JAVA
OD PODSTAW

Tom 1 – od pierwszego programu do pisania metod

Instalacja Java
Pierwszy program
Komentarze i formatowanie kodu
Zmienne, stałe, typy prymitywne
Typ String
Operatory i rzutowanie
Instrukcje warunkowe
Pętle
Tablice
Metody

Przemysław Kruglej

Wydanie własne, pierwsze (04-2024)

Numer ISBN „Java od podstaw – Tom 1”: 978-83-968045-3-2
Numer ISBN kolekcji „Java od podstaw”: 978-83-968045-2-5

Kontakt z autorem: przemyslaw.kruglej@gmail.com

Strona Internetowa tej książki: https://kursjava.com/ksiazki/java-od-podstaw#tom-1

Kody źródłowe oraz rozwiązania zadań dostępne są na poniższej stronie w Internecie:

https://github.com/przemyslaw-kruglej/java-od-podstaw-przyklady

Errata dostępna jest w sekcji „Tom 1 – Wydanie pierwsze – 04-2024” na stronie:

https://github.com/przemyslaw-kruglej/java-od-podstaw-errata

Zagnieżdżanie instrukcji warunkowych

Zagnieżdżone instrukcje warunkowe mogą także zawierać w sobie zagnieżdżone instrukcje warunkowe itd., jednak zbyt wiele zagnieżdżonych instrukcji warunkowych sprawia, że kod jest trudny do zrozumienia. Dobrą praktyką jest zapisywanie kodu w taki sposób, aby ograniczyć liczbę zagnieżdżonych w sobie bloków kodu, takich jak instrukcje warunkowe. Często fragmenty kodu umieszcza się w takich sytuacjach w osobnych metodach. Nadanie takiej metodzie dobrej nazwy pomaga w zrozumieniu działania kodu. Metody są tematem ostatniego rozdziału.

7.6. Instrukcja i wyrażenie switch

Najpierw zaznajomimy się z instrukcją `switch`, która jest w języku Java od początku jego istnienia, a następnie przedstawię Ci wyrażenie `switch`, które zostało dodane do Javy relatywnie niedawno.

7.6.1. Instrukcja switch

Poza instrukcją `if`, mamy do dyspozycji jeszcze inny rodzaj instrukcji warunkowej – jest to instrukcja `switch`. Jej składnia jest następująca:

```
switch (wyrażenie) {
  case staleWyrazenie:
    instrukcja;
    instrukcja;
    break;
  case staleWyrazenie2:
    instrukcja2;
    break;
  default:
    instrukcja3;
}
```

Instrukcja `switch` ma za zadanie porównać wartość wyrażenia do podanych stałych wartości w sekcjach `case`. Jeżeli wartość wyrażenia będzie pasować do którejś z porównywanych wartości, to wykonana zostanie skojarzona z nią instrukcja (bądź instrukcje). Zwróć uwagę, że instrukcje powiązane z daną sekcją `case` nie są objęte w nawiasy klamrowe `{ }`.

Jeżeli natomiast żadna z wartości nie będzie pasować, wykonane zostaną instrukcje z bloku `default`, który nie jest konieczny w instrukcji `switch` – można go pominąć. O znaczeniu słowa `break` opowiem Ci w podrozdziale „7.6.1.1 Użycie break w instrukcji switch”.

Java 14 i usprawnienia instrukcji switch

W wersji 14 języka Java instrukcja `switch` została uproszczona. Jednakże, wiele firm korzysta z poprzednich wersji Javy, więc ta nowa wersja instrukcji `switch` może nie zawsze być dostępna. Dlatego najpierw przedstawię podstawową wersję `switch` dostępną w Javie od początku jej istnienia, a potem zaprezentuję Ci jej nową, uproszczoną wersję.

