres **przed** wywołaniem `get()`:

```
System.out.print("Numer ksiazki: ");
try {
  int numer = Integer.parseInt(
      scanner.nextLine().trim());
  if (numer >= 1 && numer <= ksiazki.size()) {
    // Wstaw fragment A albo B.
  } else {
    System.out.println("Niepoprawny numer!");
  }
} catch (NumberFormatException e) {
  System.out.println("Wpisz liczbe calkowita!");
}
```

## CZĘŚĆ 2 — WYPOŻYCZANIE KSIĄŻKI

**FRAGMENT A.** Po otrzymaniu poprawnego numeru, wewnątrz warunku zakresu:

```
Ksiazka k = ksiazki.get(numer - 1);
if (k.wypozyczona) {
  System.out.println(
      "Ksiazka juz wypozyczona!");
} else {
  k.wypozyczona = true;
  System.out.println("Wypozyczono ksiazke!");
}
```

## CZĘŚĆ 3 — ZWROT KSIĄŻKI

**FRAGMENT B.** Po wybraniu poprawnego numeru, zamiast fragmentu A:

```
Ksiazka k = ksiazki.get(numer - 1);
if (!k.wypozyczona) {
  System.out.println(
      "Ksiazka jest juz dostepna!");
} else {
  k.wypozyczona = false;
  System.out.println("Zwrocono ksiazke!");
}
```

### WAŻNE

Próba wypożyczenia już wypożyczonej książki nie powinna zmieniać jej statusu. Podobnie zwrot książki już dostępnej nie powinien powodować żadnej zmiany.

### PRZYKŁAD

```
1. Pan Tadeusz [Dostepna]
Wybor: Wypozycz 1
Wypozyczono ksiazke!
1. Pan Tadeusz [Wypozyczona]
Wybor: Zwroc 1
Zwrocono ksiazke!
```




**JAK POŁĄCZYĆ FRAGMENTY?** Użyj klasy `Ksiazka`, listy i `Scanner` ze strony 85. Fragment A albo B wstaw do warunku poprawnego numeru. To alternatywne operacje; nie wklejaj obu deklaracji `k` do jednego bloku.

### DODATKOWE WYZWANIE

Dodaj licznik pokazujący, ile książek jest aktualnie dostępnych.

# ZAPISUJ KSIĄŻKI DO PLIKU I ODCZYTUJ JE PONOWNIE

**DLACZEGO ZAPIS DO PLIKU JEST WAŻNY?** Dotychczas dane naszych aplikacji znikały po ich zamknięciu. W mini bibliotece chcemy zachować katalog między kolejnymi uruchomieniami.

## KROK 1 — IMPORTUJ NARZĘDZIA

```
import java.util.Properties;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.charset.StandardCharsets;
import java.io.Reader;
import java.io.Writer;
import java.io.IOException;
```

## KROK 2 — UTWÓRZ ŚCIEŻKĘ PLIKU

```
Path plik =
    Path.of("biblioteka.properties");
```

## KROK 3 — PRZYGOTUJ DANE

```
Properties dane = new Properties();
dane.setProperty("n",
    String.valueOf(ksiazki.size()));
for (int i = 0; i < ksiazki.size(); i++) {
  Ksiazka k = ksiazki.get(i);
  dane.setProperty("t" + i, k.tytul);
  dane.setProperty("a" + i, k.autor);
  dane.setProperty("w" + i,
      String.valueOf(k.wypozyczona));
}
```

## KROK 4 — ZAPISZ DO PLIKU

```
try (Writer zapis = Files.newBufferedWriter(
    plik, StandardCharsets.UTF_8)) {
  dane.store(zapis, "Mini biblioteka");
} catch (IOException e) {
  System.out.println("Blad zapisu danych!");
}
```

## CO OZNACZA TRY-WITH-RESOURCES?

W kroku 5 zapis `try (Reader czytnik = Files.newBufferedReader(plik, StandardCharsets.UTF_8))` otwiera zasób i automatycznie zamyka go po zakończeniu operacji. Blok `catch` obsługuje `IOException`.

## CO TO JEST PLIK PROPERTIES?

To plik, który przechowuje informacje w postaci par `klucz=wartosc`. Przykład:

```
n=1
t0=Pan Tadeusz
a0=Adam Mickiewicz
w0=false
```

**ZNACZENIE KLUCZY:**
`n` — liczba książek.
`t0` — tytuł pierwszej książki.
`a0` — autor pierwszej książki.
`w0` — status pierwszej książki.

## KROK 5 — ODCZYTAJ ZAPISANY PLIK

```
Properties odczyt = new Properties();
if (Files.exists(plik)) {
  try (Reader czytnik =
      Files.newBufferedReader(
          plik, StandardCharsets.UTF_8)) {
    odczyt.load(czytnik);
  } catch (IOException e) {
    System.out.println(
        "Blad odczytu danych!");
  }
}
```

## KROK 6 — ODTWÓRZ INFORMACJE

Po odczytaniu pliku pobierz liczbę zapisanych książek. Następnie odczytuj odpowiadające sobie klucze tytułu, autora i statusu. Na tej podstawie twórz nowe obiekty `Ksiazka` i dodawaj je do kolekcji. Pełna implementacja znajduje się na następnej stronie.

