Orkiestra

Wiele pojęć z rzeczywistego świata jesteśmy w stanie przenieść bezpośrednio na język obiektowo-orientowany. Rozważmy na przykład kilka instrumentów muzycznych, np. gitarę, flet i puzon. Każdy z nich wygląda i działa inaczej, ale wszytkie posiadają jedną istotną cechę - można na nich grać! Powiemy zatem, że:

- Gitara jest Instrumentem i można na niej grać.
- Flet jest Instrumentem i można na nim grać.
- Puzon jest Instrumentem i można na nim grać.

Najczęściej słowo **jest** określa nam relację między typami danych, która odpowiada dziedziczeniu. Widzimy więc, że w tym przypadku Instrument będzie pełnił rolę klasy bazowej... ale czy na pewno klasy? Czy możemy powiedzieć, czym jest i w jaki sposób zachowuje się po prostu "instrument"? Wiemy jedynie, że może grać, ale nie wiemy w jaki sposób. W takiej sytuacji powinniśmy posłużyć się **interfejsem**.

Napisz deklaracje dla wszystkich wymaganych typów i przetestuj ich działanie:

1. Stwórz interfejs `Instrument` i zadeklaruj w nim metodę `play()`.
2. Stwórz klasy `Guitar`, `Flute` i `Trombone`, które realizują interfejs `Instrument` i definiują metodę `play()`
3. Stwórz klasę `Orchestra`, która:
    - również realizuje interfejs `Instrument`
    - posiada atrybut `instruments`, który jest listą obiektów typu `Instrument`
    - posiada metodę `addInstrument(Instrument)`, która dodaje do orkiestry nowy instrument
    - definiuje metodę `play()`, w której gra na wszystkich instrumentach - wołając w pętli na każdym z nich `play()`.
4. Dodaj metodę `main`, w której utworzysz instancje wszystkich rodzajów instrumentów oraz jeden obiekt orkiestry, do której dodasz utworzone instrumenty. Następnie wywołaj na orkiestrze `play()`. Co obserwsujesz? Jaką rolę w realizacji zadania pełni polimorfizm?

Przydatne informacje

Interfejsy

Interfejs to specjalny typ obiektowy, który nie posiada atrybutów ani definicji metod - może jedynie deklarować (publiczne) nagłówki metod, a więc "określać możliwe zachowania" danego typu. Intefejsy nie posiadają też konstruktorów więc nie można utworzyć obiektów takiego typu (nie miałoby to sensu).

W programowaniu obiektowym interfejsy służą najczęściej do pokazania możliwych do wykonania operacji bez udostępniania ich realizacji.

Przekładają się też naturalnie na różne abstrakcyjne pojęcia. Wiemy na przykład, jak działa Samochód, w jaki sposób się porusza. Wiemy również, jak porusza się Rower. Zarówno Samochód, jak i Rower są Pojazdami. Pojazdy generalnie potrafią się poruszać... ale jak? To zależy właśnie od rodzaju pojazdu. Nie wiedząc, z jakim Pojazdem mamy do czynienia nie jesteśmy w stanie nic na ten temat powiedzieć.

W naszym przykładzie interfejsem będzie oczywiście Pojazd. W Javie zdefiniujemy go w następujący sposób:

```
public interface Vehicle {
    void drive();
}
```

Definicja wygląda bardzo podobnie jak klasa, ale różni się słowem kluczowym (`interface` zamiast `class`), metody nie mają modyfikatora dostępu (domyślnie są `public`) i przede wszystkim nie posiadają ciała - tuż po sygnaturze metody występuje średnik.

Sam interfejs nie ma większego znaczenia, dopóki nie stworzymy dla niego choć jednej realizacji:

```
public class Car implements Vehicle {
    public void drive() {
       // tu odpalamy silnik i jedziemy
    }
}

public class Bike implements Vehicle {
    public void drive() {
       // tu pedałujemy!
    }
}
```

Realizacja interfejsu wygląda bardzo podobnie jak dziedziczenie - inne jest słowo kluczowe (`implements` zamiast `extends`) i tym razem **musimy** zdefiniować metody zadeklarowane w interfejsie, inaczej program się nie skompiluje.

Interfejsy mogą się również wzajemnie rozszerzać (dziedziczyć - słowo kluczowe `extends`). Od Javy 8 możliwe jest również częściowe definiowanie zachowań w interfejsach, ale nie będziemy tego szczegółowo omawiać.