Lekcja 3 Część 1: Wstęp i metody

Krzysztof Gębicz

Co to jest programowanie obiektowe (OOP)?

Programowanie obiektowe (OOP) to sposób pisania programów, w którym myślimy w kategoriach obiektów. Obiekt to element programu, który posiada:

- Stan (opisany przez pola/atrybuty)
- Zachowanie (opisane przez metody)

Filary OOP:

- Enkapsulacja - ukrywanie danych i kontrola dostępu.
- Abstrakcja - skupienie się na najważniejszych cechach obiektu.
- Dziedziczenie - możliwość tworzenia nowych klas na bazie istniejących.
- Polimorfizm - różne zachowania tej samej metody w zależności od kontekstu.

W tej części skupimy się na enkapsulacji i metodach.

Klasa i obiekt

Teoria:

- Klasa to wzór/szablon, według którego tworzymy obiekty.
- Obiekt to instancja klasy, czyli realne „wcielenie” szablonu.
- Klasa może zawierać pola (stan) i metody (zachowania).

Fragment Kodul:

```
public class Osoba {

    private String imie;

    private String nazwisko;

    private int wiek;
```

Deklarujemy pola prywatne - dostęp do nich będzie kontrolowany metodami.

Konstruktor

Teoria:

- Konstruktor to specjalna metoda wywoływana przy tworzeniu obiektu.
- Ma tę samą nazwę co klasa i nie zwraca żadnego typu.
- Odpowiada za poprawną inicjalizację pól.

Fragment kodu:

```
public Osoba(String imie, String nazwisko, int wiek) {
    this.imie = imie;
    this.nazwisko = nazwisko;
    this.wiek = wiek;
}
```

`this` oznacza bieżący obiekt i rozróżnia pola od parametrów.

Modyfikatory dostępu

Teoria:

- Modyfikatory określają, kto ma dostęp do pól i metod.
- Najczęściej używane:
  - public - dostęp z dowolnego miejsca w programie.
  - private - dostęp tylko wewnątrz danej klasy.
  - protected - dostęp w klasie i klasach dziedziczących.
  - domyślny (package-private) - dostęp tylko w obrębie pakietu.

Dlaczego to ważne?

- Pozwala kontrolować, które elementy klasy są widoczne na zewnątrz.
- Chroni przed przypadkową zmianą wewnętrznych danych.
- Wspiera zasadę enkapsulacji.

Metody obiektu

Teoria:

- Metody definiują, co obiekt może robić.
- Mogą zwracać wartości (`return`) albo być typu `void` (tylko efekt).
- Zwykle działają na polach obiektu.

Fragment kodu:

```
public void przedstawSie() {

    System.out.println("Cześć, mam na imię " +
    imie + " " + nazwisko + ", mam " + wiek + "
    lat.");

}
```

Metoda korzysta z pól obiektu, by coś wypisać.

Tworzenie obiektu

Teoria:

- Do utworzenia obiektu używamy słowa kluczowego new.
- new wywołuje konstruktor.
- Wynik zapisywany jest w zmiennej referencyjnej.

Fragment kodu:

```
public static void main(String[] args) {
    Osoba a = new Osoba("Jan", "Kowalski", 33);
    a.przedstawSie();
}
```

Tworzymy obiekt i wywołujemy na nim metodę.

Pola statyczne i metody statyczne

Teoria:

- Pola instancyjne należą do obiektu.
- Pola statyczne (`static`) są wspólne dla całej klasy.
- Metody statyczne można wywołać bez tworzenia obiektu.

Fragment kodu:

```
private static int licznikStudentow = 0;


public static int getLicznikStudentow() {
    return licznikStudentow;
}
```

Licznik studentów rośnie przy każdym nowym obiekcie.

Budowa metody

Teoria: Każda metoda składa się z:

- Modyfikatora dostępu (`public`, `private`)
- Typu zwracanego (`int`, `void`, itp.)
- Nazwy (np. `obliczSume`)
- Listy parametrów (dane wejściowe)

Fragment kodu:

```
public int suma(int a, int b) {
    return a + b;}
public void wypiszPowitanie(String imie) {
    System.out.println("Cześć, " + imie + "!");}
```

Pierwsza metoda zwraca wynik, druga tylko coś wypisuje.

Parametry i zwracanie wartości

Teoria:

- Typy proste (np. `int`) są przekazywane przez wartość (kopiowane).
- Obiekty są przekazywane przez referencję (kopiowany jest adres).
- `return` przerywa działanie metody i zwraca wynik.

Fragment kodu:

```
public void modifyPrimitive(int x) {

    x = 99; // nie wpływa na oryginał}

public void modifyObject(StringBuilder sb) {

    sb.append(" dopisane"); // zmiana widoczna poza
metodą

}
```

Dla obiektów metoda może zmienić stan, dla prymitywów — nie.

Enkapsulacja: gettery i settery

Teoria:

- Pola ustawiamy jako `private`.
- Dostęp do nich zapewniają gettery i settery.
- Dzięki temu możemy dodać walidację.

Fragment kodu:

```
public int getWiek() {
    return wiek;}

public void setWiek(int wiek) {
    if (wiek < 0) throw new BłądArgumentu("Wiek nie może być ujemny");
    this.wiek = wiek;
}
```

Przeciążanie metod

Teoria:

- Metoda może mieć tę samą nazwę, ale inny zestaw parametrów.
- To ułatwia pracę - różne sposoby wywołania tej samej operacji.

Fragment kodu:

```
public void log(String msg) {
    System.out.println(msg);}
public void log(String msg, int level) {
    System.out.println("[" + level + "] " + msg);}
```

this i final

Teoria:

- `this` wskazuje na bieżący obiekt (np. w konstruktorze).
- `final` oznacza, że wartość nie może się zmienić (dla pól) lub parametr nie może zostać nadpisany.

Fragment kodu:

```
public Konto(String numer, double poczatkowyStan) {
        this.numer = numer;
        this.stan = poczatkowyStan;}
    public void wplac(final double kwota) {
        if (kwota <= 0) throw new BłądArgumentu("Kwota musi być >
        0");
        this.stan += kwota;}
```