## CO OZNACZA IOEXCEPTION?

IOException reprezentuje błędy związane między innymi z operacjami wejścia i wyjścia, na przykład odczytem lub zapisem plików.

**WAŻNE INFORMACJE.** Plik `biblioteka.properties` będzie tworzony w katalogu roboczym uruchomionego programu. W przykładzie używamy kodowania UTF-8. Klasa Properties zapisuje również informacje techniczne i odpowiednio przekształca znaki specjalne. Dlatego rzeczywista zawartość pliku może różnić się wizualnie od uproszczonego schematu. Nie należy traktować lokalnego pliku jako zabezpieczonej bazy danych.



# ZBUDUJ MINI BIBLIOTEKĘ Z ZAPISEM DANYCH

## KROK 1 — UTWÓRZ PLIK Main.java

**CZĘŚĆ 1 — STRUKTURA, PLIK I METODY.** Czytaj lewą kolumnę, potem prawą. Część 2 — menu i obsługa użytkownika — znajduje się na dołączonym arkuszu kodu do strony 88.

### 1 — WIERSZE 01-62

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;
import java.util.*;
public class Main {
  static final Path PLIK =
  Path.of("biblioteka.properties");
  static class Ksiazka {
    String tytul, autor;
    boolean wypozyczona;
    Ksiazka(String t, String a) {
      tytul = t;
      autor = a;
    }
  }
  static void zapisz(List<Ksiazka> lista) {
    Properties p = new Properties();
    p.setProperty("n", "" + lista.size());
    for (int i = 0; i < lista.size(); i++) {
      Ksiazka k = lista.get(i);
      p.setProperty("t" + i, k.tytul);
      p.setProperty("a" + i, k.autor);
      p.setProperty("w" + i,
        "" + k.wypozyczona);
    }
    try (Writer w = Files.newBufferedWriter(
      PLIK, StandardCharsets.UTF_8)) {
      p.store(w, "Mini biblioteka");
    } catch (IOException e) {
      System.out.println("Blad zapisu!");
    }
  }
  static void wczytaj(List<Ksiazka> lista) {
    if (!Files.exists(PLIK)) return;
    Properties p = new Properties();
    try (Reader r = Files.newBufferedReader(
      PLIK, StandardCharsets.UTF_8)) {
      p.load(r);
      int n = Integer.parseInt(
        p.getProperty("n", "0"));
      if (n < 0) throw new IOException(
        "Zla liczba");
      List<Ksiazka> tmp = new ArrayList<>();
      for (int i = 0; i < n; i++) {
        String t = p.getProperty("t" + i);
        String a = p.getProperty("a" + i);
        String w = p.getProperty("w" + i);
        if (t == null || a == null ||
          (!"true".equals(w) &&
          !"false".equals(w))) {
          throw new IOException("Zle dane");
        }
        Ksiazka k = new Ksiazka(t, a);
        k.wypozyczona = Boolean.parseBoolean(w);
        tmp.add(k);
      }
      lista.addAll(tmp);
    } catch (IOException | NumberFormatException
      e) {
      System.out.println("Blad odczytu danych!");
    }
  }
```

## KROK 2 — WPISZ KOMPLETNY KOD

### 2 — WIERSZE 63-101

```
  static void pokaz(List<Ksiazka> lista) {
    if (lista.isEmpty()) {
      System.out.println("Brak ksiazek.");
      return;
    }
    for (int i = 0; i < lista.size(); i++) {
      Ksiazka k = lista.get(i);
      String s = k.wypozyczona
      ? "Wypozyczona" : "Dostepna";
      System.out.println((i + 1) + ". " +
        k.tytul + " - " + k.autor +
        " [" + s + "]");
    }
  }
  static void zmien(List<Ksiazka> lista,
    Scanner sc, boolean status) {
    pokaz(lista);
    if (lista.isEmpty()) return;
    System.out.print("Numer ksiazki: ");
    try {
      int nr = Integer.parseInt(
        sc.nextLine().trim());
      if (nr < 1 || nr > lista.size()) {
        System.out.println("Zly numer!");
        return;
      }
      Ksiazka k = lista.get(nr - 1);
      if (k.wypozyczona == status) {
        System.out.println("Status bez zmian.");
        return;
      }
      k.wypozyczona = status;
      zapisz(lista);
      System.out.println(status
        ? "Wypozyczono!" : "Zwrocono!");
    } catch (NumberFormatException e) {
      System.out.println("Wpisz liczbe!");
    }
  }
```

## KROK 3 — URUCHOM

