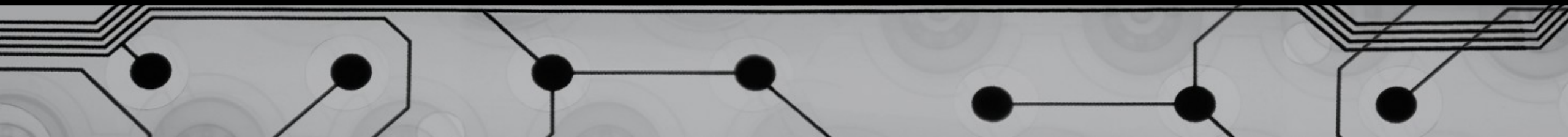
CST 243-3 Rapid Application Development

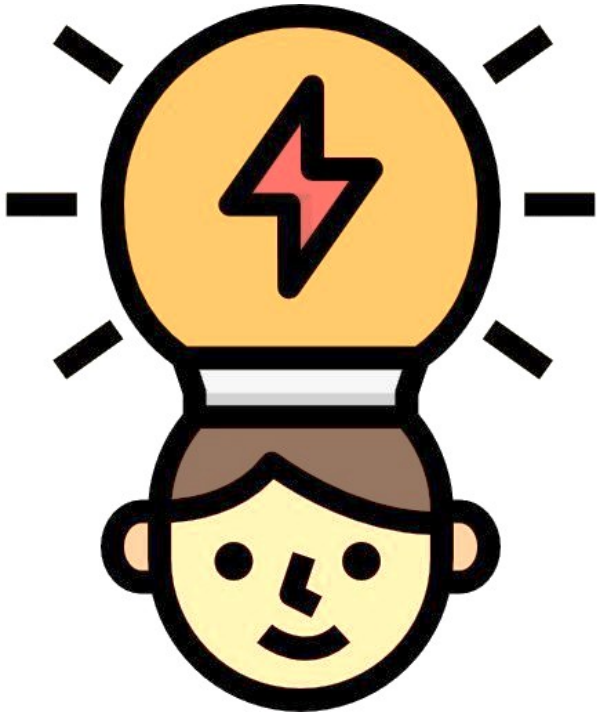
Lesson 05: Developing Localized Applications

By Subhash Ariyadasa



Lesson Learning Outcomes

- After successful completion of this lesson you will be able to,
  - **Demonstrate** knowledge of **localization** and **internationalization** concepts
  - **Explain** various **techniques used to handle language-specific resources** in Java applications
  - **Use Java's built-in Locale and ResourceBundle classes** effectively to implement localized Java applications
  - **Explain the different language-specific data handling techniques in databases** for selecting the most suitable approach for managing localized content effectively
  - **Develop Java applications that** can seamlessly **support multiple languages**



Lesson Outline

- Part I: Introduction to Localization
  - I18n, L10n and G11n
  - Locale
  - Need of I18n, L10n
  - Contributors
  - Translation
- Part II: Localizing Java Applications
  - Locale in Java Environment
  - ResourceBundle Class
  - Internationalizing a Simple Program
  - Formatting Numbers
  - Formatting Currency
  - Dates and Time Formatting
  - Message Formatting

Lesson Outline…

- Part III: Data Modeling for Multiple Languages
  - Database Localization
  - Field Database
  - Table Database
  - Row Database
  - Clone Database
  - Advantages and disadvantages of each type

Part I

Introduction to Localization

Localization (l10n)

- Localization involves **taking a product** and **making it linguistically and culturally appropriate to the target locale** (country/region and language) where it will be used and solved

Linguistic and Cultural Diversity

- **Different languages** are spoken/written in **different geographical regions** of the world
- **Different cultures** have different religious, political, beliefs, attitudes, practices, etc
- Dimensions of Diversity:
  - Names, measurement units, currency, time and date formats, language usage, music, colors, geographical locations

Only in Software??? Noooo…

- **Literary works** are adapted to make them appropriate to the target audience by changing names, concepts, etc
  - Short stories, novels, poems, dramas, etc
- **Electronic devices** such as televisions, iPods, mobile phone have interfaces in local languages such as Arabic, Chinese, French, Italian, etc
- **User and service manuals** are available in local languages such as Arabic, Chinese, French, Italian, etc

Locale

- Locale is a **set of parameters** that defines the **user's language, country** and **any special variant preferences** that the user wants to see in their user interface
- Usually a locale identifier consists of at least a **language identifier** and a **region identifier**
  - E.g. : en_US, en_UK, si_LK, ta_LK, en_AU.UTF-8

Locale...

