Zaawansowane programowanie obiektowe i funkcyjne

Wykład 8: Lokalizacja

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.1
4 марта 2021 г.

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Lokalizacja

- Lokalizacja zapewnia uruchamianie aplikacji w lokalnym języku.
- Wiąże się także z bardziej subtelnymi kwestiami:
  - format zapisu liczb,
  - waluta,
  - format prezentacji daty.

Lokalizator

- Lokalizator to klasa `Locale` przekazująca informacje o lokalizacji.
- Możemy pobrać domyślny lokalizator z systemu.

```
1   Locale default = Locale.getDefault()
```

- Możemy pobrać dopuszczalne lokalizatory dla danego zastosowania.

```
1   Locale[] supportedLocales =
        NumberFormat.getAvailableLocales();
```

- Możemy użyć predefiniowanych lokalizatorów np. `Locale.ENGLISH`.
- Możemy utworzyć lokalizator dla danego identyfikatora języka.

```
1   Locale thai =
        Locale.forLanguageTag("th-TH-x-lvariant-TH")
```

Identyfikacja języka

- Identyfikator języka jest ciągiem znaków jednoznacznie określającym język.
- Zawiera informacje o języku, jego regionalnej odmianie, użytym alfabecie, wariancie i rozszerzeniu.
- Identyfikator jest zgodny z dokumentami
  - IETF BCP 47, "Tags for Identifying Languages"
  - UTS#35, "Unicode Locale Data Markup Language".

Składnia identyfikatora języka

1. Język
   - "en"(angielski), "pl"(polski), "kok"(konkani)
2. Alfabet
   - "Latn"(łaciński), "Cyrl"(cyrylica), "Hant"(tradycyjny alfabet chiński).
3. Kraj (rejon)
   - "US"(Stany Zjednoczone), "PL"(Polska), "029"(Karaiby)
4. Wariant
   - "polyton"(Politoniczny grecki), "POSIX"
5. Rozszerzenie
   - "u-ca-japanese"(kalendarz japoński), "u-nu-thai"(liczby tajskie), "x-java-1-7"(dowolne rozszerzenie)

Dostępne lokalizacje zapisu liczb

Dostępnych jest 736 lokalizacji, w tym dla angielskiego 106

Niue
Montserrat
Guernsey
Jamaica
Zambia
Malta
Liberia
Ghana
Israel
Palau
St. Vincent & Grenadines
United States,Computer
Europe
St. Kitts & Nevis
Macau SAR China
Belize
Nauru
Northern Mariana Islands
Grenada
Botswana
Australia
Cyprus
Rwanda
Ireland
Kiribati
Swaziland
United Kingdom

American Samoa
Jersey
Christmas Island
Austria
Sint Maarten
Tanzania
Puerto Rico
Bahamas
Kenya
Netherlands
South Sudan
Madagascar
South Africa
Tuvalu
Pitcairn Islands
Marshall Islands
Guyana
Nigeria
Pakistan
St. Lucia
Trinidad & Tobago
Bermuda
Vanuatu
United States
Norfolk Island
Guam
Anguilla

Cameroon
Tonga
Papua New Guinea
Eritrea
Philippines
Dominica
Cook Islands
Burundi
Antigua & Barbuda
Samoa
Namibia
Sierra Leone
St. Helena
Cayman Islands
Denmark
Zimbabwe
U.S. Outlying Islands
Tokelau
Slovenia
Micronesia
Belgium
Singapore
Switzerland
Sudan
Malaysia
Falkland Islands

Gambia
Diego Garcia
Sweden
World
Solomon Islands
Malawi
British Indian Ocean Territory
Germany
Cocos Keeling Islands
Finland
Seychelles
U.S. Virgin Islands
Uganda
New Zealand
Fiji
Barbados
Mauritius
Isle of Man
Lesotho
Hong Kong SAR China
Gibraltar
Canada
British Virgin Islands
Turks & Caicos Islands
India

Identyfikacja lokalizatora

- Możemy odczytać przyjazną nazwę lokalizatora korzystając z metody `getDisplayName`.

```
1  Locale pl = Locale.forLanguageTag("pl-PL");
2  pl.getDisplayName();
3  //Polish (Poland)
```

