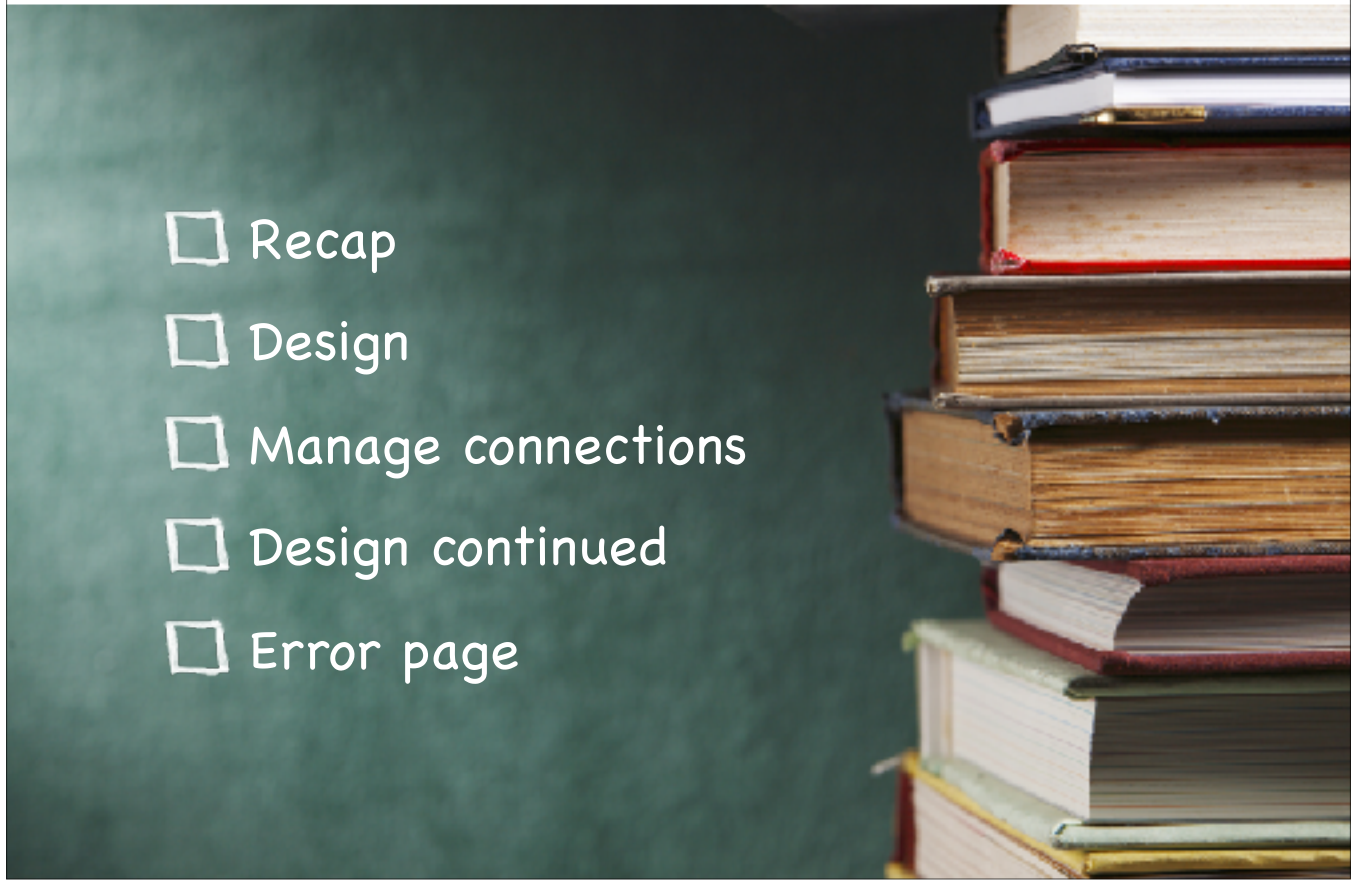
Web 3

Lesson 3: Database Design

EXAM QUESTIONS…

- ☑ Is JDBC used in View, Model or Controller?
- ☑ If I use try-with-resources, where is the connection closed?
- ☑ When should you use an error page?
- ☑ What makes an error page different from an normal JSP page?
- ☑ How long should you keep your database connection open for a web application? Why?
- ☑ …

