HO
GENT

Werken met bestanden

Table of Contents

1. Doelstellingen . . . 1
2. Inleiding . . . 1
3. Files en streams . . . 1
4. Streams . . . 2
5. De klasse Files . . . 3
6. Werken met tekstbestanden . . . 4
    6.1. Schrijven naar een tekstbestand . . . 4
    6.2. Lezen van een tekstbestand . . . 5
    6.3. Openen, verwerken en sluiten van een file via try-with-resources . . . 7
    6.4. Lezen van een tekstbestand via Streams . . . 7
    6.5. Samengevat . . . 8
7. Binair bestand - serialisatie . . . 9
    7.1. Schrijven naar een binair bestand - serialisatie . . . 9
    7.2. Lezen van een geserialiseerd bestand - deserialisatie . . . 10
    7.3. Samengevat . . . 12

1. Doelstellingen

- Kan een tekstbestand lezen
- Kan een tekstbestand schrijven
- Kan objecten serialiseren
- Kan een geserialiseerd bestand deserialiseren

2. Inleiding

- Lange termijn opslag van grote hoeveelheden gegevens
- Persistente gegevens (blijven bestaan na beëindiging van een programma)
- Opgeslagen op secundaire geheugenmedia:
  - Magnetische schijven of tapes
  - Optische schijven
- Bestaan uit records (gerelateerde velden)
- Organisatie van de records geeft
  - Sequentiële files
  - Random access files

Bij een sequentiële file doorloop je de gegevens van voor naar achter. Het duurt dus langer om een gegeven dat zich achteraan in het bestand bevindt, te bereiken.

Bij een random access file krijg je directe toegang tot de gegevens (daarom ook wel direct access genoemd). De tijd om het gegeven te bereiken is dus even groot, om het even op welke positie het gegeven zich bevindt.

3. Files en streams

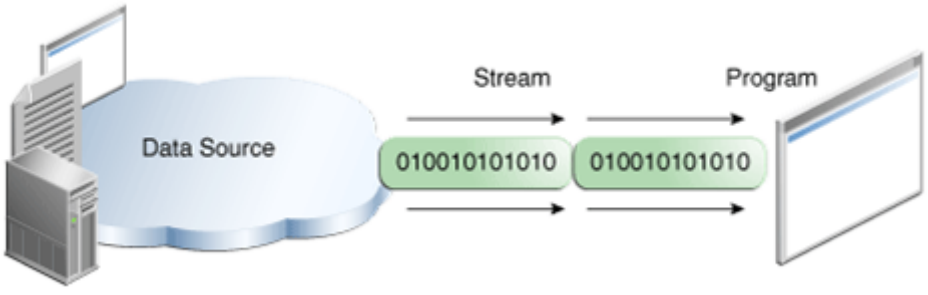
- Java ziet elke file als een stream van bytes.

- Elk besturingssysteem bepaalt het einde van een file door:
  - end-of-file merkteken
  - het totaal aantal bytes van de file bij te houden
- Een java programma krijgt een signaal van het besturingssysteem wanneer het programma het einde van een stream bereikt door middel van
  - een EOFExceptie of

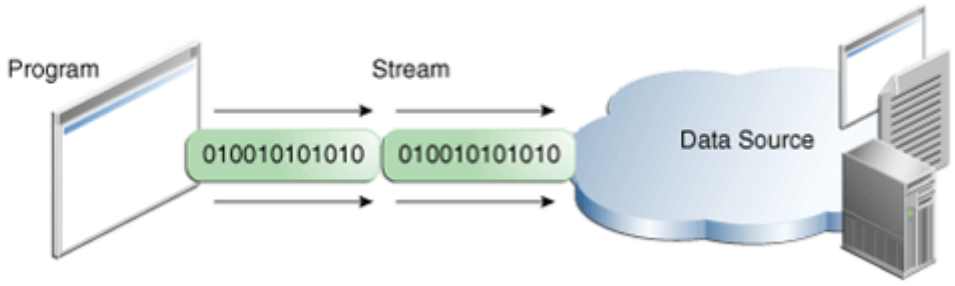
    - een specifieke returnwaarde van een methode
- Streams ondersteunen meerdere vormen van gegevens: van eenvoudige bytes tot geavanceerde objecten.
    - Streams die bytes in- en uitvoeren zijn byte-based streams, de gegevens worden voorgesteld in een binair format vb. 2 bytes voor een char, 4 bytes voor een int, 8 bytes voor een double, …
      ⇒ Files die gecreëerd worden door gebruik te maken van byte-based streams zijn **binary** files.
    - Streams die karakters in- en uitvoeren zijn character-based streams, de gegevens worden voorgesteld als een sequentie van karakters. vb. Het getal 1 miljoen bevat 7 cijfers (een 1 gevolgd door 7 nullen), dus dit zijn 7 karakters, die elk 2 bytes innemen: in totaal zijn dus 14 bytes nodig om dit getal op te slaan. Als je daarentegen het getal 8 wilt opslaan, dan is dat slechts 1 karakter, dat dus slechts 2 bytes inneemt.
      ⇒ Files die gecreëerd worden door gebruik te maken van character-based streams zijn **text** files.

