



Java 9+ main features



Michał Nowakowski

Lead Software Engineer @EPAM

michal@nowakowski.me.uk

Timeline

// Java 8 – 08.2014 (**LTS – 03.2025**)

// Java 9 – 09.2017

// Java 10 – 03.2018

// Java 11 – 09.2018 (**LTS – 09.2026**)

// Java 12 – 03.2019

LTS – Long Term Support (8 years)



Java 9

Kolekcje – factory methods

// Metody statyczne do tworzenia kolekcji z podanymi elementami:
Set.of(), List.of(), Map.of(), Map.ofEntries()

```
List<Integer> list1 = Arrays.asList(1, 2, 3, 4, 5);  
Set<Integer> set1 = new HashSet<>();
```

```
set1.add(1);  
set1.add(2);
```

```
List<Integer> list2 = List.of(1, 2, 3, 4, 5);  
Set<Integer> set2 = Set.of(1, 2);
```

Kolekcje – factory methods

// Metody statyczne do tworzenia kolekcji z podanymi elementami:
Set.of(), List.of(), Map.of(), Map.ofEntries()

```
Map<Integer, String> map1 = new HashMap<>();  
map1.put(1, "Adam");  
map1.put(2, "Asia");
```

```
Map<Integer, String> map2 = Map.of(1, "Adam", 2, "Asia");  
Map<Integer, String> map3 = Map.ofEntries(Map.entry(1, "Adam"),  
    Map.entry(2, "Asia"));
```

REPL - jshell

// Dynamiczny interpreter kodu: **R**ead, **E**valuate, **P**rint, **L**oop

```
jshell> boolean isLongString(String s)
{
    ...> return s.length() > 10;
    ...> }
```

```
|   created method isLongString(String)
```

```
jshell> String s1 = "shortOne";
s1 ==> "shortOne"
```

```
jshell> "very long one"
$3 ==> "very long one"
```

```
jshell> isLongString(s1)
$4 ==> false
```

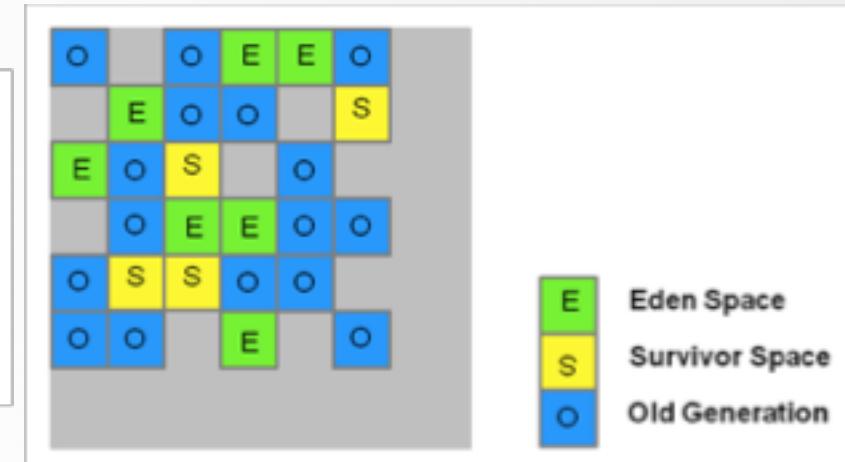
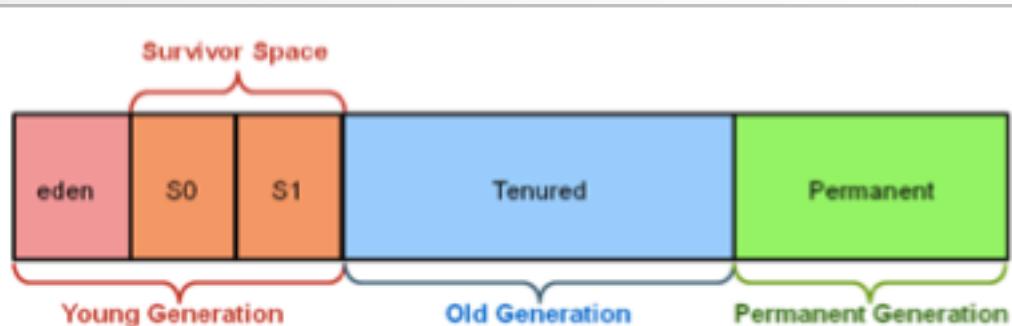
Metody prywatne w interfejsach

// Możliwość definiowania metod prywatnych w interfejsach
(widziane tylko w metodach default lub statycznych)

```
interface Printer {  
  
    default void print(String content) {  
        printToConsole(content);  
    }  
  
    private void printToConsole(String content) {  
        System.out.println(content);  
    }  
}
```

Domyślny GC – G1

// GC używany dla wielkich obszarów pamięci, podobny do klasycznego GC, ale pamięć jest podzielona na wiele regionów zarządzanych osobno (**-XX:+UseG1GC**)



Moduły

// Klasa to pojemnik na pola i metody

// Pakiet to pojemnik na klasy i interfejsy

// Moduł to pojemnik na pakiety

// Moduły mogą importować pakiety, inne moduły, i wystawiać na zewnątrz określone pakiety – publikować je

Moduły

- // Możliwość odchudzenia JDK – nasza aplikacja nie musi wykorzystywać i być dostarczana ze wszystkimi modułami Javy
- // Opis modułu umieszczamy w pliku **module-info.java**

```
module mymodule {  
    requires java.base;  
    exports com.infoshareacademy.service;  
}
```