- A language (e.g. English)
  - Often expressed as an ISO-639-1 code: **de**, **en**, **fr**, **si**, **ta**
- A region or location (e.g. United States, UK)
  - Often expressed as an ISO-3166-1 code: **CA**, **US**, **GB**, **DE**, **LK**
- Why isn't it enough to specify just the language?
  - Different locations may use different conventions, spelling, etc.
  - "color" (US) vs. "colour" (UK)
  - "localize" (US) vs. "localise" (UK)
  - Some locations use dialects of a given language
  - Other differences (dates, currency, numbers, time zone, etc.)

Locale Example

- si_LK

```
$string['access'] = 'පිවිසුම් හැකියාව';
$string['accesshelp'] = 'පිවිසුම් හැකියාව - උදව්';
$string['accesskey'] = 'පිවිසුම් යතුර, $a';
$string['accessstatement'] = 'පිවිසුම් හැකියාවේ ප්‍රකාශනය';
$string['activitynext'] = 'මීලඟ ක්‍රියාකාරකම';
$string['activityprev'] = 'පෙර ක්‍රියාකාරකම';
$string['breadcrumb'] = 'Breadcrumb පියමග';
$string['currenttopic'] = 'මෙම මාතෘකාව';
$string['currentweek'] = 'මෙම සතිය';
$string['hideblocka'] = '$a කොටස සඟවන්න';
$string['monthnext'] = 'ඉදිරි මාසය';
$string['monthprev'] = 'පසුගිය මාසය';
$string['showblocka'] = '$a කොටස පෙන්වන්න';
```

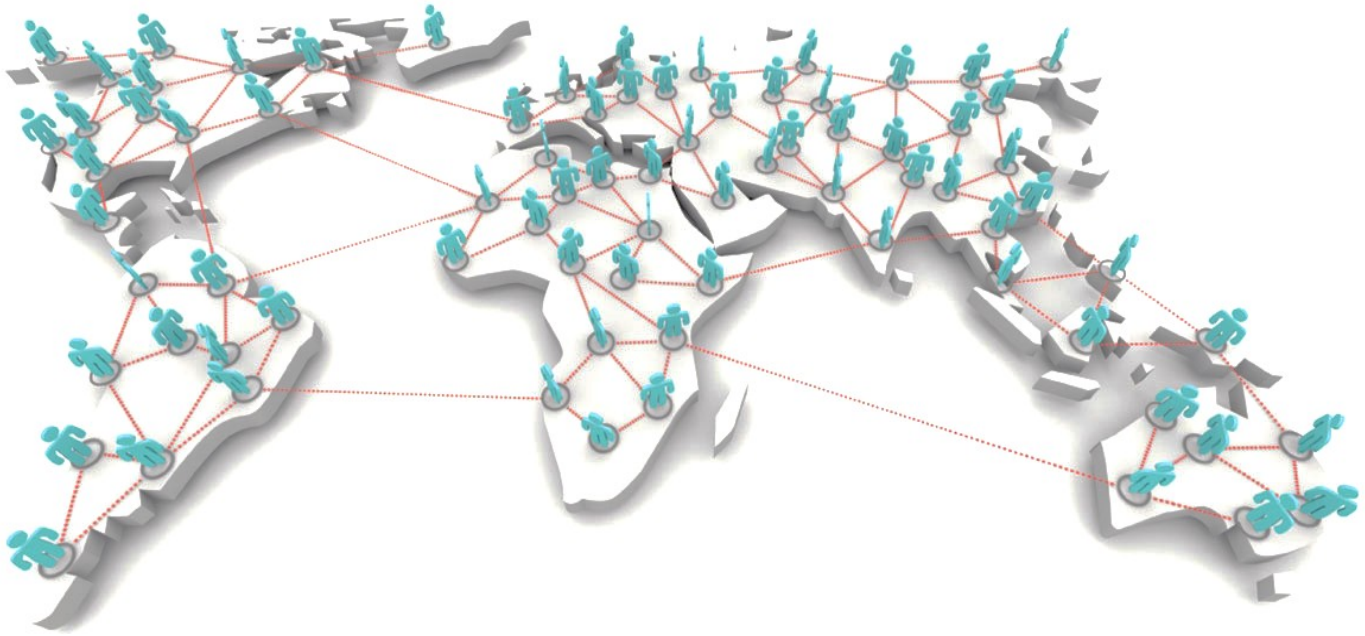
Locale Example...

