Databanken en webtoepassingen in Java

K. Coolsaet

Universiteit Gent

20 maart 2017

WiFi — UGentGuest
Gebruikersnaam: `guestM3java`
Wachtwoord: `ag4oZnpQ`
Eerst surfen naar http://www.ugent.be

Cursuswebsite: `http://inigem.ugent.be/moevie.html`

En toen ging het fout...

Uitzonderingen — *Exceptions*

Als er iets fout gaat in een programma *gooit* Java een *uitzondering op*

- Delen door 0

  ```
  java.lang.ArithmeticException: / by zero
  ```

- Ongeldige array-index

  ```
  java.lang.ArrayIndexOutOfBoundsException: 3
  ```

- Ongeldige lijst-index

  ```
  java.lang.IndexOutOfBoundsException: Index: 3, Size: 0
  ```

- Object bevat verwijzing ‘naar nergens’

  ```
  java.lang.NullPointerException
  ```

En toen ging het fout...

Uitzonderingen — *Exceptions*

Als er iets fout gaat in een programma *gooit* Java een *uitzondering op*

- Delen door 0

  ```
  java.lang.ArithmeticException: / by zero
  ```

- Ongeldige array-index

  ```
  java.lang.ArrayIndexOutOfBoundsException: 3
  ```

- Ongeldige lijst-index

  ```
  java.lang.IndexOutOfBoundsException: Index: 3, Size: 0
  ```

- Object bevat verwijzing ‘naar nergens’

  ```
  java.lang.NullPointerException
  ```

- **In de praktijk:** vaak invoer/uitvoer-fouten (bestanden, databank, netwrk, . . .)

En toen ging het fout...

Uitzonderingen — *Exceptions*

Als er iets fout gaat in een programma *gooit* Java een *uitzondering op*

- Delen door 0

  ```
  java.lang.ArithmeticException: / by zero
  ```

- Ongeldige array-index

  ```
  java.lang.ArrayIndexOutOfBoundsException: 3
  ```

- Ongeldige lijst-index

  ```
  java.lang.IndexOutOfBoundsException: Index: 3, Size: 0
  ```

- Object bevat verwijzing 'naar nergens'

  ```
  java.lang.NullPointerException
  ```

- **In de praktijk:** vaak invoer/uitvoer-fouten (bestanden, databank, netwrk, . . .)

Uitzonderingen worden voorgesteld als objecten van één of andere *uitzonderingsklasse*.

Zelf opgooien

Je kan ook zelf uitzonderingen opgooien met **throw**:

```
public double gemiddelde (double[] tab) {
    if (tab.length == 0) {
        throw new Exception ("Gemiddelde van lege tabel");
    } else {
        int som = 0.0;
        for (double w : tab) {
            som += w;
        } ...
        return som / tab.length;
    }
}
```

of

```
... throw new IllegalArgumentException
                        ("Tabel mag niet leeg zijn");
```

Opvangen

Opvangen met een **try–catch**-opdracht

```
try {
   ... // Opdrachten
} catch (Exception ex) {
   ... // Enkel uitgevoerd als er iets fout loopt
}
```

Opvangen

Opvangen met een **try–catch**-opdracht

```
try {
   ... // Opdrachten
} catch (Exception ex) {
   ... // Enkel uitgevoerd als er iets fout loopt
}
```

of meer specifiek:

```
try {
   ...
} catch (NullPointerException ex) {
   ...
} catch (IllegalArgumentException ex) {
    ...
}
```

Niet vernoemde uitzonderingen worden **niet** opgevangen!

