Programowanie w języku Java - Klasy i metody abstrakcyjne, interfejsy

mgr inż. Maciej Lasota <m.lasota@tu.kielce.pl>

Version 1.0, 16-04-2017

Spis treści

Zajęcia **laboratoryjne nr 6** z programowania w języku Java - jedne zajęcia **1,5h**.

Klasy i metody abstrakcyjne

Klasy abstrakcyjne są to klasy, która nie mogą mieć swoich reprezentantów w postaci *obiektów*. Klasa abstrakcyjna jest pewnym uogólnieniem innych klas (na przykład dla występujących w rzeczywistości obiektów), lecz sama jako taka nie istnieje. W Javie klasę abstrakcyjną możemy stworzyć za pomocą słowa kluczowego `abstract`, które dodajemy do definicji standardowej klasy.

Klasy abstrakcyjne służą jedynie do definiowania klas o *charakterze abstrakcyjnym* wykorzystywanym do **dziedziczenia** przez inne klasy. Abstrakcyjność klasy oznacza, iż nie można tworzyć jej egzemplarzy (obiektów).

Przykład 1. Definiowanie klasy abstrakcyjnej

```
[ public ] abstract class NazwaKlasy {
  // definicje pól
  // definicje metod
  // definicje metod abstrakcyjnych
}
```

Przykład 2. Przykład klasy abstrakcyjnej

```
abstract class Samochod {
  String marka
  int color;

  public void hamuje() {
    // deklaracja metody
  }

  abstract public void trabi(); //deklaracja metody abstarkcyjnej
}
```

Metoda abstrakcyjna nie ma *implementacji (ciała)* i winna być zadeklarowana ze specyfikatorem `abstract`. Metody abstrakcyjne to takie, co do których nie wiemy jeszcze jaka może być ich konkretna implementacja (lub nie chcemy tego przesądzać), ale wiemy, że powinny wystąpić w zestawie metod każdej konkretnej **klasy dziedziczącej** klasę abstrakcyjną. Konkretna implementacja (definicja w klasie kodu metody) może być bardzo różna, w zależności od konkretnego rodzaju obiektów, które opisuje dana klasa.

Klasa abstrakcyjna nie musi mieć metod abstrakcyjnych.

Klasa dziedzicząca klasę abstrakcyjną musi zdefiniować wszystkie abstrakcyjne metody tej klasy, albo sama będzie *klasą abstrakcyjną* i wtedy jej definicja musi być opatrzona specyfikatorem `abstract`.

Przykład 3. Przykład dziedziczenia klasy abstrakcyjnej

```
class Limuzyna extends Samochod {
  public void trabi() {
    //implementacja metody
  }
}
```

Interfejsy

Interfejs to definicja *abstrakcyjnego typu* posiadającego jedynie operacje, a nie dane. Klasy są definicją typów. Typów, jakie mogą mieć obiekty. Interfejsy także są definicją typów, tyle że nieco okrojoną. Klasy zawierają zmienne i stałe oraz deklaracje metod i ich implementacje. Interfejsy mogą zawierać tylko stałe i deklaracje metod bez implementacji. Interfejsy definiujemy podobnie do klas, tyle że używamy słówka kluczowego `interface`.

Interfejs to:

- zestaw publicznych abstrakcyjnych metod,
- i/lub publicznych statycznych stałych.

Implementacja interfejsu w klasie to zdefiniowanie w tej klasie wszystkich metod interfejsu. To że klasa ma implementować dany interfejs oznaczamy słowem kluczowym `implements`. Interfejs zawiera deklaracje publicznych metod abstrakcyjnych oraz ew. publicznych stałych statycznych.

Każda klasa implementująca interfejs musi zdefiniować **wszystkie** jego metody albo będzie klasą abstrakcyjną.

W Javie nie ma **wielodziedziczenia implementacji**, ale za to jest możliwe **wielodziedziczenie interfejsów**.