- ta_LK

```
$string['access'] = 'அணுகற்திறன்';
$string['accesshelp'] = 'அணுகலுதவி';
$string['accesskey'] = 'அணுகற்குறி $a';
$string['accessstatement'] = 'அணுகற்திறன் கூற்று';
$string['activitynext'] = 'அடுத்த செயல்பாடு';
$string['activityprev'] = 'முன்னைய செயல்பாடு';
$string['breadcrumb'] = 'Breadcrumb பாதை';
$string['monthnext'] = 'அடுத்த மாதம்';
$string['monthprev'] = 'முன்னைய மாதம்';
$string['sitemap'] = 'தள அமைப்பு';
$string['skipa'] = '$a ஐத் தவிர்';
$string['skipblock'] = 'கட்டத்தைத் தவிர்';
$string['skipnavigation'] = 'வழிச்செலுத்தல் தவிர்';
```

Internationalization (i18n)

- Internationalization is the **process of generalizing a product** so that it can handle multiple languages and cultural conventions **without the need for re-design**
- Internationalization takes place at the **level of program design**



Globalization (g11n)

- Globalization addresses the **business issues associated with taking a product global**
- In the globalization of high-tech products this involves integrating localization through out a company, after **proper internationalization and product design**, as well as **marketing**, **sales**, and **support in the world market**

I18n, L10n and G11n

- Internationalization ("I18n"):
  - **Process of designing software** so that it can be **adapted to various languages and regions**
  - Done **once per product** (ideally) and updated as code is added
- Localization ("L10n"):
  - **Process of adapting/translating** internationalized software for a **specific region or language**
  - Done once per locale and each locale is updated as text is added
- Globalization ("G11n"): I18n + L10n
  - **Process of taking a product to global**
  - Less commonly used term, but many companies use it

Who Cares?

- Why should a team want to internationalize / localize its app?
  - Reach a wider audience
  - Make more **$$$**
- Is it worth to localize?
  - May **need to evaluate cost/benefit**:
    - What fraction of our users speak that language?
    - Are they also fluent in English?
    - Are they already able to use the site now?
- Open-source software is often **translated for free** by community
  - Maybe you can post your code and let them do it

Who is doing this?

- Developers
  - **Internationalize** the app's code
  - **Pull all strings out of code** and into separate resource files
  - **Call methods** that localize/format strings, numbers before printing
  - **Use libraries** (e.g. gettext) to help localize messages
- Localizers (maybe not programmers)
  - **Localize** the app's text
  - Often **hired to localize an app** for a particular locale at a time
  - Desktop apps: possibly **compile a different binary for each locale**
  - Web app: **look up localized strings** when generating each page

Translation

- Translation is **only one of the activities** in localization
- In addition to translation, a localization project includes **many other tasks**
  - Project Management
  - Software Engineering
  - Testing

Part II

Localizing Java Applications

Locale in Java

- Locale is an **identifier for a language**, an **optional country** (or region), and an **optional variant code**
- In the Java programming language, a **locale is represented by a Locale object**
- **java.util.Locale** is a lightweight object that contains only a few important members:
  - A **language code**
  - An **optional country or region code**
  - An **optional variant code**

