Programowanie - Java

Jakub Mazurek
kuba@fenix.club

13 grudnia 2016

Spis treści

1 Programowanie obiektowe

Termin **programowanie obiektowe** to jeden z wielu tzw. paradygmatów programowania.
W podejściu obiektowym program komputerowy jest zbiorem wielu **obiektów** komunikujących się między sobą.

Każdy obiekt:

- Ma swój **stan**, wyrażany przy pomocy pól, które zawierają dane obiektu.
- Ma swoje **zachowanie**, czyli metody (funkcje) jakie zawiera.

Ze względu na swoje cechy, programowanie obiektowe bardzo dobrze sprawdza się przy modelowaniu naszej rzeczywistości.

2 Obiektowość w Javie

Java jest **językiem obiektowym**. Wszystkie typy jakie w niej napotkamy są **obiektami** (jedynymi wyjątkami sa **typy proste**, np. `int` lub `float`)[1].

2.1 Klasy

Klasa to jedno z najbardziej fundamentalnych pojęć w Javie. Jest ona definicją tego jak powinny wyglądać (co powinny zawierać w sobie) jej obiekty. Poniżej mamy przykład klasy w Javie:

```
1  public class MobilePhone {
2      public String deviceModel;
3      public String serialNumber;
4      public int productionYear;
5
6
7      public void call(String phoneNumber) {
8          // Some logic to call the provided phone number
9      }
10 }
```

Powyższa klasa opisuje nam (w dużym uproszczeniu) pewne cechy telefonu komórkowego. Przyjrzyjmy się jej teraz dokładniej:

- Pierwsza linijka to **definicja** klasy. W naszym przypadku tworzymy klasę o nazwie `MobilePhone`. Warto zaznaczyć, że nazwy klas w Javie muszą zaczynać się wielką literą.
- Linijki 2,3 oraz 4 zawierają definicje zmiennych, które będą **polami** klasy `MobilePhone`. To one będą opisywały **stan** obiektów tej klasy.
- W linijce nr 7 mamy deklarację **metody**. Metoda ta nosi nazwę `call` i przyjmuje jeden argument typu `String`. Za pomocą metod definiujemy co obiekty naszej klasy są w stanie zrobić (jakie jest ich **zachowanie**).

[1] W Javie istnieją klasy (np. `Integer`), które nadają typom prostym cechy obiektów.

2.2 Tworzenie obiektów

Po zdefiniowaniu klasy w Javie można tworzyć jej **obiekty** (instancje). Oto jak możemy stworzyć obiekt klasy `MobilePhone` z wcześniejszego przykładu:

```
MobilePhone myPhone = new MobilePhone();
```

Po powyższym wywołaniu zmienna pod nazwą `myPhone` będzie odnosiła się do nowego obiektu klasy `MobilePhone`.
Oznacza to, że nasz obiekt będzie miał pola oraz metody, które zdefiniowaliśmy w jego klasie:

```
1 MobilePhone myPhone = new MobilePhone();
2
3 System.out.println("Device model: " + myPhone.deviceModel);
4
5 myPhone.call("+48606123456");
```

Powstaje teraz pytanie: jakie wartości będą miały pola naszego obiektu?

Wszystkie pola będą miały odpowiadające swoim typom wartości domyślne. W naszym przypadku:

```
1 System.out.println(myPhone.deviceModel); // null
2 System.out.println(myPhone.serialNumber); // null
3 System.out.println(myPhone.productionYear); // 0
```

Wynika to z faktu, że przy tworzeniu naszego obiektu skorzystaliśmy z domyślnego **konstruktora**.

2.3 Konstruktory

Konstruktorem nazywamy specjalną metodę klasy, która służy do tworzenia nowych obiektów tej klasy.
Każda klasa posiada przynajmniej jeden konstruktor, nawet jeśli nie jest on zdefiniowany bezpośrednio w kodzie. W takim przypadku (tak jak ma to miejsce w klasie `MobilePhone`) Java dodaje do klasy tzw. **domyślny konstruktor**.
Konstruktory to metody klasy o specjalnej sygnaturze - nie mają one określonego zwracanego typu, a ich nazwa musi być dokładnie taka sama jak nazwa klasy:

```java
public class MobilePhone {
    public String deviceModel;
    public String serialNumber;
    public int productionYear;

    // Konstruktor bezargumentowy
    public MobilePhone() {
        deviceModel = "A0003T";
        serialNumber = "XDA-123-010";
        productionYear = 2016;
    }

    // Metoda
    public void call(String phoneNumber) {
        // Some logic to call the provided phone number
    }
}
```

Po dodaniu naszego konstruktora spróbujmy stworzyć obiekt jeszcze raz:

```
MobilePhone myPhone = new MobilePhone();

System.out.println(myPhone.deviceModel); // A0003T
System.out.println(myPhone.serialNumber); // XDA-123-010
System.out.println(myPhone.productionYear); // 2016
```

Tym razem wartości pól obiektu `myPhone` są inne i pokrywają się z wartościami, których użyliśmy w naszym konstruktorze.
Konstruktory mogą również przyjmować argumenty:

```
public class MobilePhone {
    public String deviceModel;
    public String serialNumber;
    public int productionYear;

    // Konstruktor z argumentami
    public MobilePhone(String model, String serial, int production) {
        deviceModel = model;
        serialNumber = serial;
        productionYear = production;
    }
}
```

3 Modyfikatory dostępu

W Javie istnieją cztery możliwe zakresy dostępu:

1. **`public`** - dany element jest zawsze widoczny, również spoza swojej paczki.
2. **`protected`** - dany element jest widoczny dla klas ze swojej paczki oraz dla klas dziedziczących.
3. **`package private`** - modyfikator domyślny (pusty), dany element jest widoczny tylko dla klas ze swojej paczki.
4. **`private`** - dany element jest widoczny jedynie w obrębie swojej klasy.

Pojęcie modyfikatora dostępu jest ściśle związanie z **paczkami (*packages*)** w Javie. Są one sposobem organizacji kodu źródłowego na mniejsze zbiory. W praktyce, paczka jest po prostu katalogiem o odpowiedniej nazwie istniejącym w naszym systemie plików. Modyfikatory dostępu nie mają związku z zabezpieczaniem kodu. Ich rolą jest zapewnianie dostępu do pól i metod klasy w odpowiedni sposób.

4 Zadania

1. Przenieś klasę `MobilePhone` do osobnego pliku w swoim projekcie.
2. W swoim projekcie stwórz paczki o nazwach `logic` i `main`, a następnie przenieś do odpowiednich paczek klasy `MobilePhone` i `Program` (zawierającą metodę `main`).
3. Korzystając z klasy `MobilePhone`, dodaj do tej klasy metodę o nazwie `getProductionYear`, która zwróci wartość pola `productionYear`.
4. Dodaj do klasy `MobilePhone` dwa publiczne konstruktory. Jeden z nich powinien nie przyjmować żadnych argumentów i nadawać polom domyślne wartości. Drugi powinien przyjmować trzy wartości i nadawać je polom swojej klasy.
5. W klasie `MobilePhone`, zmień dostęp do pola `productionYear` na `private`. Następnie dodaj metodę `setProductionYear` w której sprawdzisz poprawność podanego argumentu i jeśli będzie poprawny - zmienisz wartość pola klasy.