Dates, Strings and Localization

Content

- Java dates and times
- String class
- Localization
- Resource bundles
- Number formatting
- Date formatting

Dates and times

- Most date and time classes are in the java.time package
- LocalDate – just the date
- LocalTime – just the time
- LocalDateTime – date and time
- ZonedDateTime – date, time, zone
- They are immutable

Creating dates and times

- LocalDate.now();
- LocalDate.of(2019, 6, 24);
- LocalDate.of(2019, Month.JUNE, 24);
- LocalTime.of(4, 12); (hour and min)
- LocalTime.of(4, 23, 23, 34); (hour, min, sec, nanosec)

- All the date types have static method now() that returns the current time
- All types have static methods to create the date/times, constructors are private
- Methods are overloaded and there are many ways to create each of them

Exercise: Dates and times

- Create a variable that holds the current time
- Create a variable that holds your birth date
- Create a variable that holds the moment your having a meeting with collegues
- Create a variable that you'll send to an offshore party to keep on online meeting

Time zones

- GMT = Greenwich Mean Time
- UTC = Coordinated Universal Time
- GMT and UTC hold the same zone as 0

- ZoneId zone = ZoneId.of(“US/Eastern”);
- ZonedDateTime z1 = ZonedDateTime.of(2019, 6, 24, 10, 23, 20, 20, zone);

- Calculate time difference between two date/times with different zones: Substract the offset of the time for both dates and calculate the difference between both UTC times

Exercise: how many hours apart?

- 2019-06-20T09:53-7:00
- 2019-06-20T03:45+2:00

- 2019-06-24T19:23+5:30
- 2019-06-25T13:56+2:00

Useful methods

- isBefore
- isAfter
- getDayOfWeek
- getDayOfMonth
- getDayOfYear

Does it compile?

```
LocalDate ld = new LocalDate(2019, 6, 24);
LocalDateTime ld1 = LocalDate.now();
LocalDateTime ld2 = LocalDateTime.of(2019, 6, 24);
LocalTime lt1 = LocalTime.of(2, 23);
LocalTime lt2 = LocalTime.of(2, 61);
ZonedDateTime z2 = ZonedDateTime.now();
ZonedDateTime z3 = ZonedDateTime.now(validZoneId);
```

Manipulating dates

- Assign to variable, else the result will be lost (because immutable)
- Obvious methods:
  - plusYears/minusYears
  - plusMonths/minusMonths
  - plusWeeks/minusWeeks
  - plusDays/minusDays
  - plusHours/minusHours
  - plusMinutes/minusMinutes
  - plusSeconds/minusSeconds
  - plusNanos/minusNanos

Exercise: Manipulating dates

Can you add years to LocalTime?

What happens if you add 24 hours and more than 24 hours to a LocalTime?

What happens if you subtract 100000 years to LocalDate? And add?

What happens if you add a month to a date and the next month has a day less (30 instead of 31)?

Periods

- Data type for storing periods of more than a day
- Period year = Period.ofDays(4); //four days
- Period week = Period.ofWeeks(1); // one week
- Period month = Period.ofMonths(2); //two months
- Period year = Period.ofYears(3); //three years
- Period yearAndMonthAndDays = Period.of(1, 1, 9);//one year, one month and nine days

- If you print the last one it looks like this: P1Y1M9D

- Periods can be used to add to or subtract from days

Exercise: Periods

- Calculate and print the difference in days between your birthday and the current date

- Create and print a period like this: Period.ofMonths(3).ofDays(1)
- Create a period of 2 weeks, what does the toString look like?
- Add a period of one month to February 28th, what happens?
- Add a period of one month to January 30th, what happens?

Duration

- To store time frames of less than a day
- Can store: number of days, hours, minutes, seconds and nanoseconds
- You can only use it for objects that store time (LocalTime, LocalDateTime, ZonedDateTime)
- You can use it to add to or subtract from these objects
- You cannot store both days and hours in a duration, only one unit at once

Duration

- Duration day = Duration.ofDays(1); //PT24H
- Duration day2 = Duration.of(1, ChronoUnit.DAYS); //also for other time units
- Duration hour = Duration.ofHours(1); //PT1H
- Duration minute = Duration.ofMinutes(1); //PT1M
- Duration second = Duration.ofSeconds(1); //PT1S
- Duration milliSec = Duration.ofMillis(1); //PT0.001S

Exercise: Duration

- Make LocalDateTime of the current time
- Create a duration of your favorite number of hours
- Add this to now

- What happens if you add a duration to a LocalDate?

