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.

Dziedziczenie

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

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

Interfejsy

Czym są?

- Specjalny typ, który definiuje zbiór metod (bez implementacji).
- Klasa, która implementuje interfejs, musi dostarczyć implementację tych metod.
- Klasa może implementować wiele interfejsów (w przeciwieństwie do dziedziczenia klas).

Zastosowania:

- Wspólne użycie dla różnych klas.
- Oddzielenie definicji od implementacji.
- Budowa elastycznych i łatwych w testowaniu systemów.

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