Uzupełnienie Informacji cz1

Krzysztof Gębicz

Co to jest wyjątek?

Wyjątek = obiekt opisujący błąd lub nietypową sytuację w czasie wykonania.

W Javie wyjątki dziedziczą z `Throwable`:

- `Throwable`
  - `Error` (błędy systemowe — zwykle ich nie obsługujemy)
  - `Exception` (obsługiwalne)

throw — kiedy i jak zgłaszać wyjątek

- `throw` służy do jawnego zgłoszenia wyjątku z kodu.
- Użycie: `throw new IllegalArgumentException("komunikat");`

Przykład:

```
if (age < 0) {
    throw new IllegalArgumentException("Wiek nie może być ujemny");
}
```

Co to jest throws

- Deklaracja w nagłówku metody.
- Informuje, że metoda może zgłosić wyjątek.
- Wymagana przy tzw. *checked exceptions* (np. `IOException`, `SQLException`).

```
void czytaj() throws IOException {
    FileReader f = new FileReader("plik.txt");
}
```

Po co throws

- Żeby nie ukrywać błędów – programista wie, że musi obsłużyć wyjątek.
- Żeby przekazać problem wyżej (do metody, która wie, co z nim zrobić).
- Kompilator wymusza bezpieczeństwo – jeśli zapomnisz, program się nie skompiluje.

Kiedy używać

Gdy:

- metoda korzysta z kodu, który rzuca *checked exceptions*,
- nie wiesz, jak sensownie obsłużyć błąd (np. I/O, baza).

Nie trzeba:

- dla `RuntimeException` (np. `NullPointerException`),
- jeśli obsługujesz wyjątek wewnątrz metody.

Najczęstsze wyjątki z throws

Biblioteka	Typ błędu	Przykład
`java.io`	`IOException`	operacje na plikach
`java.sql`	`SQLException`	baza danych
`java.text`	`ParseException`	parsowanie dat
`java.net`	`MalformedURLException`	błędny adres URL

Po co nam try / catch / finally

Umożliwia bezpieczne reagowanie na błędy bez przerywania programu.

- `try` — kod, który może spowodować wyjątek
- `catch` — reakcja, gdy wystąpi błąd
- `finally` — kod wykonywany zawsze (sprzątanie, zamykanie plików)

try — blok chroniony

- W `try` umieszczamy kod, który *może rzucić wyjątek*.
- Jeśli nic się nie stanie → `catch` jest pomijany.

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

Gdy błąd wystąpi, sterowanie *skacze* do pasującego `catch`.

Catch — obsługa błędu

- `catch` przechwytuje wyjątek danego typu i pozwala go obsłużyć.
- Można mieć kilka `catch` dla różnych wyjątków.

```
try {
    int x = Integer.parseInt("abc");
} catch (NumberFormatException e) {
    System.out.println("Niepoprawny format liczby!");
}
```

Kolejność catch

- Zawsze od najbardziej szczegółowego do ogólnego.
- Jeśli odwrotnie → błąd kompilacji.

```
try {
    ...
} catch (FileNotFoundException e) { ... }
  catch (IOException e) { ... } // poprawnie
```

`catch (Exception e)` łapie *wszystko*, ale lepiej łapać konkretny typ.

finally — kod wykonywany zawsze

- Wykonuje się:
  - po `try`, jeśli nie było błędu,
  - po `catch`, jeśli błąd był.
- Idealne miejsce na zamykanie plików, połączeń, czyszczenie danych.

finally — kod wykonywany zawsze

```
try {
    FileReader fr = new FileReader("plik.txt");
} catch (IOException e) {
    System.out.println("Błąd pliku!");
} finally {
    System.out.println("Zawsze się wykona");
}
```

Co to jest tablica

Tablica to zbiór elementów tego samego typu, ułożonych w pamięci jeden po drugim.

- Stały rozmiar – ustalany przy tworzeniu.
- Dostęp przez indeks (zaczyna się od 0).
- Typ może być prymitywny (`int[]`) lub obiektowy (`String[]`).

```
int[] liczby = new int[5];

String[] imiona = {"Ala", "Ola", "Jan"};
```

Każdy element tablicy ma domyślną wartość (`0`, `false`, `null`).

Deklaracja i inicjalizacja

- Dwa sposoby tworzenia:

```
int[] a = new int[5];        // pusta tablica 5 elementów

int[] b = {1, 2, 3, 4, 5}; // od razu z wartościami
```

- Rozmiar:

```
System.out.println(b.length); // 5
```

- Nie można zmienić długości po utworzeniu!

Dostęp do elementów

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

System.out.println(t[0]); // 10

t[1] = 99; // zmiana elementu
```

Tablice wielowymiarowe

- Tablica tablic, np. dwuwymiarowa:

```
int[][] macierz = new int[2][3];

macierz[0][1] = 5;
```

- Różne długości wierszy są dozwolone:

```
int[][] nieregularna = { {1,2}, {3,4,5} };
```

Zalety tablic

- Bardzo szybki dostęp przez indeks.
- Niskie zużycie pamięci.
- Idealne, gdy rozmiar danych jest znany.
- Dobrze współpracują z typami prymitywnymi (`int`, `double`).

Wady tablic

Stały rozmiar — nie można dodać/usunąć elementu.

Brak wygodnych metod (`add`, `remove`, `contains`).

Trzeba samodzielnie kopiować (`System.arraycopy`, `Arrays.copyOf`).

Mniej elastyczne niż kolekcje (`ArrayList`, `List`).

Kiedy używać tablic

Gdy:

- rozmiar danych jest znany i stały,
- potrzebna maksymalna wydajność,
- dane to typy proste (int, double itp.).

Unikaj, gdy:

- rozmiar ma się zmieniać,
- potrzebujesz metod kolekcji (`add`, `remove`),
- łatwość obsługi ważniejsza niż szybkość.

java.util.Arrays

Zawiera przydatne metody statyczne dla tablic:

1. Arrays.sort(a);
2. Arrays.toString(a);
3. Arrays.copyOf(a, newSize);
4. Arrays.binarySearch(a, key);

Ułatwia operacje, które tablice same w sobie nie posiadają.

ArrayList — dynamiczna tablica

- Część kolekcji (`java.util`), rośnie automatycznie.
- Można dodawać i usuwać elementy w locie.

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

imiona.add("Jan");

imiona.add("Anna");

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

Łatwe w użyciu, działa jak „rozszerzalna tablica”.

Porównanie: Tablica vs ArrayList

Cecha	Tablica	ArrayList
Rozmiar	Stały	Dynamiczny
Typ danych	Prymitywy i obiekty	Tylko obiekty
Metody	Brak	add, remove, get, set
Wydajność	Bardzo wysoka	Wysoka (trochę wolniejsza)
Kiedy użyć	Dane o znanym rozmiarze	Zmienna liczba elementów