4. Streams

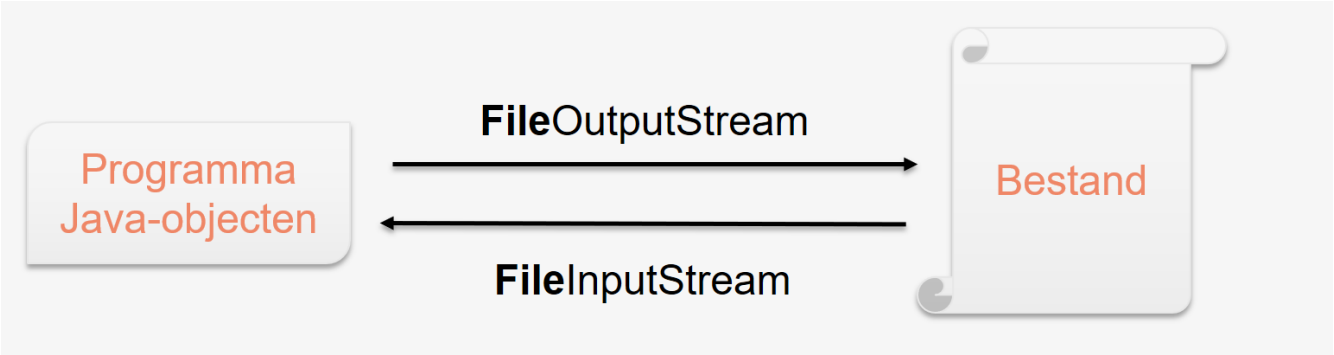
- Een stream is een manier om allerlei vormen van gegevenstransport in een computer of tussen computers voor te stellen: gegevens die via het toetsenbord binnenkomen, of via een netwerkkabel, of gegevens die via twee programma's worden uitgewisseld.
- Java associeert streams met devices:
    - System.in: standaard input stream object, laat toe om bytes via het toetsenbord in te lezen
    - System.out: standaard output stream object, laat toe om bytes weer te geven op het scherm
    - System.err: standaard error stream object, laat toe om foutboodschappen weer te geven op het scherm
- Een programma gebruikt een input stream om gegevens te lezen van een source, één item per keer:



- Een programma gebruikt een output stream om gegevens te schrijven naar een bestemming, één item per keer:



- **FileInput**- en **FileOutputStream** specifiek voor het werken met bestanden (zie package java.io).



5. De klasse Files

- Sinds Java SE 6 kunnen we naast de package java.io ook gebruik maken van java.nio, de nieuwe I/O API:
- De klasse **Files** (package java.nio.file) vervangt de klasse File.
  Ze is efficiënter en bevat onder andere static methodes om:
  - Bestanden en directories aan te maken en te verwijderen.
  - Eigenschappen van bestanden en directories op te vragen en aan te passen.
  - Streams aan te maken om bestanden te lezen en te schrijven – zie verder.
  - Voorbeelden van bewerkingen:
    - Path f = Files.createFile(“MyCode.txt”);
    - Files.delete(f);
  - Compatibiliteit met de klasse File
    Wanneer een programma toch nog gebruik moet maken van File-objecten, kunnen we een Path omzetten naar een File met de methode **toFile()** van de interface Path.
    Ook de omgekeerde bewerking is mogelijk met de methode **toPath()** van de klasse File.
  - Path Strings in verschillende besturingssystemen
    Als je het Path opgeeft naar een bepaalde file of directory, dan gebruik je een bepaalde separator. In Windows is dit een backslash (\), maar in Linux of op MacOS is het een gewone slash (/).
    Om geen problemen te veroorzaken, kan je best het scheidingsteken van het lokale systeem opvragen via **File.separator**