Java 10

Wnioskowanie typu

// Wnioskowanie typu dla zmiennych - nowe słowo kluczowe **var**

// Zmienne lokalne, lokalne finalne i operator diamentowy:

```
List<Integer> list1 = Arrays.asList(1, 2, 3, 4);  
var list2 = Arrays.asList(1, 2, 3, 4);
```

```
Map<String, String> map1 = new HashMap<>();  
var map2 = new HashMap<>();
```

```
final String s1 = "Nowy string";  
final var s2 = "Nowy string";
```

Wnioskowanie typu

// Pętla for-each:

```
var list1 = Arrays.asList(1, 2, 3, 4);
for (Integer i : list1) {
    System.out.println(i);
}

for (var i : list1) {
    System.out.println(i);
}
```

Wnioskowanie typu

// Klasy anonimowe:

```
var task1 = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello");
    }
};
```

Bezargumentowy **Optional.orElseThrow**

// Dodano bezargumentową wersję metody **orElseThrow()** w klasie **Optional**:

```
Optional o = Optional.of(123);  
o.orElseThrow();
```

// Rzuca wyjątek typu **NoSuchElementException**

Kopiowanie kolekcji

// Dodano metody służące do tworzenia niemodyfikowalnych kopii istniejący kolekcji (`Set.copyOf()`, `List.copyOf()`, `Map.copyOf()`):

```
var list1 = Arrays.asList(1, 2, 3, 4);
var copy = List.copyOf(list1);
```

Nowe kolektory

// Dodano nowe kolektory, służące do tworzenia niemodyfikowalnych kolekcji ze strumienia (Collectors.toUnmodifiableSet(), Collectors.toUnmodifiableMap(), Collectors.toUnmodifiableList()):

```
var list1 = Arrays.asList(1, 2, 3, 4);

var copy = list1.stream()
    .map(i -> i*2)
    .sorted()
    .collect(toUnmodifiableList());
```



Java 11

String – dodatkowe metody

// **String.strip()** – usuwa białe znaki na początku i na końcu tekstu

// **String.stripLeading()** – usuwa białe znaki na początku tekstu

// **String.stripTrailing()** – usuwa białe znaki na końcu tekstu

```
var text = "    example text      ";
System.out.println(text.strip()); // "example text:
System.out.println(text.stripLeading()); // "example text      "
System.out.println(text.stripTrailing()); // "    example text"
```

String – dodatkowe metody

// String.isBlank() – sprawdza, czy tekst jest pusty (zawiera tylko białe znaki)

// String.lines() – zwraca strumień linii tekstu

// String.repeat(n) – powiela tekst n razy

```
var text = "name";
System.out.println(text.isBlank()); // false
```

```
Stream<String> stream = text.lines();
```

```
System.out.println(text.repeat(2)); // namename
```

Files – dodatkowe metody

// **Files.writeString()** – zapisuje ciąg znaków do pliku

// **Files.readString()** – czyta ciąg znaków z pliku

// **Files.isSameFile()** – sprawdza, czy podane ścieżki wskazują na ten sam plik

```
var path1 = Paths.get("/tmp/plik1");
var path2 = Paths.get("/tmp/plik1");
```

```
Files.writeString(path1, "test string");
System.out.println(Files.readString(path1)); // "test string"
```

```
System.out.println(Files.isSameFile(path1, path2)); // true
```

var – parametry wejściowe lambd

// Umożliwiono wykorzystanie słowa kluczowego **var** w kontekście parametrów wejściowych wyrażeń lambda

```
var list1 = Arrays.asList(1, 2, 3, 4);  
  
Collections.sort(list1, (var o1, var o2) -> o1 - o2);
```

Optional.isEmpty

// Dodano metodę sprawdzającą, czy dany **Optional** jest pusty (przeciwieństwo **isPresent()**):

```
Optional o = Optional.of(123);
o.isEmpty(); // false
o.isPresent(); // true
```



Java 12

Rozszerzony switch (preview)

// Rozszerzono funkcjonalność polecenia **switch**

// Zamiast kilku instrukcji **case** zastosowano składnię przypominającą wyrażenie lambda

// **Switch** może być także traktowany jako metoda, która zwraca wynik

Rozszerzony switch (tak było)

```
var s = scanner.nextLine();
switch (s) {
    case "A":
    case "a":
        System.out.println("Wpisano a lub A");
        break;

    default:
        System.out.println("Nieznana wartosc");
        break;
}
```

Rozszerzony switch (tak jest w 12)

```
var s = scanner.nextLine();
switch (s) {
    case "A", "a" -> System.out.println("WPisano a lub A");

    default -> System.out.println("Nieznana wartosc");
}
```

Rozszerzony switch (zwracanie wyniku)

```
var wynik = switch (s) {  
    case "A", "B" -> 2;  
    default -> -1;  
};
```

Microbenchmark Suite

// Narzędzia bazujące na **Java Microbenchmark Harness (JMH)**, do wygodnego testowanie wydajności kodu źródłowego JDK.

// Głównie dla programistów rozwijających JDK.

```
# Run progress: 0.00% complete, ETA 01:31:40
# Fork: 1 of 5
# Warmup Iteration 1: 19.849 ns/op
# Warmup Iteration 2: 19.067 ns/op
# Warmup Iteration 3: 20.044 ns/op
# Warmup Iteration 4: 20.050 ns/op
# Warmup Iteration 5: 20.061 ns/op
Iteration 1: 20.037 ns/op
Iteration 2: 20.019 ns/op
Iteration 3: 20.070 ns/op
Iteration 4: 20.052 ns/op
Iteration 5: 20.024 ns/op
```

Q&A

// THANKS!