Programowanie w języku Java - Kompozycja, dziedziczenie

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

Version 1.0, 13-03-2017

Spis treści

Podejście obiektowe umożliwia **ponowne wykorzystanie** *(ang. reusing)* już gotowych klas przy tworzeniu klas nowych, co znacznie oszczędza pracę przy kodowaniu, a także czyni programowanie mniej podatne na błędy.

Istnieją dwa sposoby ponownego wykorzystania klas:

- **kompozycja**,
- **dziedziczenie**.

Kompozycja

Z koncepcyjnego punktu widzenia **kompozycja** oznacza, że *"obiekt jest zawarty w innym obiekcie"* . Jest to relacja *"całość – część"* (B "zawiera" A). Np. obiekty typu Pojazd zawierają obiekty typu Rozmiar, Koła, Silnik itd.. Kompozycję uzyskujemy poprzez definiowanie w nowej klasie pól, które są obiektami istniejących klas.

Przykład 1. Przykład kompozycji

```
// Klasa Size
public class Size {
  //...
}

// Klasa Wheels
public class Wheels {
  //...
}

// klasa Engine
public class Engine {
  //...
}

// Klasa Vehicle
public class Vehicle {
  String color;
  int speed;
  Size size;
  Engine engine

  public Vehicle(String color, int speed, Size size, Engine engine) {
    //...
  }
  //...
  public void vehicleAttributes() {
    System.out.println("Color : " + color);
    System.out.println("Speed : " + speed);
    System.out.println("Size : " + size.toString());
    System.out.println("Engine : " + engine.toString());
  }
}
```

Dziedziczenie

Dziedziczenie jest jednym z podstawowych mechanizmów programowania obiektowego. Mechanizm ten umożliwia definiowanie nowych klas na bazie istniejących.

- Dziedziczenie jest w języku Java mechanizmem wszechobecnym i niezwykle potężnym. Prawie każda klasa a mówiąc precyzyjniej każda klasa z wyjątkiem klasy `java.lang.Object` – dziedziczy z jakiejś innej klasy, każda bowiem klasa dziedziczy w sposób niejawny ze wspomnianej klasy **Object**.
- W Javie nie ma **wielodziedziczenia** - klasa może bezpośrednio odziedziczyć tylko jedną klasę.

Dziedziczenie - podobnie jak **kompozycja** *(a nawet w większym stopniu)* - pozwala na zmniejszanie nakładów na kodowanie (reusing). Jest to również odzwierciedlenie naturalnych sytuacji.

Dziedziczenie polega na przejęciu **właściwości** i **funkcjonalności** obiektów innej klasy i ewentualnej modyfikacji tych właściwości i funkcjonalności w taki sposób, by były one bardziej wyspecjalizowane. Jest to relacja, nazywana *generalizacją-specjalizacją*: B "jest typu" A, "B jest A", a jednocześnie B specjalizuje A. A jest generalizacją B.

Przykład 2. Przykład dziedziczenia

```
public class Car extends Vehicle {
  String model;
  int seats;

  public Car(String model, int seats , String color, int speed, Size size, Engine
engine) {
    super(color, speed, size, engine);
    this.seats = seats;
    this.model = model;
    //...
   }

   public void carAttributes() {
     System.out.println("Model of Car : " + model);
     System.out.println("Seats in Car : " + seats);

     // The subclass refers to the members of the superclass
     System.out.println("Color of Car : " + color);
     System.out.println("Speed of Car : " + speed);
     System.out.println("Size of Car : " + size.toString());
     System.out.println("Engine of Car : " + engine.toString());
   }
}
```

- **this** - wskazanie na aktualny obiekt
- **super** – wskazanie na obiekt klasy bazowej (nadrzędnej)

Jeśli nie chcemy aby dana klasa **nie była dziedziczona** przez inne klasy podczas definicji klasy możemy użyć specyfikatora `final`. Wiele klas standardowych jest zabezpieczonych przed dziedziczeniem np. klasa `String`.

Przedefiniowanie (przesłanianie, nadpisywanie) metod

Przedefiniowanie metody *(ang. overriding)* nadklasy w klasie pochodnej oznacza dostarczenie w klasie pochodnej definicji nieprywatnej i niestatycznej metody z taką samą **sygnaturą** *(czyli nazwą*

i listą parametrów) jak sygnatura nieprywatnej i niestatycznej metody nadklasy, ale z ew. inną definicją **ciała metody** *(innym kodem, który jest wykonywany przy wywołaniu metody)*, przy czym:

- typy wyników tych metod muszą być **takie same** lub **kowariantne** (co oznacza m.in., że typ wyniku metody z podklasy może być podtypem wyniku metody nadklasy),
- przedefiniowanie **nie może ograniczać dostępu**: specyfikator dostępu metody przedefiniowanej w podklasie musi być taki sam lub szerszy (np. public zamiast protected) niż metody przedefiniowywanej,
- metoda przedefiniowana (z podklasy) **nie może zgłaszać** więcej lub bardziej ogólnych wyjątków kontrolowanych niż metoda przedefiniowywana (z nadklasy).

Przedefiniowując metody w podklasach warto używać **adnotacji** `@Override`. Daje ona sygnał kompilatorowi, że intencją programisty jest nadpisanie metody z klasy bazowej.

Przykład 3. Przykład przedefiniowania metody

```
class Car extends Vehicle {
  //...
  @Override
  public void vehicleAttributes() {
    //...
  }
}
```

Słowo kluczowe **final**, użyte w deklaracji metody zabrania jej przedefiniowania w klasie pochodnej (dziedziczącej).

Rzutowanie obiektów, stwierdzanie typu

Rzutowanie *(ang. cast)* jest to operacja polegająca na zmianie zmiennej referncyjenj jednego typu na zmienna referencyjną innego typu. W Javie wyróżniamy dwa rodzaj rzutowania obiektów:

- **rzutowanie w górę** *(ang. upcasting)* – bezpieczne,
- **rzutowanie w dół** *(ang. downcasting)* – wymaga testowania (stwierdzenia typu instancji obiektu).

Stwierdzeniu jakiego typu jest referencja służy operator `instanceof`.

```
ref  instanceof  T
```

Przy tym:

- wyrażenie `null instanceof dowolny_typ` zawsze ma wartość **false**,
- wyrażenie `x instanceof T`, będzie błędne składniowo (wystąpi błąd w kompilacji), jeśli typ referencji `x` i typ `T` nie są związane stosunkiem **dziedziczenia**,

- wyrażenie `x instanceof T` będzie miało wartość **false**, jeśli faktyczny typ referencji `x` jest **nadtypem** typu `T`.

Przykład 4. Przykład rzutowania w górę

```
Car car = new Car();
Vehicle veh = (Vehicle) car;    //rzutowanie w gore Car -> Vehicle
veh.vehicleAttributes();
```

Przykład 5. Przykład rzutowania w dół

```
Vehicle veh = (Vehicle) new Car();
if (veh instanceof Car) {   //sprawdzenie przed rzutowaniem
  Car car = (Car) veh;      //rzutowanie w dol Vehicle -> Car
  car.carAttributes();
}
```

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/