Locale in Java…

- An **operation that requires a locale to perform its task** is called **locale-sensitive**
  - For example, displaying a number is a locale-sensitive operation
- Locale.Builder() approach can be used when creating the Locale object

// A very specific Sinhala-speaking, Sinhala locale with a custom variant of traditional.

```
Locale locale = new Locale.Builder()
                .setLanguage("si")
                .setRegion("LK")
                .setVariant("TRADITIONAL")
                .build();
```

Default Locale

- Locale objects are used **throughout the Java class libraries**
- The default locale is used by locale-sensitive objects **whenever your application doesn't specify an explicit locale choice**
  - System.out.println(Locale.getDefault());
- The initial **default locale is normally determined from the host operating system's locale**

ResourceBundle Class

- A ResourceBundle object allows you to **isolate localizable elements from the rest of the application**
- With all resources separated into a bundle, the **application simply loads the appropriate bundle for the active locale**
- If the user switches locales, the application just **loads a different bundle**

Internationalizing a Simple Java Program

- Create the **Properties Files**
- Define the **Locale**
- Create a **ResourceBundle**
- **Fetch the Text** from the ResourceBundle

Creating the Properties File

- Properties file **stores information about the characteristics of a program or environment**
- It is in **plain-text format**
- For example, if all the text are in a properties file, they can be translated into various languages
- **No changes to the source code** are required
  - MessagesBundle_en_US.properties
  - MessagesBundle_si_LK.properties
  - MessagesBundle_ta_LK.properties

Define the Locale

- Use locale object from **java.util.Locale** class
  - **String** language = "si";
  - **String** country = "LK";
  - Locale currentLocale = **new Locale.Builder()**
    .setLanguage(language**)**
    .setRegion(country**)**
    .build();

- Locale objects are **only identifiers**
- After defining a Locale, you **pass it to other objects** that perform useful tasks
  - locale-sensitive objects: behavior varies according to Locale
  - i.e.: formatting dates and numbers

Create a ResourceBundle

- The ResourceBundle is created as follows:

```
ResourceBundle res =
ResourceBundle.getBundle("MessagesBundle", currentLocale);
```

- Arguments passed to the getBundle method **identify which properties file will be accessed**
  - First argument: **refers to the family of properties** files
  - Second argument: **specifies which property file is chosen**
    - e.g.:
      - MessagesBundle_si_LK.properties

Fetch the Text from the ResourceBundle

- The properties files **contain key-value pairs**
- The values **consist of the translated text** that the program will display
- You **specify the keys when fetching the translated messages from the ResourceBundle** with the getString method

```
String msg = res.getString("message");
```

Formatting Numbers

- In France the number **123456.78** should be formatted as **123 456,78**, and in **Germany** it should appear as **123.456,78**
- Before displaying or printing a number, a program must **convert it to a String** that is **in a locale-sensitive format**
- The **NumberFormat class** can be **used to format primitive-type numbers**

Formatting Numbers…

```
int quantity = 123456;
double amount = 345987.246;
NumberFormat numberFormatter;
NumberFormatter =
        NumberFormat.getNumberInstance(currentLocale);
String quantityOut = numberFormatter.format(quantity);
String amountOut =  numberFormatter.format(amount)
```

Formatting Currencies

- Format currencies in the **same manner** as numbers, except that you call **getCurrencyInstance** to create a formatter

```
double currencyAmount = 9876543.21;
Currency currentCurrency =
    Currency.getInstance(currentLocale);
NumberFormat currencyFormatter =
NumberFormat.getCurrencyInstance(currentLocale);
System.out.println(currentCurrency.getDisplayName() + ": " +
    currencyFormatter.format(currencyAmount));
```

Formatting Dates

- **Germans** recognize **13.2.24** as a valid date, but **Americans** expect that same date to appear as **2/13/24**
- The **DateFormat class** allows you to **format dates and times** with predefined styles in a locale-sensitive manner
- Formatting dates with the **DateFormat** class is a **two-step process**
  - **Create a formatter** with the **getDateInstance** method
  - **Invoke the format method**, which returns a String containing the formatted date