- Jednakże zwracana nazwa jest lokalizowana przez lokalizator domyślny.
- Możemy odczytać zlokalizowaną nazwę lokalizatora przekazując go do metody.

```
1  Locale pl = Locale.forLanguageTag("pl-PL");
2  pl.getDisplayName(pl);
3  //polski (Polska)
```

Testowanie działania lokalizacji

- Możemy przetestować działanie programu dla danej lokalizacji wywołując program z odpowiednimi parametrami.
  - `Duser.language.xx` pozwala określić język lokalizatora.
  - `Duser.region=XX` pozwala określić kraj lokalizatora

```
java -Duser.language.de -Duser.region=CH MojProgram
```

Lokalizacja liczb i walut

- Lokalizacja zawiera w sobie informacje dotyczące lokalnej waluty i sposobu formatowania liczb.

```
1  NumberFormat currFmt =
       NumberFormat.getCurrencyInstance(loc);
2  double amt = 123456.78;
3  System.out.println(currFmt.format(amt));
4  //123 456,78 zl
```

- Powiązanie tych dwóch aspektów nie zawsze jest porządane.
- Napotkamy problemy, gdy będziemy chcieli zapisać wartość w dolarach, ale z zachowaniem polskiego formatowania liczb.

Określanie waluty

- Możemy zadeklarować walutę w oderwaniu od lokacji.
- Służy do tego klasa `Currency`.
- Potrzebujemy także znać oznaczenie waluty.
  - Identyfikatory walut określone są przez standard ISO 4217.[1]

```
1  NumberFormat usdFormater =
       NumberFormat.getCurrencyInstance(pl);
2  usdFormater.setCurrency(Currency.getInstance("USD"));
3  //123 456,78 USD
```

- Zachowujemy formatowanie liczb charakterystyczne dla lokacji, ale używamy obcej waluty.

[1] http://www.currency-iso.org/en/home/tables/table-a1.html

Data i czas

- Lokalizacja daty i czasu dotyczy następujących aspektów
  - Tłumaczenia nazw miesięcy i dni tygodnia.
  - Kolejności prezentacji roku, miesiąca i dnia.
  - Obowiązującego kalendarza.
  - Strefy czasowej lokalizacji.

Formatowanie daty

- Lokalizacja pozwala na formatowanie daty i czasu korzystając ze stylów omówionych na wykładzie o czasie.
- Służą do tego statyczne metody `ofLocalizedDate`, `ofLocalizedTime` i `ofLocalizedDateTime` klasy `DateTimeFormatter`.
- Wymuszenie obcej lokalizacji następuje poprzez wywołanie metody `withLocale`.

```
1  DateTimeFormatter dateFormatter =
2  DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL)
3    .withLocale(Locale.of("RU"));
```

пятница, 13 сентября 2019 г.

Nazwy dni i miesięcy daty

- Możemy odczytać nazwy miesięcy i dni tygodnia dla danego języka.
- Możemy odczytać je z trybów wyliczeniowych `Month` i `DayOfWeek` określając lokalizację.

```
1   for (Month month : Month.values())
2    String inside = month.getDisplayName(FULL, locale);
```

- W niektórych językach nazwy mają zmienne formy w zależności czy są częścią daty, czy też występują samodzielnie.
- Możemy odczytywać samodzielne formy używając stylu `FULL_STANDALONE`.

```
1  String standalone =
       month.getDisplayName(FULL_STANDALONE, locale);
```

Przykłady zróżnicowanych form

Miesiące

- górnołużycki: januara <> januar
- łotewski: janvāris <> Janvāris
- słowacki: januára <> január
- czeski: ledna <> leden
- dolnołużycki: januara <> januar
- baskijski: urtarrilak <> Urtarrila
- rosyjski: января <> январь
- azerski: yanvar <> Yanvar
- galicyjski: xaneiro <> Xaneiro
- białoruski: студзеня <> студзень
- fiński: tammikuuta <> tammikuu
- osetyjski: январы <> Январь
- asturyjski: de xineru <> xineru
- albański: janar <> Janar
- wietnamski: tháng 1 <> Tháng 1
- **polski: stycznia <> styczeń**
- włoski: gennaio <> Gennaio
- ukraiński: січня <> січень
- chorwacki: siječnja <> siječanj
- szkocki gaelicki: dhen Fhaoilleach <> Am Faoilleach

