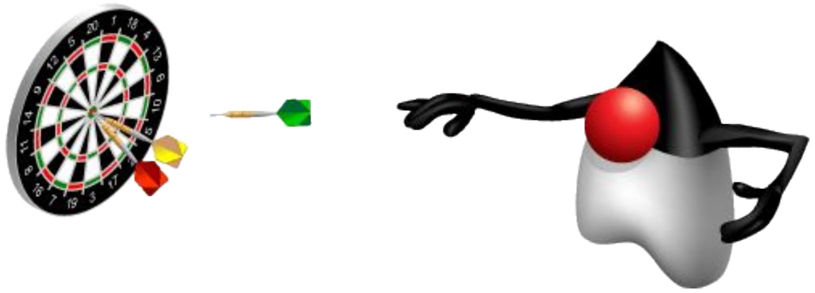
Lesson 14
Localization

Objectives

After completing this lesson, you should be able to:

- Describe the advantages of localizing an application
- Define what a locale represents
- Read and set the locale by using the `Locale` object
- Build a resource bundle for each locale
- Call a resource bundle from an application
- Change the locale for a resource bundle
- Format text for localization by using `NumberFormat` and `DateFormat`



Why Localize?

The decision to create a version of an application for international use often happens at the start of a development project.

- Region- and language-aware software
- Dates, numbers, and currencies formatted for specific countries
- Ability to plug in country-specific data without changing code

A Sample Application

Localize a sample application:

- Text-based user interface
- Localize menus
- Display currency and date localizations

```
=== Localization App ===
1. Set to English
2. Set to French
3. Set to Chinese
4. Set to Russian
5. Show me the date
6. Show me the money!
q. Enter q to quit
Enter a command:
```

`Locale`

A `Locale` specifies a particular language and country:

- Language
    - An alpha-2 or alpha-3 ISO 639 code
    - “en” for English, “es” for Spanish
    - Always uses lowercase
- Country
    - Uses the ISO 3166 alpha-2 country code or UN M.49 numeric area code
    - "US" for United States, "ES" for Spain
    - Always uses uppercase
- See The Java Tutorials for details of all standards used

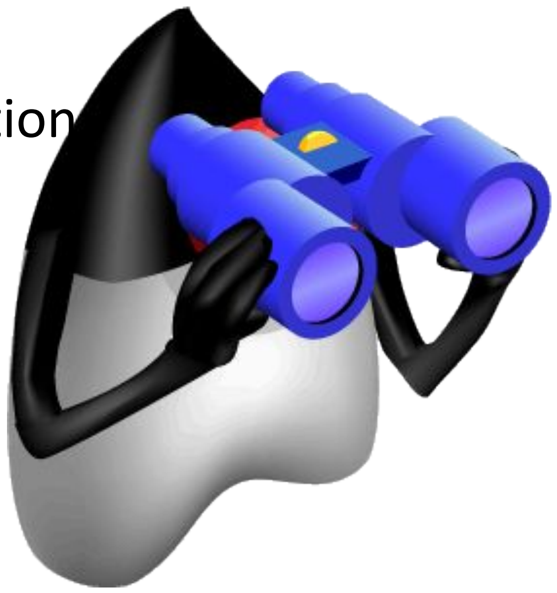
Resource Bundle

The `ResourceBundle` class isolates locale-specific data:

- Returns key/value pairs stored separately
- Can be a class or a `.properties` file

Steps to use:

- Create bundle files for each locale.
- Call a specific locale from your application.



Resource Bundle File

Properties file contains a set of key/value pairs.

- Each key identifies a specific application component.
- Special file names use language and country codes.

Default for sample application:

- Menu converted into resource bundle

MessageBundle.properties

```
menu1 = Set to English
menu2 = Set to French
menu3 = Set to Chinese
menu4 = Set to Russian
menu5 = Show the Date
menu6 = Show me the money!
menuq = Enter q to quit
```

Sample Resource Bundle Files

Samples for French and Chinese

```
MessagesBundle_fr_FR.properties
menu1 = Régler à l'anglais
menu2 = Régler au français
menu3 = Réglez chinoise
menu4 = Définir pour la Russie
menu5 = Afficher la date
menu6 = Montrez-moi l'argent!
menuq = Saisissez q pour quitter
```

```
MessagesBundle_zh_CN.properties
menu1 = 设置为英语
menu2 = 设置为法语
menu3 = 设置为中文
menu4 = 设置到俄罗斯
menu5 = 显示日期
menu6 = 显示我的钱！
menuq = 输入q退出
```

Quiz

Which bundle file represents a language of Spanish and a country code of US?

a. `MessagesBundle_ES_US.properties`
b. `MessagesBundle_es_es.properties`
c. `MessagesBundle_es_US.properties`
d. `MessagesBundle_ES_us.properties`