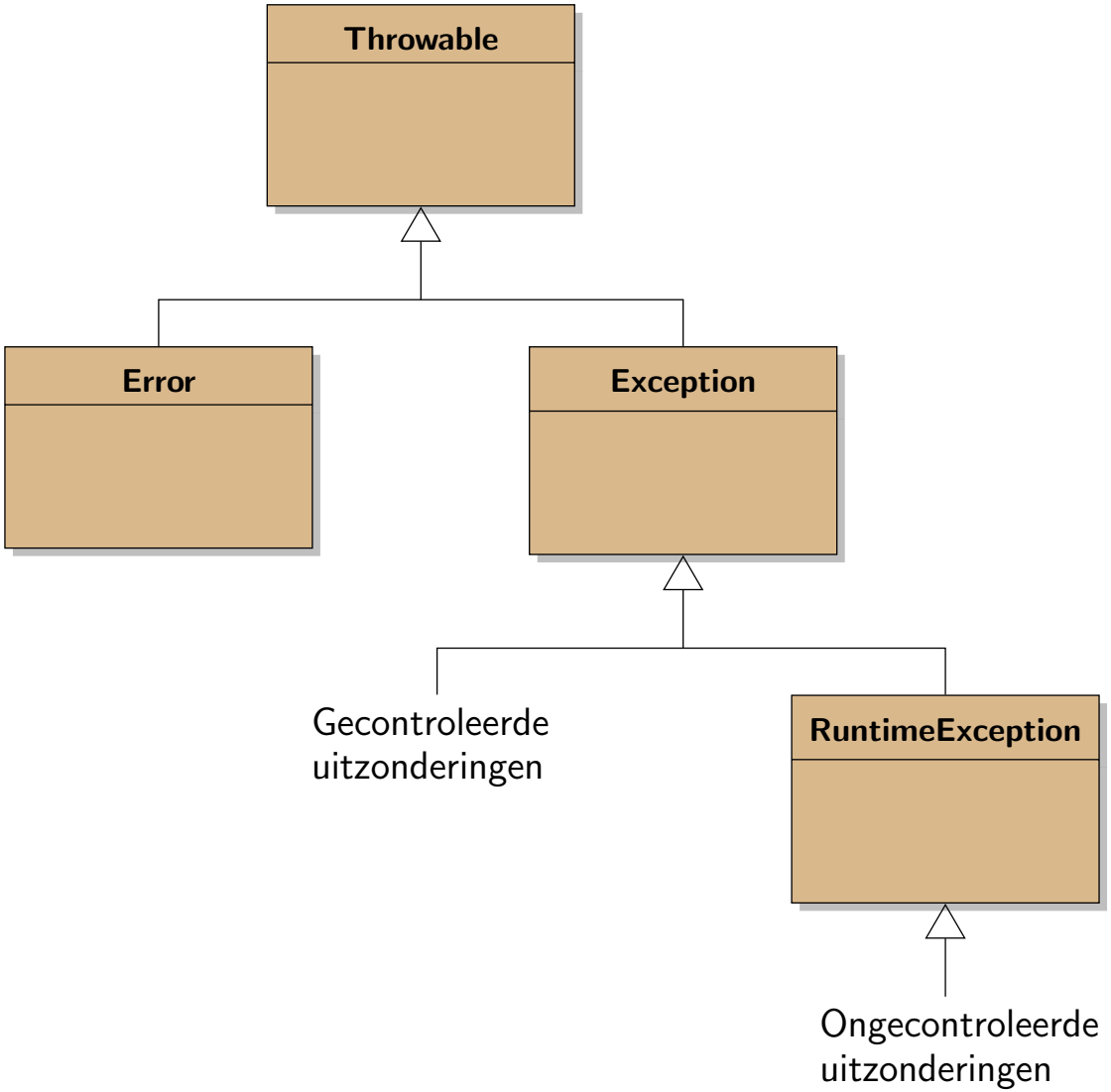
Fout kan ook 'diep' veroorzaakt zijn

```
public void doeA () {
    ... throw new ArithmeticException ("Getal < 0")
}

public void doeB () {
    doeA();
}

public void doeC() {
    try {
        doeB();
    } catch (Exception ex) {
        ...
    }
}
```

Klassenhiërarchie



Gecontroleerde uitzonderingen

Komen meestal voor bij invoer/uitvoer of netwerkcommunicatie.

Je kan ze niet negeren. Je moet ze

- Opvangen met **try–catch**, of

Gecontroleerde uitzonderingen

Komen meestal voor bij invoer/uitvoer of netwerkcommunicatie.

Je kan ze niet negeren. Je moet ze

- Opvangen met **try–catch**, of
- Doorgeven naar de oproeper met **throws**

public void *verwerkBestand*(*String name*) **throws** *IOException* {

... // Gebruikt diverse invoer/uitvoer-methoden

}

De methode die *verwerkBestand* oproept, moet nu de uitzondering behandelen (of opnieuw doorgeven).