Dni tygodnia

- łotewski: pirmdiena <> Pirmdiena
- lapoński Inari: vuossaargâ <> vuossargâ
- baskijski: astelehena <> Astelehena
- kazachski: дүйсенбі <> Дүйсенбі
- galicyjski: luns <> Luns
- fiński: maanantaina <> maanantai
- osetyjski: къуырисæр <> Къуырисæр
- albański: e hënë <> E hënë
- włoski: lunedì <> Lunedì

Kolejność dni tygodnia

- Musimy pamiętać, że w zależności od lokalizatora pierwszym dniem tygodnia może być sobota, poniedziałek, niedziela, a nawet piątek.

```
1  DayOfWeek first =
       WeekFields.of(locale).getFirstDayOfWeek();
```

```
MONDAY -> 359 //Polska
FRIDAY -> 1 //Bangladesz
SATURDAY -> 33 //Zjednoczone Emiraty Arabskie
SUNDAY -> 343 //Malta (domyślny)
```

Lokalizacja, a sortowanie

- Wybrana lokalizacja wpływa na sposób sortowania tekstów.
- W skrócie, jeżeli w danej lokalizacji nie występują pewne znaki to mechanizm sortowania nie wie jak porównywać je z innymi.
- Do określania lokalizacji przy sortowaniu służy klasa `Collator`.

```
1  Collator coll = Collator.getInstance(Locale.US);
2  list.sort(coll);
```

Porównanie sortowania

`Locale.US`

1. sambala
2. samburu
3. sango
4. sangu
5. sena
6. serbski
7. soga
8. somalijski
9. sorani
10. standardowy marokański tamazight
11. staro-cerkiewno-słowiański
12. suahili
13. syczuański
14. syngaleski
15. szkocki gaelicki
16. szona
17. szwajcarski niemiecki
18. szwedzki
19. słowacki
20. słoweński

`Locale.of("pl")`

1. sambala
2. samburu
3. sango
4. sangu
5. sena
6. serbski
7. słowacki
8. słoweński
9. soga
10. somalijski
11. sorani
12. standardowy marokański tamazight
13. staro-cerkiewno-słowiański
14. suahili
15. syczuański
16. syngaleski
17. szkocki gaelicki
18. szona
19. szwajcarski niemiecki
20. szwedzki

Dokładniej o porównywaniu znaków

- Klasa `Collator` pozwala na określenie stopnia rozróżniania znaków.
- Określamy czy mają być brane pod uwagę cechy pierwszorzędne, drugorzędne, czy trzeciorzędne.
  - Różnica miedzy A i Z jest pierwszorzędna.
  - Różnica między A i Å jest drugorzędna.
  - Różnica miedzy A i a jest trzeciorzędna.
- Poziom określamy za pomocą stałych klasy `Collator`.

```
1  Collator coll = Collator.getInstance(locale);
2  coll.setStrength(Collator.PRIMARY);
```

PRIMARY	SECONDARY	TERTIARY
Angstrom = Ångström	Angstrom ≠ Ångström	Angstrom ≠ Ångström
Able = able	Able = able	Able ≠ able

Рис. 1: *Przykłady cech [Horstmann and Cornell, 2017]*

Dalsze problemy z porównywaniem znaków

- Unicode pozwala na niejednoznaczny zapis.
- Przykładowo możemy zapisać literę Å jako jeden znak lub jako dwa osobne A ° uzyskując ten sam efekt.
- Istnienie ligatury powoduje, że jeden znak może łączyć nawet trzy inne znaki *ffi*.
- Problem z porównywaniem znaków rozwiązuje się wprowadzając cztery rodzaje dekompozycji.

