Programowanie Obiektowe

Lekcja 1

Krzysztof Gębicz

Wprowadzenie do Javy

Cel lekcji

- Zrozumieć podstawy jest Java
- Poznać podstawową składnię w Javie

Czym jest Java?

- Język wysokiego poziomu, obiektowy
- Działa na platformie JVM (Java Virtual Machine)
- Zastosowania: aplikacje desktopowe, webowe, Android, konsolowe

Co jest wbudowane w Javę?

Liczba klas wynosi tysiące, a pakietów setki.

Biblioteka obejmuje różne obszary, takie jak:

- **Kolekcje** (np. `java.util`)
- **Wejście/Wyjście** (np. `java.io`, `java.nio`)
- **Obsługa sieci** (np. `java.net`)
- **Bezpieczeństwo** (np. `java.security`)
- **Wielowątkowość** (np. `java.util.concurrent`)
- **Obsługa baz danych** (np. `java.sql`)
- **Interfejsy graficzne** (np. `java.awt`, `javax.swing`)

Przykład

```java
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import javax.swing.JOptionPane;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Kolekcje - lista imion
        ArrayList<String> imiona = new ArrayList<>();
        imiona.add(e:"Ala");
        imiona.add(e:"Ola");
        imiona.add(e:"Jan");

        // Wejście/Wyjście - zapis do pliku
        try (FileWriter fw = new FileWriter(fileName:"imiona.txt")) {
            for (String imie : imiona) {
                fw.write(imie + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Wielowątkowość - uruchomienie wątku
        ExecutorService executor = Executors.newSingleThreadExecutor();
        executor.submit(() -> System.out.println(x:"Wątek działa w tle..."));
        executor.shutdown();

        // Interfejs graficzny - prosty komunikat
        JOptionPane.showMessageDialog(parentComponent:null, message:"Zapisano imiona do pliku!");
    }
}
```

Typy danych i zmienne

- **int** – liczby całkowite, podstawowy typ.
- **double** – liczby zmiennoprzecinkowe, najczęściej stosowany.
- **boolean** – wartości logiczne (`true`, `false`).
- **char** – pojedynczy znak (przydatny przy operacjach na tekście).
- **String** – tekst, absolutnie podstawowy w praktyce.
- **Tablice** (np. `int[]`) – podstawowa struktura danych.
- **Kolekcje** (np. `ArrayList`, `List`, `Map`) – często używane w zadaniach.
- **Obiekty własnych klas**.

Operatory

Arytmetyczne – do operacji matematycznych

- `+` – dodawanie
- `-` – odejmowanie
- `*` – mnożenie
- `/` – dzielenie
- `%` – reszta z dzielenia
- `++` – inkrementacja (zwiększenie o 1)
- `--` – dekrementacja (zmniejszenie o 1)

Porównania (relacyjne) – do sprawdzania relacji między wartościami

- `==` – równe
- `!=` – różne
- `<` – mniejsze
- `>` – większe
- `<=` – mniejsze lub równe
- `>=` – większe lub równe

Logiczne – do łączenia warunków

- `&&` – AND (i)
- `||` – OR (lub)
- `!` – NOT (negacja)

Przypisania – skróty do zmiany wartości zmiennych

- `=` – przypisanie
- `+=` – dodanie i przypisanie
- `-=` – odjęcie i przypisanie
- `*=` – mnożenie i przypisanie
- `/=` – dzielenie i przypisanie
- `%=` – reszta z dzielenia i przypisanie

Przykład Użycia Operatorów

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Arytmetyczne
        int a = 10, b = 3;
        System.out.println(a + b); // 13
        System.out.println(a - b); // 7
        System.out.println(a * b); // 30
        System.out.println(a / b); // 3
        System.out.println(a % b); // 1

        a++; // a = 11
        b--; // b = 2
        System.out.println("a++ = " + a + ", b-- = " + b);

        // Porównania
        System.out.println(a == b); // false
        System.out.println(a != b); // true
        System.out.println(a > b);  // true
        System.out.println(a <= b); // false
    }
}
```

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Logiczne
        boolean x = true, y = false;
        System.out.println(x && y); // false
        System.out.println(x || y); // true
        System.out.println(!x);     // false

        // Przypisania
        int c = 5;
        c += 3; // c = 8
        c *= 2; // c = 16
        System.out.println("c = " + c);
    }
}
```