```
javac Main.java
java Main
```

## PRZYKŁADOWA SESJA

```
MINI BIBLIOTEKA
Wybor: 1
Tytul: Pan Tadeusz
Autor: Adam Mickiewicz
Dodano!
Wybor: 2
1. Pan Tadeusz - Adam Mickiewicz [Dostepna]
Wybor: 4
1. Pan Tadeusz - Adam Mickiewicz [Dostepna]
Numer ksiazki: 1
Wypozyczono!
Wybor: 2
1. Pan Tadeusz - Adam Mickiewicz [Wypozyczona]
```

## CO POTRAFI PROGRAM?

**1.** Przechowuje obiekty książek.
**2.** Pozwala dodawać nowe pozycje.
**3.** Wyświetla katalog.
**4.** Wyszukuje książki po tytule lub autorze.
**5.** Obsługuje wypożyczenia.
**6.** Obsługuje zwroty.
**7.** Sprawdza poprawność numeru książki.
**8.** Zapisuje dane po dodaniu lub zmianie statusu.
**9.** Wczytuje dane podczas uruchamiania.
**10.** Pozwala zakończyć pracę przez menu.

**WAŻNE OGRANICZENIA:** To edukacyjna aplikacja konsolowa przeznaczona dla jednego użytkownika. Zapis do pliku nie jest transakcyjny: awaria podczas zapisu może uszkodzić plik. Jeżeli zapis się nie powiedzie, program wyświetla komunikat o błędzie, ale bieżąca lista nadal pozostaje zmieniona w pamięci. Program nie obsługuje równoczesnej pracy wielu użytkowników ani nie przechowuje danych osób wypożyczających książki.

ARKUSZ KODU DO STRONY 88

# MINI BIBLIOTEKA — KOD, CZĘŚĆ 2

Dopisz poniższy kod bezpośrednio po metodzie `zmien` ze strony 88, w tym samym pliku `Main.java`. Ostatni nawias zamyka klasę `Main`. Wiersze 102–170.

```
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    List<Ksiazka> lista = new ArrayList<>();
    wczytaj(lista);
    while (true) {
      System.out.println("\nMINI BIBLIOTEKA");
      System.out.println("1 - Dodaj");
      System.out.println("2 - Pokaz");
      System.out.println("3 - Szukaj");
      System.out.println("4 - Wypozycz");
      System.out.println("5 - Zwroc");
      System.out.println("0 - Koniec");
      System.out.print("Wybor: ");
      String wybor = sc.nextLine().trim();
      switch (wybor) {
        case "1":
          System.out.print("Tytul: ");
          String t = sc.nextLine().trim();
          System.out.print("Autor: ");
          String a = sc.nextLine().trim();
          if (t.isEmpty() || a.isEmpty()) {
            System.out.println("Puste dane!");
          } else {
            lista.add(new Ksiazka(t, a));
            zapisz(lista);
            System.out.println("Dodano!");
          }
          break;
        case "2":
          pokaz(lista);
          break;
        case "3":
          System.out.print("Szukaj: ");
          String q = sc.nextLine().trim()
          .toLowerCase(Locale.ROOT);
          if (q.isEmpty()) {
            System.out.println("Pusta fraza!");
            break;
          }
          boolean znaleziono = false;
          for (Ksiazka k : lista) {
            if (k.tytul.toLowerCase(Locale.ROOT)
              .contains(q)
              || k.autor.toLowerCase(Locale.ROOT)
              .contains(q)) {
              System.out.println(
                k.tytul + " - " + k.autor);
              znaleziono = true;
            }
          }
          if (!znaleziono) {
            System.out.println("Nie znaleziono.");
          }
          break;
        case "4":
          zmien(lista, sc, true);
          break;
        case "5":
          zmien(lista, sc, false);
          break;
        case "0":
          System.out.println("Do zobaczenia!");
          return;
        default:
          System.out.println("Nieznana opcja!");
      }
    }
  }
}
```

# PRZETESTUJ MINI BIBLIOTEKĘ I UKOŃCZ SWÓJ DZIESIĄTY PROJEKT

## TESTY PROGRAMU · 01–04

### 01 — DODAWANIE KSIĄŻKI

Dodaj książkę: **Lalka**, autor: **Bolesław Prus**
**Wynik:** Książka pojawia się w katalogu ze statusem dostępna.

### 02 — WYSZUKIWANIE

Wpisz fragment: **lal**
**Wynik:** Program odnajduje książkę „Lalka”.

### 03 — WYPOŻYCZENIE

Wypożycz dostępną książkę.
**Wynik:** Status zmienia się na wypożyczona.

### 04 — PODWÓJNE WYPOŻYCZENIE

Spróbuj wypożyczyć tę samą książkę ponownie.
**Wynik:** Program informuje, że status nie może zostać ponownie zmieniony na wypożyczony.

### WYZWANIE A — LICZNIK DOSTĘPNYCH KSIĄŻEK

Policz obiekty, dla których:
`!k.wypozyczona`

### WYZWANIE B — NOWE POLE

Rozbuduj klasę `Ksiazka` o rok wydania:
`int rokWydania;`
Zaktualizuj konstruktor, dodawanie książek oraz zapis i odczyt informacji.

## TESTY PROGRAMU · 05–08

### 05 — ZWROT

Zwróć wypożyczoną książkę.
**Wynik:** Status zmienia się na dostępna.

### 06 — ZAPIS DANYCH

Dodaj książkę i zakończ program. Następnie uruchom go ponownie.
**Wynik:** Wcześniej zapisane książki powinny pojawić się w katalogu, jeśli plik został prawidłowo zapisany i odczytany.

### 07 — ZACHOWANIE STATUSU

Wypożycz książkę i zakończ program. Uruchom go ponownie.
**Wynik:** Status książki pozostaje wypożyczona.

### 08 — BRAK PLIKU

Przed pierwszym uruchomieniem plik biblioteki może nie istnieć.
**Wynik:** Program uruchamia się z pustym katalogiem i tworzy plik po pierwszej zmianie danych.

### WYZWANIE C — POTWIERDZENIE USUNIĘCIA

Przed usunięciem książki wyświetl pytanie:
`Czy na pewno usunac ksiazke? (T/N)`

### WYZWANIE D — KOPIA ZAPASOWA

Przed nadpisaniem pliku spróbuj utworzyć jego kopię. Zastanów się, dlaczego zapisywanie danych bez kopii zapasowej może być ryzykowne.

## TABELA KONTROLNA

| Test | Oczekiwany rezultat |
|---|---|
| Dodanie książki | Nowa pozycja w katalogu |
| Wyszukiwanie | Znalezienie pasującej książki |
| Wypożyczenie | Status: wypożyczona |
| Ponowne wypożyczenie | Brak zmiany |
| Zwrot | Status: dostępna |
| Ponowne uruchomienie | Odczyt zapisanych danych |
| Błędny numer | Komunikat błędu |

## WYZWANIE — USUWANIE KSIĄŻEK

Dodaj do menu nową opcję:

```
6 - Usun ksiazke
```

**KROK 1** · Wyświetl katalog z numerami.

**KROK 2** · Poproś użytkownika o numer książki.

**KROK 3** · Sprawdź poprawność numeru.

**KROK 4** · Usuń wskazany obiekt z listy.

```
lista.remove(numer - 1);
```

**KROK 5** · Zapisz zmieniony katalog do pliku:

```
zapisz(lista);
```

## CHECKLISTA UMIEJĘTNOŚCI

- ☐ Potrafię tworzyć klasy i obiekty.
- ☐ Umiem pracować z `ArrayList`.
- ☐ Potrafię wyszukiwać dane.
- ☐ Rozumiem zmienne logiczne.
- ☐ Umiem zmieniać stan obiektu.
- ☐ Potrafię tworzyć metody.
- ☐ Rozumiem obsługę wyjątków.
- ☐ Umiem zapisywać dane do pliku.
- ☐ Potrafię odczytywać dane.
- ☐ Rozumiem podstawy trwałego przechowywania informacji.
- ☐ Potrafię rozbudować aplikację konsolową.

**10 / 10 PROJEKTÓW**

## GRATULACJE! UKOŃCZYŁEŚ 10 PROJEKTÓW W JAVIE!

Od prostego kalkulatora po mini bibliotekę z zapisem danych — stworzyłeś serię praktycznych programów, które pomogły Ci poznać najważniejsze podstawy języka Java.

To dopiero początek Twojej przygody z programowaniem.

MODUŁ

11

# TESTOWANIE I PORTFOLIO

**Sprawdź swoje umiejętności, uporządkuj projekty i przygotuj się na kolejne wyzwania.**






**POSTĘP: 10 / 10**

| | | | | |
|---|---|---|---|---|
| 01 Kalkulator | 02 Zgadnij liczbę | 03 Konwerter | 04 Generator haseł | 05 Quiz |
| 06 Lista zadań | 07 Kontakty | 08 Budżet | 09 Nawyki | 10 Biblioteka |

## 10 PROJEKTÓW ZA TOBĄ!

Teraz czas sprawdzić, czego się nauczyłeś, uporządkować swój kod i przygotować go do dalszego rozwoju.

### TRZY CELE TEGO MODUŁU

| | | |
|---|---|---|
| 01 | **TESTOWANIE** | Nauczysz się sprawdzać, czy Twoje programy działają poprawnie. |
| 02 | **POPRAWIANIE KODU** | Dowiesz się, jak rozpoznawać typowe błędy i stopniowo ulepszać rozwiązania. |
| 03 | **PORTFOLIO** | Przygotujesz uporządkowaną kolekcję projektów, którą możesz rozwijać i prezentować. |

**Dobry programista nie tylko pisze kod. Potrafi także go zrozumieć, przetestować i ulepszyć.**

# JAK TESTOWAĆ WŁASNE PROGRAMY W JAVIE?

## DLACZEGO TESTOWANIE JEST WAŻNE?

Program może uruchamiać się bez błędów, ale nadal wykonywać niepoprawne obliczenia albo reagować niewłaściwie na dane użytkownika. Testowanie pomaga wykrywać takie problemy.




## KROK 1 — OKREŚL OCZEKIWANY WYNIK

Zanim uruchomisz program, zastanów się, jaki rezultat powinien zwrócić.

**PRZYKŁAD — KALKULATOR NAPIWKU**
Rachunek: **100 zł** · Napiwek: **10%**
Oczekiwany wynik: **110 zł do zapłaty.**

## KROK 2 — URUCHOM PROGRAM

W terminalu, w katalogu zawierającym plik `Main.java`, wpisz:

```
javac Main.java
java Main
```

## KROK 3 — PORÓWNAJ WYNIKI

Sprawdź, czy rezultat programu odpowiada wcześniej obliczonej wartości.

## KROK 5 — ZAPISZ WYNIKI

Dla każdego testu zanotuj: **wprowadzone dane**, oczekiwany rezultat, otrzymany rezultat oraz informację, czy test został zaliczony.

## TABELA TESTOWA

| Dane wejściowe | Oczekiwany rezultat |
|---|---|
| 100 zł, 10% | 110 zł |
| 50 zł, 0% | 50 zł |
| -20 zł, 10% | Komunikat błędu |
| abc, 10% | Komunikat błędu |

## WAŻNE POJĘCIE — DEBUGOWANIE

Debugowanie to proces wyszukiwania przyczyn błędów i ich usuwania. Możesz używać debuggera w edytorze albo tymczasowych instrukcji:

```
System.out.println("Wartosc: " + zmienna);
```

Takie komunikaty pomagają śledzić zmiany wartości podczas wykonywania programu.

## KROK 4 — PRZETESTUJ RÓŻNE DANE

Nie sprawdzaj tylko jednego przypadku. Przygotuj przynajmniej cztery rodzaje testów.

### TEST A — TYPOWE DANE

Dane, które użytkownik prawdopodobnie wpisze najczęściej.
**Przykład:** Rachunek: 100 zł.

### TEST B — WARTOŚCI GRANICZNE

Dane znajdujące się na granicy dozwolonego zakresu.
**Przykład:** Napiwek: 0%.

### TEST C — NIEPOPRAWNE DANE

Dane, których program nie powinien zaakceptować.
**Przykład:** Ujemna kwota rachunku.

### TEST D — NIEPRAWIDŁOWY TYP

Dane w formacie innym niż oczekiwany.
**Przykład:** Tekst zamiast liczby.

## KROK 6 — POPRAW I SPRAWDŹ PONOWNIE

Jeśli test wykryje problem:

1. Znajdź fragment kodu odpowiedzialny za wynik.
2. Sprawdź wartości zmiennych.
3. Popraw jedną rzecz.
4. Uruchom test ponownie.
5. Upewnij się, że pozostałe funkcje nadal działają.

## MINIWYZWANIE

Wybierz jeden z dziesięciu ukończonych projektów i przygotuj pięć testów, zanim wprowadzisz kolejną zmianę.

# 8 NAJCZĘSTSZYCH BŁĘDÓW POCZĄTKUJĄCYCH PROGRAMISTÓW

### BŁĄD 01 — BRAKUJĄCY ŚREDNIK

**NIEPOPRAWNIE:** `int liczba = 10`

**POPRAWNIE:** `int liczba = 10;`

**WYJAŚNIENIE:** Wiele instrukcji Javy musi kończyć się średnikiem.

### BŁĄD 02 — NIEZGODNA NAZWA KLASY I PLIKU

Jeśli klasa publiczna nazywa się:
`public class Main { }`

Plik powinien nazywać się: `Main.java`

**WYJAŚNIENIE:** Nazwa publicznej klasy najwyższego poziomu musi odpowiadać nazwie pliku źródłowego, z uwzględnieniem wielkich i małych liter.

### BŁĄD 03 — PORÓWNYWANIE NAPISÓW OPERATOREM ==

**NIEPOPRAWNIE — PORÓWNANIE TREŚCI:**
`tekst == "Java"`

**POPRAWNIE:** `"Java".equals(tekst)`

**WYJAŚNIENIE:** Do porównywania zawartości napisów używaj metody `equals()`. Operator == porównuje referencje.

### BŁĄD 04 — NIEPOPRAWNY INDEKS

**PROBLEM — PRZYKŁAD:**
```
int[] liczby = {10, 20, 30};
System.out.println(liczby[3]);
```

Tablica ma trzy elementy, ale ich indeksy to 0, 1 i 2. Odwołanie do indeksu 3 spowoduje wyjątek.

**POPRAWNIE:**
```
System.out.println(liczby[2]);
```

**WYNIK: 30**

### BŁĄD 05 — DZIELENIE PRZEZ ZERO

**NIEPOPRAWNIE:** `int wynik = 10 / 0;`

**PROBLEM:** Dzielenie całkowite przez zero powoduje `ArithmeticException`.

**ROZWIĄZANIE:** Sprawdź dzielnik przed obliczeniem.
```