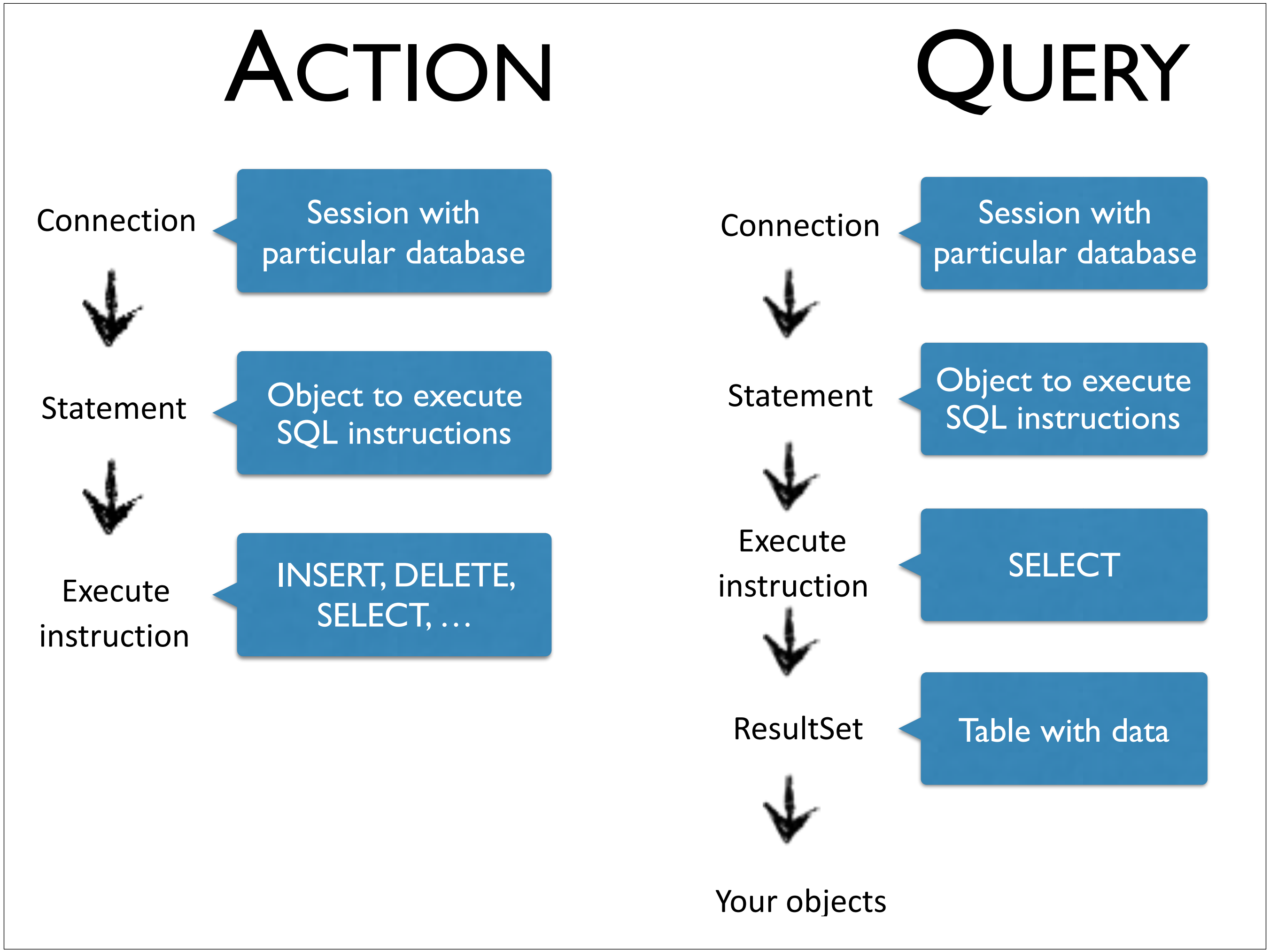
AGENDA



- ☐ Recap
- ☐ Design
- ☐ Manage connections
- ☐ Design continued
- ☐ Error page

JDBC

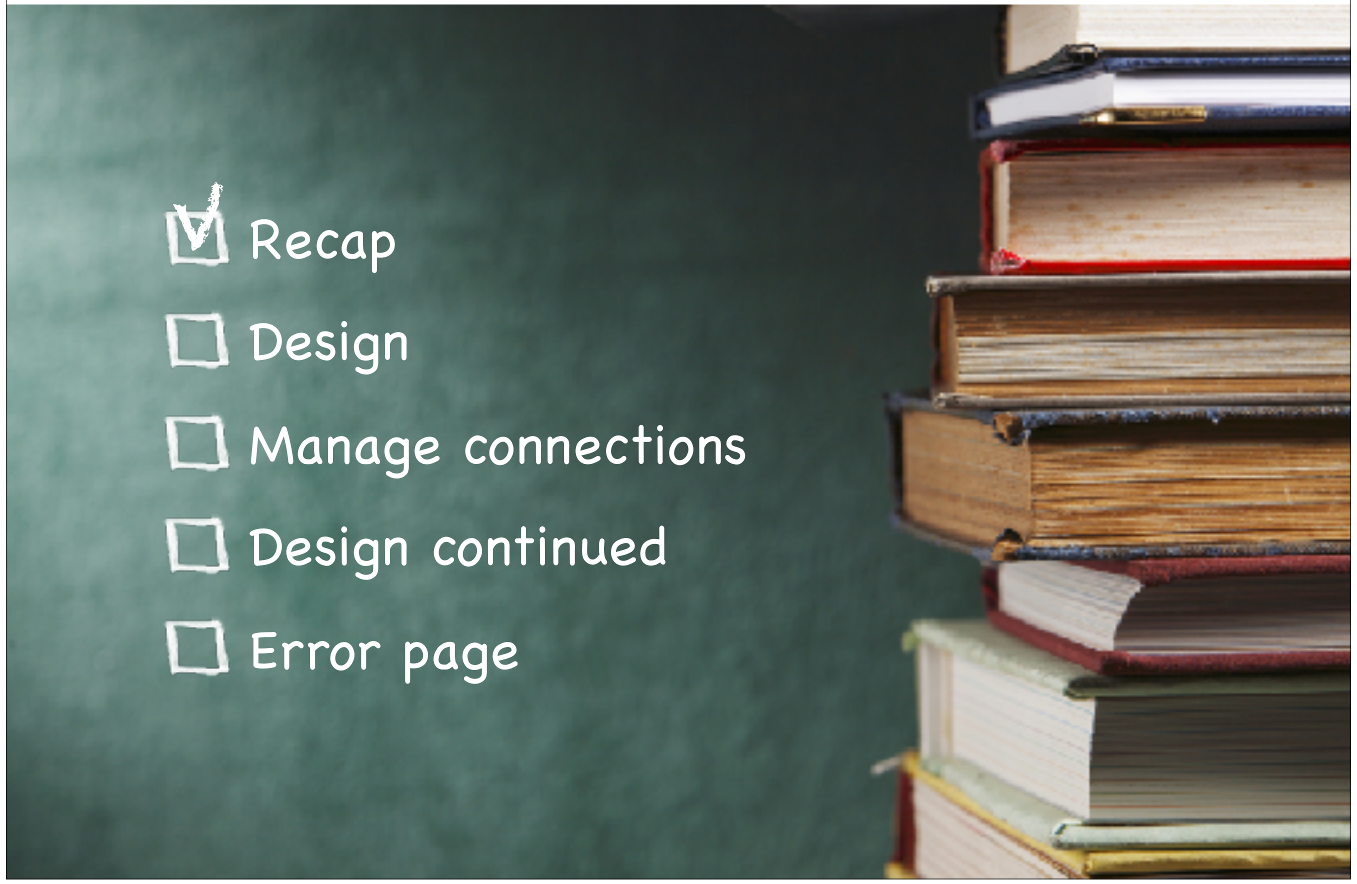
- Java DataBase Connectivity
- Java API $\longrightarrow$ Set of classes you can use
- Allows Java program to communicate
  - with **database**
  - via **SQL**


ACTION
Connection
Session with particular database
Statement
Object to execute SQL instructions
Execute instruction
INSERT, DELETE, SELECT, …
QUERY
Connection
Session with particular database
Statement
Object to execute SQL instructions
Execute instruction
SELECT
ResultSet
Table with data
Your objects