Spójrzmy na przykład użycia instrukcji `switch`:

Rozdzial_07_Instrukcje_warunkowe/Switch/InstrukcjaSwitchDzienTygodnia.java

```
import java.util.Scanner;

public class InstrukcjaSwitchDzienTygodnia {
  public static void main(String[] args) {
    System.out.print("Podaj dzień tygodnia: ");
    int dzienTygodnia = pobierzLiczbe();

    switch (dzienTygodnia) {
      case 1:
        System.out.println("Poniedziałek.");
        break;
      case 2:
        System.out.println("Wtorek.");
        break;
      case 3:
        System.out.println("Środa.");
        break;
      case 4:
        System.out.println("Czwartek.");
        break;
      case 5:
        System.out.println("Piątek.");
        break;
      case 6:
        System.out.println("Sobota.");
        break;
      case 7:
        System.out.println("Niedziela.");
        break;
      default:
        System.out.println(
            "Nieznany dzień tygodnia: " + dzienTygodnia
        );
    }
  }
```

```
  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

W tym przykładzie pobieramy od użytkownika numer dnia tygodnia i wypisujemy jego nazwę przy użyciu instrukcji `switch`. Porównujemy wartość zmiennej `dzienTygodnia` do liczb w kolejnych sekcjach `case`. Jeżeli żadna z wyszczególnionych liczb nie będzie zgadzała się z wartością zmiennej `dzienTygodnia`, wypisany zostanie komunikat z sekcji `default`.

Spójrzmy na dwa przykładowe wykonania tego programu:

```
Podaj dzień tygodnia: 2
Wtorek.
```

```
Podaj dzień tygodnia: -5
Nieznany dzień tygodnia: -5
```

Do niedawna instrukcji `switch` (a także wyrażenia `switch`, które poznasz niebawem) można było używać tylko z kilkoma typami:

- `byte` i `Byte`
- `short` i `Short`
- `char` i `Character`
- `int` i `Integer`
- `String`
- `enum`

Typy `byte`, `short`, `char`, `int`, oraz `String` już znasz. Typy `Byte`, `Short`, `Character`, oraz `Integer` to typy złożone odpowiadające typom prymitywnym – ich wstępny opis przedstawiłem w rozdziale „5.4 Typy złożone odwzorowujące typy prymitywne”. Typ wyliczeniowy `enum` będzie tematem jednego z rozdziałów w drugim tomie tej książki. Zwróć uwagę, że na tej liście nie ma typów `double`, `float`, oraz `long` – nie są one wspierane przez instrukcję `switch`.

W poprzednim podrozdziale w ramach prezentacji zagnieżdżonych instrukcji `if` przedstawiłem przykład prostego kalkulatora, który wykonywał jedną z czterech operacji na wczytanych od użytkownika liczbach. Zobaczmy, jak wygląda ten program, gdy użyjemy w nim instrukcji `switch`:

Rozdzial_07_Instrukcje_warunkowe/Switch/WykonajDzialanieSwitch.java

```
import java.util.Scanner;

public class WykonajDzialanieSwitch {
  public static void main(String[] args) {
    System.out.print("Podaj pierwszą liczbę: ");
    int liczba1 = pobierzLiczbe();

    System.out.print("Podaj drugą liczbę: ");
    int liczba2 = pobierzLiczbe();

    System.out.print(
        "Podaj operację do wykonania (+ - * /): "
    );
    String operacja = pobierzSlowo();

    int wynik = 0;
    boolean nieprawidlowaOperacja = false;

    switch (operacja) {
      case "+":
        wynik = liczba1 + liczba2;
        break;
      case "-":
        wynik = liczba1 + liczba2;
        break;
      case "*":
        wynik = liczba1 + liczba2;
        break;
      case "/":
        if (liczba2 != 0) {
          wynik = liczba1 / liczba2;
        } else {
          nieprawidlowaOperacja = true;
          System.out.println(
              "Dzielnik nie może być zerem!"
          );
        }
        break;
      default:
        nieprawidlowaOperacja = true;
        System.out.println(
          "Nieprawidłowa operacja: " + operacja
        );
    }
```

```
    if (!nieprawidlowaOperacja) {
      System.out.printf(
          "%d %s %d = %d", liczba1, operacja, liczba2, wynik
      );
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

Ta wersja programu jest mniej czytelna, ponieważ jest dłuższa i zawiera wiele instrukcji `break`, które tak naprawdę nic do programu nie wnoszą, powodując jedynie, że jest on trudniejszy w zrozumieniu.