if (dzielnik != 0) {
    int wynik = 10 / dzielnik;
}
```

### BŁĄD 06 — NIEKOŃCZĄCA SIĘ PĘTLA

**PROBLEM**
```
int i = 0;
while (i < 5) {
  System.out
    .println(i);
}
```

**POPRAWNIE**
```
int i = 0;
while (i < 5) {
  System.out
    .println(i);
  i++;
}
```

**PROBLEM:** W lewej wersji zmienna `i` nigdy się nie zmienia.

### BŁĄD 07 — TEKST ZAMIAST LICZBY

**PROBLEM — PRZYKŁAD:**
```
int liczba = Integer.parseInt("abc");
```

Program zgłasza `NumberFormatException`.

**ROZWIĄZANIE:** Obsłuż wyjątek za pomocą `try` i `catch` albo sprawdź dane przed konwersją.

### BŁĄD 08 — NIEZAINICJALIZOWANY OBIEKT

**PROBLEM — PRZYKŁAD:**
```
String tekst = null;
System.out.println(tekst.length());
```

Próba wywołania metody na referencji `null` powoduje `NullPointerException`.

**ROZWIĄZANIE:** Upewnij się, że zmienna odwołuje się do właściwego obiektu przed użyciem.

## TRZY RODZAJE BŁĘDÓW

**BŁĘDY KOMPILACJI**
Kod nie spełnia reguł języka i nie może zostać poprawnie skompilowany.

**BŁĘDY WYKONANIA**
Problem pojawia się podczas działania programu.

**BŁĘDY LOGICZNE**
Program działa, ale daje nieprawidłowy wynik.

**ZASADA:** Nie ignoruj komunikatów o błędach. Czytaj ich treść, sprawdzaj wskazany fragment i próbuj zrozumieć przyczynę problemu.

## MINIWYZWANIE — ZNAJDŹ I POPRAW BŁĘDY

**NIEPOPRAWNIE**
```
int liczba = 5
if (liczba = 5) {
    System.out.println("Pieć");
}
```

**ROZWIĄZANIE**
```
int liczba = 5;
if (liczba == 5) {
    System.out.println("Piec");
}
```

# ZBUDUJ PORTFOLIO SWOICH 10 PROJEKTÓW W JAVIE

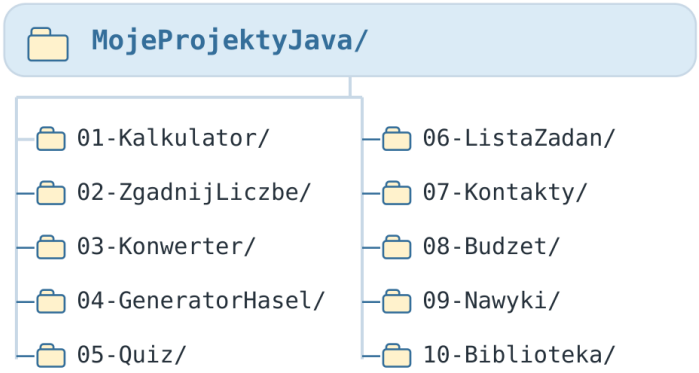
## CO TO JEST PORTFOLIO PROGRAMISTYCZNE?

Portfolio to uporządkowany zbiór projektów przedstawiających Twoje umiejętności i postępy. Nie muszą to być duże aplikacje. Ważne, aby projekty działały, miały jasny cel i zawierały informacje o sposobie uruchomienia.

## KROK 1 — UTWÓRZ GŁÓWNY FOLDER

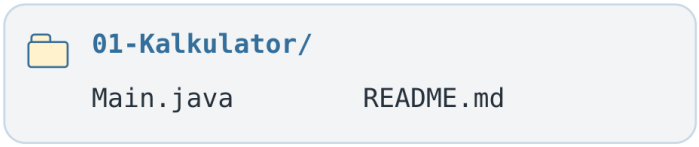
Nazwij go: `MojeProjektyJava`

## KROK 2 — UTWÓRZ 10 PODFOLDERÓW




## KROK 3 — UMIEŚĆ KOD W FOLDERZE

W każdym folderze umieść odpowiedni plik źródłowy. Przykład:




## KROK 6 — ZADBAJ O PORZĄDEK

Przechowuj kod źródłowy i dokumentację, a nie wszystkie pliki tymczasowe powstające podczas kompilacji.

`Main.java` — kod źródłowy.
`README.md` — dokumentacja.
`Main.class` — wynik kompilacji, który zwykle można odtworzyć.

## KROK 4 — DODAJ PLIK README

README to dokument opisujący projekt. Może zawierać:

Nazwę programu; krótki opis działania; wykorzystane elementy Javy; instrukcję uruchomienia; przykładowe dane wejściowe; przykładowy wynik; możliwe dalsze ulepszenia.

## PRZYKŁAD OPISU PROJEKTU

**NAZWA:** Kalkulator rachunku i napiwku
**OPIS:** Program oblicza napiwek oraz całkowitą kwotę rachunku.
**WYKORZYSTANE ELEMENTY:**
- Scanner.
- Zmienne double.
- Operatory matematyczne.
- Instrukcje warunkowe.

**URUCHOMIENIE** · W folderze danego projektu wpisz:

```
javac Main.java
java Main
```

**PRZYKŁADOWY WYNIK — SKRÓT**

```
Rachunek: 100
Napiwek: 10
Do zaplaty: 110.00 zl
```

## KROK 5 — SPRAWDŹ KAŻDY PROJEKT

Przed umieszczeniem programu w portfolio:

**1.** Skompiluj kod. **2.** Uruchom aplikację. **3.** Sprawdź podstawowe funkcje. **4.** Usuń niepotrzebne komunikaty testowe. **5.** Uzupełnij README.

**WAŻNE ZASADY** · Nie umieszczaj w publicznym portfolio prawdziwych haseł, prywatnych numerów telefonów, danych osobowych ani innych informacji poufnych. Jeśli projekt zapisuje dane w pliku, wykorzystuj przykładowe dane testowe. Nie udostępniaj lokalnych plików użytkowników bez ich zgody.

## DODATKOWY KROK — KONTROLA WERSJI

Kiedy opanujesz podstawy, możesz nauczyć się korzystać z Git. Git pomaga zapisywać historię zmian w projekcie. Platformy przechowujące repozytoria, takie jak GitHub, mogą służyć do prezentowania kodu i współpracy.

**MINIWYZWANIE** · Przygotuj plik README dla trzech ulubionych projektów. Opisz, czego nauczyłeś się podczas ich tworzenia i co chciałbyś w nich jeszcze poprawić.

# CO DALEJ? TWÓJ PLAN ROZWOJU W JAVIE

## UKOŃCZENIE 10 PROJEKTÓW TO DOPIERO POCZĄTEK

Masz już podstawy potrzebne do tworzenia prostych aplikacji konsolowych. Teraz warto utrwalić wiedzę, poprawić swoje projekty i poznać kolejne narzędzia.

TYDZIEŃ 1 → TYDZIEŃ 2 → TYDZIEŃ 3 → TYDZIEŃ 4

### TYDZIEŃ 1 — UTRWAL PODSTAWY

**CEL:** Samodzielnie rozumieć i modyfikować proste programy.

- Powtórz zmienne i typy danych.
- Przećwicz `if`, `else` i `switch`.
- Rozwiąż kilka zadań z pętlami.
- Utwórz własne metody.
- Zmodyfikuj kalkulator i konwerter.

**REZULTAT:** Potrafisz wyjaśnić działanie najważniejszych instrukcji w swoich pierwszych projektach.

### TYDZIEŃ 2 — PRACUJ Z DANYMI

**CEL:** Swobodniej korzystać z kolekcji i obiektów.

- Powtórz `ArrayList`.
- Przećwicz `HashSet`.
- Utwórz własną klasę.
- Dodaj nowe pola do obiektów.
- Rozbuduj książkę kontaktów.
- Dodaj nowe funkcje listy zadań.

**REZULTAT:** Potrafisz przechowywać i przetwarzać wiele powiązanych informacji.

### TYDZIEŃ 3 — DOSKONAL TESTOWANIE

**CEL:** Tworzyć bardziej niezawodne programy.

- Testuj poprawne dane.
- Sprawdzaj wartości graniczne.
- Obsługuj niepoprawne dane.
- Korzystaj z debuggera.
- Poznaj podstawy testów jednostkowych.
- Zapoznaj się z narzędziem JUnit.

**REZULTAT:** Potrafisz wykrywać błędy i systematycznie sprawdzać poprawność działania programu.

### TYDZIEŃ 4 — ROZWIJAJ PORTFOLIO

**CEL:** Przygotować czytelną kolekcję projektów.

- Uporządkuj foldery.
- Napisz README.
- Ulepsz trzy wybrane aplikacje.
- Poznaj podstawy Git.
- Zapisuj zmiany kolejnych wersji.
- Zaplanuj jeden nowy projekt.

**REZULTAT:** Masz uporządkowane portfolio i konkretny pomysł na dalszy rozwój.

## TRZY KIERUNKI DALSZEJ NAUKI

### 1 · PROGRAMOWANIE OBIEKTOWE

Poznaj dokładniej: enkapsulację; dziedziczenie; interfejsy; polimorfizm.

### 2 · DANE I PLIKI

Rozwijaj umiejętności związane z: zapisywaniem danych; odczytem plików; obsługą wyjątków; podstawami baz danych.

### 3 · WIĘKSZE APLIKACJE

W przyszłości możesz poznawać: graficzne interfejsy użytkownika; testy automatyczne; narzędzia do budowania projektów; podstawy aplikacji internetowych.

