
# Programowanie - Java

Jakub Mazurek
kuba@fenix.club

24 listopada 2016

## Spis treści

# 1 Dziedziczenie w Javie

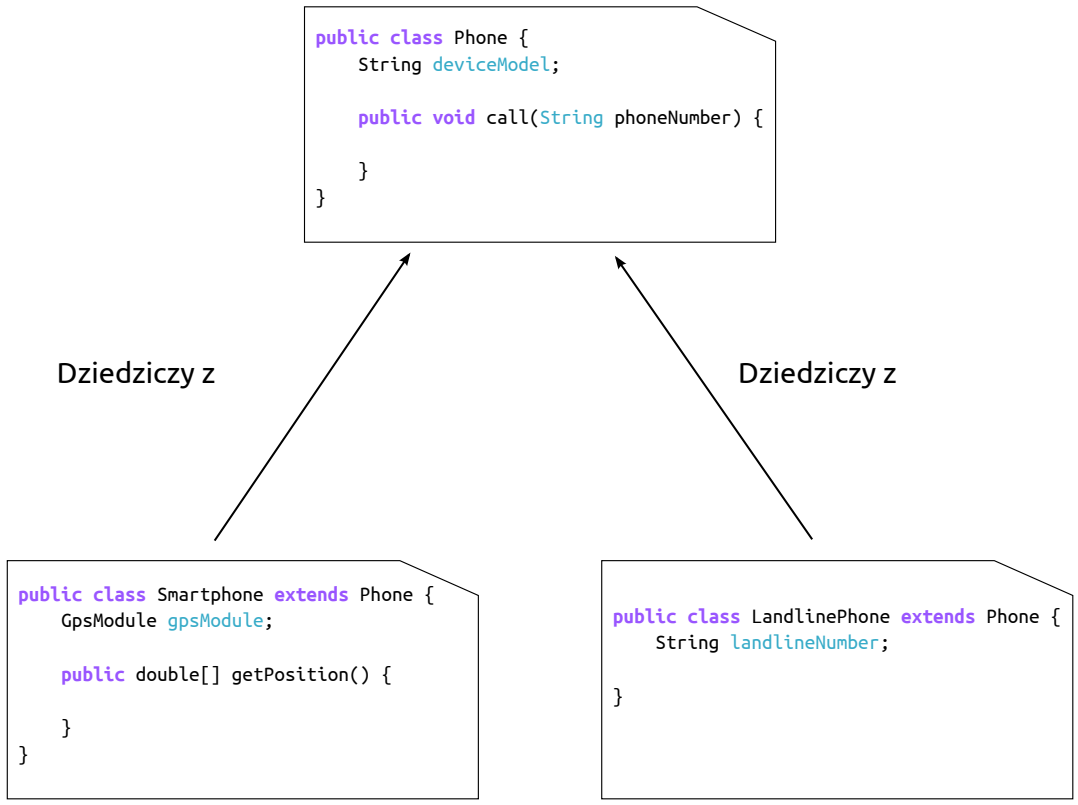
**Dziedziczenie** (ang. *inheritance*) jest jednym z najważniejszych aspektów programowania obiektowego. Mamy z nim do czynienia wtedy, kiedy nowa klasa **nie jest** budowana od zera, ale tworzy się ją na podstawie już istniejącej.
W takiej sytuacji mówimy, że nowa klasa **dziedziczy po** swojej **nadklasie** (ang. *superclass*).

W Javie dziedziczenie implementuje się za pomocą słowa kluczowego **`extends`** (ang. *to extend* - rozszerzać, przedłużać). Za pomocą tego słowa kluczowego wskazujemy na klasę, z której chcemy dziedziczyć. Przykładowo:

```
public class MobilePhone extends Phone
```

W powyższym przykładzie widzimy definicję klasy o nazwie `MobilePhone`, która dziedziczy po klasie `Phone`.

W przeciwieństwie do np. `C++`, Java nie pozwala nam na dziedziczenie z wielu klas (ang. *multi-inheritance*). Klasa w Javie może mieć **tylko jedną** nadklasę (rodzica).




Rysunek 1: Przykładowa hierarchia dziedziczenia

Dlaczego dziedziczenie jest przydatne?

- Możliwość stosowania **polimorfizmu**.
- Mniejsza ilość duplikowanego lub bardzo podobnego kodu w wielu miejscach.
- Modelowanie rzeczywistości w intuicyjny sposób.
- Lepsza czytelność i organizacja kodu źródłowego.