Instrukcja `switch` jest raczej rzadko stosowana, ale warto ją znać, ponieważ czasem się przydaje. W szczególności nowa wersja `switch` – wyrażenie `switch` – jest bardziej przydatna. Zaraz Ci o niej opowiem, ale najpierw omówię jeszcze kilka zagadnień związanych z instrukcją `switch`.

Null w switch, dopasowanie wzorców i nowa klauzula when

Od wersji Java 21 instrukcję `switch` możemy stosować także z dowolnymi typami złożonymi, jeżeli sekcje `case` będą korzystać z nowej składni dopasowania do wzorca (ang. *pattern matching*). Planowałem to zagadnienie omówić na końcu podrozdziału o instrukcji `switch`, jednakże warto podejść do niego dopiero po zaznajomieniu się z wartością `null`, którą dokładnie omówię w rozdziale o tablicach. Ponadto, *pattern matching* jest bardziej zaawansowaną funkcjonalnością i wrócę do niej w drugim tomie tej książki. „Dodatek G” na końcu książki przeznaczyłem, by pokrótce opisać zmiany do instrukcji i wyrażenia `switch`, które zostały wprowadzane w Javie 21, które mogą okazać się przydatne już na tym etapie nauki.

7.6.1.1. Użycie break w instrukcji switch

Zadaniem słowa kluczowego `break` jest przerwanie wykonania kolejnych, następujących po nim operacji.

Spójrzmy ponownie na instrukcję `switch` z przykładu wypisującego nazwę dnia tygodnia – gdy usuniemy z niej instrukcje `break`, to będzie ona wyglądać następująco:

Rozdzial_07_Instrukcje_warunkowe/Switch/InstrukcjaSwitchDzienTygodniaBezBreak.java

```
import java.util.Scanner;

public class InstrukcjaSwitchDzienTygodniaBezBreak {
  public static void main(String[] args) {
    System.out.print("Podaj dzień tygodnia: ");
    int dzienTygodnia = pobierzLiczbe();

    switch (dzienTygodnia) {
      case 1: System.out.println("Poniedziałek.");
      case 2: System.out.println("Wtorek.");
      case 3: System.out.println("Środa.");
      case 4: System.out.println("Czwartek.");
      case 5: System.out.println("Piątek.");
      case 6: System.out.println("Sobota.");
      case 7: System.out.println("Niedziela.");
      default:
        System.out.println(
            "Nieznany dzień tygodnia: " + dzienTygodnia
        );
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Zobaczmy jaki teraz będzie efekt działania tego programu, gdy podamy dzień o numerze np. `3`:

```
Podaj dzień tygodnia: 3
Środa.
Czwartek.
Piątek.
Sobota.
Niedziela.
Nieznany dzień tygodnia: 3
```

Brak słowa kluczowego `break` powoduje, że po dopasowaniu wartości zmiennej `dzienTygodnia` do wartości z jednej z sekcji `case`, wykonywane są nie tylko powiązane z nią instrukcje, ale także instrukcje wszystkich sekcji `case` po niej następujące.

Użycie `break` nie jest wymagane, ale zazwyczaj jego brak świadczy o tym, że programista zapomniał go po prostu dopisać. Czasami opisane działanie instrukcji `switch` jest przydatne, ale niekiedy powoduje też trudne do wychwycenia błędy.

Spójrz na jeszcze jeden przykład uruchomienia tego programu:

```
Podaj dzień tygodnia: -5
Nieznany dzień tygodnia: -5
```

Tym razem wypisany został tylko ostatni komunikat, ponieważ po sekcji `default` nie ma już więcej innych sekcji instrukcji `switch`.

7.6.1.2. Kilka sekcji case skojarzonych z tymi samymi instrukcjami

Czasem możemy chcieć wykonać takie same instrukcje po dopasowaniu do kilku różnych wartości – możemy to osiągnąć zapisując kilka sekcji `case` jedna po drugiej:

Rozdzial_07_Instrukcje_warunkowe/Switch/KilkaCaseDoJednejInstrukcji.java

```
import java.util.Scanner;

public class KilkaCaseDoJednejInstrukcji {
  public static void main(String[] args) {
    System.out.print("Podaj dzień tygodnia: ");
    int dzienTygodnia = pobierzLiczbe();

    switch (dzienTygodnia) {
      case 1: case 2: case 3: case 4: case 5:
        System.out.println("Dzień roboczy.");
        break;
      case 6: case 7:
        System.out.println("Weekend.");
        break;
      default:
        System.out.println("Nieznany dzień tygodnia.");
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

W tym przykładzie wypiszemy na ekran:

- `"Dzień roboczy"`, gdy użytkownik poda numer od `1` do `5`,
- `"Weekend"`, gdy użytkownik poda numer `6` lub `7`,
- `"Nieznany dzień tygodnia"`, gdy użytkownik poda inną liczbę.

Sekcje `case` w tym przykładzie moglibyśmy zapisać także w następujący sposób:

```
switch (dzienTygodnia) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
    System.out.println("Dzień roboczy.");
    break;
  case 6:
  case 7:
    System.out.println("Weekend.");
    break;
  default:
    System.out.println("Nieznany dzień tygodnia.");
}
```

Chociaż taki zapis jest poprawny, to z mojego punktu widzenia wydaje się nadmiarowo długi. Jeżeli takie formatowanie jest dla Ciebie bardziej czytelne, to możesz je oczywiście stosować.

Takie zachowanie instrukcji `switch` związane jest z omówionym słowem kluczowym `break` – skoro w danej sekcji `case` nie ma `break`, to zostanie wykonana następna sekcja. W tym przykładzie sekcje `case 1`, `case 2`, `case 3`, oraz `case 4` nie mają ani powiązanych z nimi instrukcji do wykonania, ani nie zostało w nich użyte słowo kluczowe `break`, więc wykonywane są sekcje znajdujące się „pod” nimi. Efekt jest taki, że dla wartości od `1` do `5` wykonywane są instrukcje związane z sekcją `case 5`, która ma przyporządkowane dwie instrukcje: wypisanie na ekran komunikatu `"Dzień roboczy."` oraz `break`. Analogicznie dzieje się z sekcjami `case 6` i `case 7`.

7.6.1.3. Sekcja default

Sekcja `default` nie jest wymagana w instrukcji `switch`, a jeżeli jest użyta, to niekoniecznie musi być na końcu – przeciwnie do instrukcji warunkowej `if`, w której sekcja `else` zawsze musi być na końcu. Wróćmy do poprzedniego przykładu `switch`:

```
switch (dzienTygodnia) {
  case 1: case 2: case 3: case 4: case 5:
    System.out.println("Dzień roboczy.");
    break;
  case 6: case 7:
    System.out.println("Weekend.");
    break;
  default:
    System.out.println("Nieznany dzień tygodnia.");
}
```

Tę instrukcję `switch` moglibyśmy także zapisać w następujący sposób:

```
switch (dzienTygodnia) {
  default:
    System.out.println("Nieznany dzień tygodnia.");
    break;
  case 1: case 2: case 3:
  case 4: case 5:
    System.out.println("Dzień roboczy.");
    break;
  case 6: case 7:
    System.out.println("Weekend.");
    break;
}
```