Formatting Dates…

```
Date today;
String dateOut;
DateFormat dateFormatter;

dateFormatter =
DateFormat.getDateInstance(DateFormat.DEFAULT,
currentLocale);
today = new Date();
dateOut = dateFormatter.format(today);
```

Formatting Time

- Date objects represent **both dates and times**
- Formatting times with the **DateFormat** class is similar to formatting dates, except that you **create the formatter with the getTimeInstance method**

```
DateFormat timeFormatter =
DateFormat.getTimeInstance(DateFormat.DEFAULT,
currentLocale);
```

Formatting Dates and Time

Sample Date and Time Formats

Style	U.S. Locale	French Locale
DEFAULT	Jun 30, 2009 7:03:47 AM	30 juin 2009 07:03:47
SHORT	6/30/09 7:03 AM	30/06/09 07:03
MEDIUM	Jun 30, 2009 7:03:47 AM	30 juin 2009 07:03:47
LONG	June 30, 2009 7:03:47 AM PDT	30 juin 2009 07:03:47 PDT
FULL	Tuesday, June 30, 2009 7:03:47 AM PDT	mardi 30 juin 2009 07 h 03 PDT

Create the formatter with the **getDateTimeInstance** method

Formatting Messages

- Dealing with **Compound Messages**
  - Contain several kinds of variables: **dates**, **times**, **strings**, **numbers**, **currencies**, and **percentages**
  - The **MessageFormat** class will be used to internationalize a compound message
  - To format a compound message in a locale-independent manner,
    - **Construct a pattern** that you apply to a **MessageFormat** object
    - **Store this pattern** in a **ResourceBundle**

You have purchased an item for $7.75 from ebay on 8/1/24 at 7:00 PM.

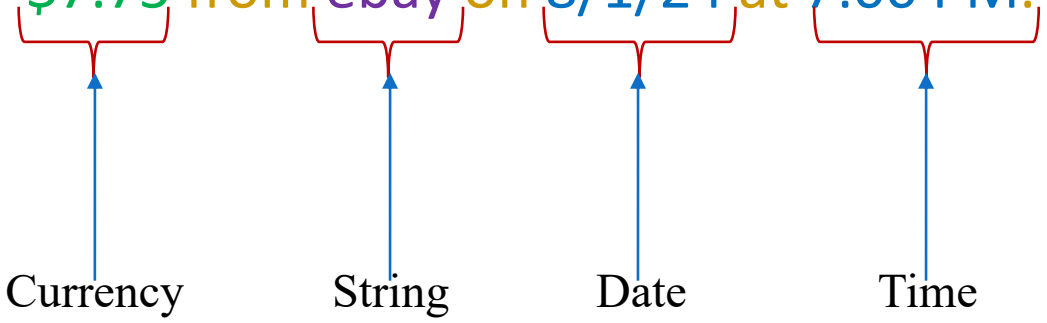
Formatting Messages…

- Formatting compound message with the **MessageFormat** class is a **five-step** process
  1. Identify the Variables in the Message
  2. Isolate the Message Pattern in a ResourceBundle
  3. Set the Message Arguments
  4. Create the Formatter
  5. Format the Message Using the Pattern and the Arguments

Formatting Messages…

1. Identify the Variables in the Message

You have purchased an item for $7.75 from ebay on 8/1/24 at 7:00 PM.



2. Isolate the Message Pattern in a ResourceBundle

- When we are creating message pattern, we need to **use language-neutral format**
- There are **three ways to write variables in a language-neutral format**

Formatting Messages...

1. { ArgumentIndex } → e.g.: {0}
2. { ArgumentIndex , FormatType } → e.g.: {1,date}
3. { ArgumentIndex , FormatType , FormatStyle } → e.g.: {1,date,short}

- **Format types** available under this class:
  - number, date, time, choice
- **Format styles** available under this class:
  - short, medium, long, full, integer, currency, percent, etc.