Een nieuw uitzonderingstype definiëren

```
public class NotPrimeException extends Exception {
    private int value;
    public int getValue() { return value; }

    public NotPrimeException(int value) {
        super ("Must be prime: " + value);
        this.value = value;
    }
}
```

Een nieuw uitzonderingstype definiëren

```
public class NotPrimeException extends Exception {
    private int value;
    public int getValue() { return value; }

    public NotPrimeException(int value) {
        super ("Must be prime: " + value);
        this.value = value;
    }
}
```

Voorbeeld van gebruik:

```
try {
    ...
} catch (NotPrimeException ex) {
    System.out.println ("Moest priem zijn: " + ex.getValue());
    System.exit(0);
}
```

Try-met-bronnen

- Invoer/uitvoer moet altijd netjes worden afgesloten,
- met gepaste *close*-methode,

Try-met-bronnen

- Invoer/uitvoer moet altijd netjes worden afgesloten,
- met gepaste *close*-methode,
- ook wanneer er iets fout gaat.

Try-met-bronnen

- Invoer/uitvoer moet altijd netjes worden afgesloten,
- met gepaste *close*-methode,
- ook wanneer er iets fout gaat.

Gebruik *try-met-bronnen*:

```
try (PrintWriter writer = new PrintWriter ("uitvoer.txt")) {
    writer.println("Dit is een eerste zin");
    ...
    writer.println("Dit is een laatste zin");
} catch (IOException ex) {
    ... // verwerk eventuele uitzonderingen
}
```

Try-met-bronnen

- Invoer/uitvoer moet altijd netjes worden afgesloten,
- met gepaste *close*-methode,
- ook wanneer er iets fout gaat.

Gebruik *try-met-bronnen*:

```
try (PrintWriter writer = new PrintWriter ("uitvoer.txt")) {
    writer.println("Dit is een eerste zin");
    ...
    writer.println("Dit is een laatste zin");
} catch (IOException ex) {
    ... // verwerk eventuele uitzonderingen
}
```

- Bronnen voldoen aan *AutoCloseable*-interface
- Meerdere bronnen toegelaten
- Mag ook zonder **catch**-clausule

Bierkiezer

Fase 4: ontwerp, aanmaak en invullen databank

Schrijf een SQL-script met

- tabeldefinities
- `INSERT`-opdrachten met initiële gegevens

Zie *createdb.sql*, *initdb.sql* in *db*-project van *Moevie*, of blz. 76,77.

(Aanmaken databank + uitvoeren script — zie later)

- Welke SQL-opdrachten zal je nodig hebben om de databanklaag te implementeren?

Databanktoegang met JDBC

Ondersteunt verschillende databanken

Databanktoegang met JDBC

Ondersteunt verschillende databanken

- Client–server (gebruikersnaam + wachtwoord)

Databanktoegang met JDBC

Ondersteunt verschillende databanken

- Client–server (gebruikersnaam + wachtwoord)
- Ingebed (*Moevie*)

Databanktoegang met JDBC

Ondersteunt verschillende databanken

- Client–server (gebruikersnaam + wachtwoord)
- Ingebed (*Moevie*)
- '*In memory*' (testen)

Databanktoegang met JDBC

Ondersteunt verschillende databanken

- Client–server (gebruikersnaam + wachtwoord)
- Ingebed (*Moevie*)
- '*In memory*' (testen)

JDBC is in principe onafhankelijk van de databanksoftware

- Java DB = Apache Derby (*Moevie*)
- MySQL, PostgreSQL, SQLite, ...
- Zelfs MS Access (via *JDBC-ODBC bridge*)

(Maar opgelet! SQL-dialecten kunnen veel verschillen.)

Drivers — JDBC-URL

Je hebt een *driver* nodig om met een specifieke databank te werken

- In een JAR-archief, in het *class path*
- (IDEA: toevoegen aan bibliotheken)
- Java DB: *derby.jar* (ingebed) of *derbyclient.jar* (client)

Drivers — JDBC-URL

Je hebt een *driver* nodig om met een specifieke databank te werken

- In een JAR-archief, in het *class path*
- (IDEA: toevoegen aan bibliotheken)
- Java DB: *derby.jar* (ingebed) of *derbyclient.jar* (client)

Verbinding wordt aangegeven door *JDBC-URL*. Vb.

```
jdbc:derby:bierdb
```

```
jdbc:mysql://dbserver.minfin.fgov.be/lonen
```

Bierkiezer

Fase 4 (vervolg)

Maak de databank aan vanop de opdrachtlijn met behulp van `ij`.