Ten fragment kodu jest poprawny i działa tak, jak jego poprzednia wersja, jednak zauważ, że musiałem dodać instrukcję `break` do sekcji `default` – w przeciwnym razie, gdyby użytkownik podał nieprawidłowy numer dnia tygodnia, wykonana zostałaby instrukcja związana nie tylko z sekcją `default`, ale także sekcją `case 5`. Sekcja `default` „nie chroni” nas przed kaskadowym wykonaniem następujących po niej sekcji instrukcji `switch` – musimy do niej dodać instrukcję `break`.

Pomimo, że sekcja `default` może być w dowolnym miejscu instrukcji `switch`, powinniśmy umieszczać ją na końcu, ponieważ w ten sposób łatwiej analizuje się kod instrukcji `switch`, a ponadto nie musimy stosować słowa kluczowego `break` w sekcji `default`, gdy ta sekcja jest na końcu (ponieważ nie następują po niej żadne inne sekcje instrukcji `switch`).

7.6.1.4. Wartości umieszczane w sekcjach case

Wspomniałem na początku tego podrozdziału, że wartości umieszczone w sekcjach `case` muszą być stałymi – inaczej kod się nie skompiluje. Dla przykładu, jeżeli wczytamy od użytkownika liczbę, to nie możemy jej umieścić w sekcji `case`:

```
int x = pobierzLiczbe();

switch (pewneWyrazenie) {
  case x: // błąd kompilacji - wymagana stała wartość
    instrukcja1;
    break;
}
```

Kompilator zaprotestuje, ponieważ zmienna `x` nie ma stałej wartości – jej wartość zależy od tego, co poda nam użytkownik.

Ponadto, wartości umieszczane w sekcjach `case` muszą być unikalne w ramach danej instrukcji `switch`:

```
switch (dzienTygodnia) {
  case 1: case 2: case 3:
  case 4: case 5:
    System.out.println("Dzień roboczy.");
    break;
  case 6: case 7:
    System.out.println("Weekend.");
    break;
  case 5: // błąd - powtórzona wartość
    System.out.println("Prawie weekend!");
    break;
  default:
    System.out.println("Nieznany dzień tygodnia.");
}
```

Wartość `5` została użyta w dwóch różnych sekcjach `case` – kompilator zaprotestuje zgłaszając błąd: `duplicate case label`.

7.6.2. [Java 14] Uproszczona instrukcja switch

Podstawowa wersja instrukcji `switch` nie jest zbyt czytelna, między innymi przez wymaganie używania instrukcji `break` w każdej sekcji `case`:

```
int dzienTygodnia = pobierzLiczbe();
String nazwaDnia;

switch (dzienTygodnia) {
  case 1:
    nazwaDnia = "Poniedziałek";
    break;
  case 2:
    nazwaDnia = "Wtorek";
    break;
  case 3:
    nazwaDnia = "Środa";
    break;
  case 4:
    nazwaDnia = "Czwartek";
    break;
  case 5:
    nazwaDnia = "Piątek";
    break;
```

```
  case 6:
    nazwaDnia = "Sobota";
    break;
  case 7:
    nazwaDnia = "Niedziela";
    break;
  default:
    nazwaDnia = "Nieprawidłowy numer dnia.";
}

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

Celem przedstawionej instrukcji `switch` jest przypisanie wartości do zmiennej `nazwaDnia` na podstawie wartości zmiennej `dzienTygodnia`. Na końcu każdej sekcji `case` musiałem dodać instrukcję `break` – gdyby jej zabrakło, to po dopasowaniu zmiennej `dzienTygodnia` do jednej z wartości wykonane zostałyby także instrukcje z kolejnych sekcji `case`.

Twórcy języka Java postanowili odświeżyć składnię instrukcji warunkowej `switch`. Od 14 wersji Javy możemy zapisać przedstawiony przykład w krótszy sposób:

Rozdzial_07_Instrukcje_warunkowe/Switch/UproszczonaInstrukcjaSwitch.java