- Bonus: use ChronoUnit the calculate how many days, seconds, weeks old you are

Instants

- Used for representing a specific moment of time
- Used a lot as a timer

- Instant i1 = Instant.now();

//do something else so that it’s later

- Instant i2 = Instant.now();
- Duration d1 = Duration.between(i1, i2);

Exercise: Instants

- Create a ZonedDateTime of the current time in your time zone
- Turn this ZonedDateTime to an instant and print this
- What happens?

- Can you add a day to the instant?
- And an hour?
- And a week?

- What happens if you want to turn a LocalDateTime to an Instant?

Daylight Savings Time

- Changing the time by an hour once a year
- Not all countries do it
- This is automatically managed by Java
- If you ask for a certain moment that does not exist (like 2.30AM when the clock goes forward), it will return 3.30AM

Exercise: Daylight Savings Time

- Exercise: figure out how it deals with the clock turning back, can you figure out whether it’s the first time 1.30AM or second time? E.g. what happens if you add an hour to 1.30AM?

String class - recap

- String class is final
- String objects are immutable
- Java uses String pool to optimize memory
- Instead of the method .concat you can use + between two strings for concatenation

What is the difference between?

- String str1 = “hi there”;
- String str2 = new String(“hi there”);

Exercise: String class

What does this print:

```
System.out.println(“hi” + 1 + 2);
System.out.println(1 + 2 + “hi”);

String s = “ hithere123 ”;
System.out.println(s.trim());
System.out.println(s.length());
System.out.println(s.charAt(2));
System.out.println(s.substring(1,8));
System.out.println(s.replace(‘3’, ‘4’));
System.out.println(s.reverse());
```

Stringbuilder

When you need to change a String a lot it is more efficient to use a StringBuilder, because a StringBuilder is mutable.

If it also needs to be thread safe, you should use StringBuffer

Exercise:

Create the String “hithere” in a loop using a StringBuilder.

When done, change it to uppercase, and put a space between hi and there and reverse it

Internationalization and localization

Internationalization (i18n):

- Design the program in a way it can adapt to the specific country it is in
  - Use property files to get the strings (so that it's possible to have multiple options depending on the language)
  - Use date formatting to use the date format of the user (e.g. avoid conflicts with the order of day and month)

Localization (l10n):

- Process of supporting regions (also called locale)
- Translating strings and using other date formats
- The Locale class stores languages and language/country pairs

Locale

“A specific geographical, political, or cultural region”

For now: just countries or languages, or just languages

Exercise: System.out.println(Locale.getDefault());

You can use setDefault to change the default for the time the program is running (and only for that program)

Format of a local:

lc_CC

Language code = lowercase

Underscore when they’re both present

Country code = uppercase

Locale

Which is the valid locale?

1. en_UK
2. UK
3. EN
4. enUK
5. UK_en

Locale

Locale class has constants for most common combinations.

Exercise:
Find out how to select the locale for Dutch language.
Find out how to select the locale for the Netherlands.

You can also create your own Locale
new Locale(“hi”)
new Locale(“hi”, “TH”)
new Locale.Builder().setLanguage(“hi”).setRegion(“TH”).build()
Also valid: new Locale.Builder().setLanguage(“HI”).setRegion(“th”).build()

Resource bundle

- Contains specific local objects for the program
- Two formats:
  - Property file: file with key/value pairs
  - Java class

Example_en.properties

```
hi=Hi
example=Example
```

Example_nl.properties

```
hi=Hoi
example=Voorbeeld
```

Resource bundle

```
public class Example{
        public static void main(String[] args){
                Locale nl = new Locale(“nl”, “NL”);
                Locale us = new Locale(“en”, “US”);
                printExample(nl);
                printExample(us);
        }
        public static void printExample(Locale l){
                ResourceBundle rb = ResourceBundle.getBundle(“Example”, l);
                System.out.println(rb.getString(“hi”) + “ “ + rb.getString(“example”);
        }
}
```

Property files

- Ways to write a key/value pair (spaces before and after separator are ignored):
  - hi=Hoi
  - hi:Hoi
  - hi Hoi
- # and ! on the beginning of the line mean it’s a comment
- Spaces at the beginning and end of the line are ignored
- End a line with a \ if you want to break the line in your string
- Normal Java escape characters like \n can be used