- Plaats *ij.bat* in het pad

  `C:\Program Files\Java\jdk1.8.0_112\db\bin`

- Kies een (werk)map voor de nieuwe databank
- Plaats script in die map

```
C:\Users\Kris\Documents\bier\etc\db>ij
ij version 10.11
ij> connect 'jdbc:derby:bierdb;create=true';
ij> run 'createdb.sql';
...
ij> exit;
C:\Users\Kris\Documents\bier\etc\db>
```

(Speciale JDBC-URL hoeft/mag slechts één keer.)

Kan ook gebruikt worden om databank te bekijken (bijv. *moevie*).

Verbinden

```
Connection conn =
    DriverManager.getConnection("jdbc:derby:moevie");
```

Verbinden

```
Connection conn =
    DriverManager.getConnection("jdbc:derby:moevie");
```

of

```
String gebruikersnaam = ...;
String wachtwoord = ...;
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://dbserver.minfin.fgov.be/lonen",
    gebruikersnaam,
    wachtwoord
);
```

Verbinden

```
Connection conn =
    DriverManager.getConnection("jdbc:derby:moevie");
```

of

```
String gebruikersnaam = ...;
String wachtwoord = ...;
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://dbserver.minfin.fgov.be/lonen",
    gebruikersnaam,
    wachtwoord
);
```

- Verbinding is object v/e klasse die **interface** *Connection* implementeert.
- Idem later voor opdrachten en resultaten van zoekopdrachten

Verbinden

Kan (gecontroleerde) uitzondering opgooien (*SQLException*) en moet gesloten worden.

Verbinden

Kan (gecontroleerde) uitzondering opgooien (*SQLException*) en moet gesloten worden.

Gebruik *try-met-bronnen*.

```
try (Connection conn = DriverManager.getConnection ("...")) {
    ...
    // gegevens ophalen, toevoegen, veranderen, enz.
} catch (SQLException e) {
    ... // in geval van databankproblemen
}
```

Opdrachten

Gebruikt (interface) *PreparedStatement*.

```
try (PreparedStatement stmt = conn.prepareStatement (
    "... SQL-opdracht ..."
)) {
    // opdrachten uitvoeren
    ...
} catch (SQLException e) {
    ...
}
```

Opdrachten

Gebruikt (interface) *PreparedStatement*.

```
try (PreparedStatement stmt = conn.prepareStatement (
    "... SQL-opdracht ..."
)) {
    // opdrachten uitvoeren
    ...
} catch (SQLException e) {
    ...
}
```

Twee soorten

- Zoekopdrachten (`SELECT`) — *executeQuery*
- Andere (`INSERT`, `UPDATE`, `DELETE`) — *executeUpdate*.

executeUpdate

Voorbeeld: wis alle beoordelingen uit de databank

```
try (Connection conn
        = DriverManager.getConnection("jdbc:derby:moevie");
     PreparedStatement ps = conn.prepareStatement(
             "DELETE FROM Beoordelingen"
     )) {
    ps.executeUpdate();
} catch (SQLException ex) {
    System.err.println ("Database error: " + ex);
}
```

(Merk op: dit is een try-met-bronnen met *twee* bronnen.)

executeQuery

ResultSet executeQuery (*String sqlOpdracht*) **throws** *SQLException*;

Resultaat is een *ResultSet*

executeQuery

ResultSet executeQuery (*String sqlOpdracht*) **throws** *SQLException*;

Resultaat is een *ResultSet*

- Stelt een 'tabel' voor

executeQuery

ResultSet executeQuery (*String sqlOpdracht*) **throws** *SQLException*;

Resultaat is een *ResultSet*

- Stelt een 'tabel' voor
- Kan slechts rij per rij bekeken worden (d.m.v. *cursor*)

executeQuery

ResultSet executeQuery (*String sqlOpdracht*) **throws** *SQLException*;

Resultaat is een *ResultSet*

- Stelt een 'tabel' voor
- Kan slechts rij per rij bekeken worden (d.m.v. *cursor*)
- Cursur begint *vóór* eerste rij
- Gebruik methode *next()* om op te schuiven. Geeft **false** terug als het einde is bereikt.

executeQuery

ResultSet executeQuery (*String sqlOpdracht*) **throws** *SQLException*;

Resultaat is een *ResultSet*

- Stelt een 'tabel' voor
- Kan slechts rij per rij bekeken worden (d.m.v. *cursor*)
- Cursur begint *vóór* eerste rij
- Gebruik methode *next()* om op te schuiven. Geeft **false** terug als het einde is bereikt.

