Eenvoudige klassen

Wrappers

- Wrapper klassen.
- Autoboxing.
- Static members.
  - ➔ De wrapper klassen hebben een aantal static members
    - **constanten**

primitive types kunnen gewrapt worden.

- ➔ Het primitieve datatype zal dan als het ware ingepakt worden in een object.
- ➔ Deze wrapper klassen maken het mogelijk om primitieve datatypes te behandelen als objecten.

Klasse	*Primitief datatype*
`Byte`	`byte`
`Short`	`short`
`Integer`	`int`
`Long`	`long`
`Float`	`float`
`Double`	`double`
`BigInteger`	Geheel getal met onbeperkte precisie.
`BigDecimal`	Decimaal getal met onbeperkte precisie.
`Boolean`	`boolean`
`Character`	`char`

Constante properties

Constante	*Betekenis*
`Integer.MAX_VALUE`	De maximale integer-waarde.
`Integer.MIN_VALUE`	De minimale integer-waarde.
`Float.NaN`	*Not a number*. Geen getal.
`Float.NEGATIVE_INFINITY`	- ∞ Min oneindig.
`Float.POSITIVE_INFINITY`	+ ∞ Plus oneindig.
`Double.NaN`	*Not a number*. Geen getal
`Double.NEGATIVE_INFINITY`	- ∞ Min oneindig.
`Double.POSITIVE_INFINITY`	+ ∞ Plus oneindig.

Static methoden

Methode	*Betekenis*
`Integer.parseInt()`	Zet een *string* om naar een integer.
`Float.parseFloat()`	Zet een *string* om naar een *float*-object.
`Double.parseDouble()`	Zet een *string* om naar een *double*-object.
`Float.isNaN()`	Gaat na of de waarde een ongeldig getal is.
`Double.isNaN()`	Gaat na of de waarde een ongeldig getal is.
`Float.isInfinite()`	Gaat na of de waarde oneindig is.
`Double.isInfinite()`	Gaat na of de waarde oneindig is.

Equals vs ==

Verder merken we nog op dat *wrapper*-objecten net als *string*-objecten onveranderbaar zijn en dat ze een eigen implementatie hebben van de methode `equals()`. Om dergelijke getallen te vergelijken, gebruik je dus best deze methode en niet het ==-teken.

```
Integer value1 = new Integer(5);
Integer value2 = new Integer(5);
System.out.println(value1 == value2);       // false
System.out.println(value1.equals(value2)); // true
```

Autoboxing

12.2.2 Autoboxing

Sinds Java 5 gebeurt de omzetting tussen een primitief datatype en het *wrapper*-object automatisch. Men noemt dit *autoboxing*. Indien een object verwacht wordt en een primitief type gegeven is, zal de compiler dit automatisch inpakken in zijn overeenkomstig *wrapper*-object. Omgekeerd, indien een primitief type verwacht wordt en een *wrapper*-object gegeven is, zal de compiler van dit object de primitieve waarde opvragen.

Dit maakt het de programmeur aanzienlijk makkelijker.

Voorbeeld:

```
public class AutoBoxing {
   public static void main(String[] args) {
      Integer intObject = 5;
      int intPrimitive = new Integer(6);
      System.out.println(intObject);
      System.out.println(intPrimitive);
   }
}
```

Vanwege de snelheid is het echter aangewezen zoveel mogelijk gebruik te maken van de primitieve datatypes en enkel de *wrapper*-objecten te gebruiken indien nodig.

Datums en tijden

- Voor java 8 **!=** succes.
  - ➔ De klasse Date.
  - ➔ De klasse Calendar.
- De klasse Calendar.
  - ➔ Beware of danger ahead.
- Sinds Java 8
  - ➔ Compleet nieuwe klasse, voor optimaal comfort!

Computer time

- Heeft geen besef van jaren maanden dagen etc.

  → Enkel van fundamentele tijdseenheden.
    - Seconden
    - Milliseconden
    - Nanoseconden

  → Een tijdstip wordt uitgedrukt in een aantal tijdseenheden voor of na een bepaald **nulpunt**

    - EPOCH: 1 januari 1970 00:00:00u.