6. Werken met tekstbestanden

- Tekstbestanden zijn **leesbare** bestanden
- Java legt geen structuur op aan een file:
  - Java kent geen records
  - Bouw zelf de structuur van een file, rekening houdend met de requirements van de applicatie

6.1. Schrijven naar een tekstbestand

- Maakt gebruik van **Formatter**
  - De klasse Formatter helpt om gegevens onder de vorm van geformatteerde data te converteren naar de interne datatypen
  - Formatter maakt gebruikt van streams → OutputStream
    - Voorbeeld: System.out
    - Voorbeeld: **Files.newOutputStream(Paths.get(bestandsnaam))** of **Files.newOutputStream(Paths.get(bestandsnaam),StandardOpenOption.APPEND)** // achteraan toevoegen
- Als het bestand nog niet bestond, wordt het gemaakt ENKEL als de tweede parameter StandardOpenOption.APPEND er niet bij staat.

```
        try
        {
            output = new Formatter(Files.newOutputStream(Paths.get
("src\\tekstbestand\\clients.txt"),StandardOpenOption.APPEND)); ①
        }
        catch (InvalidPathException ie) ②
        {
            System.err.println("Error finding file.");
            System.exit(1);
        }
        catch (IOException ex) ③
        {
            System.err.println("Error creating file.");
            System.exit(1);
        }
```

① clients.txt wordt klaargezet om naar te schrijven: er wordt een output stream gekoppeld aan het bestand om bytes ernaar te schrijven.
Een Formatter-object wordt aan de gespecifieerde output stream gekoppeld; hierdoor wordt alles via het Formatter-object naar de file weggeschreven.

② InvalidPathException
Het opgegeven pad klopt niet

③ IOException
In geval je geen schrijfrechten hebt voor de file. De file wordt niet gevonden of kan niet gecreëerd worden.

```
        try
        {
            output.format("%d %s %s %.2f%n", record.getAccount(), ①
                    record.getFirstName(), record.getLastName(),
                    record.getBalance());
        } catch (FormatterClosedException formatterClosedException) ②
        {
            System.err.println("Error writing to file.");
        }
```

① via format-methode wordt de data van het record op de gewenste manier weggeschreven

② FormatterClosedException
In geval Formatter reeds afgesloten (close) terwijl je schrijft.

6.2. Lezen van een tekstbestand

- Maakt gebruik van **Scanner**
  - De klasse Scanner helpt om gegevens te converteren.
  - Scanner maakt gebruikt van streams → InputStream
    - Voorbeeld: System.in
    - Voorbeeld: **Files.newInputStream(Paths.get(bestandsnaam))**
  - Scanner splitst de inputstream op in tokens gescheiden door whitespace karakters.
  - Enkele methoden van Scanner:
    - hasNext(), hasNextInt() geven een boolean terug
    - next(), nextLine() geven een String terug
    - nextInt() geeft een int terug
    - …

```
        try
        {
            input = new Scanner(Files.newInputStream(Paths.get
("src\\tekstbestand\\clients.txt"))); ①
        }
        catch (InvalidPathException ie) ②
        {
            System.err.println("Error finding file.");
            System.exit(1);
        }
        catch (IOException ex) ③
        {
            System.err.println("Error opening file.");
            System.exit(1);
        }
```

① clients.txt wordt klaargezet om te lezen: er wordt een input stream gekoppeld aan het bestand om bytes te lezen.
Een Scanner-object wordt aan de gespecifieerde input stream gekoppeld; hierdoor wordt alles via het Scanner-object uit de file gelezen.