Formatting Messages…

- We can **convert** our **variables** like below **to apply language-neutral format**

You have purchased an item for $7.75 from ebay on 8/1/23 at 7:00 PM.

- Currency → {0,number,currency}
- String → {1}
- Date → {2,date,short}
- Time → {2,time,short}

You have purchased an item for {0,number,currency} from {1} on {2,date,short} at {2,time,short}.

Formatting Messages…

3. Set the Message Arguments

```
Object[] messageArguments = {
177.5,
messages.getString("ebay"),
new Date()
};
```

4. Create the Formatter

```
MessageFormat formatter = new MessageFormat("");
formatter.setLocale(currentLocale);
```

Formatting Messages...

5. **Format the Message** using the Pattern and the Arguments

```
formatter.applyPattern(messages.getString("template"));
String output = formatter.format(messageArguments);
```

- Then, you can **display the message**
  - If you used en_US:

    You have purchased an item for $177.75 from ebay on 8/1/24 at 7:00 PM.

  - If you used si_LK:

ඔබ LKR 177.75 ක් වටිනා භාණ්ඩයක් 8/1/24 වන දින 7:00 PM ට පමණ ebay මගින් මිලට ගන්නා ලදී.

Part III

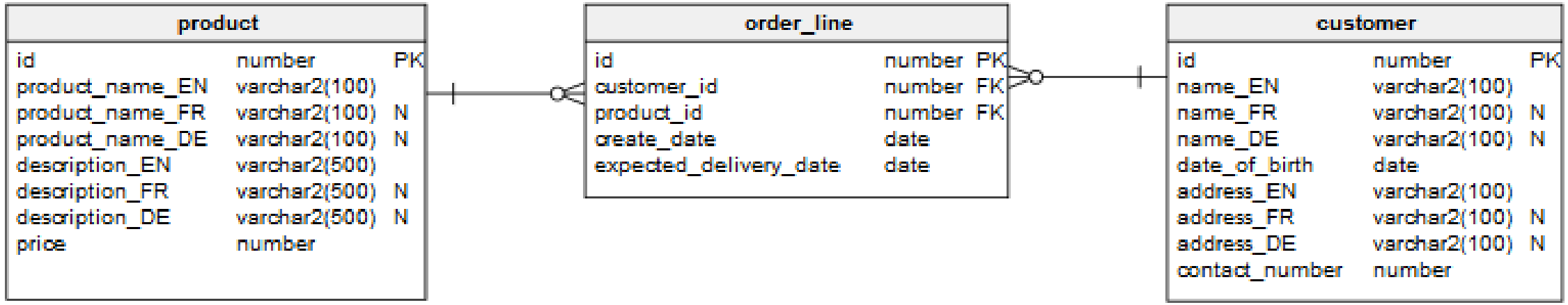
Data Modeling for Multiple Languages

Database Localization

- There are **several ways** to perform database localization
- But mainly we can identify **4 models**
  - **Field Database** Localization Method
    - Adding **separate language columns** for each intended field
  - **Table Database** Localization Method
    - Creating a **separate table** for translated text
  - **Row Database** Localization Method
    - A translation **table with rows for each language**
  - **Clone Database** Localization Method
    - Creates a **complete clone of the database**

Field Database Localization Method

- **Simplest approach** in terms of development
- Implemented by **adding one language column for each field**



Field Database Localization Method...

- Advantages
  - **Easy** to implement
  - **No complexity in writing SQL** to fetch the underlying data in any language
- Disadvantages
  - **No scalability**
  - **Time-consuming**
  - **Need changes in each and every query** if you introduced a new language

Table Database Localization Method

- A **separate table** is used to store translated text
  - Contains one **column** for **each language**
  - Values that have been translated from field values into **all applicable languages are stored as records**