## 1.1 Zadania

1. Korzystając z projektu z ostatnich zajęć, stwórz w nim (w paczce `logic`) klasę o nazwie `Phone`.
2. Wybierz te pola, metody i konstruktory klasy `MobilePhone`, które według Ciebie powinny być wspólnymi cechami dowolnego telefonu. Przenieś te elementy do klasy `Phone`.
3. Zmień klasę `MobilePhone` w taki sposób, aby dziedziczyła po klasie `Phone`. Przetestuj swoje rozwiązanie w metodzie `main` Twojego programu.

# 2 Klasy abstrakcyjne

Poznaliśmy już nieco lepiej mechanizm dziedziczenia w języku Java. Przejdziemy teraz do omówienia specjalnego rodzaju klas w Javie - **klas abstrakcyjnych**.
Klasy abstrakcyjne definiuje się przy pomocy słowa kluczowego **`abstract`**:

```
public abstract class Phone
```

Klasy abstrakcyjne mają kilka unikalnych cech:

- Tworzenie nowych obiektów klasy abstrakcyjnej jest **niemożliwe**.
- Tylko klasy abstrakcyjne (i interfejsy) mogą zawierać w sobie **metody abstrakcyjne**.
- Klasy dziedziczące po klasie abstrakcyjnej albo implementują wszystkie jej metody abstrakcyjne, albo same są abstrakcyjne.

**Metody abstrakcyjne** to metody, które są zadeklarowane, ale nie mają (jeszcze) implementacji.
Do czego służą klasy abstrakcyjne i dlaczego są przydatne?

Ich głównym zastosowaniem w praktyce jest określanie funkcjonalności i cech, które będą wspólne dla grupy klas dziedziczących po klasie abstrakcyjnej.
Można je rozumieć jako swojego rodzaju szkielet klasy.

## 2.1 Zadania

1. Korzystając z kodu zadań poprzedniej sekcji, zmień klasę `Phone` tak, aby była klasą abstrakcyjną.
2. Które z metod klasy `Phone` powinny stać się abstrakcyjne, a które dalej mieć implementację? Wybierz i dokonaj odpowiednich zmian w kodzie.
3. Dostosuj klasę `MobilePhone` do zmian z poprzedniego punktu.

# 3 Interfejsy

**Interfejsy** w Javie można nazwać okrojonymi klasami abstrakcyjnymi. Mają one jednak wiele ważnych zastosowań i będziemy się z nimi często spotykać.

Głównym zastosowaniem interfejsów jest definiowanie możliwych do wykonania akcji. Analogią do rzeczywistości są ogólnie znane nam **interfejsy użytkownika**, np. interfejs kontrolera konsoli. Składa się on z kilku przycisków - jako użytkownicy kontrolera wchodzimy w interakcję jedynie z tymi przyciskami i nie musimy wiedzieć, jak taki kontroler działa w środku.
Interfejsy mają swoje słowo kluczowe **`interface`** i są definiowane w poniższy sposób:

```
public interface GamePad {

    void pressA();
    void pressB();

    void turnAnalog(double angle);
}
```

Mogą one zawierać wyłącznie stałe pola oraz deklaracje metod (które domyślnie są abstrakcyjne i publiczne).

Klasa może implementować dowolnie wiele interfejsów. W poniższym przykładzie rozbijemy funkcjonalności składanego roweru na bardziej abstrakcyjne interfejsy:

```
public interface Rideable {

    void ride(double acceleration);
    void stop();
    void turn(double angle);
}
```

Powyższy interfejs opisuje w abstrakcyjny sposób akcje związane z jazdą prostym pojazdem naziemnym. Nazwa naszego interfejsu zgadza się z często stosowaną konwencją w której nazwy interfejsów mają sufiks ***-able***.

Drugą z naszych funkcjonalności możemy opisać innym interfejsem, który będzie zawierał jedną metodę:

```
public interface Foldable {

    void fold();
}
```

Możemy teraz skorzystać z naszych interfejsów w konkretnej klasie:

```java
public class FoldableBike implements Rideable, Foldable {

    @Override
    public void fold() {

    }

    @Override
    public void ride(double acceleration) {

    }

    @Override
    public void stop() {

    }

    @Override
    public void turn(double angle) {

    }
}
```

## 3.1 Zadania

1. Wybierz dwie z cech telefonu komórkowego (np. dzwonienie i pobieranie fizycznej lokalizacji). Stwórz dwa oddzielne interfejsy, które będą reprezentowały dwie wybrane przez Ciebie cechy.

2. Zaimplementuj obydwa interfejsy w odpowiedniej klasie z zajęć.