B

Referentie

B.1 Inleiding

Met deze appendix willen we een paar veel voorkomende methoden oplijsten en en het gebruik er van illustreren. Het is geenszins de bedoeling om volledig te zijn, daarvoor bestaat de Java API. We presenteren hier enkel een selectie van methoden die je frequent zal nodig hebben.

B.2 Typeconversies

Als er iets is wat je als programmeur frequent doet, dan zijn het wel typeconversies. In Java is het op zich vrij eenvoudig:

- tussen primitieve types kan je casten (zie cursus Java Basisconcepten)
- tussen objecttypes die in een overervingsrelatie met elkaar zitten kan je up- of downcasten (zie hoofdstuk 3)
- tussen primitieve types en object types moet je conversiemethoden gebruiken

A deze conversiemethoden zijn statische methoden uit de klassen `String`, `Integer`, `Long`, `Double`, ... Al deze klassen maken deel uit van de `java.lang` package. Merk op dat er een duidelijk verschil is tussen bv. `long`, het primitief type, en `Long`, de klasse. Voor alle primitieve types bestaan er dergelijke klassen, ook wel `wrapper classes` genoemd omdat je ze o.a. kan gebruiken om primitieve waarden mee in te pakken als objecttype.

Om een String naar een primitief getaltype te converteren voorzien de verschillende wrapper classes `parseXXX` methoden:

```
int i = Integer.parseInt("34");
long l = Long.parseLong("23");
double d = Double.parseDouble("45.6");
boolean b = Boolean.parseBoolean("true");
```

Merk op dat al die methoden een `NumberFormatException` kunnen opgooien, wat vrij logisch is aangezien niet alle strings naar een getal om te zetten zijn.

Voor de omgekeerde weg, van primitief naar String, heb je verschillende mogelijkheden. Ofwel maak je gebruik van de verschillende overloads van de `valueOf` methode uit de klasse String, ofwel van de `toString` methoden uit de respectievelijke wrapper classes, ofwel tel je bij je primitieve waarde gewoon een lege string bij:

```
String s1 = String.valueOf(32);
String s2 = Integer.toString(32);
String s3 = 32 + "";
```

Ten slotte hebben we nog van `String` naar `char` en omgekeerd:

```
String s = "hallo";
char c = s.charAt(0);
String s1= String.valueOf(c);
String s2 = Character.toString(c);
String s3 = c + "";
```

B.3 String manipulaties

In de cursus Java Basisconcepten kwamen al een aantal methoden uit de klasse `String` aan bod: `length`, `charAt`, `substring`, `indexOf`, `toUpperCase` en `toLowerCase`. Iets minder frequent gebruikt, maar daarom zeker niet minder nuttig zijn:

- `trim`

  Met de `trim` methode worden alle spaties verwijderd aan het begin en het einde van de tekst waarop ze wordt toegepast.

  ```
  String s = "  hallo ";
  String s1 = s.trim(); // "hallo"
  ```

- `equalsIgnoreCase`

  Deze versie van `equals` maakt geen onderscheid tussen hoofdletters en kleine letters.

```
String s = "hallo";
String s1 =  "HALLO";
if (s.equalsIgnoreCase(s1)) { // true
  // doe iets
}
```

- `replace`

  Met `replace` kan je alle voorkomens van een letter in een tekst veranderen in een andere letter.

```
String s = "mesquite in your cellar";
String s1 =  s.replace('e', 'o');
// resultaat: "mosquito in your collar"
```

- `split`

  Een zeer handige methode om een tekst te splitsen op een bepaald karakter. Het resultaat is een `String[]` die alle afzonderlijke stukjes bevat.

```
String s = "boo:and:foo";
String[] s1 =  s.split(":");
// resultaat: { "boo", "and", "foo" }
```

B.4 Getallen afronden

Om een getal af te ronden tot bv. 2 cijfers na de komma heb je verschillende opties. Je kan het verwezenlijken via een eenvoudige berekening:

```
float unrounded = 3.6353f;
float roundedTwoDecimals = Math.round(unrounded*100)↘
  →/100f;

double unrounded = 3.6353;
double roundedTwoDecimals = Math.round(unrounded * ↘
  →100) / 100.0;
```

Of je maakt gebruik van de `java.text.DecimalFormat` klasse. De `format` methode uit die klasse kan je gebruiken om een decimaal getal met een gegeven formaat op te maken. Volgende code illustreert een paar van die format strings:

```
String myFormattedValue;
double q = 1.0/3.0;
System.out.println ("1.0/3.0 = " + q );

// Define the format pattern in a string
String format = "0.000";

// Create a DecimalFormat instance for this format
DecimalFormat df = new DecimalFormat(format);

// Create a string from the double according to the ↘
   →format
myFormattedValue = df.format(q);
System.out.println ("1.0/3.0 = " + myFormattedValue )↘
   →;

// Can change the format pattern :
df. applyPattern("0.00000");
myFormattedValue = df.format(q);
System.out.println ("1.0/3.0 = " + myFormattedValue )↘
   →;
myFormattedValue = df.format(1.0/2.0);
System.out.println ("1.0/2.0 = " + myFormattedValue )↘
   →;

// The # symbol indicates trailing blanks
df.applyPattern("0.######");
myFormattedValue = df.format(1.0/2.0) ;
System.out.println("1.0/2.0 = " + myFormattedValue + ↘
   →"!" );

// Use of the E-notation (scientific notation)
df.applyPattern("0.00E0");
myFormattedValue = df.format(1000.0/3.0) ;
System.out.println("1000.0/3.0 = " + myFormattedValue↘
   → );

// output
1.0/3.0 = 0.3333333333333333
1.0/3.0 = 0.333
1.0/3.0 = 0.33333
1.0/2.0 = 0.50000
1.0/2.0 = 0.5!
1000.0/3.0 = 3.33E2
```