Properties

```
Properties props = new Properties();
rb.keySet().stream().foreach(k -> props.put(k, rb.getString(k)));

System.out.println(props.getProperty(“hi”));
System.out.println(props.getProperty(“bye”)); //null
System.out.println(props.getProperty(“bye”, “byebye”)); //default value
                                                        byebye used
```

Exercise: Resource bundle

- Make a class person
- Create a resource bundle for person in two languages for printing the welcome and goodbye text, also make a property for saying hi, bye, can i help you and thank you
- Make a method that prints all the key/value pairs for both locales
- (Bonus points if you use streams for this)

Java class resource bundle

- When just strings is not enough, you can create a java class as resource bundle
- Values are created at runtime, could also be objects
- Has a .java extension instead of .properties, naming the same (Name_lc)
- Extends ListResourceBundle
- One method to be implemented: protected Object[][] getContents()
- Creates 2D array

Java class resource bundle

```
public class Example_nl extends ListResourceBundle{
        protected Object[][] getContents(){
                return new Object[][]{{“hi”, “Hoi”}, {“example”, ”Voorbeeld”}};
        }

        public static void main(String[]args){
                ResourceBundle rb = ResourceBundle.getBundle(
                        “resourcebundles.Example”);
                System.out.println(rb.getObject(“hi”) + rb.getObject(“example”));
        }
}
```

Find matching bundle

Steps Java uses to find matching resource file:

- It will look for the .java if file and next for a .properties
- First Java tries to find match on language and region
- If there’s no matching region, it will look for the language only
- If there’s no match on language, it will use default language and region
- If there’s no match on default language and region, it will look for default language only
- If there’s no match on language, but there is a default bundle (Example.java or Example.properties) it will use that one
- If there’s no default resource bundle, it will throw an exception

Find matching bundle for each property

If a property is not in the found resource bundle, Java will look for another resource bundle that is the next close match to the locale to get the property

Find matching bundle

- Methods for getting resource bundle:

  ResourceBundle.getBundle(“name”); //uses default locale

  ResourceBundle.getBundle(“name”, localeName); uses specified locale

Formatting numbers

- Different regions use different formats of certain numbers. For example, currency and dates.
- java.text package has classes to format numbers such as dates and prices

Steps to formatting/parsing numbers:

- Create NumberFormat
- Format/parse the number

Creating NumberFormat

Descriptions	Works without input parameter: uses default as Locale Works with Locale as input parameter
General purpose formatting	NumberFormat.getInstance()
Same	NumberFormat.getNumberInstance()
Monetary / currency formatting	NumberFormat.getCurrencyInstance()
Percentage formatting	NumberFormat.getPercentageInstance()
Rounds decimal values before displaying	NumberFormat.getIntegerInstance()

Formatting and parsing

- On the NumberFormat instance, .parse() and .format() can be called

Exercise: what is the difference between these two?

```
NumberFormat de = NumberFormat.getInstance(Locale.GERMAN);
System.out.println(de.format(longNumber));

NumberFormat us = NumberFormat.getInstance(Locale.US);
System.out.println(us.format(longNumber));
```

Example: format number

See the difference in format for a country in Europe and the US for each of these number formats when formatting a number:

- NumberFormat.getInstance()
- NumberFormat.getNumberInstance()
- NumberFormat.getCurrencyInstance()
- NumberFormat.getPercentageInstance()
- NumberFormat.getIntegerInstance()

Example: parse number

Parse is used to turn a string into a number

Handle ParseException when using parse

```
String str = “1.25”;
System.out.println(de.parse(str));
System.out.println(us.parse(str));
```

What does it do? And what to remember from this?

Formatting dates and times

- Java has a DateTimeFormatter to format date times
- This DateTimeFormatter is in the package java.time.format

- Predefined formats:
  - ISO_LOCAL_DATE/ ISO_LOCAL_TIME / ISO_LOCAL_DATE_TIME
  - DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT)
  - DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)

Example: Formatting dates and times

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

System.out.println(dtf.format(dateTime)); //mm/dd/yy
System.out.println(dtf.format(date); //mm/dd/yy
System.out.println(dtf.format(time); //exception

System.out.println(dateTime.format(dtf)); //mm/dd/yy 10:10 AM
System.out.println(date.format(dtf)); //mm/dd/yy
System.out.println(time.format(dtf)); //exception
```

Create your own date/time format

DateTimeFormatter dtf = DateTimeFormatter.ofPattern(“MMM dd, yyyy, hh:mm”);

Exercise:

- Create your favorite date/time format and print a date/time with this format
- Create the uggliest date/time format you can think of and print a date/time with this format