② InvalidPathException
Het opgegeven pad klopt niet

③ IOException
Het opgegeven bestand wordt niet gevonden.

```
        try
        {
            while (input.hasNext())
            {
                lijst.add(new AccountRecord(input.nextInt(), input.next(), input.
next(), input.nextDouble())); ①
            }
        } catch (InputMismatchException elementException) ②
        {
            System.err.println("File improperly formed.");
            input.close();
            System.exit(1);
        } catch (NoSuchElementException elementException) ③
        {
            System.err.println("Element missing");
            input.close();
            System.exit(1);
        } catch (IllegalStateException stateException) ④
        {
            System.err.println("Error reading from file.");
            System.exit(1);
        }
```

① via de gepaste next-methode wordt de data van het record uitgelezen

② InputMismatchException
Indien de organisatie/type gegevens niet overeenstemmen.

③ NoSuchElementException
Er ontbreken elementen.

④ IllegalStateException
In geval van lezen terwijl Scanner reeds gesloten is.

6.3. Openen, verwerken en sluiten van een file via try-with-resources

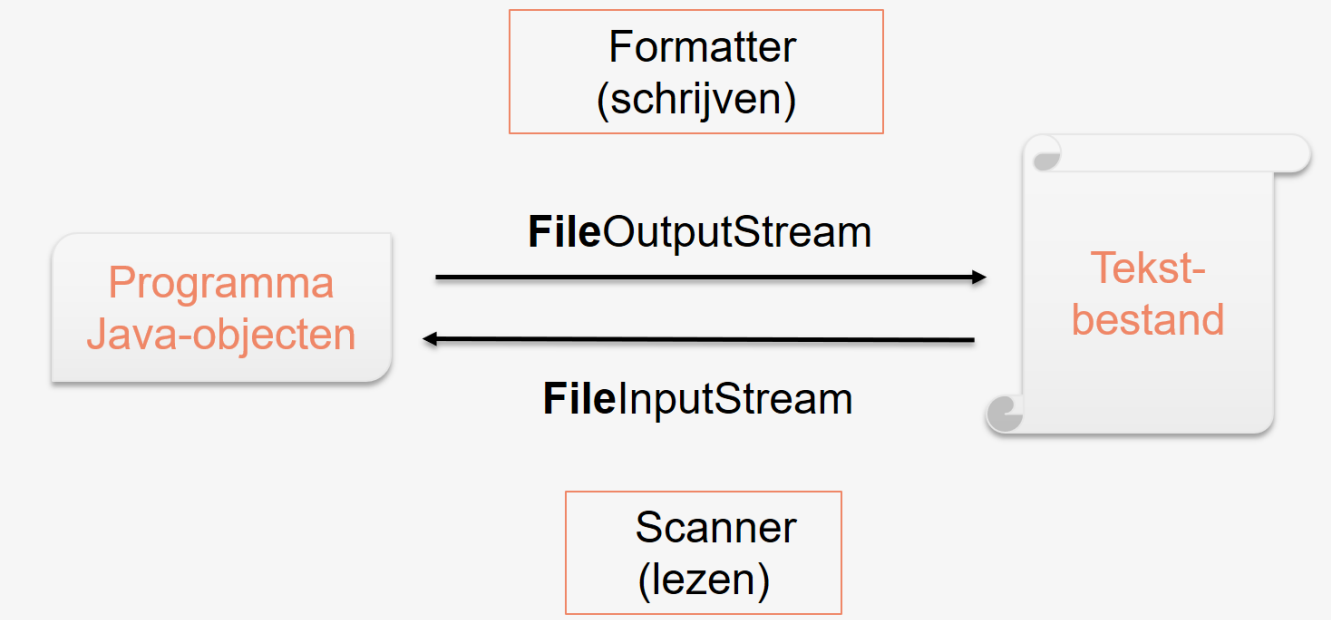
- try **(Formatter out = new Formatter(.....))**
  { //Werken met de stream }
  catch(IOException ex)
  { //De gepaste exceptions opvangen en afhandelen }
- In deze notatie openen we de stream tussen de haakjes van de try-structuur. De stream die we hier openen, zal automatisch worden afgesloten na afloop van de try-structuur, ook wanneer er exceptions optreden.