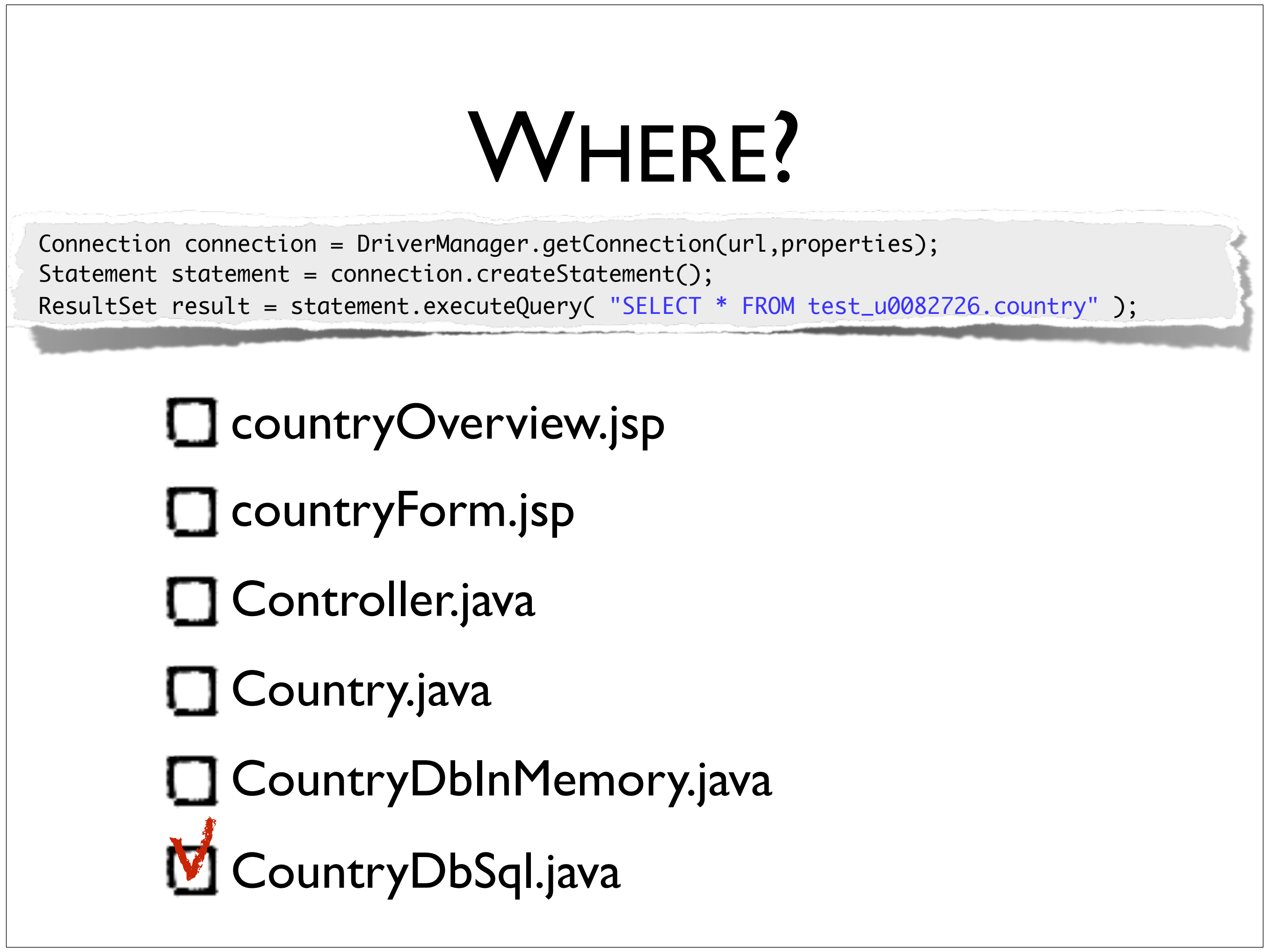
```java
public class CountryDbDemo {
    public static void main(String[] args) throws SqlException {
        Properties properties = new Properties();
        String url = "jdbc:postgresql://gegevensbanken.khleuven.be:51314/webontwerp?
             currentSchema=<name of your schema>";
        properties.setProperty("user", <userid>);
        properties.setProperty("password", <password>);
        properties.setProperty("ssl", "true");
        properties.setProperty("sslfactory", “org.postgresql.ssl.NonValidatingFactory");
        Class.forName("org.postgresql.Driver");
        Connection connection = DriverManager.getConnection(url,properties);

        Statement statement = connection.createStatement();
        ResultSet result = statement.executeQuery( "SELECT * FROM country" );

        while(result.next()){
            String name = result.getString("name");
            String capital = result.getString("capital");
            int numberOfVotes = Integer.parseInt(result.getString("votes"));
            int numberOfInhabitants = Integer.parseInt(result.getString("inhabitants"));

            Country country= new Country(name, numberOfInhabitants,capital, numberOfVotes);
            System.out.println(country);
        }

        statement.close();
        connection.close();
    }
}
```

Class.forName(“org.postgresql.Driver"); not needed for main
getType_name(“columnName”), e.g. getInt(“votes”), getString(“name”)

AGENDA



- [✓] Recap
- [ ] Design
- [ ] Manage connections
- [ ] Design continued
- [ ] Error page



WHERE?

```
Connection connection = DriverManager.getConnection(url,properties);
Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery( "SELECT * FROM test_u0082726.country" );
```

- ☐ countryOverview.jsp
- ☐ countryForm.jsp
- ☐ Controller.java
- ☐ Country.java
- ☐ CountryDbInMemory.java
- ✓ CountryDbSql.java