Instrukcje warunkowe

```java
int liczba = 10;

// Instrukcja if-else
if (liczba > 10) {
    System.out.println(x:"Liczba jest większa od 10");
}
else if (liczba < 10) {
    System.out.println(x:"Liczba mniejsza od 10");
}
else {
    System.out.println(x:"Liczba jest rowna 10");
}
```

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

Pętle

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


int i = 1;
while (i <= 5) {
    System.out.println("Iteracja: " + i);
    i++;
}


int j = 1;
do {
    System.out.println("Iteracja: " + j);
    j++;
} while (j <= 5);
```

1. **`for`**
   - **Do czego się sprawdza:** gdy znamy liczbę powtórzeń z góry.
   - **Najczęstsze użycie:** iteracja po tablicach, listach, liczbach w określonym zakresie.
   - **Charakterystyka:** licznik inicjalizowany w nagłówku pętli, zwiększany po każdej iteracji.

2. **`while`**
   - **Do czego się sprawdza:** gdy liczba iteracji zależy od warunku, który może zmieniać się dynamicznie.
   - **Najczęstsze użycie:** wczytywanie danych do momentu spełnienia warunku, powtarzanie działań dopóki coś jest prawdą.
   - **Charakterystyka:** warunek sprawdzany przed wykonaniem pętli.
3. **`do-while`**
   - **Do czego się sprawdza:** gdy chcemy wykonać blok kodu przynajmniej raz, a następnie sprawdzić warunek.
   - **Najczęstsze użycie:** menu w konsoli, powtarzanie operacji aż użytkownik poda poprawną wartość.
   - **Charakterystyka:** najpierw wykonuje ciało pętli, potem sprawdza warunek – gwarantuje przynajmniej jedno wykonanie.

Aplikacje konsolowe

Zalety Javy w kontekście:

- Automatyczne zarządzanie pamięcią (Garbage Collector) – mniej problemów z wskaźnikami.
- Prosta obsługa wejścia/wyjścia (`Scanner`, `System.out.println`)
- Bezpieczne typy – mniej błędów w obliczeniach na zmiennych.

Wady Javy w kontekście:

- Wolniejsze od natywnego C++
- Wymaga klasy `Scanner` do wejścia – nieco więcej kodu niż w C++.

Zalety C++ w kontekście:

- Bardzo szybkie wykonywanie prostych obliczeń.
- Bezpośredni dostęp do pamięci – przydatne w zadaniach wymagających optymalizacji.
- Proste wejście/wyjście (`cin` / `cout`)

Wady C++ w kontekście:

- Wskaźniki i ręczne zarządzanie pamięcią – większe ryzyko błędów.
- Składnia może być trudniejsza dla początkujących.

Przykład

```java
import java.util.Scanner;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print(s:"Podaj swoje imię: ");
        String imie = scanner.nextLine();
        System.out.println("Witaj, " + imie + "!");
        scanner.close(); // zamknięcie skanera
    }
}
```

Program w Javie pobiera od użytkownika imię i wypisuje powitanie. Wykorzystuje klasę `Scanner` do odczytu danych z konsoli. `System.out.println` wypisuje wynik na ekranie.

Program w C++ pobiera imię od użytkownika i wypisuje powitanie. Do wczytywania danych używa `getline(cin, imie)`, a do wyświetlania wyników `cout`.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string imie;
    cout << "Podaj swoje imię: ";
    getline(cin, imie); // wczytanie linii z wejścia
    cout << "Witaj, " << imie << "!" << endl;
    return 0;
}
```

Aplikacje Desktopowe

Zalety Javy

- **Uniwersalność**: Dzięki JVM (Java Virtual Machine) aplikacje działają na Windows, Linux i macOS bez zmian w kodzie.
- **Bogate biblioteki GUI**:
  - **Swing**: Stabilna, choć starsza biblioteka do tworzenia prostych interfejsów graficznych.
  - **JavaFX**: Nowoczesne narzędzie z obsługą multimediów, animacji i efektów wizualnych.
- **Automatyczne zarządzanie pamięcią**: Garbage Collector minimalizuje ryzyko błędów związanych z pamięcią.
- **Szybkie prototypowanie**: Gotowe komponenty GUI (okna, przyciski, formularze) przyspieszają tworzenie aplikacji.
- **Bezpieczeństwo**: Silny system typów i zarządzanie wyjątkami zmniejszają ryzyko błędów w trakcie działania.