De klasse Instant

- instanties van deze klasse zijn onveranderlijk.

Methode	*Betekenis*
`long getEpochSecond()`	Geeft het aantal seconden sinds de EPOCH.
`int getNano()`	Geeft het aantal nanoseconden sinds de laatste seconde.
`boolean isAfter()`	Gaat na of dit tijdstip later is dan het opgegeven tijdstip.
`boolean isBefore()`	Gaat na of dit tijdstip vroeger is dan het opgegeven tijdstip.
`Instant minusSeconds()`	Geeft een nieuwe instantie terug die een aantal seconden vroeger ligt.
`Instant plusSeconds()`	Geeft een nieuwe instantie terug die een aantal seconden later ligt.

Tabel 40: Methoden van de klasse Instant

Methoden Instant.

De klasse heeft tevens een aantal statische eigenschappen die constanten bevatten:

Eigenschap	*Betekenis*
`static Instant EPOCH`	Het tijdstip van de EPOCH.
`static Instant MAX`	Het laatste tijdstip dat met deze klasse voorgesteld kan worden.
`static Instant MIN`	Het vroegste tijdstip dat met deze klasse voorgesteld kan worden.

Tabel 41: Eigenschappen van de klasse Instant

Human time.

- Niet in nanoseconden.
- Dagen en maanden betekenen niets voor een computer.

Enum DayOfWeek

Methode	*Betekenis*
`int length(boolean l)`	Geeft het aantal dagen in de maand. Als argument moet je meegeven of het een schrikkeljaar is of niet
`int maxLength()`	Het maximaal aantal dagen in deze maand.
`int minLength()`	Het minimaal aantal dagen in deze maand.
`Month plus(long m)`	Geeft de nieuwe maand als we bij deze maand een aantal maanden optellen
`Month minus(long m)`	Geeft de nieuwe maand als we van deze maand een aantal maanden aftrekken.
`getDisplayName()`	Geeft de naam van de maand weer volgens de lokale gebruiken.

Tabel 42: Methoden van de enumeratie Month

Hetzelfde geldt voor de weekdagen die opgesomd zijn in de enumeratie `DayOfWeek` met instanties als `DayOfWeek.MONDAY` enz... Ook hier zijn er een aantal interessante methoden:

Methode	*Betekenis*
`DayOfWeek plus(long m)`	Geeft de nieuwe weekdag als we bij deze weekdag een aantal dagen optellen
`DayOfWeek minus(long m)`	Geeft de nieuwe weekdag als we van deze weekdag een aantal dagen aftrekken.
`getDisplayName()`	Geeft de naam van de dag weer volgens de lokale gebruiken.

Lokale Datums en tijden.

- ➔ LocalTime
- ➔ LocalDate
- ➔ LocalDateTime
- ❑ Deze bovenstaand klassen houden rekening met de lokale tijdrekening;
- ❑ Deze klassen worden niet met een constructor geïnitialiseerd maar met een statische methode
  - ➔ **now( )**

Tijdzones

- ZoneId
  - ➔ Instanties maken op basis van een regio
- ZoneOffset
  - ➔ Subklasse ZoneId
  - ➔ Instanties maken door tijdsverschil met UTC/Greenwich mee te geven

Lees de documentatie

Datums en tijden met tijdzones.

Student.takeMegaBook(219);

Tijdsduur

- Duration
  - ➔ Wordt gebruikt voor verschillen in machinetijden.
- ChronoUnit
  - ➔ Enumeratie die de verschillende tijdseenheden bevat.
  - ➔ Een tijdsverschil kan worden omgezet in een eenheden met de methode between( )
- Period
  - ➔ Wordt gebruikt voor verschillen in mensentijden.

Omzetting van en naar Date en Calendar

```
Calendar.toInstant()
GregorianCalendar.toZonedDateTime()
GregorianCalendar.from(ZonedDateTime)
Date.from(Instant)
Date.toInstant()
```

Formattering van datums en tijden

Hiervoor hebben we het object **DateTimeFormatter**.