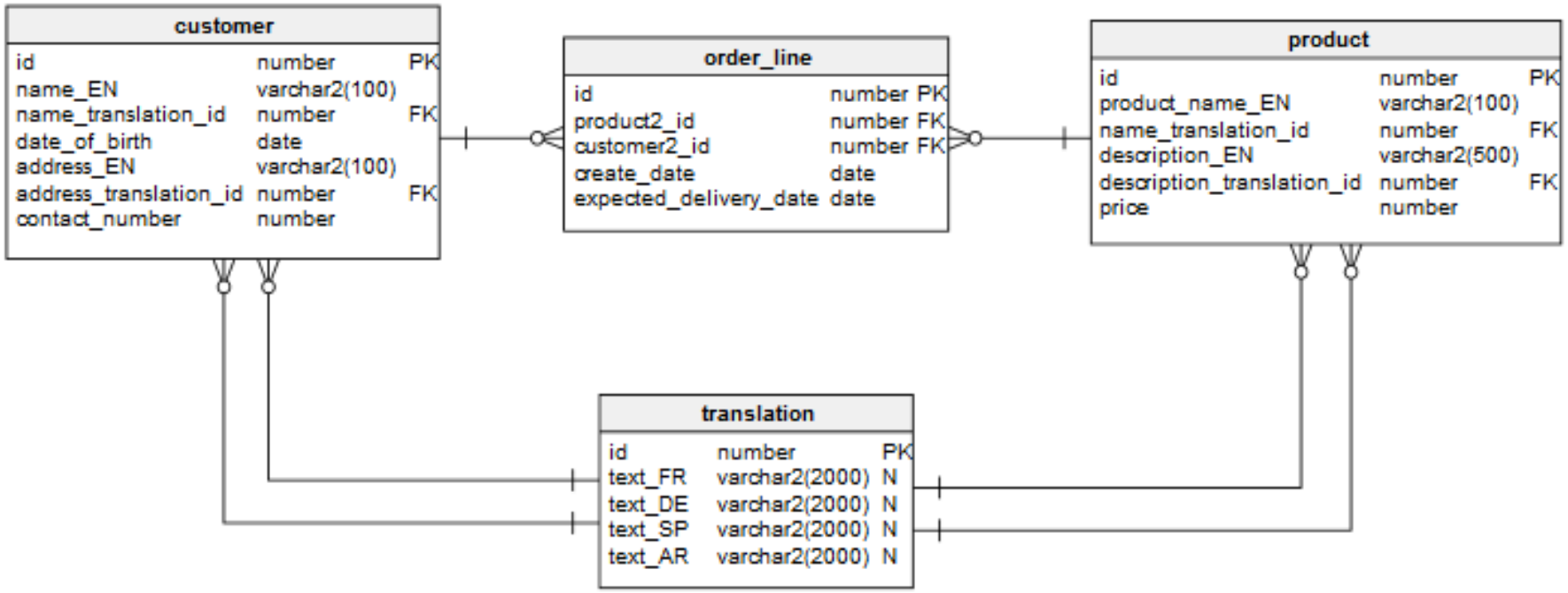
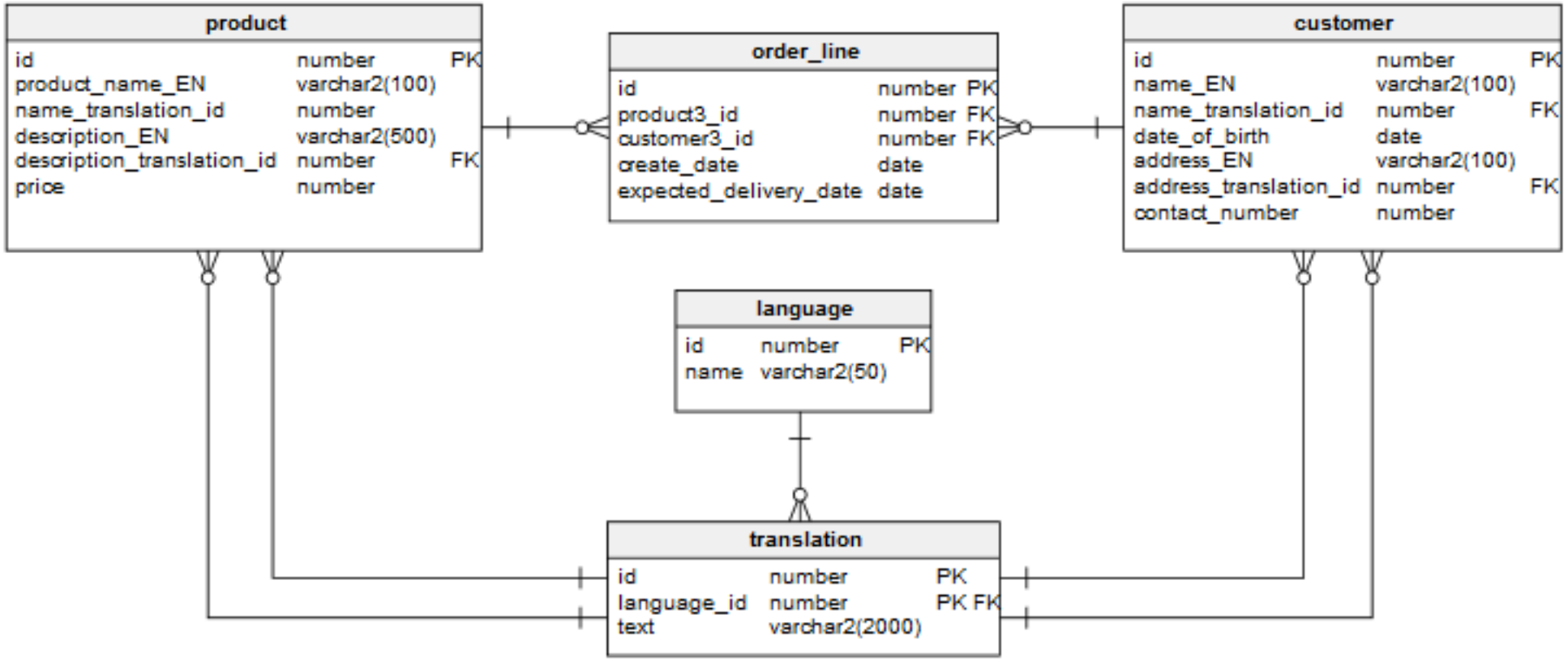


