Notatki - Resource Bundle

Spis treści

Resource Bundle

Wspomnieliśmy o **i18n** oraz **l10n** i pojawia się pytanie, po co nam to? Pokażmy przykład użycia tego w praktyce - **Resource Bundle** ☺.

`Resource Bundle` służy do zapisu konkretnych ustawień aplikacji i użycia ich w zależności od podanego `Locale`. Resource Bundle może być przedstawione za pomocą plików zwanych `property file` albo ustawienia mogą też być definiowane w plikach `.java`.

Na co dzień zamiast `property file` używa się stwierdzenia **plik z propertasami** ☺.

Plik taki jest zdefiniowany na zasadzie Mapy/Słownika - `klucz=wartość`

```
property1=someValue1
property2=someValue2
```

Dotychczas, jeżeli pisalibyśmy jakiś program, który używa Stringów, to Stringi były pisane od razu w kodzie po polsku. Teraz dowiemy się w jaki sposób można napisać aplikację, gdzie Stringi mogą zostać wyciągnięte "poza" aplikację, żeby mieć je w oddzielnym pliku i jak będziemy potrzebowali dodać kolejny język do aplikacji to dodajemy kolejny plik z językiem. Ale przechodząc do konkretów.

Sklep Internetowy!

Przejdźmy do przykładu sklepu internetowego, który ma działać i w języku polskim i angielskim.

Tutaj chcę jeszcze dodać, że nie będziemy się zagłębiać w różne sposoby zapisu danych w plikach `.properties`. Dodam tylko, że można tam pisać komentarze zaczynając linijkę od `#` lub `!`

Pliki `.properties` nie są jedynym możliwym sposobem na zdefiniowanie ustawień/tłumaczeń. Możliwe jest to również do osiągnięcia w samych plikach `.java`. Nie pokazujemy jednak tego sposobu, ze względu na raczej mało popularne jego zastosowanie.

Przykładowe pliki `.properties`:

Plik Store.properties

```
defaultProperty=Default Value
```

Plik Store_en.properties

```
mainPage=Main Page!
footer=And here we have our footer
```

Plik Store_pl.properties

```
mainPage=Strona Główna!
footer=A oto jest stopka
```

Wartość domyślna

Mając już określone pliki z ustawieniami jak w przykładzie powyżej, możemy przejść do napisania kodu, który odczyta te wartości z plików. W tym celu wykorzystamy klasę `ResourceBundle`:

```
public class InternetStoreExample {

    public static void main(String[] args) {
        printValuesForDefault();
    }

    private static void printValuesForDefault() {
        System.out.println("locale: " + Locale.getDefault());

        ResourceBundle resourceBundle = ResourceBundle.getBundle("Store");
        String mainPage = resourceBundle.getString("mainPage");
        String footer = resourceBundle.getString("footer");

        System.out.println("mainPage: " + mainPage);
        System.out.println("footer: " + footer);
    }

}
```

W przykładzie powyżej drukowane wartości na ekranie będą pobrane z pliku `.properties` odpowiedniego do ustawionego domyślnego `Locale`. Można to sprawdzić wywołując: `Locale.getDefault()`.

Konkretne wartości dla danego Locale

Istnieje również możliwość pobrania wartości z pliku dla konkretnego, narzuconego przez nas locale. Przykład poniżej.

```
public class InternetStoreExample {

    public static void main(String[] args) {
        Locale pl = new Locale("pl", "PL");
        Locale en = Locale.UK;

        printValues(pl);
        printValues(en);
    }

    private static void printValues(Locale locale) {
        System.out.println("locale: " + locale);

        ResourceBundle resourceBundle = ResourceBundle.getBundle("Store", locale);
        String mainPage = resourceBundle.getString("mainPage");
        String footer = resourceBundle.getString("footer");

        System.out.println("mainPage: " + mainPage);
        System.out.println("footer: " + footer);
    }

}
```

Wydrukowanie wszystkich wartości

Poniżej znajdziesz fragment kodu, w którym drukowane są wszystkie wartości dla podanego locale.

```
public class InternetStoreExample {

    public static void main(String[] args) {
        Locale pl = new Locale("pl", "PL");
        Locale en = Locale.UK;

        printWholeMap(pl);
        printWholeMap(en);
    }

    private static void printWholeMap(Locale locale) {
        System.out.println("locale: " + locale);

        ResourceBundle resourceBundle = ResourceBundle.getBundle("Store", locale);

        Map<String, String> withValues = resourceBundle.keySet().stream()
            .collect(Collectors.toMap(key -> key, key -> resourceBundle.getString(key)));
        System.out.println(withValues);
    }

}
```

Zauważ, że mimo, że w plikach `Store_en.properties` ani `Store_pl.properties` nie ma klucza, `defaultProperty`, a został on wydrukowany na ekranie. W tym mechanizmie działa swojego rodzaju "dziedziczenie", dlatego klucz, który nie istniał w plikach `Store_en.properties` ani `Store_pl.properties`, ale istniał w pliku `Store.properties` został wydrukowany. Możemy równie dobrze ten klucz nadpisać defininiując go np. w pliku `Store_pl.properties`.

Hierarchia poszukiwań pliku .properties

Poniżej została umieszczona tabela, w której została rozpisana hierarchia poszukiwań odpowiedniego pliku `.properties`. W krokach dodane są również pliki `.java`, ale tak jak zostało wspomniane wcześniej, nie poruszaliśmy tematyki definiowania takich ustawień w plikach `.java`. Natomiast dla porządku, pliki `.java` również zostały umieszczone w tabeli.

Lp.	Szukamy pliku	Co się dzieje pod spodem?
1	Store_en_US.java	Szukamy podanego pliku
2	Store_en_US.properties	Szukamy podanego pliku
3	Store_en.java	Jeżeli pliku wyżej nie ma, szukamy pliku tylko z językiem, bez kraju
4	Store_en.properties	Jeżeli pliku wyżej nie ma, szukamy pliku tylko z językiem, bez kraju
5	Store_pl_PL.java	Jeżeli pliku wyżej nie ma, szukamy defaultowego Locale
6	Store_pl_PL.properties	Jeżeli pliku wyżej nie ma, szukamy defaultowego Locale
7	Store_pl.java	Jeżeli pliku wyżej nie ma, szukamy defaultowego Locale, ale bez kraju
8	Store_pl.properties	Jeżeli pliku wyżej nie ma, szukamy defaultowego Locale, ale bez kraju
9	Store.java	Jeżeli pliku wyżej nie ma, szukamy pliku bez Locale
10	Store.properties	Jeżeli pliku wyżej nie ma, szukamy pliku bez Locale
11	-	Jeżeli nie znaleziono żadnego pliku to zostaje wyrzucone `MissingResourceException`