Programowanie w języku Java

LABORATORIUM NR 3

Semestr Zimowy

Autor:
Krzysztof Muller

Kontakt:
krzysztof.muller1@gmail.com

Enumy

- enumy to typy wyliczeniowe,
- pozwalają na zadeklarowanie ograniczonej liczby możliwych wartości,
- enum deklarujemy poza klasą (można w osobnym pliku),

```
/* Prosty enum */
enum KolorEnum {
    CZERWONY, ZIELONY, NIEBIESKI;
}
```

- w enum-ach możemy tworzyć metody i przypisywać wartości,

```
/* enum z metodą i zmienną */
enum WielkoscEnum {
    MALY(false), DUZY(true), SREDNI(false), OGROMNY(true);

    boolean wielkosc;

    WielkoscEnum(boolean czyDuzy) { wielkosc = czyDuzy; }
}
```

Ćw.1

https://github.com/kmuller86/java_lab_3/blob/przyklad_1/src/Main.java

https://github.com/kmuller86/java_lab_3/blob/cw_1/src/Main.java

Klasy

- klasy – to byty które mogą zawierać zmienne i metody,
- reprezentacją klasy są obiekty np. Pojazd **pojazd**,

```
/* Deklaracja z inicjalizacją pola 'pojazd' typem Pojazd */
Pojazd pojazd = new Pojazd();
```

```
/* Prosta klasa */
class Pojazd {
    String nazwa;
    KolorEnum kolor;
    int iloscKol;
}
```

- Klasę traktujemy jak typ a obiekt jak zmienną,
- Za pomocą obiektu możemy się odwoływać do zmiennych i metod z klasy,
- Klasa rozpoczyna się słowem **class,**
- Może mieć modyfikator dostępu (**public, protected …**),
- Klasę inicjujemy przez słowo **new** i nazwę klasy z nawiasami okrągłymi (), **pojazd = new Pojazd();**

```
Pojazd pojazd = new Pojazd();
pojazd.kolor = KolorEnum.NIEBIESKI;
pojazd.iloscKol = 4;
pojazd.nazwa = "Traktor";
```

Ćw.2

https://github.com/kmuller86/java_lab_3/blob/cw_2/src/Main.java

https://github.com/kmuller86/java_lab_3/blob/cw_2/src/Main.java

Constructor, Settery i Gettery

- Konstruktor jest specjalną metodą która ma nazwę taką samą jak klasa w której deklarujemy konstruktor,
- Konstruktor nie posiada typu zwracanego,
- Konstruktor może posiadać operator widoczności **private**, **protected**, **public** i być wywoływany zgodnie z widocznością tych operatorów …
- Konstruktor może przyjmować parametry, lub być bezparametrowy,
- W klasie może być wiele konstruktorów i możemy je przeciążać,
- Jeśli nie zadeklarujemy w klasie konstruktora, to zostanie dodany domyślny konstruktor bezparametrowy niejawny np. **Punkt(){}**
- Jeśli stworzymy jakiś konstruktor parametrowy, to konstruktor bezparametrowy domyślny nie zostanie dodany, jeśli go potrzebujemy, to musimy go jawnie zadeklarować,
- Konstruktor służy głównie do inicjalizacji pól w klasie lub ustawienia ich domyślnych wartości,

```java
/* Pusty konstruktor */
Punkt() {
}

/* konstruktor z parametrami */
Punkt(int x, int y) {
    this.x = x;
    this.y = y;
}

/* przeładowanie konstruktorów - wiele konstruktorów
z inną liczbą parametrów lub różnymi typami*/
Punkt(int x, int y, int z) {
    this.x = x;
    this.y = y + z;
}
```

- Na ogół chcemy mieć ograniczony dostęp do pól w klasie, aby ich zmiana była pewna, że użytkownik ma zamiar zmienić ich wartość , dlatego zazwyczaj ustawiamy widoczność pól w klasie na **private** lub **protected** (widoczne w dziedziczeniu),
- **Settery** są metodami które umożliwiają nadpisanie wartości pól w klasie niezależnie od ustawionej na nich widoczności,
- **Settery** zwyczajowo nadajemy nazwę **set +** ‘nazwa zmiennej z dużej litery’ → **setZmienna1**,
- **Gettery** są to metody które umożliwiają dostęp do danych z klasy niezależnie od ustawionej na nich widoczności,
- **Gettery** zwyczajowo nadajemy nazwę **get +** ‘nazwa zmiennej z dużej litery’ → **getZmienna1**,
- Settery i gettery mogą zmieniać pola, lub zwracać ich zmienioną/sformatowaną wartość

```java
/* settery - metody modyfikujące zmienne */
public void setX(int x) {
    this.x = x;
}
```

```java
public int getY() {
    return y;
}
```

```java
public int getY() {
    return y + 1;
}
```

Ćw.3

https://github.com/kmuller86/java_lab_3/blob/przyklad_3/src/Main.java

https://github.com/kmuller86/java_lab_3/blob/cw_3/src/Main.java

- Można używać stworzoną przez nas klasę jako typ w innej klasie

```java
public class ObjektPunktu {
    /* zadeklarowanie zmiennej typu klasy Punkt */
    Punkt punkt;
    private int z;
```

Ćw.4

https://github.com/kmuller86/java_lab_3/tree/przyklad_4/src

https://github.com/kmuller86/java_lab_3/tree/cw_4

Dziedziczenie

- Dziedziczenie jest podstawowym mechanizmem programowania obiektowego, dzięki niemu możemy utworzyć zrozumiałe, spójne i elastyczne struktury, które łatwiej się modyfikuje w kolejnych krokach rozwijania kodu,
- W java jedna klasa może dziedziczyć tylko jedną klasę na raz,
- Dziedziczymy inne klasy za pomocą operatora **extends** (extends Nazwa_Klasy),
- Dzięki dziedziczeniu klasa potomna może używać wszystkie pola (zmienne) i metody(funkcje) z klasy nadrzędnej, oraz może dodawać swoje własne pola i metody,
- Jedna klasa może być dziedziczona wiele razy,
- Dzięki dziedziczeniu możemy deklarować obiekty za pomocą najbardziej generycznej klasy (najwyżej w hierarchii, po której dziedziczą inne klasy) i implementować za pomocą klas potomnych, sprawia to, że kod jest bardziej elastyczny,
- Za pomocą operatora **super()** w konstruktorze możemy wywołać konstruktory z klasy nadrzędnej, lub wywołać metody z klasy nadrzędnej,
- Operator **protected/public** na polach i metodach umożliwia korzystanie z nich w klasach potomnych (dziedziczących po nich), brak operatora oznacza widoczność **public** w obszarze tego samego foldera (paczki). Jeśli użyjemy pul / metod z operatorem **private** to te pola / metody nie będą widoczne w klasie potomnej,

```
public class Kierownik extends Osoba {
```

```
Osoba osoba2 = new Prezes();
```

```
public Prezes() {
    super();
    this.przychod = 0;
}
```

```
public Prezes(String imie, String nazwisko, int pieniadze, int premia, int przychod) {
    super(imie, nazwisko, pieniadze, premia);
    this.przychod = przychod;
}
```

```
@Override
public int wyliczPieniadze() {
    return super.wyliczPieniadze() + przychod;
}
```

Ćw 5

https://github.com/kmuller86/java_lab_3/tree/przyklad_5/src

https://github.com/kmuller86/java_lab_3/blob/cw_5/src/Main.java