6.4. Lezen van een tekstbestand via Streams

```
        try(Stream<String> lines = Files.lines(Path.of
("src\\tekstbestand\\clients.txt")))
        {
            lijst = lines.map(line ->
                        {
                            String[] arr = line.split(" ");
                            return new AccountRecord(Integer.parseInt(arr[0]), arr[1],
arr[2], Double.parseDouble(arr[3].replace(',', '.')));
                        })
                    .collect(Collectors.toList());
        } catch (InputMismatchException elementException) ②
        {
            System.err.println("File improperly formed.");
            input.close();
            System.exit(1);
        } catch (NoSuchElementException elementException) ③
        {
            System.err.println("Element missing");
            input.close();
            System.exit(1);
        } catch (IllegalStateException stateException) ④
        {
            System.err.println("Error reading from file.");
            System.exit(1);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

6.5. Samengevat

- Om een tekstbestand te kunnen lezen en/of schrijven moet je in het algemeen deze 3 stappen doorlopen:
  - Openen bestand om te lezen of te schrijven
  - Bewerkingen uitvoeren
  - Sluiten van het geopende bestand
- De mogelijke exceptions veroorzaakt door de vorige handelingen opvangen.
- Schematisch:



7. Binair bestand - serialisatie

- Om een object te lezen uit of te schrijven in een file, voorziet Java object serialisatie.
- Een geserialiseerd object wordt voorgesteld door een sequentie van bytes die naast de **data** ook het **type** van de informatie van het object bevat.

7.1. Schrijven naar een binair bestand - serialisatie

- Objecten van klassen die de **interface Serializable** implementeren, kunnen geserialiseerd worden via **ObjectOutputStream**.
- Serializable is een tagging interface ⇒ bevat geen abstracte methodes; klassen die deze interface implementeren behoren tot een bepaalde set van klassen. Als een klasse de interface Serializable implementeert, is het lid van de Serializable klassen.
- Alles wat **transient** wordt gedeclareerd, zal genegeerd worden tijdens het serialisatieproces.
- Alle primitieve-type variabelen zijn serializable.

```
        try
        {
            output = new ObjectOutputStream(Files.newOutputStream(
                    Paths.get("src\\serialisatie\\accounts.ser"))); ①

        } catch (InvalidPathException ie) ②
        {
            System.err.println("Error finding file.");
            System.exit(1);
        } catch (IOException ex) ③
        {
            System.err.println("Error opening file.");
            System.exit(1);
        }
```

① accounts.ser wordt klaargezet om naar te schrijven: er wordt een output stream gekoppeld aan het bestand om bytes ernaar te schrijven.
Een ObjectOutputStream-object wordt aan de gespecifieerde output stream gekoppeld; hierdoor wordt alles via het ObjectOutputStream-object geserialiseerd.

② InvalidPathException
Het opgegeven pad klopt niet

③ IOException
In geval je geen schrijfrechten hebt voor de file. De file wordt niet gevonden of kan niet gecreëerd worden.

```
        try
        {
            output.writeObject(record); ①
        } catch (IOException ex) ②
        {
            System.err.println("Error writing to file.");
        }
```

① via writeObject-methode wordt het record geserialiseerd

② IOException
In geval iets fout loopt bij het serialiseren.

7.2. Lezen van een geserialiseerd bestand - deserialisatie

- Nadat een geserialiseerd object in een file geschreven is, kan het gelezen worden uit de file en gedeserialiseerd om het object opnieuw in het geheugen te creëren.
- Objecten van klassen die de interface **Serializable** implementeren, kunnen gedeserialiseerd worden via **ObjectInputStream**.

```
        try
        {
            input = new ObjectInputStream(Files.newInputStream(
                    Paths.get("src\\serialisatie\\accounts.ser"))); ①
        } catch (InvalidPathException ie) ②
        {
            System.err.println("Error finding file.");
            System.exit(1);
        } catch (IOException io) ③
        {
            System.err.println("Error opening file.");
            System.exit(1);
        }
```

① accounts.ser wordt klaargezet om te deserialiseren (= uitlezen en objecten terug opbouwen): er wordt een input stream gekoppeld aan het bestand om bytes uit te lezen.
Een ObjectInputStream-object wordt aan de gespecifieerde input stream gekoppeld; hierdoor wordt alles via het ObjectInputStream-object gedeserialiseerd.

② InvalidPathException
Het opgegeven pad klopt niet

③ IOException
De file wordt niet gevonden.