Initializing the Sample Application

```
PrintWriter pw = new PrintWriter(System.out, true);
    // More init code here

    Locale usLocale = Locale.US;
    Locale frLocale = Locale.FRANCE;
    Locale zhLocale = new Locale("zh", "CN");
    Locale ruLocale = new Locale("ru", "RU");
    Locale currentLocale = Locale.getDefault();

    ResourceBundle messages =
  ResourceBundle.getBundle("MessagesBundle", currentLocale);
  // more init code here

    public static void main(String[] args){
        SampleApp ui = new SampleApp();
        ui.run();
    }
```

Sample Application: Main Loop

```
public void run(){
    String line = "";
    while (!(line.equals("q"))){
        this.printMenu();
        try { line = this.br.readLine(); }
        catch (Exception e){ e.printStackTrace(); }

        switch (line){
            case "1": setEnglish(); break;
            case "2": setFrench(); break;
            case "3": setChinese(); break;
            case "4": setRussian(); break;
            case "5": showDate(); break;
            case "6": showMoney(); break;
        }
    }
}
```

The `printMenu` Method

Instead of text, resource bundle is used.

- `messages` is a resource bundle.
- A key is used to retrieve each menu item.
- Language is selected based on the `Locale` setting.

```
public void printMenu(){
    pw.println("=== Localization App ===");
    pw.println("1. " + messages.getString("menu1"));
    pw.println("2. " + messages.getString("menu2"));
    pw.println("3. " + messages.getString("menu3"));
    pw.println("4. " + messages.getString("menu4"));
    pw.println("5. " + messages.getString("menu5"));
    pw.println("6. " + messages.getString("menu6"));
    pw.println("q. " + messages.getString("menuq"));
    System.out.print(messages.getString("menucommand")+" ");
 }
```

Changing the `Locale`

To change the `Locale`:

- Set `currentLocale` to the desired language.
- Reload the bundle by using the current locale.

```
public void setFrench(){
    currentLocale = frLocale;
    messages =
  ResourceBundle.getBundle("MessagesBundle",
  currentLocale);
}
```

Sample Interface with French

After the French option is selected, the updated user interface looks like the following:

```
=== Localization App ===
1. Régler à l'anglais
2. Régler au français
3. Réglez chinoise
4. Définir pour la Russie
5. Afficher la date
6. Montrez-moi l'argent!
q. Saisissez q pour quitter
Entrez une commande:
```

Format Date and Currency

Numbers can be localized and displayed in their local format.

Special format classes include:

- `DateFormat`
- `NumberFormat`

Create objects using `Locale`.

Initialize Date and Currency

The application can show a local formatted date and currency. The variables are initialized as follows:

```
// More init code precedes
NumberFormat currency;
Double money = new Double(1000000.00);

Date today = new Date();
DateFormat df;
```

Displaying a Date

Format a date:

- Get a `DateFormat` object based on the `Locale`.
- Call the `format` method passing the date to format.

```
public void showDate(){

    df = DateFormat.getDateInstance(DateFormat.DEFAULT,
   currentLocale);
    pw.println(df.format(today) + " " + currentLocale.toString());
}
```

Sample dates:

```
20 juil. 2011 fr_FR
20.07.2011 ru_RU
```

Customizing a Date

`DateFormat` constants include:

- SHORT: Is completely numeric, such as 12.13.52 or 3:30pm
- MEDIUM: Is longer, such as Jan 12, 1952
- LONG: Is longer, such as January 12, 1952 or 3:30:32pm
- FULL: Is completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST

`SimpleDateFormat`:

- A subclass of a `DateFormat` class

Letter	Date or Time	Presentation	Examples
G	Era	Text	AD
y	Year	Year	1996; 96
M	Month in Year	Month	July; Jul; 07

Displaying Currency

Format currency:

- Get a currency instance from `NumberFormat`.
- Pass the `Double` to the `format` method.

```
 public void showMoney(){
    currency = 
  NumberFormat.getCurrencyInstance(currentLocale);
    pw.println(currency.format(money) + " " + 
  currentLocale.toString());
}
```

Sample currency output:

```
1 000 000 руб. ru_RU
1 000 000,00 € fr_FR
￥1,000,000.00 zh_CN
```

Quiz

Which date format constant provides the most detailed information?

a. `LONG`
b. `FULL`
c. `MAX`
d. `COMPLETE`

Summary

In this lesson, you should have learned how to:

- Describe the advantages of localizing an application
- Define what a locale represents
- Read and set the locale by using the `Locale` object
- Build a resource bundle for each locale
- Call a resource bundle from an application
- Change the locale for a resource bundle
- Format text for localization by using `NumberFormat` and `DateFormat`