# Java training

Internationalization and localization, formatting dates and numbers

# Session overview

- Internationalization and localization
- Formatting dates and numbers
- Hands-on examples

# Internationalization

**Internationalization** (aka i18n) - designing an application to be adaptable to multiple languages & regions, without making development changes

- The same executable can run worldwide, just needs the localized data
- Textual elements (messages, GUI component labels) are not hard-coded
  - Stored outside the source code and retrieved dynamically
- Support for new languages does not require recompilation
- Culturally-dependent data (dates, currencies) appear in formats that conform to the end user's region and language
- It can be localized quickly

# Localization

**Localization** (aka l10n) - adapting software for a specific region or language
→ adding locale-specific components and translating text

- The primary task- translating the user interface elements
  - Changing the language interaction
  - Displaying values localized:
    - Numbers
    - Dates
    - Currency
    - Other localized

# Classes used for i18n / l10n

- **`ResourceBundle`** - aggregates locale-specific objects:
  - Used to isolate locale-sensitive data (ex: translatable text)
  - It loads the properties files that contain the l10n messages (further detailed)
  - Creation:

```
ResourceBundle rb = ResourceBundle.getBundle("messages");
```

- **`Locale`** - class used to identify a particular language and country
  - After creating a `Locale` object, it is passed to other objects, to:
    - Retrieve l10n messages
    - Format dates and numbers
  - Creation: `Locale usLocale = new Locale("en","US");`

# Hands-on - localization setup

1. Create the session folder (d03/s06) and the source folder (d03/s06/src)
2. Create the package `com.cerner.training.d07` in the source folder
3. Create the following files in the 'd03/s06/src' folder:
   a. messages.properties → holds the default (fallback) translations
   b. messages_en_US.properties → holds the translations for the English (US) locale
   c. messages_de_DE.properties → holds the translations for the German (DE) locale
4. Add a 'greeting' property in every file:
   a. greeting=Hello, everyone! → in the default and _en_US properties file
   b. greeting=Hallo, zusammen! → in the _de_DE properties file
5. Create a class in the package, named `LocalizationSample` + a `main` method
6. Write the code to use them → (the next slide)

# Hands-on - a program w/o and with i18n / l10n

```
// without l10n
String greeting = “Hello, everyone!”; // hard-coded in English (US)
System.out.println(greeting);          // will greet everyone in English :-)

// with l10n
Locale usLocale = new Locale("en", "US");
Locale deLocale = new Locale("de", "DE"); // + create a locale for ro + RO
ResourceBundle resourceBundle =
                    ResourceBundle.getBundle("messages", usLocale);
System.out.println(resourceBundle.getMessage(“greeting”);
```

# Formatting dates

**`DateFormat`** - main class used for formatting dates / times (on `Locale`):

```
Locale usLocale = new Locale("en", "US");
DateFormat.getDateInstance(DateFormat.DEFAULT, usLocale);
String l10nDate = dateFormat.format(new Date());
```

- Formatting styles (for US):
  - `DEFAULT`: Feb 21, 2018
  - `SHORT`: 2/21/18
  - `MEDIUM`: Feb 21, 2018
  - `LONG`: February 21, 2018
  - `FULL`: Wednesday, February 21, 2018

**+ Hands-on →**

# Formatting times

```
Locale deLocale = new Locale(“de”, “DE”);
DateFormat.getTimeInstance(DateFormat.DEFAULT, deLocale);
String l10nTime = dateFormat.format(new Date());
```

- Formatting styles (for DE):
  - DEFAULT: 7:33:47
  - SHORT: 07:33
  - MEDIUM: 07:33:07
  - LONG: 07:33:45 OEZ
  - FULL: 7.33 Uhr OEZ

**+ Hands-on →**

# Formatting both date and time

```
DateFormat.getDateTimeInstance(DateFormat.LONG,
                               DateFormat.LONG,
                               locale);
```

# Formatting numbers

**`NumberFormat`** - class used to format numbers, based on `Locale` objects

- It has several static methods for creating instances, accepting a `Locale` as parameter

```
NumberFormat formatter = NumberFormat.getInstance(deLocale);
System.out.println(formatter.format(1000));   // 1.000
```

- Instances of `NumberFormat` can be also used to parse `Strings` into numbers:

```
Number number = formatter.parse(“10.000”);
```

**`DecimalFormat`** – class used to display decimal numbers in a given format

```
DecimalFormat decimalFormat = new DecimalFormat("##,###");
System.out.println(decimalFormat.format(21340)); // 21,340
```

# Further reading

- https://docs.oracle.com/javase/8/docs/api/java/util/ResourceBundle.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Locale.html
- https://docs.oracle.com/javase/8/docs/api/java/text/DateFormat.html
- https://docs.oracle.com/javase/8/docs/api/java/text/NumberFormat.html
- https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html

# Q&A session

1. You ask, I answer
2. I ask, you answer