Lekcja 4 Część 2: Rozwinięcie Klas

Krzysztof Gębicz

Enkapsulacja

Enkapsulacja to nie tylko prywatne pola i gettery/settery. To cała filozofia projektowania klas, która zakłada ukrywanie szczegółów implementacyjnych.

Korzyści:

- Bezpieczeństwo danych - użytkownik klasy nie może przypadkowo nadpisać stanu obiektu.
- Łatwiejsze utrzymanie - możemy zmienić sposób przechowywania danych wewnątrz klasy, bez zmiany w kodzie korzystającym z tej klasy.
- Dodawanie logiki - np. walidacja, logowanie zmian, automatyczne obliczenia.

Przykład

Dziedziczenie

- Dziedziczenie to jeden z głównych filarów OOP. Umożliwia tworzenie hierarchii klas i wielokrotne wykorzystanie istniejących fragmentów kodu.

Ważne elementy:

- super – odwołanie do metod i konstruktorów klasy bazowej.

- @Override – informacja, że metoda nadpisuje zachowanie klasy bazowej.

Przykład

Dziedziczenie - Ważne elementy

Ważne elementy:

- super – odwołanie do metod i konstruktorów klasy bazowej.

- @Override – informacja, że metoda nadpisuje zachowanie klasy bazowej.

- protected – pozwala klasom dziedziczącym korzystać z pól, które są niedostępne na zewnątrz.

Dziedziczenie - Praktyczne zastosowanie

- Tworzenie hierarchii obiektów (np. Pojazd → Samochód → ElektrycznySamochód).

- Eliminacja powtarzalności kodu.

Abstrakcja

Abstrakcja pozwala skupić się na tym, *co* obiekt robi, a nie *jak* to robi.

Klasy abstrakcyjne:

- Mogą zawierać zarówno metody z implementacją, jak i metody abstrakcyjne (bez ciała).

- Nie można utworzyć obiektu klasy abstrakcyjnej

Przykład

Polimorfizm

Polimorfizm oznacza możliwość wywołania tej samej metody na różnych obiektach i uzyskanie różnych wyników.

```
Figura[] figury = {
    new Prostokat(3, 4),
    new Kolo(5),
    new Trojkat(6, 2)
};
for(Figura f : figury) {
    System.out.println("Pole: " + f.obliczPole());
}
```

Słowo kluczowe this

`this` odnosi się do aktualnego obiektu klasy.

Słowo kluczowe this zastosowania

Rozróżnianie pól od parametrów:

```
public class Osoba {
    private String imie;

    public Osoba(String imie) {
        this.imie = imie; // this odróżnia pole klasy od parametru
    }
}
```

Słowo kluczowe this zastosowania

Wywoływanie innych konstruktorów:

```
public class Punkt {
    private int x, y;
    public Punkt() {
        this(0, 0); // domyślna wartość
    }
    public Punkt(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

Słowo kluczowe this zastosowania

Przekazywanie obiektu jako argument:

```
public void porownaj(Punkt inny) {
    if(this.x == inny.x && this.y == inny.y)
{
        System.out.println("Punkty są
równe");
    }
}
```

Słowo kluczowe final

`final` ogranicza możliwość zmiany zachowania klas, metod i pól.

Słowo kluczowe final - Przykłady

final w polu:

```
public class Stałe {
    public static final double PI =
3.14159;
}
```

Pole jest stałe – nie można przypisać nowej wartości.

Słowo kluczowe final - Przykłady

final w metodzie:

```
public class Zwierze {
    public final void oddychaj() {
        System.out.println("Oddycham tlenem");
    }
}
```

Metoda nie może być nadpisana w klasach dziedziczących.

Słowo kluczowe final - Przykłady

final w klasie:

```
public final class Matematyka {
    public static int dodaj(int a, int b) {
        return a + b;
    }
}
```

Klasa nie może być rozszerzana.

Podsumowanie

- Enkapsulacja chroni dane i pozwala je kontrolować.
- Dziedziczenie umożliwia ponowne wykorzystanie kodu.
- Abstrakcja pozwala projektować systemy bardziej elastyczne.
- Polimorfizm daje możliwość różnego zachowania tej samej metody w zależności od obiektu.
- this odnosi się do bieżącego obiektu i pomaga w wielu sytuacjach.
- final ogranicza możliwość zmiany pól, metod lub klas.

Zadania Utrwalające

Ćwiczenie 1 – Enkapsulacja

Napisz klasę `Samochod`, która posiada pola: `marka`, `model`, `predkosc`.

- Wszystkie pola ustaw jako `private`.
- Utwórz konstruktor ustawiający wartości.
- Dodaj gettery i settery do pola `predkosc`, które będą uniemożliwiały ustawienie wartości mniejszej niż 0 lub większej niż 300.
- Przetestuj klasę w nowym pliku `Main`.

Ćwiczenie 2 – Dziedziczenie

Utwórz klasę `Osoba` z polami `imie`, `nazwisko`.

- Utwórz klasę `Nauczyciel` dziedziczącą po `Osoba`, dodaj pole `przedmiot`.
- Utwórz klasę `Uczen` dziedziczącą po `Osoba`, dodaj pole `klasaSzkolna`.
- W każdej klasie napisz metodę `przedstawSie()`, która wypisuje inne informacje w zależności od typu obiektu.

Ćwiczenie 3 – Abstrakcja (klasy abstrakcyjne)

Stwórz klasę abstrakcyjną `Figura` z metodą abstrakcyjną `double obliczPole()`.

- Utwórz klasy `Kolo` i `Prostokat`, które dziedziczą po `Figura` i implementują metodę `obliczPole()`.

- Utwórz tablicę figur i wypisz pola każdej figury w pętli.

Ćwiczenie 4 – Abstrakcja (interfejsy)

Zdefiniuj interfejs `Pojazd` z metodami: `przyspiesz()` i `zatrzymaj()`.

- Utwórz klasy `Samochod` i `Rower`, które implementują ten interfejs.
- W metodzie `main` utwórz listę pojazdów i wywołaj metody interfejsu dla każdego elementu

Ćwiczenie 5 – Polimorfizm

Stwórz klasę `Zwierze` z metodą `dajGlos()`.

- Utwórz klasy `Kot` i `Pies`, które nadpisują tę metodę.
- Utwórz tablicę zwierząt i wywołaj `dajGlos()` w pętli.
- Zastanów się: dlaczego każde zwierzę reaguje inaczej na to samo wywołanie metody?

Ćwiczenie 6 – this

Napisz klasę `Punkt` z polami `x`, `y`.

- Utwórz dwa konstruktory:
  - bezparametrowy (ustawia wartości na 0,0),
  - z parametrami (ustawia wartości na podane liczby).
- W konstruktorze bezparametrowym użyj `this(...)` do wywołania drugiego konstruktora.
- Dodaj metodę `czyRowne(Punkt inny)`, która sprawdza, czy bieżący punkt (`this`) jest równy innemu punktowi.

Ćwiczenie 7 – final

Utwórz klasę `Matematyka` z polem `PI` oznaczonym jako `public static final`.

Utwórz metodę `poleKola(double r)`, która korzysta z tego pola.

Utwórz klasę `Zwierzeta` z metodą `oddychaj()`, oznaczoną jako `final`.

- Spróbuj nadpisać tę metodę w klasie `Kot` i zobacz, co zrobi kompilator.

Ćwiczenie 8 – Mini-projekt (łączenie wszystkiego)

Zaprojektuj prosty system rejestracji pojazdów:

- Klasa abstrakcyjna `Pojazd` z polami: `numerRejestracyjny`, `predkoscMax` i metodą abstrakcyjną `opis()`.
- Klasy dziedziczące: `Samochod`, `Motocykl`, `Rower`.
- Interfejs `Ruch` z metodami `przyspiesz()` i `zatrzymaj()`.
- Każda klasa pojazdu implementuje interfejs.
- W `main` utwórz tablicę pojazdów, wywołaj metody `opis()` i `przyspiesz()` dla każdego.