### ZŁOTA ZASADA NAUKI

Lepiej regularnie tworzyć i poprawiać małe programy niż tylko czytać o zaawansowanych technologiach.

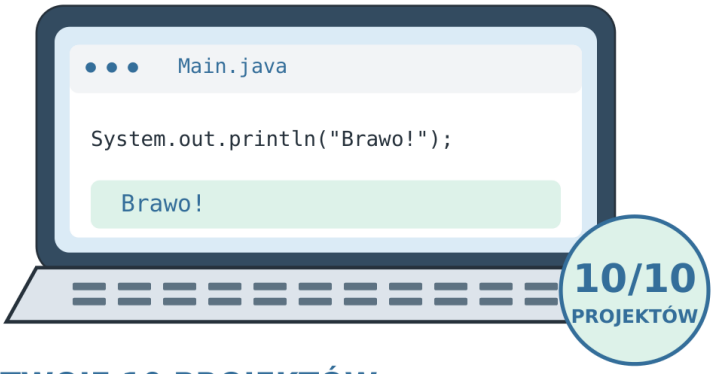
## TWOJE NOWE WYZWANIE — LISTA ZADAŃ 2.0

Wybierz jeden z ukończonych projektów. Zaplanuj trzy ulepszenia. Następnie zaimplementuj je samodzielnie, po jednym na raz. **Przykład:** Dodaj terminy realizacji. Wprowadź priorytety. Zapisuj zadania w pliku.

**Ważne:** Plan czterech tygodni jest propozycją. Dostosuj tempo do własnych możliwości i wracaj do trudniejszych tematów, gdy potrzebujesz dodatkowej praktyki.
**Dalsza nauka:** JUnit i Git to opcjonalne narzędzia. Dziesięć projektów z tego materiału korzysta wyłącznie ze standardowej biblioteki Javy.

# GRATULACJE! TWOJA PRZYGODA Z JAVĄ TRWA DALEJ!

**Od pierwszej linijki kodu do dziesięciu praktycznych projektów.**




## UDAŁO SIĘ!

Przeszedłeś przez dziesięć praktycznych projektów w Javie. Na początku tworzyłeś proste programy wykonujące obliczenia. Później nauczyłeś się korzystać z warunków, pętli, metod, tablic, kolekcji i obiektów. W ostatnich projektach poznałeś pracę z datami oraz podstawy zapisywania danych w plikach. To solidny początek dalszej nauki programowania.

## TWOJE 10 PROJEKTÓW

### 01 — KALKULATOR RACHUNKU I NAPIWKU
Zmienne, dane wejściowe i obliczenia.

### 02 — GRA „ZGADNIJ LICZBĘ”
Losowanie, warunki i pętle.

### 03 — KONWERTER JEDNOSTEK
Menu, metody i wzory matematyczne.

### 04 — GENERATOR HASEŁ
Napisy, losowanie znaków i SecureRandom.

### 05 — QUIZ WIEDZY
Tablice, pytania i punktacja.

### 06 — LISTA ZADAŃ
ArrayList i zarządzanie elementami.

### 07 — KSIĄŻKA KONTAKTÓW
Klasy, obiekty i wyszukiwanie.

### 08 — MENEDŻER BUDŻETU
Transakcje, metody i podsumowania.

### 09 — DZIENNIK NAWYKÓW
Daty, zbiory i analiza regularności.

### 10 — MINI BIBLIOTEKA
Katalog książek, stany obiektów i zapis danych.

## CO JUŻ WIESZ? POTRAFISZ:

- [ ] Tworzyć i uruchamiać programy Java.
- [ ] Korzystać ze zmiennych i typów danych.
- [ ] Pisać instrukcje warunkowe.
- [ ] Tworzyć pętle.
- [ ] Definiować metody.
- [ ] Pracować z napisami.
- [ ] Przechowywać dane w kolekcjach.
- [ ] Tworzyć klasy i obiekty.
- [ ] Korzystać z dat.
- [ ] Zapisywać i odczytywać proste pliki.
- [ ] Testować i poprawiać kod.

**NAJWAŻNIEJSZA LEKCJA.** Nie musisz znać odpowiedzi na każde pytanie. Ważne, aby umieć analizować problem, dzielić go na mniejsze części, próbować różnych rozwiązań i uczyć się na błędach.

**PAMIĘTAJ.** Każdy większy program powstaje z mniejszych elementów. Każdy błąd może być okazją do lepszego zrozumienia kodu. Każdy ukończony projekt rozwija Twoje umiejętności.

## OSTATNIE WYZWANIE

Wymyśl swój jedenasty projekt. Zapisz:

**1.** Jaki problem rozwiązuje?
**2.** Jakie dane będą potrzebne?
**3.** Jakie funkcje powinien zawierać?
**4.** Jakie poznane elementy Javy wykorzystasz?
**5.** Jak sprawdzisz, czy działa poprawnie?

Następnie zacznij tworzyć pierwszą wersję.

**10 PROJEKTÓW. NOWE UMIEJĘTNOŚCI. KOLEJNE MOŻLIWOŚCI.**

**NIE PRZESTAWAJ PROGRAMOWAĆ!**