C	Brak rozkładu znaków akcentowanych
D	Rozkład znaków akcentowanych (kanoniczny)
KC,KD	Rozkład ciągów znaków (ligatury i ™)

Brak rozkładu	Rozkład kanoniczny	Pełen rozkład
$\text{Å} \neq \text{A°}$	$\text{Å} = \text{A°}$	$\text{Å} = \text{A°}$
$\text{™} \neq \text{TM}$	$\text{™} \neq \text{TM}$	$\text{™} = \text{TM}$

Рис. 2: *Przykłady rozkładów [Horstmann and Cornell, 2017]*

Stosowanie dekompozycji

- Dekompozycję określamy za pomocą stałych klasy `Collator`.

```
1  Collator coll = Collator.getInstance(locale);
2  coll.setDecomposition(Collator.CANONICAL_DECOMPOSITION);
3  CollationKey aKey = coll.getCollationKey("A");
```

- Wynik dekompozycji może być przechowywany jako obiekt klasy `CollationKey`.

```
1  CollationKey aKey = coll.getCollationKey("a");
2  aKey.compareTo(coll.getCollationKey("A"));
```

- Możemy także dokonywać normalizacji tekstu korzystając z klasy `java.text.Normalizer`.

```
1  String normalized = Normalizer.
2    normalize(text, Normalizer.Form.NFKD);
```

- Zgodnie z zaleceniami W3C należy stosować normalizację C przy przekazywaniu tekstów.

Formatowanie liczb i dat

- Java posiada klasę `MessageFormat` pozwalającą na wstawianie parametrów do tekstu.

```
1  String message = "{0}. You have a new wire transfer
       for {1} at {0}.";
2  MessageFormat.format(message, new Date(), 123.45);
3  //11/22/18, 7:59 PM. You have a new wire transfer for
       123.45 at 11/22/18, 7:59 PM.
```

- Komunikat podaje dwukrotnie datę razem z czasem.
- Możemy rozbić te dwa elementy definiując formaty.

```
1  message = "{0,date}. You have a new wire transfer for
       {1,number} at {0,time}.";
2  //Nov 22, 2018. You have a new wire transfer for
       123.45 at 7:59:09 PM.
```

- Dodatkowo można określić styl prezentacji elementów w tym wyświetlanie jako walutę.

```
1  message = "{0,date,long}. You have a new wire transfer
       for {1,number,currency} at {0,time,short}.";
2  //November 22, 2018. You have a new wire transfer for
       $123.45 at 7:59 PM.
```

Warunkowe modyfikowanie tekstu

- Przekształćmy komunikat, aby podawać informację o liczbie przelewów.

```
1  message = "{0}. You have {2} new wire transfer. The
       total amount is {1}. The last transfer at {0}.";
2  System.out.println(MessageFormat.format(message, new
       Date(), 123.45,1));
3  //11/22/18, 8:30 PM. You have 1 new wire transfer. The
       total amount is 123.45. The last transfer at
       11/22/18, 8:30 PM.
```

- Pojawią się problemy, gdy będzie więcej przelewów. Możemy je wyeliminować dodając nowy parametr.

```
1  message = "{0}. You have {2} new wire {3}. The total
       amount is {1}. The last transfer at {0}.";
2  System.out.println(MessageFormat.format(message, new
       Date(), 123.45,2,"transfers"));
3  //11/22/18, 8:30 PM. You have 2 new wire transfers.
       The total amount is 123.45. The last transfer at
       11/22/18, 8:30 PM.
```

- Jednakże musimy teraz pilnować, aby panowała zgodność między dwoma ostatnimi parametrami.

Formatowanie warunkowe

- Java wprowadza parametr formatowania warunkowego.
- Pole typu choice pozwala zdefiniować parametr w zależności od wartości zmiennej.

```
1  message = "{0}. You have {2} new wire
       {2,choice,1#transfer|2#transfers}. The total amount
       is {1}. The last transfer at {0}.";
2  System.out.println(MessageFormat.format(message, new
       Date(), 123.45,5));
3  //11/22/18, 8:30 PM. You have 5 new wire transfers.
       The total amount is 123.45. The last transfer at
       11/22/18, 8:30 PM.
```

- Formatowanie pozwala także definiować przedziały dla wartości stosując znak $<$.