```
                AccountRecord record = (AccountRecord) input.readObject(); ①
```

① via readObject-methode wordt het record gedeserialiseerd. Teruggeefwaarde van readObject() is een Object, vandaar de downcasting.

- Deserialiseren van een objecten die één per een werden geserialiseerd:

```
        try
        {
            while (true)
            {
                AccountRecord record = (AccountRecord) input.readObject(); ①
                lijst.add(record);

            }
        } catch (EOFException e) ②
        {
        }catch (ClassNotFoundException ex) ③
        {
            System.err.print("ongeldige objectstream");
            System.exit(1);
        } catch (IOException e) ④
        {
            System.err.println("Error reading file.");
            System.exit(1);
        }
        return lijst;
```

① via readObject-methode wordt het record gedeserialiseerd en toegevoegd aan de lijst

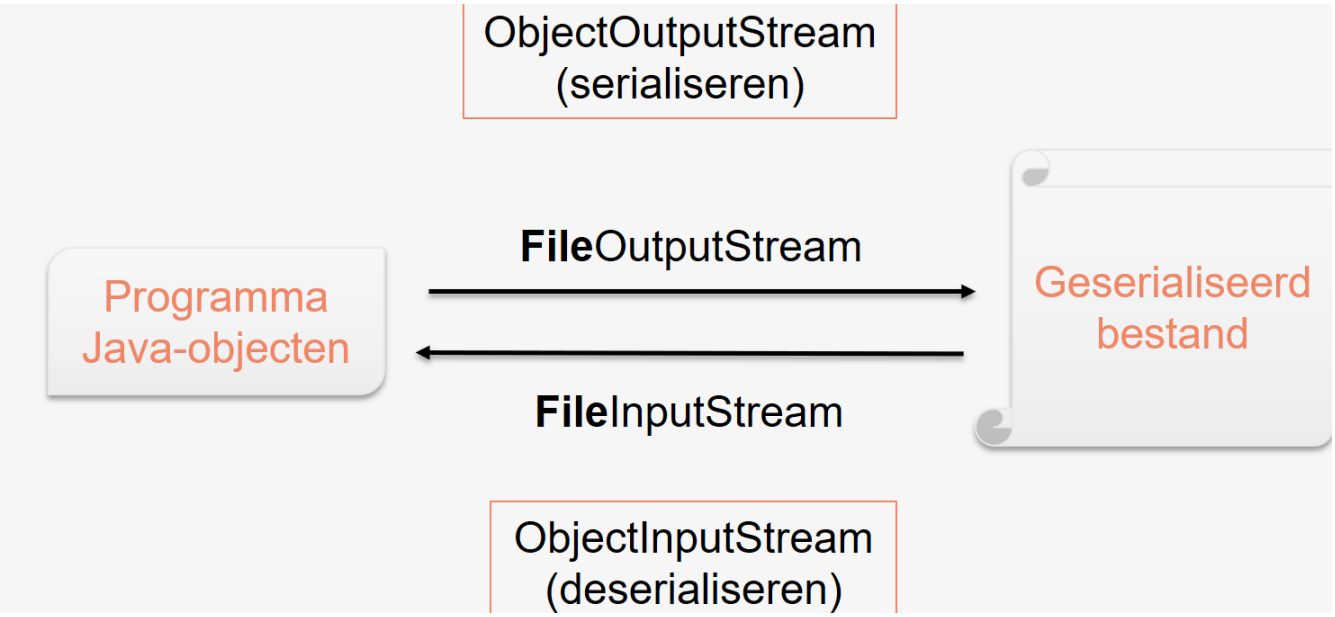
② EOFException
Bij het bereiken van het einde van het bestand wordt een EOFException gegooid.

③ ClassNotFoundException
Indien iets fout loopt bij het downcasten.

④ IOException
De file wordt niet gevonden.

7.3. Samengevat

- Om een binair bestand te kunnen lezen en/of schrijven moet je in het algemeen deze 3 stappen doorlopen:
  - Openen bestand om te serialiseren of te deserialiseren
  - Bewerkingen uitvoeren
  - Sluiten van het geopende bestand
- De mogelijke exceptions veroorzaakt door de vorige handelingen opvangen.
- Schematisch:


ObjectOutputStream
(serialiseren)
Programma
Java-objecten
FileOutputStream
FileInputStream
Geserialiseerd
bestand
ObjectInputStream
(deserialiseren)