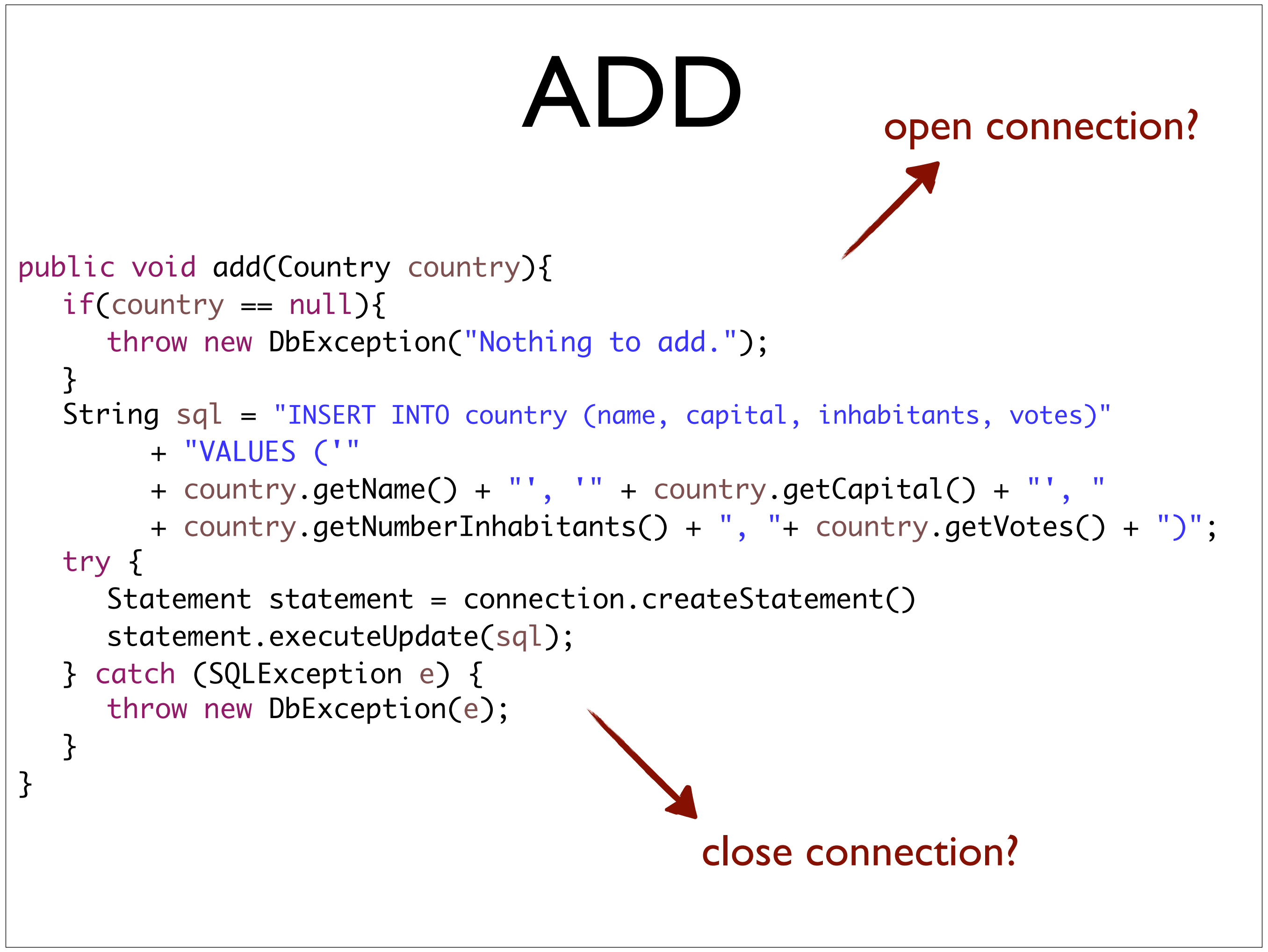
Last week, we just had 1 main method, containing all database code.

Where do we put the code with the database instructions in our web application? We don't have a main method…

COUNTRYDBSQL.JAVA

```
public class CountryDbSql {

   public void add(Country country){
     //create insert query based on properties of country
     //use statement object to execute the action
   }

   public List<Country> getAll(){
     //use statement object to execute a select query
     //convert result to list of countries
     //return list of countries;
   }

   public Country get(String id){
     //use statement object to execute a select query
     //convert result to country
     //return country;
   }
   …
}
```

Create a new database class, with the same methods as in the in-memory database class.



ADD

open connection?