Table Database Localization Method…

- Advantages
  - Good approach if localization is to be implemented on an **existing data model**
  - **Easy to change** when new languages are introduced
- Disadvantages
  - **Still requires a change** in the data model
  - The **complexity of writing SQL increases** as number of joins increases

Row Database Localization Method

- Similar to the previous approach, but it stores the **values for translated text in rows** rather than columns



Row Database Localization Method…

- Advantages
  - **No data model changes** are needed when you add a new language
  - The **complexity** of data-retrieval SQLs is **reduced**
- Disadvantages
  - A **relatively high number of joins** is required to retrieve translated data
  - If application supports a large number of languages, then querying one table for a translation would be **a time-consuming activity**

Clone Database Localization Method

- Creates a **complete clone of the database**
- An **exact copy of the original structure**
  - All **table** and **field names** of the database of the clone, **is the same**
- The databases **differ only in the database name**
  - shop
  - shop_SI
  - shop_TA

Clone Database Localization Method…

- Advantages
  - **No need to change** the database structure
  - **Easy to implement** when new languages are introduced
- Disadvantages
  - **Data redundancy** will be high
  - There will be some **unnecessary memory consumption** due to cloning

Things We Covered

- Part I: Introduction to Localization
  - I18n, L10n and G11n
  - Locale
  - Need of I18n, L10n
  - Contributors
  - Translation
- Part II: Localizing Java Applications
  - Locale in Java Environment
  - ResourceBundle Class
  - Internationalizing a Simple Program
  - Formatting Numbers
  - Formatting Currency
  - Dates and Time Formatting
  - Message Formatting

Things We Covered…

- Part III: Data Modeling for Multiple Languages
  - Database Localization
  - Field Database
  - Table Database
  - Row Database
  - Clone Database
  - Advantages and disadvantages of each type