Typ wyliczeniowy (enum)

Podczas programowania bardzo często pojawia się potrzeba stworzenia zmiennej/pola/parametru metody, który powinien przyjmować tylko jedną z kilku z góry określonych wartości.

- Rozmiary ubrań - XS,S,M,L,XL,XXL
- Typ karty płatniczej - CREDIT,DEBIT
- Typ napędu samochodu - DIESEL,GASOLINE,HYBRID,LPG,EV

Oczywiście w takiej sytuacji można użyć typu tekstowego String i zapisywać poszczególne wartości jako zwyczajne napisy, jednak takie podejście ma szereg wad:

- brak zdefiniowanej liczby dopuszczalnych wartości
- bez wnikania w dokumentację/komentarze programista używający parametru nie ma pojęcia, że nie powinno się podawać jako wartości dowolnego napisu
- wrażliwość na literówki, wielkość liter - jeśli programista lub użytkownik programu zrobi literówkę (np. DEISEL zamiast DIESEL) to kod używający tego parametru nie rozpozna poprawnie tej wartości

```
public class CarApp {
    public static void main(String[] args) {
        refuel("GASOLINE");
        refuel("DIESEL");
        refuel("DEISEL");
    }

    public static void refuel(String carType){
        switch (carType){
            case "DIESEL":
                System.out.println("Tankuję olej napędowy");
                break;
            case "GASOLINE":
                System.out.println("Tankuję benzynę");
                break;
            default:
                System.out.println("Nie rozpoznano paliwa :(");
        }
    }
}
```

Przed wprowadzeniem do programowania omawianych dzisiaj typów wyliczeniowych bardzo często ten problem był "rozwiązywany" poprzez tworzenie stałych z dopuszczalnymi wartościami (zazwyczaj w osobnym interfejsie) - najczęściej typu String lub int.

```
public class CarApp {
    public static void main(String[] args) {
        refuel(CarType.GASOLINE);
        refuel(CarType.DIESEL);
        refuel(CarType.GASOLINE);
    }

    public static void refuel(int carType){
```

```
        switch (carType){
            case CarType.DIESEL:
                System.out.println("Tankuję olej napędowy");
                break;
            case CarType.GASOLINE:
                System.out.println("Tankuję benzynę");
                break;
            default:
                System.out.println("Nie rozpoznano paliwa :(");
        }
    }
}

// plik CarType.java
public interface CarType {
    public static final int GASOLINE = 0;
    public static final int DIESEL = 1;
    public static final int EV = 2;
}
```

Powyższe podejście jest o wiele wygodniejsze niż pierwsza wersja, jednak niestety rozwiązuje tylko część problemów. Kod taki czyta się oczywiście o wiele lepiej, jednak ciągle nie mamy pewności, że do metody nie trafi niepoprawna wartość i programista, który napotka metodę void refuel(int carType) po raz pierwszy ciągle musi na własną rękę poszukać, czy przypadkiem gdzieś nie istnieją stałe przeznaczone do użytku z nią.

Typy wyliczeniowe (enumy)

Typ wyliczeniowy - enum (enum - skrót od enumerated (wyliczony)) - mechanizm wprowadzony w Javie 1.5 (2004r.), pozwalający na stworzenie nowego typu danych (podobnie jak klasa), który może przyjmować tylko jedną z określonych wartości.

Enumy tworzy się praktycznie identycznie jak klasy - jedyną różnicą jest zamiana słowa class na enum. Ciało typu wyliczeniowego powinno zawierać nazwy wartości, jakie może reprezentować (oddzielone przecinkami).

```
public class CarApp {
    public static void main(String[] args) {
        refuel(CarType.GASOLINE);
        refuel(CarType.DIESEL);
        refuel(CarType.GASOLINE);
    }

    public static void refuel(CarType carType){
        switch (carType){
            case DIESEL:
                System.out.println("Tankuję olej napędowy");
                break;
            case GASOLINE:
                System.out.println("Tankuję benzynę");
                break;
            default:
                System.out.println("Nie rozpoznano paliwa :(");
        }
```

```
    }
}

// plik CarType.java
enum CarType {
    GASOLINE,
    DIESEL,
    GASOLINE_LPG,
    EV
}
```

Zwróć uwagę, że gdy używasz switcha z parametrem typu enum każdy case powinien zawierać wyłącznie nazwę dopuszczalnej wartości (DIESEL zamiast CarType.DIESEL). Dodatkowo switch nie obsługuje nigdy wartości null!

Kiedy używać enumów?

Typy wyliczeniowe powinny być użyte, gdy jesteśmy praktycznie pewni, że zbiór wartości się nie zmieni lub będzie się zmieniał bardzo rzadko. Jest to mechanizm bardzo wygodny i bezpieczny, jednak dodanie nowej wspieranej wartości do enuma wymaga modyfikacji kodu i przekompilowania całego projektu oraz uruchomienie nowej wersji aplikacji, co jest dość dużą wadą!

Przykładowe typy danych przy tworzeniu modelu klienta:

CUSTOMER

- firstName: String
- lastName: String
- city: String (!) (chyba, że Twoja aplikacja wspiera i ma w przyszłości wspierać np. tylko 5 miast +- kilka)
- role: enum (USER, ADMIN, MODERATOR)
- maritalStatus: enum (MARRIED,SINGLE)
- activated: boolean

Zauważ, że teoretycznie każdą wartość typu boolean można by zastąpić odpowiednim enumem (np. activated: enum (ENABLED, NOT_ENABLED)). Kiedy zatem najlepiej to zrobić? - gdy wartość true/false w przypadku danej zmiennej nie mówi nam za wiele. Przykładowo jeśli produkt występuje tylko w dwóch rozmiarach (S,M), moglibyśmy zapisać to w postaci medium: boolean, jednak przyjęcie, że medium==false oznacza small jest oczywiste wyłącznie dla programisty piszącego tę klasę w momencie tworzenia, dla każdego innego programisty czytelniej będzie, jeśli zapiszemy to jako size: enum (SMALL,MEDIUM).

Metody dostępne w enumach

Każdy zdefiniowany przez nas enum posiada dwie często przydatne metody:

- values() - zwraca w postaci tablicy wszystkie dostępne wartości danego enuma
- valueOf(String text) - parsuje enuma ze Stringa, tj. zwraca wartość typu dany enum na podstawie przekazanego napisu