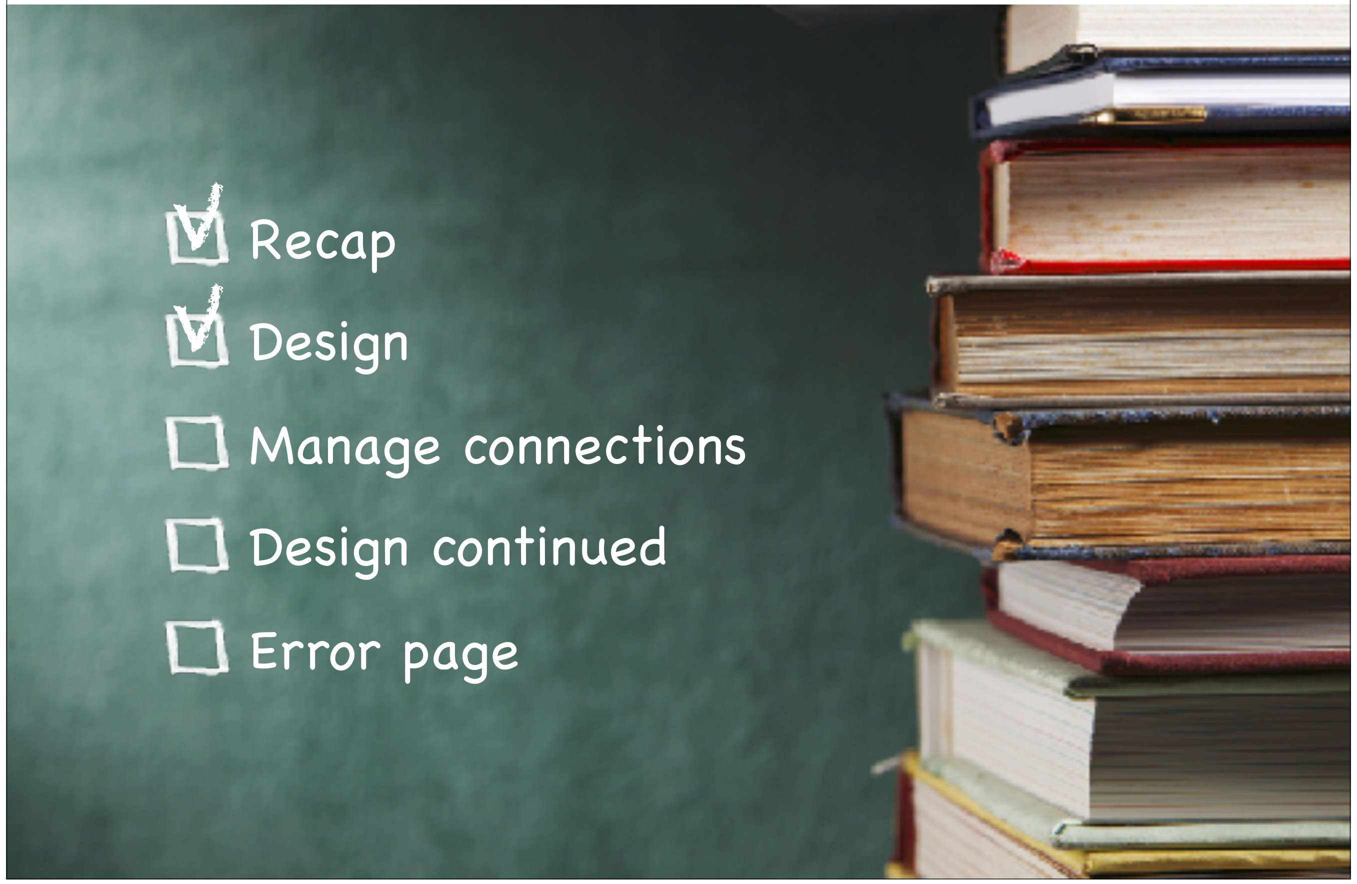
```
public void add(Country country){
   if(country == null){
      throw new DbException("Nothing to add.");
   }
   String sql = "INSERT INTO country (name, capital, inhabitants, votes)"
         + "VALUES ('"
         + country.getName() + "', '" + country.getCapital() + "', "
         + country.getNumberInhabitants() + ", "+ country.getVotes() + ")";
   try {
      Statement statement = connection.createStatement()
      statement.executeUpdate(sql);
   } catch (SQLException e) {
      throw new DbException(e);
   }
}
```

close connection?

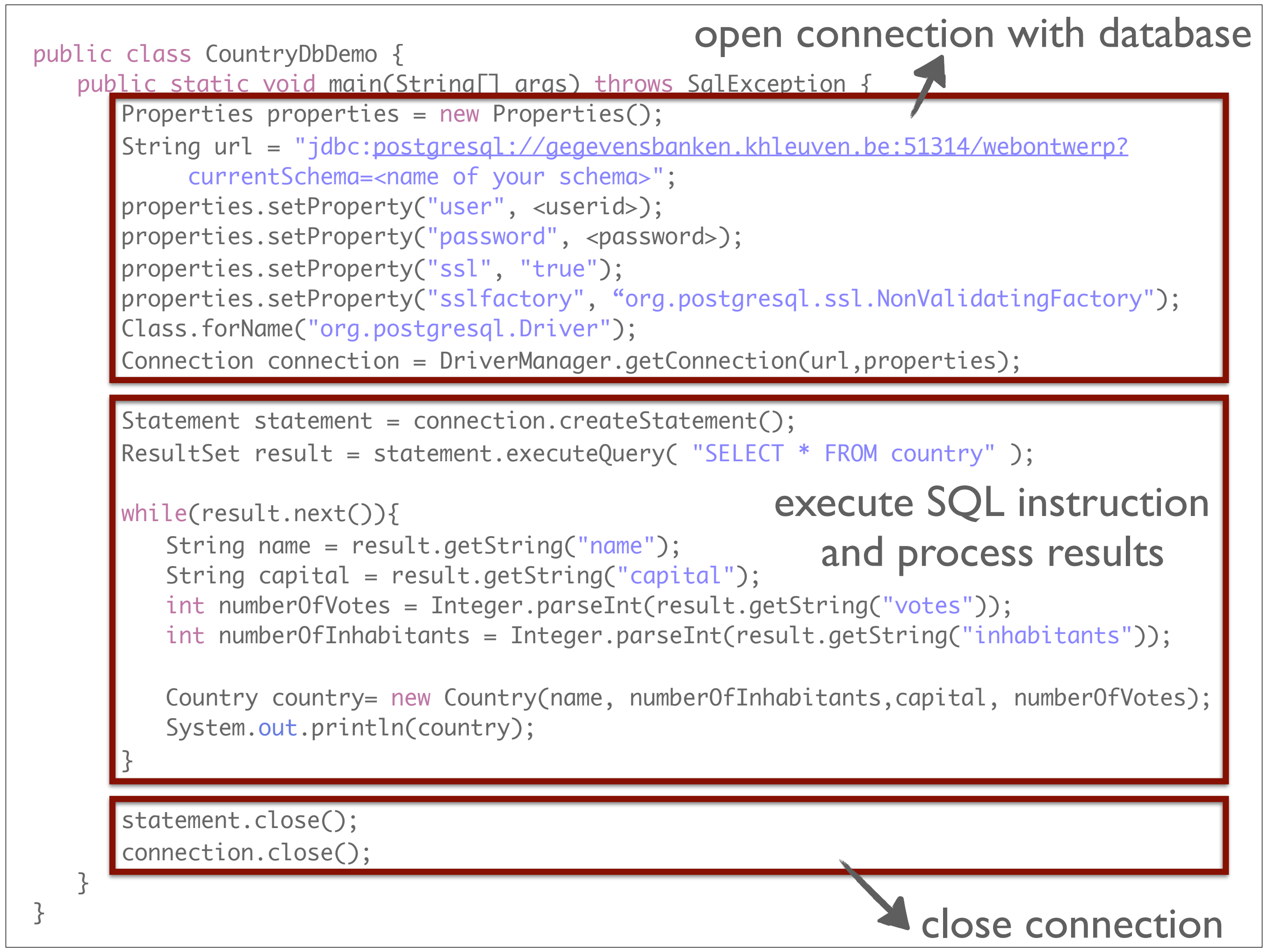
Example: in each method we create a statement and execute it.

