Typy generyczne

Czego się dowiesz

- Czym są typy i metody generyczne,
- jak zdefiniować typ lub metodę generyczną,
- do czego się je wykorzystuje w praktyce.

Wstęp

Możliwe, że już kilkukrotnie miałeś wrażenie, że po raz kolejny piszesz kod bardzo podobny do tego, który pisałeś kilka lekcji wcześniej. Sytuacja taka mogła nastąpić, gdy definiujemy jakąś klasę gromadzącą informacje, czy to o osobach, czy też książkach w jakiejś klasie, w której znajduje się tablica (klasy Library, Company, etc.).

Problemem tych klas było to, że były one zdefiniowane w taki sposób, że nijak nie dało się ich wykorzystać w innych sytuacjach, ponieważ zdefiniowane były na konkretnych typach - czyli np. potrafiły przechowywać obiekty klasy `Person` lub `Publication`, ale nie jednych i drugich.

Teoretycznie dałoby się napisać ogólną klasę-kontener, w której zdefiniowalibyśmy tablicę typu `Object` i skorzystali z polimorfizmu:

Container.java

```
class Container {
    private Object[] array;

    public Object[] getArray() {
        return array;
    }

    public void setArray(Object[] array) {
        this.array = array;
```

```
    }

    public void printObjects() {
        for (Object o : array) {
            System.out.println(o);
        }
    }
}
```

Jednak pomimo wysokiego poziomu ogólności takiej klasy sprawia ona dużo problemów, związanych np. z ciągłą koniecznością rzutowania obiektów, braku informacji o faktycznym typie obiektu.

W Javie w wersji 5 wprowadzono jedną z największych zmian od początku istnienia tego języka, a mianowicie **typy i metody generyczne (*eng. generics*)**.

Typy generyczne

Typami generycznymi (zwanymi też sparametryzowanymi) nazywamy takie klasy lub interfejsy, w których pewne rzeczy możemy określić dopiero podczas tworzenia obiektów danego typu lub ich rozszerzania / implementowania.

Podstawowa struktura klasy generycznej wygląda następująco:

```
class SomeClass<T>{
  //pola i metody
}
```

W ostrych nawiasach na poziomie sygnatury klasy podany jest parametr `<T>`, który może być wykorzystywany do definicji typów zmiennych, czy tablic. W miejsce T możesz wstawić także więcej różnych parametrów, wymieniając je po przecinku, np. `<T, K, V>`.

Przeróbmy naszą poprzednią klasę Container, na klasę generyczną.

Container.java

```
class Container<T> {
    private T[] array;

    public T[] getArray() {
        return array;
    }

    public void setArray(T[] array) {
        this.array = array;
    }

    public T get(int index) {
        return array[index];
    }

    public void printObjects() {
        for (T o : array) {
            System.out.println(o);
        }
    }
}
```

Pozbyliśmy się już informacji o tym, ze przechowujemy dane w tablicy typu Object. W zamian za to mamy tablicę typu **T** - czyli dowolnego typu, który podamy przy tworzeniu obiektu klasy Container. Dodatkowo dodaliśmy metodę get(), która zwraca element z tablicy o indeksie index. Co ważne metoda ta także zwraca "coś" typu T.

GenericTester.java

```
class GenericTester {
```

```
    public static void main(String[] args) {
        // definiujemy kontener przechowujący liczby całkowite
        Container<Integer> integers = new Container<Integer>();
        // przypisujemy nową tablicę typu Integer
        integers.setArray(new Integer[5]);
        // do pierwszego elementu przypisujemy liczbę 5
        integers.getArray()[0] = 5;

        // tworzymy kontener przechowujący obiekty String
        Container<String> strings = new Container<String>();
        // przypisujemy tablicę typu String
        strings.setArray(new String[10]);
        // przypisujemy 1 i 2 element tablicy
        strings.getArray()[0] = "Ania";
        strings.getArray()[1] = "Basia";

        // odczytanie danych bez konieczności rzutowania!
        Integer num = integers.get(0);
        String str = strings.get(0);

        // wyświetlenie wartości
        System.out.println(num);
        System.out.println(str);
    }
}
```

Na początku zdefiniowaliśmy obiekt `Container`, parametryzując go typem `Integer`. **Do typów generycznych można używać tylko typów obiektowych**. Jest to jedno z miejsc, gdzie wymagane jest korzystanie z klas opakowujących dla typów prostych.

Zauważ, że ponieważ w nawiasach ostrych podaliśmy typ `Integer`, to od tego momentu w klasie `Container` posiadamy pustą referencję typu `Integer[]`, do której za pomocą metody `setArray()` przypisujemy

nową 5 elementową tablicę liczb całkowitych, a dalej do jej 1 elementu liczbę 5 (możliwe dzięki autoboxingowi). W skrócie wszędzie tam, gdzie w klasie Container pojawiał się parametr T, od teraz wyobrażamy sobie w tych miejscach Integer.

W dalszej części kodu robimy analogiczne czynności, tym razem parametryzując typ Container za pomocą typu <String>.

Metoda *get()* pokazuje całe piękno typu generycznego - nie jest wymagane żadne rzutowanie z typu Object na Integer, czy String, bo my dokładnie wiemy, że obiekt integers przechowuje wartości Integer, a strings napisy typu String.

Metody generyczne

Generyczne mogą być nie tylko typy / klasy, ale również pojedyncze metody. Załóżmy dosyć abstrakcyjną sytuację, że chcemy napisać metodę, która wyświetla 5 razy przekazany argument. Nie wiemy jednak jakiego dokładnie typu wartości chcemy wyświetlić - użyjemy więc metody generycznej.

GenericMethods.java

```
class GenericMethods {
    public static void main(String[] args) {
        print5Times(2);
        print5Times("Krzysio");
    }

    public static <T> void print5Times(T arg) {
        for (int i = 0; i < 5; i++)
            System.out.println(arg);
    }
}
```

Metoda print5Times() przyjmuje argument dowolnego typu obiektowego, a następnie wyświetla go 5 razy. Zauważ, że najpierw musieliśmy jednak zadeklarować w nawiasach ostrych w definicji metody (przed typem zwracanym), że jest to metoda generyczna.

W metodzie main() możemy ją teraz wywołać z dowolnym typem standardowym lub zdefiniowanym przez siebie. Pamiętaj, że w powyższym przykładzie przekazując wartość 2 dochodzi do autoboxingu i w rzeczywistości przekazywany jest tam obiekt typu Integer.

Kiedy wykorzystuje się typy i metody generyczne

W praktyce z typami generycznymi będziesz spotykać się na każdym kroku przy okazji korzystania z kolekcji, które już niebawem ci przedstawię. W przypadku, gdy sam tworzysz kod, typy i metody generyczne wykorzystuj przede wszystkim tam, gdzie liczy się dla ciebie uniwersalność. Zamiast tworzyć przykładowo metodę w 10 przeciążonych wersjach różnych typów, która oprócz przyjmowania parametrów innych typów robi dokładnie to samo, możesz uzyskać ten sam efekt, definiując jedną metodę uogólnioną.

Typy generyczne to złożone i skomplikowane zagadnienie, jednak podstawy, które przedstawiłem w tej lekcji powinny ci wystarczyć do zrozumienia kolejnych lekcji.