Typisch gebruik:

```
try (ResultSet res = stmt.executeQuery ()) {
    while (res.next ()) {
        // haal de gegevens op van de huidige rij
        ...
    }
}
```

(Dit is een voorbeeld van een try-met-bronnen zonder **catch**-gedeelte.)

executeQuery

Kolommen van huidige rij worden opgehaald met speciale 'getters':

boolean *getBoolean* (*String columnName*);
double *getDouble* (*String columnName*);
int *getInt* (*String columnName*);
String getString (*String columnName*);

Type moet overeenkomen met type van de databankkolom!

(. . .)

executeQuery

Voorbeeld: druk alle titels af van alle films

```
try (Connection conn
        = DriverManager.getConnection("jdbc:derby:moevie");
        PreparedStatement ps = conn.prepareStatement(
                "SELECT titel FROM Films " +
                        "ORDER BY Aangemaakt DESC";
        ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                System.out.println(rs.getString("Titel"));
            }
} catch (SQLException ex) {
        ...
}
```

Parameters

SQL-statements hebben meestal parameters.

Bij JDBC worden dit vraagtekens:

```
DELETE FROM Beoordelingen WHERE Film=? AND Gebruiker=?
INSERT INTO Films(Titel,Hoofdrol,Eigenaar) VALUE (?,?,?)
SELECT Commentaar, Sterren FROM Beoordelingen
    WHERE Film=? AND Gebruiker=?
```

Parameters

SQL-statements hebben meestal parameters.

Bij JDBC worden dit vraagtekens:

```
DELETE FROM Beoordelingen WHERE Film=? AND Gebruiker=?
INSERT INTO Films(Titel,Hoofdrol,Eigenaar) VALUE (?,?,?)
SELECT Commentaar, Sterren FROM Beoordelingen
    WHERE Film=? AND Gebruiker=?
```

Vraagtekens worden in de *PreparedStatements* ingevuld met

public void *setXxx* (**int** *indexParameter*, *xxx waardeParameter*)

Index telt vanaf 1 (hoeveelste vraagteken?)

Voorbeelden

Maak een nieuwe film.

```
public void nieuweFilm(String titel, String eigenaar, String hoofdrol) {
    try (Connection conn = DriverManager.getConnection("...");
         PreparedStatement ps = conn.prepareStatement(
             "INSERT INTO Films(Titel,Hoofdrol,Eigenaar) "
             + "VALUES (?,?,?)"
         )) {
        ps.setString(1, titel);
        ps.setString(2, hoofdrol);
        ps.setString(3, eigenaar);
        ps.executeUpdate();
    } catch (SQLException ex) {
        ...
    }
}
```

Voorbeelden

Verwijder een specifieke beoordeling

```
public void verwijderBeoordeling(int film, String gebruiker) {
    try (Connection conn = DriverManager.getConnection("...");
         PreparedStatement ps = conn.prepareStatement(
             "DELETE FROM Beoordelingen "
             + "WHERE Film=? AND Gebruiker=?"
         )) {
        ps.setInt(1, film);
        ps.setString(2, gebruiker);
        ps.executeUpdate();
    } catch (SQLException ex) {
        ...
    }
}
```

Voorbeelden

Toon alle films van één specifieke eigenaar

```
public List<Film> lijstFilms(String eigenaar) {
    try (Connection conn = DriverManager.getConnection("...");
        PreparedStatement ps = conn.prepareStatement(
            "SELECT ID,Titel,Hoofdrol FROM Films WHERE Eigenaar=?"
        )) {
        ps.setString(1, eigenaar);
        try (ResultSet rs = ps.executeQuery()) {
            List<Film> lst = new ArrayList<>();
            while (rs.next()) {
                Film film = new Film (
                    rs.getInt("ID"), rs.getString("Titel"),
                    rs.getString("Hoofdrol"), eigenaar
                );
                lst.add(film);
            }
            return lst;
        }
    } catch (SQLException ex) { ... }
}
```

Bierkiezer

Fase 5: Bierdatabank bevragen via JDBC

- Kies een `SELECT`-opdracht mét parameter die nuttig is voor de bierkiezer
- Schrijf een Java-methode die deze opdracht uitvoert (met JDBC)
- Schrijf een hoofdprogramma *Main* (in *db*-project) die deze methode 2 keer oproept en voldoende uitvoer geeft om te kunnen testen
- Voer het programma uit. Vindt JDBC de juiste driver? En de juiste database?
- Doe hetzelfde met een `INSERT`- of `UPDATE`-opdracht.

