Programowanie w języku Java - Polimorfizm (wielopostaciowość)

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

Version 1.0, 13-03-2017

Spis treści

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

Polimorfizm (wielopostaciowość)

Polimorfizm oznacza możliwość traktowania obiektów różnych podtypów pewnego wspólnego typu w taki sam sposób.

Termin *"polimorfizm"* nie jest jednoznaczny. Polimorfizm został wprowadzony w językach programowania obiektowego dla oznaczenia pewnej własności systemu typów. W obiektowości termin ten został zaadaptowany dla określenia możliwości **dynamicznego wyboru metody** *(wiązania metody)* wykonywanej na obiekcie po skierowaniu do niego komunikatu zawierającego nazwę tej metody. Na ogół te dwa znaczenia są różne i prawie niezależne.

W przypadku polimorfizmu wiązanie odwołań z kodem następuje w fazie wykonania, a nie w fazie kompilacji. Nazywa się to **dynamicznym wiązaniem** *(ang. dynamic binding)* lub **późnym wiązaniem** *(ang. late binding)*.

Przykład 1. Przykład demonstrujący działanie polimorfizmu

```
// Klasa Vehilce
public class Vehicle {
  //...
  public void start() {
    // metod uruchamiajaca pojazd
  }

  public void stop() {
    // metoda zatrzymujaca pojazd
  }

  public String toString() {
    // Implementacja metody toString()
  }
}

// Klasa Rower
public class Bicycle extends Vehicle {
  //...
  @Override
  public void start() {
    // metod uruchamiajaca Rower
  }

  @Override
  public void stop() {
    // metoda zatrzymujaca Rower
  }
}
```

```
// Klasa Motocykl
public class Motorcycle extends Vehicle {
  //...

  @Override
  public void start() {
    // metod uruchamiajaca Motocykl
  }

  @Override
  public void stop() {
    // metoda zatrzymujaca Motocykl
  }
}

// Klasa Samochód
public class Car extends Vehicle {
  //...
  @Override
  public void start() {
    // metod uruchamiajaca Samochód
  }

  @Override
  public void stop() {
    // metoda zatrzymujaca Samochód
  }
}

Vehicle[] veh = { new Bicycle(), new Motorcycle(), new Car() };

veh[0].start(); // zostanie wywołana metoda start() dla obiektu Bicycle
veh[1].start(); // zostanie wywołana metoda start() dla obiektu Motorcycle
veh[2].start(); // zostanie wywołana metoda start() dla obiektu Car
```

Kompilator nie wie jaki konkretnie jest **typ obiektu** wskazywanego przez `veh[i]` *(Bicycle, Motorcycle, Car)*. Metoda `start()` z klasy `Vehicle` jest **metodą wirtualną**, a dla takich metod wiązanie odwołań z kodem następuje w fazie wykonania, a nie w fazie kompilacji.

Z polimorfizmem związanych jest kilka bardzo istotnych pojęć:

- **metody wirtualne**,
- **przedefiniowanie (przesłanianie, nadpisywanie) metod**,
- **przeciążanie metod**.

Metody wirtualne

Metoda wirtualna *(funkcja wirtualna)* to metoda, której wywołanie jest polimorficzne. Wszystkie metody w Javie domyślnie są wirtualne, za wyjątkiem:

- **metod statycznych** (bo przecież nie dotyczą obiektów),
- **metod deklarowanych** ze specyfikatorem **final** (co oznacza, że postać metody jest ostateczna i nie może być ona przedefiniowana w klasie pochodnej, a jak nie ma przedefiniowania, to niepotrzebna jest wirtualność),
- **metod prywatnych** (do których odwołania z innych metodach danej klasy nie są polimorficzne, bo metody prywatne nie mogą być przedefiniowane).

Przedefiniowanie (przesłanianie, nadpisywanie) metod

Obiektowość dopuszcza sytuację, w której jakiś byt programistyczny będący własnością klasy podrzędnej ma tę samą nazwę jak byt klasy nadrzędnej. Pomimo identyczności nazw mogą to być byty o różnych implementacjach. Dotyczy to w szczególności metod. Taką sytuację określa się jako **przesłanianie** *(ang. overriding)*.

Przesłanianie dotyczy sytuacji, kiedy różne metody o tej samej nazwie np. `start()` znajdują się w klasach powiązanych **dziedziczeniem** (nadklasie i jej podklasie lub podklasach). Po wysłaniu do obiektu będącego członkiem tych klas komunikatu `start(⋯)` wybierana jest metoda `start()` znajdująca się najniżej w hierarchii tych klas. Mówi się wtedy, że metoda ta przesłania metodę dziedziczoną z nadklasy.

Metoda prywatna nigdy **nie może być przedefiniowana w podklasie**. Ponieważ metody prywatne nie są dziedziczone.

Więcej o przesłanianiu metod można znaleźć w poprzedniej instrukcji laboratoryjnej.

Przeciążanie metod

Pojęcie **przeciążania** *(ang. overloading)* jest podobne do przesłaniania, ale posiada nieco różną semantykę. Przy przeciążaniu na tym samym poziomie widoczności występują dwie (lub więcej) różne operacje o tej samej nazwie, lecz różnej sygnaturze.

Powszechne jest przeciążanie operatora równości `=`, który służy do porównania *liczb całkowitych, liczb rzeczywistych, stringów, identyfikatorów, struktur itd.* Podobnie, operator `+` może oznaczać **dodawanie** lub **konkatenację**.

W przypadku **metod** przeciążanie ma miejsce, gdy co najmniej dwie różne metody o tej samej nazwie, ale o różnych sygnaturach zdefiniowane są w tej samej klasie.

Przeciążanie metod polega na wywołaniu metod o tej samej nazwie, ale innej liczbie i/lub typy parametrów przyjmowanych oraz wartości zwracanej:

- po pierwsze, przeciążone metody mogą należeć do tej samej lub różnych klas (z których jedna pośrednio lub bezpośrednio dziedziczy inną),

- po drugie przeciążanie nie wyklucza przedefiniowania: jeśli np. w klasie A zdefiniowano dwie publiczne metody z tą samą nazwą (co oznacza, że są one przeciążone, bo sygnatury metod deklarowanych w jednej klasie muszą się różnić), to w klasie B dziedziczącej klasę A możemy jedną z nich dodatkowo przeciążyć (czyli podać w deklaracji inny zestaw parametów, a drugą przedefiniować - pozostawiając jej sygnatirę bez zmian i dostarczając innej definicji kodu metody). W tym względzie Java różni się od C++.

Metoda prywatna **nigdy nie może być przeciążona w podklasie**. Ponieważ metody prywatne nie są dziedziczone.

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/