Kodowanie

- Java używa kodowania UTF-16 co pozwala na zapisanie komunikatów w różnych językach.
- Należy jednak pamiętać, że teksty z zewnętrznych źródeł mogą być zapisane w innym kodowaniu.
- Także system operacyjny najpewniej używa innego kodowania.
- Kodowanie możemy uwzględnić podczas otwierania strumieni wejścia/wyjścia.

```
1  PrintWriter out = new PrintWriter(filename,
       "Windows-1252");
```

- W przypadku systemu operacyjnego możemy odczytać jego domyślne kodowanie.

```
1  Charset platformEncoding = Charset.defaultCharset();
```

- W przypadku tekstów zewnętrznych może okazać się konieczny eksperymentalny dobór kodowania.

Zasoby

- Tworząc nową lokalizację musimy zadbać, aby dostarczyć do aplikacji tłumaczenia wszystkich elementów tekstowych.
- Ze względu na utrzymanie kodu musimy przechowywać te teksty odrębnie od kodu.
- Dla tego tworzymy osobne pliki zwane zasobami (ang. *Bundle*), które można edytować bez ingerencji w kod programu.

Plik właściwości

- Plik właściwości zawiera pary klucz wartość.

```
refreshButton=Refresh
wireTransferMessage={0}. You have a new wire transfer for {1} at {0}.
initialMessage=Click the button to refresh the wire transfer list.
```

- Zasoby trzymamy w oddzielnych plikach dla każdej z lokalizacji.

  `MyProgramStrings.properties`
  `MyProgramStrings_en.properties`
  `MyProgramStrings_en_US.properties`
- Ważne jest, aby zachować plik domyślny i plik językowy, a nie tylko pliki regionalne.

Odczyt zasobów

- Na początku należy wskazać właściwy plik z zasobami.

```
1  ResourceBundle bundle =
2  ResourceBundle.getBundle("MyProgramStrings", locale);
```

- Następnie można odczytywać z obiektu klasy `Bundle` wartości odwołując się do kluczy.

```
1  String refreshButtonLabel =
       bundle.getString("refreshButton");
```

Klasy zasobów

- Zasoby inne niż teksty przechowujemy w klasach pochodnych klasy `ResourceBundles`.

  `ProgramResources.java`
  `ProgramResources_en.java`
  `ProgramResources_en_US.java`

- Ładowanie zasobów odbywa się analogicznie jak dla tekstów.

```
1  ResourceBundle bundle =
       ResourceBundle.getBundle("ProgramResources",
       locale);
```

- Nadal będziemy odwoływać się do par klucz wartość, ale tym razem wartość będzie obiektem.

Definicja zasobów

- Zasoby definiujemy nadpisując klasę `ListResourceBundle`.
- Klasa powinna posiadać obiekt `Object[][] contents` posiadający wpisy klucz, wartość.
- Dodatkowo musimy zaimplementować metodę

```
1  public Object[][] getContents() {return contents;}
```

Przykład definicji zasobów

Nadpisanie klasy zasobów

```
1  public class ProgramResources_en_US extends
       ListResourceBundle{
2  private static final Object[][] contents =
3  {
4    { "backgroundColor", Color.blue },
5    { "defaultPaperSize", new double[] { 216, 279 } }
6  }
7  public Object[][] getContents() {
8    return contents; }
9  }
```

- Po nadpisaniu klasy zasobów możemy odwoływać się kodzie do jej elementów.

```
1  ResourceBundle bundle = ResourceBundle
2    .getBundle("ProgramResources", locale);
3  Color background =
       (Color)bundle.getObject("backgroundColor");
4  double[] paperSize =
       (double[])bundle.getObject("defaultPaperSize");
```

Bibliografia

[Horstmann and Cornell, 2017] Horstmann, C. S. and Cornell, G. (2017).
Java. Techniki zaawansowane.
Helion.