```
import java.util.Scanner;

public class UproszczonaInstrukcjaSwitch {
  public static void main(String[] args) {
    System.out.print("Podaj dzień tygodnia: ");
    int dzienTygodnia = pobierzLiczbe();

    String nazwaDnia;

    switch (dzienTygodnia) {
      case 1 -> nazwaDnia = "Poniedziałek";
      case 2 -> nazwaDnia = "Wtorek";
      case 3 -> nazwaDnia = "Środa";
      case 4 -> nazwaDnia = "Czwartek";
      case 5 -> nazwaDnia = "Piątek";
      case 6 -> nazwaDnia = "Sobota";
      case 7 -> nazwaDnia = "Niedziela";
      default -> nazwaDnia = "Nieprawidłowy numer dnia.";
    }

    System.out.println(nazwaDnia);
  }
```

```
  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Ta wersja instrukcji `switch` korzysta z nowej składni do wskazywania instrukcji związanej z daną sekcją `case`:

```
case wartość ->
```

Ponadto, nie musimy już sami wstawiać do sekcji `case` instrukcji `break` – zostanie ona tam dodana automatycznie dla naszej wygody. Ta wersja instrukcji `switch` jest krótsza i zdecydowanie bardziej klarowna.

Przedstawiony, nowy sposób kojarzenia sekcji `case` i instrukcji do wykonania pozwala na umieszczenie w sekcji `case` tylko jednej instrukcji:

```
switch (dzienTygodnia) {
  case 1 -> nazwaDnia = "Poniedziałek";
  case 2 -> nazwaDnia = "Wtorek";
  case 3 -> nazwaDnia = "Środa";
  case 4 -> nazwaDnia = "Czwartek";
  case 5 -> // błąd kompilacji
      nazwaDnia = "Piątek";
      System.out.println("Jutro weekend!");
  case 6 -> nazwaDnia = "Sobota";
  case 7 -> nazwaDnia = "Niedziela";
  default -> nazwaDnia = "Nieprawidłowy numer dnia.";
}
```

Do sekcji `case 5` spróbowałem dodać jeszcze jedną instrukcję – kompilator zgłosi błąd kompilacji w takim przypadku. Możemy jednak ten problem rozwiązać, jeżeli otoczymy instrukcje w blok kodu za pomocą nawiasów klamrowych:

```
switch (dzienTygodnia) {
  case 1 -> nazwaDnia = "Poniedziałek";
  case 2 -> nazwaDnia = "Wtorek";
  case 3 -> nazwaDnia = "Środa";
  case 4 -> nazwaDnia = "Czwartek";
  case 5 -> {
    nazwaDnia = "Piątek";
    System.out.println("Jutro weekend!");
  }
  case 6 -> nazwaDnia = "Sobota";
  case 7 -> nazwaDnia = "Niedziela";
  default -> nazwaDnia = "Nieprawidłowy numer dnia.";
}
```

Dla sekcji `case 5` otoczyliśmy jej instrukcje w blok kodu:

```
case 5 -> {
  nazwaDnia = "Piątek";
  System.out.println("Jutro weekend!");
}
```

Tym razem kompilator nie będzie protestował i kod skompiluje się bez problemów.