Moevie DB — Uitzonderingen

Hoe uitzonderingen opvangen?

Moevie DB — Uitzonderingen

Hoe uitzonderingen opvangen?

- Bericht afdrukken en programma beëindigen, of

Moevie DB — Uitzonderingen

Hoe uitzonderingen opvangen?

- Bericht afdrukken en programma beëindigen, of
- bericht afdrukken en fout negeren, of

Moevie DB — Uitzonderingen

Hoe uitzonderingen opvangen?

- Bericht afdrukken en programma beëindigen, of
- bericht afdrukken en fout negeren, of
- uitzondering doorgeven aan logicalaag met **throws**, of

Moevie DB — Uitzonderingen

Hoe uitzonderingen opvangen?

- Bericht afdrukken en programma beëindigen, of
- bericht afdrukken en fout negeren, of
- uitzondering doorgeven aan logicalaag met **throws**, of
- eigen uitzondering genereren:

```
public class DataAccessException extends RuntimeException {
    public DataAccessException (String methode, SQLException ex) {
        super ("Databankfout in '" + methode + "'", ex);
    }
}
```

Moevie DB — Uitzonderingen

Hoe uitzonderingen opvangen?

- Bericht afdrukken en programma beëindigen, of
- bericht afdrukken en fout negeren, of
- uitzondering doorgeven aan logicalaag met **throws**, of
- eigen uitzondering genereren:

```
public class DataAccessException extends RuntimeException {
    public DataAccessException (String methode, SQLException ex) {
        super ("Databankfout in '" + methode + "'", ex);
    }
}
```

en opgooien:

```
catch (SQLException ex) {
    throw new DataAccessException("getBeoordeling", ex);
}
```

DAO's en DAP

Voorbeeldmethode's hierboven zijn bijna bruikbaar in DAO-implementaties

- Naam/URL van de databank mag niet hard gecodeerd zijn
- URL is een veld van de data-access-provider
- DAO-implementaties bevatten verwijzing naar DAP
- Gebruiken DAP-methode *getConnection*

cf. *db*-project van *Moevie*

Bierkiezer

Fase 6: JDBC-implementatie van databanklaag

- Schrijf JDBC-implementatieklassen voor de DAO's
- Schrijf JDBC-implementatieklasse voor DAP
- (Schrijf testprogramma — cf. *db-test*-project van *Moevie*)

Enkele extra's:

- Automatisch gegenereerde sleutels (blz. 74).
- Dezelfde *PreparedStatement* mag meerdere keren gebruikt worden (blz. 70).
- Ondersteuning voor *batch insert* en *batch update* (blz. 75–76).

Bierkiezer

Laatste fase: afwerken van de logica- en presentatielagen.

Extra's:

- Aanmelden en het sessie-object (blz. 38).
- Een minimale webtoepassing (blz. 41).
- Configuratie van de webtoepassing (blz. 43).
- Foutafhandeling bij formulieren (blz. 47).
- FTL-extra's: for-lussen en `?then` (blz. 54 & 52).
- Meerdere checkboxen (blz. 55).
- Java in de presentatielaag — sterrenreeksen (blz. 58)

De sessie

Het probleem:

- Elke browser-aanvraag = nieuw controller-object
- Je kan in het controller-object geen blijvende informatie bewaren

De sessie

Het probleem:

- Elke browser-aanvraag = nieuw controller-object
- Je kan in het controller-object geen blijvende informatie bewaren

De oplossing: het **sessie**-object

- Één object per 'browser'
- Bewaart een reeks *attributen* = sleutel/waarde-paren. (Waarde is altijd een string.)
- *Moevie* gebruikt dit om te onthouden wie er ingelogd is

(Zie broncode *Moevie*)