UNIWERSYTETY WSB MERITO

Programowanie aplikacji Java

Maciej Gowin

Zjazd 3 - zadania dodatkowe

Linki

Opis

https://maciejgowin.github.io/wsb-java/

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java

Zadania dodatkowe

Wszystkie zadanie dodatkowe zostały stworzone z myślą utrwalenia materiału przerobionego na zjeździe. Chociaż mogą one zostać rozwiązane na wiele sposobów, zostały skonstruowane w sposób umożliwiający ukończenie ich jedynie przy użyciu poznanych już technik.

Programowanie: zadanie dodatkowe 13

Zaimplementuj klasy `Route` oraz `Airport`:

```
public class Airport {
    private final String code;
    (...)
```

```
public class Route {
    private final Airport departure;
    private final Airport arrival;
    (...)
```

W programie głównym stwórz kolekcję dostępnych połączeń pomiędzy lotniskami. Przekonwertuj listę oraz wypisz wszystkie możliwe połączenia z danego lotniska w formacie:

```
Departure: Wroclaw: arrivals: [Dublin, Krakow]
Departure: Krakow: arrivals: [Wroclaw, Dublin]
Departure: Warszawa: arrivals: [Krakow]
```

Programowanie: zadanie dodatkowe 14

Stwórz adnotację pozwalającą na porównywanie dwóch pól danego obiektu wraz z deklaracją pól porównywanych:

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
public @interface FieldEquals {
    String firstField();
    String secondField();
}
```

Zaimplementuj walidator sprawdzający, czy dany obiekt, którego klasa jest oznaczona adnotacją jest poprawny:

```
public class FieldEqualsValidator {

    public static boolean validate(Object object) throws Exception {
        /* ... */
    }
}
```

Programowanie: zadanie dodatkowe 15

Zaimplementuj dwa funkcjonały, które:

`String upperCase(String value)` - konwertuje ciąg znaków do ciągu znaków z wielkimi literami

`String letterSpaceSeparated(String value)` - konwertuje ciąg znaków, dodając spację pomiędzy każdy znak.

Przetestuj konwersję kilku ciągu znaków przy użyciu strumieni i zdefiniowanych funkcjonałów.

Programowanie: zadanie dodatkowe 16

Zaimplementuj strumień konwertujący mapę cen dla produktów w mapę średnich cen dla każdego z produktów. Dla przykładu:

```
Prices:
{Apple=[1.1, 2.3], Banana=[2.5, 4.6]}
Average prices:
{Apple=1.7, Banana=3.55}
```

Programowanie: zadanie dodatkowe 17

Zaimplementuj własny kolektor grupujący imiona na podstawie ich pierwszych liter niezależnie od wielkości pierwszej litery. Użyj pełnej definicji `Collector`:

```
List<String> names = List.of("gary", "George", "Anna", "", "");
Map<String, List<String>> customByFirstLetter = names.stream()
    .collect(new Collector<String, Map<String, List<String>>, Map<String, List<String>>>() {
        /* ... */
    });
```

Programowanie: zadanie dodatkowe

Do listy imion i nazwisk oddzielonych białymi znakami zdefiniuj strumień konwertujący ciąg do ciągu posortowanych inicjałów z wielkich liter.

```
Wejście: ["Maciej Gowin", "andrzej   nowak", "  jan kowalski"]
Wyjście: ["MG", "AN", "JK"]
```

Programowanie: zadanie dodatkowe

Zdefiniuj kolektor (lub użyj istniejącego) zliczający ilość wystąpień danego słowa w liście.

```
Wejście: ["ccc", "aaa", "bbb", "ccc", "aaa", "ccc"]
Wyjście: [{"aaa": 2}, {"bbb": 1}, {"ccc": 3}]
```

Programowanie: zadanie dodatkowe

Dla listy obiektów klasy `Rectangle` posiadających metodę definiującą pole powierzchni `int getArea()` oraz używając strumieni oblicz całkowitą powierzchnię figur w liście.

Programowanie: zadanie dodatkowe

Zamodeluj strukturę grafu przy pomocy własnych klas oraz przedstawionych wbudowanych struktur danych.

Programowanie: zadanie dodatkowe: future-year-validator

Stwórz adnotację @FutureYear działającą na typie Integer. Utwórz walidator sprawdzający czy dany rok jest w przyszłości na podstawie pól oznaczonych zadaną adnotacją.

Do sprawdzenia obecnego roku użyj `Year.now()`.