Do instrukcji `switch` dodane zostało jeszcze jedno usprawnienie – spójrzmy na jeden z poprzednich przykładów:

```
switch (dzienTygodnia) {
  case 1: case 2: case 3: case 4: case 5:
    System.out.println("Dzień roboczy.");
    break;
  case 6: case 7:
    System.out.println("Weekend.");
    break;
  default:
    System.out.println("Nieznany dzień tygodnia.");
    break;
}
```

Jeżeli mamy kilka sekcji `case` skojarzonych z tymi samymi instrukcjami do wykonania, to możemy użyć nowego, skrótowego zapisu:

Rozdzial_07_Instrukcje_warunkowe/Switch/UproszczonaInstrukcjaSwitchCaseRazem.java

```
import java.util.Scanner;

public class UproszczonaInstrukcjaSwitchCaseRazem {
  public static void main(String[] args) {
    System.out.print("Podaj dzień tygodnia: ");
    int dzienTygodnia = pobierzLiczbe();

    switch (dzienTygodnia) {
      case 1, 2, 3, 4, 5 ->
          System.out.println("Dzień roboczy.");
      case 6, 7 -> System.out.println("Weekend.");
      default ->
          System.out.println("Nieznany dzień tygodnia.");
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Zamiast powtarzać słowo kluczowe `case` z kolejnymi wartościami, wylistowaliśmy je po przecinku:

```
case 1, 2, 3, 4, 5 -> System.out.println("Dzień roboczy.");
case 6, 7 -> System.out.println("Weekend.");
```

Jeszcze jedna uwaga – jeżeli korzystamy z nowej składni instrukcji `switch`, to nie możemy jej użycia mieszać ze starą składnią:

```
switch (dzienTygodnia) {
  case 1, 2, 3, 4, 5 ->
      System.out.println("Dzień roboczy.");
  // błąd - albo używamy case wartość: albo case wartość ->
  case 6:
  case 7: System.out.println("Weekend.");
  default -> System.out.println("Nieznany dzień tygodnia.");
}
```

Ten fragment kodu spowodowałby błąd kompilacji, ponieważ albo powinniśmy stosować starą składnię:

```
case wartość:
```

albo nową składnię:

```
case wartość ->
```

7.6.3. [Java 14] Wyrażenie switch

W 14 wersji poza uproszczeniem składni instrukcji `switch` do Javy zostało dodane *wyrażenie* `switch`. Oznacza to, że możemy teraz korzystać ze `switch` jako sposobu na wyznaczenie pewnej wartości i bezpośrednio przypisać ją np. do zmiennej.

Jednym z przykładów uproszczonej wersji instrukcji `switch` była zamiana numeru dnia tygodnia na jego nazwę:

```
String nazwaDnia;

switch (dzienTygodnia) {
  case 1 -> nazwaDnia = "Poniedziałek";
  case 2 -> nazwaDnia = "Wtorek";
  case 3 -> nazwaDnia = "Środa";
  case 4 -> nazwaDnia = "Czwartek";
  case 5 -> nazwaDnia = "Piątek";
  case 6 -> nazwaDnia = "Sobota";
  case 7 -> nazwaDnia = "Niedziela";
  default -> nazwaDnia = "Nieprawidłowy numer dnia.";
}
```

Od 14 wersji języka Java możemy tę instrukcję `switch` zapisać jako wyrażenie:

fragment pliku Rozdzial_07_Instrukcje_warunkowe/Switch/WyrazenieSwitchDzienTygodnia.java

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1 -> "Poniedziałek";
  case 2 -> "Wtorek";
  case 3 -> "Środa";
  case 4 -> "Czwartek";
  case 5 -> "Piątek";
  case 6 -> "Sobota";
  case 7 -> "Niedziela";
  default -> "Nieprawidłowy numer dnia.";
};
```

Zauważ, że wynik wyrażenia `switch` przypisujemy bezpośrednio do zmiennej `nazwaDnia`. W sekcjach `case` nie przypisujemy już wartości do zmiennej `nazwaDnia`, lecz wskazujemy wartość, która ma zostać zwrócona z danej sekcji. Ponadto, na końcu klamry `}` zamykającej wyrażenie `switch` znajduje się średnik – jest on wymagany w tym przypadku, ponieważ oznacza on koniec instrukcji jako całości.