Problem: to create a statement, we need a connection. Where do we create the connection? Where do we close it?

AGENDA



- [✓] Recap
- [✓] Design
- [ ] Manage connections
- [ ] Design continued
- [ ] Error page


open connection with database
execute SQL instruction and process results
close connection

```
public class CountryDbDemo {
    public static void main(String[] args) throws SqlException {
        Properties properties = new Properties();
        String url = "jdbc:postgresql://gegevensbanken.khleuven.be:51314/webontwerp?
             currentSchema=<name of your schema>";
        properties.setProperty("user", <userid>);
        properties.setProperty("password", <password>);
        properties.setProperty("ssl", "true");
        properties.setProperty("sslfactory", “org.postgresql.ssl.NonValidatingFactory");
        Class.forName("org.postgresql.Driver");
        Connection connection = DriverManager.getConnection(url,properties);

        Statement statement = connection.createStatement();
        ResultSet result = statement.executeQuery( "SELECT * FROM country" );

        while(result.next()){
            String name = result.getString("name");
            String capital = result.getString("capital");
            int numberOfVotes = Integer.parseInt(result.getString("votes"));
            int numberOfInhabitants = Integer.parseInt(result.getString("inhabitants"));

            Country country= new Country(name, numberOfInhabitants,capital, numberOfVotes);
            System.out.println(country);
        }

        statement.close();
        connection.close();
    }
}
```

This can be considered as an example of 'application scope': the connection is …

- created once when the application is started
- open during the entire 'life' of the application
- close once when the application is closed

Application Scope

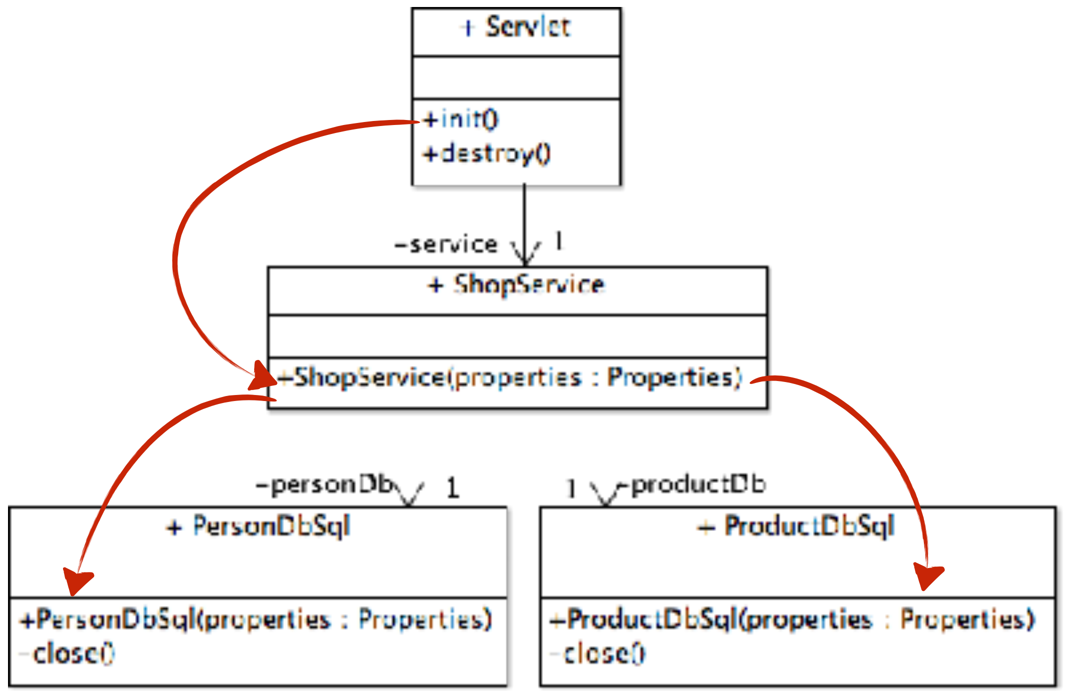
- 1 connection per application
- 100 simultaneous users:
  - how many connections ? $\rightarrow$ 1
  - time lost creating connections ? $\rightarrow$ not much
  - how many statements per connection? $\rightarrow$ 100
  - scalable ? $\rightarrow$ No, threads will wait for each other !

