Programowanie obiektowe

Laboratorium 3 i 4 - przypomnienie wiadomości o OOP na przykładzie Javy

mgr inż. Krzysztof Szwarc

krzysztof@szwarc.net.pl

Sosnowiec, 8 marca 2017

Informacje wstępne

Konwencja nazewnictwa

- Nazwy klas zapisywane są w notacji PascalCase (NazwaKlasy).
- Nazwy metod i zmiennych zapisywane są w notacji camelCase (jakasZmienna).
- Nazwy stałych zapisywane są w notacji SCREAMING_SNAKE_CASE (TO_STALA).

Klasy

Struktura klasy

Klasa składa się z pól (zmiennych) oraz metod (funkcji).
Ogólna struktura klasy:

```
modyfikatorDostepu class NazwaKlasy
{
    modyfikatorDostepu typPola nazwaPola;

    modyfikatorDostepu zwracanyTyp
       nazwaMetody(parametry)
    { return zmiennaOZwracanymTypie; }
}
```

Klasa może nie zawierać metod lub pól. Do pól odwołujemy się tak jak do zmiennych lub korzystając ze słowa this w następujący sposób: this.nazwaPola;

Klasy

Modyfikator dostępu

W Javie wyróżniamy cztery modyfikatory dostępu:

1. public - dostęp do niego ma każdy obiekt.
2. private - dostęp do niego ma wyłącznie właściciel.
3. protected - dostęp do niego ma właściciel, klasy dziedziczące po nim oraz klasy w tym samym pakiecie.
4. default - ustawiany jeżeli nie zadeklarujemy modyfikatora dostępu. Widoczność ograniczona jest do klas znajdujących się w tym samym pakiecie.

Klasy

Struktura klasy

Każda klasa zawiera konstruktor - specyficzną metodę wywoływaną w momencie tworzenia obiektu. Sposób definiowania konstruktora:

```
class NazwaKlasy
{
    NazwaKlasy()
    { }
    NazwaKlasy(String zmienna)
    { }
}
```

Konstruktor, tak jak inne metody można przeciążać (ta sama nazwa metody przyjmująca różne parametry).

Klasy

Opis

Hermetyzacja (enkapsulacja) polega na odizolowaniu pól i metod od innych klas udostępniając tylko niezbędne elementy. W praktyce polega na stosowaniu modyfikatorów typu private lub protected dla pól i niektórych metod (które nie muszą być używane z zewnątrz). Dostęp do prywatnych pól powinien odbywać się za pomocą metod zwracających i przypisujących odpowiednie wartości (tzw. gettery i settery).

Przykład - hermetyzacja

```
public class Pracownik
{
    private String imie;

    public void ustawImie(String imie)
    {
        this.imie=imie;
    }

    public String zwrocImie()
    {
        return this.imie;
        // return imie;
    }
}
```

Przykład - konstruktor

```
public class Pracownik
{
    private String imie;

    Pracownik(String imie)
    {
        this.imie=imie;
    }

    public void ustawImie(String imie)
    {
        this.imie=imie;
    }

    public String zwrocImie()
    {
        return this.imie;
    }
}
```

Przykład - użycie klasy

```
public static void main(String[] args)
{
    Pracownik naszPracownik = new
        Pracownik("Jan");
    System.out.println(naszPracownik.zwrocImie());
    // Jan
    naszPracownik.ustawImie("Andrzej");
    System.out.println(naszPracownik.zwrocImie());
    // Andrzej
}
```

Klasy

Modyfikator static

Modyfikator static wykorzystywany jest do stworzenia pola lub metody dostępnej dla każdej instancji klasy (obiektu). Takie pole/metoda istnieje zawsze (nawet jeżeli nie został utworzony obiekt danej klasy; analogicznie do klasy Math). Pole statyczne może być wykorzystywane jako licznik utworzonych instacji klasy.

Dziedziczenie

Dziedziczenie

Dziedziczenie polega na przekazaniu wybranych cech danej klasy nadrzędnej (pól/metod) innym klasom. Klasy potomne mogą posiadać swoje pola oraz metody. Aby zastosować mechanizm dziedziczenia należy dopisać po nazwie klasy słowo extends i dopisać nazwę klasy, po której ma ona dziedziczyć. W Javie jedna klasa nie może dziedziczyć z kilku klas.

Przykład

```
public class Pracownik extends Czlowiek
{
}
```

Konstrukcja super

Konstrukcja super

Aby jawnie wywołać konstruktor klasy bazowej możemy użyć konstrukcji super() (super jest referencją do rodzica - za jej pomocą możemy też wywołać np. przesłoniętą metodę klasy bazowej). Przyjmuje ona jako parametry odpowiednie wartości wymagane przez dany konstruktor klasy bazowej.

Przykład

```
public class Pracownik extends Czlowiek
{
    Pracownik(String imie, String
        nazwisko, char plec)
    {
        super(imie, nazwisko, plec);
    }
}
```

Zadania

1. Napisz klasę Czlowiek zawierającą pola reprezentujące imię oraz nazwisko. Pamiętaj o hermetyzacji i stwórz konstruktor przyjmujący obie wartości oraz konstruktor bezparametrowy ustawiający domyślne wartości pól oraz wypisujący informację „Stworzono obiekt klasy Czlowiek".
2. Napisz klasę Pracownik dziedziczącą po klasie Czlowiek, zawierającą dodatkowo pole pensja. Pamiętaj o hermetyzacji. Niech zawiera ona konstruktor przyjmujący wartość reprezentującą pensję. Utwórz obiekt klasy Pracownik.

Przesłanianie metod

Przesłonięcie metody polega na ponownym zaimplementowaniu odziedziczonej metody. Aby przesłonić metodę wystarczy ją zapisać „tradycyjnie” w klasie potomnej, jednakże zaleca się dodanie wyrażenia @Override. Niezależnie od sposobu zapisu musi zawierać ona taką samą nazwę, liczbę oraz typ argumentów, zwracać taki sam typ (lub podtyp) oraz nie może zawierać zawężonego specyfikatora dostępu. Przy wywołaniu metody klasy potomnej wykona się przesłoniony kod.

Przykład

```
public class Pracownik extends Czlowiek
{
    @Override
    public String zwrocImie()
    {
        return this.imie;
    }
}
```

Zadania

1. Napisz publiczną metodę przedstawSie w klasie Czlowiek, która wypisze na ekran jego imię oraz nazwisko. Niech konstruktor domyślny wywołuje ją.
2. Przeciąż metodę przedstawSie w klasie Pracownik. Niech wypisuje ona na ekran informację „Wywołał Pracownik”.

Część praktyczna

1. Wykonaj zadania ze strony 116 i 117 ze skryptu: `http://w.s.w.w.interia.pl/skrypt.pdf`

Dziękuję za uwagę