Z wyrażeniami `switch` związany jest jeszcze jeden nowy mechanizm: instrukcja `yield`. Służy ona do zwracania wartości z bloku `case` wyrażenia `switch` (nie instrukcji `switch`) w jednym z dwóch przypadków:

- gdy korzystamy ze starej składni `case wartość:` zamiast `case wartość ->`
- gdy chcemy wykonać więcej niż jedną instrukcję korzystając ze składni `case wartość ->`

Pierwszy z powyższych przypadków obrazuje poniższy fragment kodu:

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1: yield "Poniedziałek";
  case 2: yield "Wtorek";
  case 3: yield "Środa";
  case 4: yield "Czwartek";
  case 5: yield "Piątek";
  case 6: yield "Sobota";
  case 7: yield "Niedziela";
  default: yield "Nieprawidłowy numer dnia.";
};
```

Zauważ, że zamiast nowej składni:

```
case wartość ->
```

użyłem starej składni:

```
case wartość:
```

W takim przypadku musiałem skorzystać z instrukcji `yield`, aby wskazać, jaka wartość ma być zwrócona z konkretnej sekcji `case`.

Przykład drugiego z opisanych przypadków gdy należy stosować instrukcję `yield`:

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1 -> "Poniedziałek";
  case 2 -> "Wtorek";
  case 3 -> "Środa";
  case 4 -> "Czwartek";
  case 5 -> "Piątek";
  case 6 -> "Sobota";
  case 7 -> "Niedziela";
  default -> {
    System.out.println("Podano nieprawidłowy numer dnia.");
    yield "?";
  }
};
```

Ponieważ w sekcji `default` przedstawionego wyrażenia `switch` chcemy wykonać więcej niż jedną instrukcję, zostały one „opakowane” w nawiasy klamrowe, a wskazanie na wartość, jaka ma być zwrócona w tym przypadku, zostało wykonane za pomocą instrukcji `yield`.

Ostatnia istotna informacja na temat nowych wyrażeń `switch`: wyrażenie `switch` *musi* zwracać wartość bądź kończyć się *rzuceniem wyjątku*[26]. Nie może być sytuacji, w której pewna wartość bądź wartości nie są brane pod uwagę w wyrażeniu `switch`, co mogłoby spowodować, że wyrażenie jako całość nie miałoby wartości – spójrz na poniższy, nieprawidłowy przykład:

fragment pliku Rozdzial_07_Instrukcje_warunkowe/Switch/WyrazenieSwitchDzienTygodniaBezDefault.java

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1 -> "Poniedziałek";
  case 2 -> "Wtorek";
  case 3 -> "Środa";
  case 4 -> "Czwartek";
  case 5 -> "Piątek";
  case 6 -> "Sobota";
  case 7 -> "Niedziela";
}; // błąd kompilacji
```

W tym przykładzie bierzemy pod uwagę jedynie siedem możliwych wartości zmiennej `dzienTygodnia`. Próba kompilacji tego fragmentu kodu zakończy się błędem: „*the*

[26] Wyjątki to specjalny mechanizm służący do wskazywania, że w kodzie programu wystąpił błąd. Wyjątki poznasz w drugim tomie tej książki.

switch expression does not cover all possible input values”. To wyrażenie `switch` nie ma sekcji `default` – istnieje więc szansa, że wyrażenie `switch` nie zwróciłoby żadnej wartości, np. gdy zmienna `dzienTygodnia` miałaby wartość `10`. Kompilator wykrywa tę sytuacją i zgłasza błąd.

Wyrażenia `switch` są wygodne w użyciu ze względu na zwięzłą składnię – podobnie jak instrukcje warunkowe `switch` po odświeżeniu i skróceniu ich składni. Jedynym problemem jest wersja Java – wiele firm nadal używa wcześniejszych wersji niż wersja 14, więc wyrażenia i instrukcje `switch` są dostępne nie na każdym projekcie.