Wady Javy

- **Wydajność**: Aplikacje GUI mogą działać wolniej niż natywne, szczególnie przy złożonych interfejsach.
- **Wygląd interfejsu**: Swing nie zawsze dobrze integruje się z natywnym wyglądem systemu. JavaFX jest lepszy, ale wymaga nauki.
- **Rozmiar aplikacji**: Potrzeba dołączenia JVM i bibliotek zwiększa rozmiar plików.

Zalety C++

- **Wysoka wydajność**: Natywne aplikacje są szybsze i bardziej responsywne niż te w Javie.
- **Kontrola nad pamięcią**: Ręczne zarządzanie pozwala na precyzyjną optymalizację zasobów.
- **Dostęp do natywnych funkcji**: Możliwość korzystania z API specyficznych dla systemu operacyjnego.
- **Zaawansowane biblioteki GUI**: Qt i wxWidgets umożliwiają tworzenie profesjonalnych, nowoczesnych interfejsów.

Wady C++

- **Złożoność**: Ręczne zarządzanie pamięcią zwiększa ryzyko błędów, takich jak wycieki pamięci.
- **Czas tworzenia GUI**: Budowanie interfejsów wymaga więcej kodu i czasu niż w Javie.
- **Ograniczona przenośność**: Aplikacje wymagają rekompilacji, a czasem dostosowania kodu dla różnych systemów.

Aplikacje Mobilne

Zalety Java

- **Popularność na Androidzie**: Java jest natywnie wspierana w Android SDK, co ułatwia rozwój aplikacji mobilnych na tę platformę.
- **Bogate API Androida**: Java oferuje łatwy dostęp do rozbudowanych bibliotek Androida, umożliwiając tworzenie zaawansowanych aplikacji.
- **Automatyczne zarządzanie pamięcią**: Garbage Collector minimalizuje ryzyko wycieków pamięci, co jest kluczowe w środowisku mobilnym.
- **Szybkie prototypowanie**: Gotowe komponenty UI w Android Studio przyspieszają tworzenie interfejsów użytkownika.
- **Społeczność i zasoby**: Ogromna liczba bibliotek, frameworków i wsparcia społeczności dla programistów Androida.

Wady Java

- **Wydajność**: Aplikacje w Javie mogą być wolniejsze od natywnych aplikacji C++ z powodu overheadu JVM.
- **Ograniczenia platformy**: Java jest głównie używana na Androidzie, co ogranicza przenośność na iOS bez dodatkowych frameworków.
- **Rozmiar aplikacji**: Potrzeba dołączenia JVM lub bibliotek Androida zwiększa rozmiar aplikacji.

Zalety C++

- **Wysoka wydajność**: Natywny kod C++ zapewnia lepszą szybkość i responsywność, kluczowe w grach mobilnych i aplikacjach wymagających dużej mocy obliczeniowej.
- **Przenośność między platformami**: Biblioteki jak Qt lub silniki gier (np. Unreal Engine, Unity z C++) umożliwiają tworzenie aplikacji na Androida i iOS z jednym kodem źródłowym.
- **Kontrola nad zasobami**: Ręczne zarządzanie pamięcią pozwala optymalizować zużycie zasobów, co jest istotne na urządzeniach mobilnych o ograniczonych możliwościach.
- **Wsparcie dla gier i multimediów**: C++ jest preferowanym językiem w silnikach gier, takich jak Unreal Engine, dzięki niskopoziomowemu dostępowi do sprzętu.

Wady C++

- **Złożoność programowania**: Ręczne zarządzanie pamięcią i wskaźnikami zwiększa ryzyko błędów, co może prowadzić do niestabilności aplikacji.
- **Trudniejsze tworzenie UI**: Budowanie interfejsów użytkownika w C++ (np. za pomocą Qt) wymaga więcej kodu i czasu niż w Javie na Androidzie.
- **Mniejsza społeczność mobilna**: Mniejsza liczba zasobów i wsparcia dla C++ w kontekście aplikacji mobilnych w porównaniu do Javy na Androidzie.