Przykład 4. Definiowanie interfejsu

```
[public] interface NazwaInterfejsu {
  // definicje pól
  // definicje metod
}
```

Przykład 5. Przykład interfejsu

```
public interface Alfabet {
  public static final int ALFA = 1;
  int BETA = 2; //automatycznie public static final

  //deklaracja metody
  public void literuj();
}
```

Przykład 6. Implementacja interfejsu

```
interface A { }
interface B { }
interface C extends A { } //rozszezanie interfejsu

class X implements B, C { } //implementacja
```

Klasy wewnętrzne

Klasa wewnętrzna to klasa zdefiniowana wewnątrz innej klasy.

Klasa wewnętrzna może:

- być zadeklarowana ze specyfikatorem **private** (normalne klasy nie!), uniemożliwiając wszelki dostęp spoza klasy otaczającej,
- odwoływać się do niestatycznych składowych klasy otaczającej (jeśli nie jest zadeklarowana ze specyfikatorem static),
- być zadeklarowana ze specyfikatorem **static** (normalne klasy nie!), co powoduje, że z poziomu tej klasy nie można odwoływać się do składowych niestatycznych klasy otaczającej (takie klasy nazywają się zagnieżdżonymi, ich rola sprowadza się wyłacznie do porządkowania przestrzeni nazw i ew. lepszej strukturyzacji kodu) mieć nazwę (klasa nazwana),
- nie mieć nazwy *(wewnętrzna klasa anonimowa)*,
- być lokalna – zdefiniowana w bloku (metodzie lub innym bloku np. w bloku po instrukcji if), odwoływać się do zmiennych lokalnych (o ile jest lokalna, a zmienne są deklarowane ze specyfikatorem final).

Zawarcie **klasy wewnętrznej** w klasie otaczającej **nie oznacza**, że obiekty klasy otaczającej zawierają elementy (pola) obiektów klasy wewnętrznej. Obiekt niestatycznej klasy wewnętrznej zawiera *referencję do obiektu* klasy otaczającej, co umożliwia odwoływanie się do jej wszystkich składowych. Między obiektami statycznej klasy wewnętrznej a obiektami klasy otaczającej nie zachodzą żadne związki.

Przykład 7. Definiowanie klasy wewnętrznej

```
[ public ] class NazwaKlasyA {

  class NazwaKlasyB {
    // definicje pól
    // definicje metod
  }

  // definicje pól
  // definicje metod
}
```

Przykład 8. Przykład klasy wewnętrznej

```
class D { }
abstract class E { }

class Y extends D {
  class I extends E {

  }
}
```

Przykład 9. Tworzenie obiektu klasy wewnętrznej

```
Y y = new Y();
Y.I i = y.new I();
```

Wewnętrzna klasa statyczna:

- nie potrzebuje obiektu zewnętrznego,
- brak dostępu do niestatycznych elementów klasy zewnętrznej.

Anonimowe klasy wewnętrzne nie mają nazwy. Najczęściej tworzymy anonimowe klasy wewnętrzne po to, by przedefiniować jakieś metody klasy dziedziczonej przez klasę wewnętrzną bądź zdefiniować metody implementowanego przez nią interfejsu na użytek jednego obiektu.

Przykład 10. Definiowanie anonimowej klasy wewnętrznej

```
new NazwaTypu(parametry) {
  // pola i metody klasy wewnętrznej
}
```

Przykład 11. Przykład anonimowej klasy wewnętrznej

```
class Z extends D {
  public E make() {
    return new E() {
      // implementacja klasy anonimowej
    };
  }
}
```

Bibliografia

- **Bruce Eckels**, *"Thinking in Java. Edycja polska. Wydanie IV"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Podstawy. Wydanie IX"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Techniki zaawansowane. Wydanie IX"*, wydawnictwo Helion.
- **Krzysztof Barteczko**, *"Podstawy programowania w języku Java, PJWSTK"*, http://edu.pjwstk.edu.pl/wyklady/ppj/scb/
- **Konrad Kurczyna**, *"Laboratorium Java"*, Politechnika Świętokrzyska w Kielcach.
- **Mariusz Lipiński**, *"Nauka Javy"*, http://www.naukajavy.pl/