Localization

Internationalization

What is localization?

- you can make your program adaptable to multiple locales of geographic regions
- this includes:
  - translating string to different languages
  - outputting dates in the correct format
  - outputting numbers in the correct format
  - etc.

```
// picking a locale
Locale myLocale = Locale.getDefault();
System.out.println(myLocale);
  => en_US
```

language (mandatory)

country (optional)

```
System.out.println(Locale.ITALIAN);
  => it
System.out.println(Locale.ITALY);
  => it_IT
System.out.println(new Locale("hi", "IN"));
  => hi_IN
```

```
// creating a locale
Locale myLocale = new Locale.Builder()
  .setLanguage("en")
  .setRegion("US")
  .build();
```

could be in any order

```
// get default locale
System.out.println(Locale.getDefault());
  => en_US

// change default locale
Locale locale = new Locale("fr");
Locale.setDefault(locale);
System.out.println(Locale.getDefault());
  => fr
```

Localizing Numbers

- different countries have different conventions when it comes to numbers
- localization can help to display the number in the appropriate locale format
- for this purpose we use `NumberFormat` factory methods

NumberFormat Factory Methods

Method	Description

`getInstance()`
`getInstance(Locale locale)`
	General purpose formatter

`getNumberInstance()`
`getNumberInstance(Locale locale)`
	Same as `getInstance`

`getCurrencyInstance()`
`getCurrencyInstance(Locale locale)`
	For formatting monetary amounts

`getPercentInstance()`
`getPercentInstance(Locale locale)`
	For formatting percentages

`getIntegerInstance()`
`getIntegerInstance(Locale locale)`
	Rounds decimal numbers before displaying

`getCompactNumberInstance()`
`getInstance(Locale l, Style s)`
	Returns compact number formatter

```
// formatting numbers
double myNum = 1234.568;


var us = NumberFormat.getInstance(Locale.US);
System.out.println(us.format(myNum));
  => 1,234.568


var it = NumberFormat.getInstance(Locale.ITALY);
System.out.println(it.format(myNum));
  => 1.234,568
var ca = NumberFormat.getInstance(Locale.CANADA_FRENCH);
System.out.println(ca.format(myNum));
  => 1 234,568
```

```
// formatting currencies
double price = 12.3;

var us = NumberFormat.getCurrencyInstance(Locale.US);
System.out.println(us.format(price));
  => $12.30

var uk = NumberFormat.getCurrencyInstance(Locale.UK);
System.out.println(uk.format(price));
  => £12.30

var ger = NumberFormat.getCurrencyInstance(Locale.GERMANY);
System.out.println(ger.format(price));
  => 12,30 €
```

```
// formatting percentages
double discount = 0.151;


var us = NumberFormat.getPercentInstance(Locale.US);
System.out.println(us.format(discount));
  => 15%


var ger = NumberFormat.getPercentInstance(Locale.GERMANY);
System.out.println(ger.format(discount));
  => 15 %
```

```
// parsing numbers
public static void main(String args[]) throws ParseException {
  String myNum = "15.72";

  var us = NumberFormat.getInstance(Locale.US);
  System.out.println(us.parse(myNum));
    //  15.72

  var fr = NumberFormat.getPercentInstance(Locale.FRANCE);
  System.out.println(fr.parse(myNum));
    //  throws java.text.ParseException
    //  (in France, decimal point is not a dot, but a comma)
}
```

```
// parsing numbers with currency
public static void main(String args[]) throws ParseException {
  String price = "$12,345.67";

  var us = NumberFormat.getInstance(Locale.US);
  double priceValue = (Double) us.parse(price);
  System.out.println(priceValue);
    //  12345.67
}

// NOTE: if you use non-US like locale, the parsing will throw an exception
// using CompactNumberFormat (new in Java 17!)
int myNum = 8_765_432;
var us1 =  NumberFormat.getCompactNumberInstance(Locale.US, NumberFormat.Style.SHORT);
System.out.println(us1.format(myNum));
  => 9M

var us2 = NumberFormat.getCompactNumberInstance(Locale.US, NumberFormat.Style.LONG);
System.out.println(us2.format(myNum));
  => 9 million

var ger1 = NumberFormat.getCompactNumberInstance(Locale.GERMAN, NumberFormat.Style.SHORT);
System.out.println(ger1.format(myNum));
  => 9 Mio.

var ger2 = NumberFormat.getCompactNumberInstance(Locale.GERMAN, NumberFormat.Style.LONG);
System.out.println(ger2.format(myNum));
  => 9 Millionen
```

DateTimeFormatter Factory Methods

Method	Description

`ofLocalizedDate(`
`  FormatStyle dateStyle)`
	For formating dates

`ofLocalizedTime(`
`  FormatStyle timeStyle)`
	For formating times

`ofLocalizedDateTime(`
`  FormatStyle dateStyle,`
`  FormatStyle timeStyle)`
	For formatting dates and times

`ofLocalizedDateTime(`
`  FormatStyle dateTimeStyle)`
	For formatting dates and times

```
// localizing dates and times

var dtf = DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT);

var fr = new Locale("fr", "FR");

var dt = LocalDateTime.of(2023, Month.SEPTEMBER, 14, 9, 14, 57);

System.out.println(dtf.withLocale(fr).format(dt));

  => 14/09/2023


var dtf2 = DateTimeFormatter.ofLocalizedDate(FormatStyle.LONG);

var us = new Locale("us", "EN");

System.out.println(dtf2.withLocale(us).format(dt));

  => 2023 Sep 14
```

```
// displaying locale
var hr = new Locale("hr", "HR");
var price = 4.32;

System.out.println(hr.getDisplayLanguage());
  => Croatian
System.out.println(hr.getDisplayCountry());
  => Croatia
System.out.println(NumberFormat.getCurrencyInstance(hr).format(price));
  => 4,32 HRK
```