Application Scope

- Connection is created once
  (i.e. when starting the application)
- Connection is closed once
  (i.e. when closing the application)

APPLICATION SCOPE

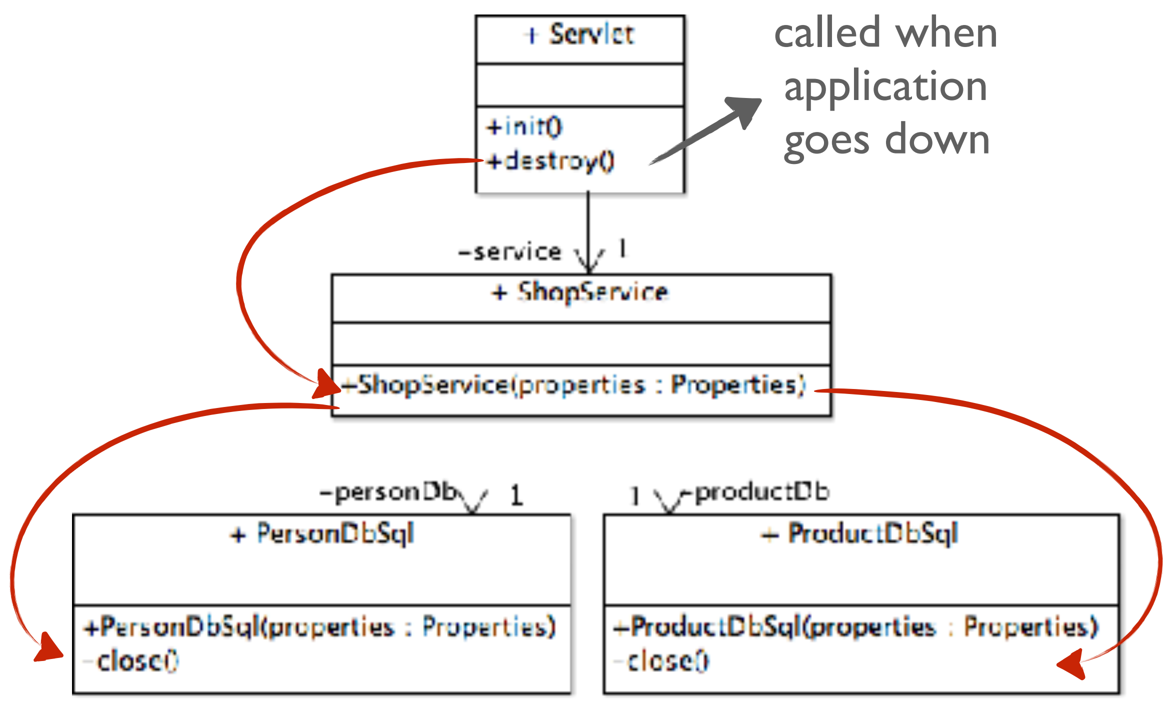
OPEN CONNECTION IN A WEB APPLICATION



+ Servlet
+init()
+destroy()
-service
1
+ ShopService
+ShopService(properties : Properties)
-personDb
1
1
-productDb
+ PersonDbSql
+PersonDbSql(properties : Properties)
-close()
+ ProductDbSql
+ProductDbSql(properties : Properties)
-close()

Application Scope

CLOSE CONNECTION IN A WEB APPLICATION


+ Servlet
+init()
+destroy()
called when application goes down
-service
1
+ ShopService
+ShopService(properties : Properties)
-personDb
1
1
-productDb
+ PersonDbSql
+PersonDbSql(properties : Properties)
-close()
+ ProductDbSql
+ProductDbSql(properties : Properties)
-close()

Application Scope

Servlet - method destroy()

```
@Override
public void destroy() {

     service.close();

     super.destroy();

}
```

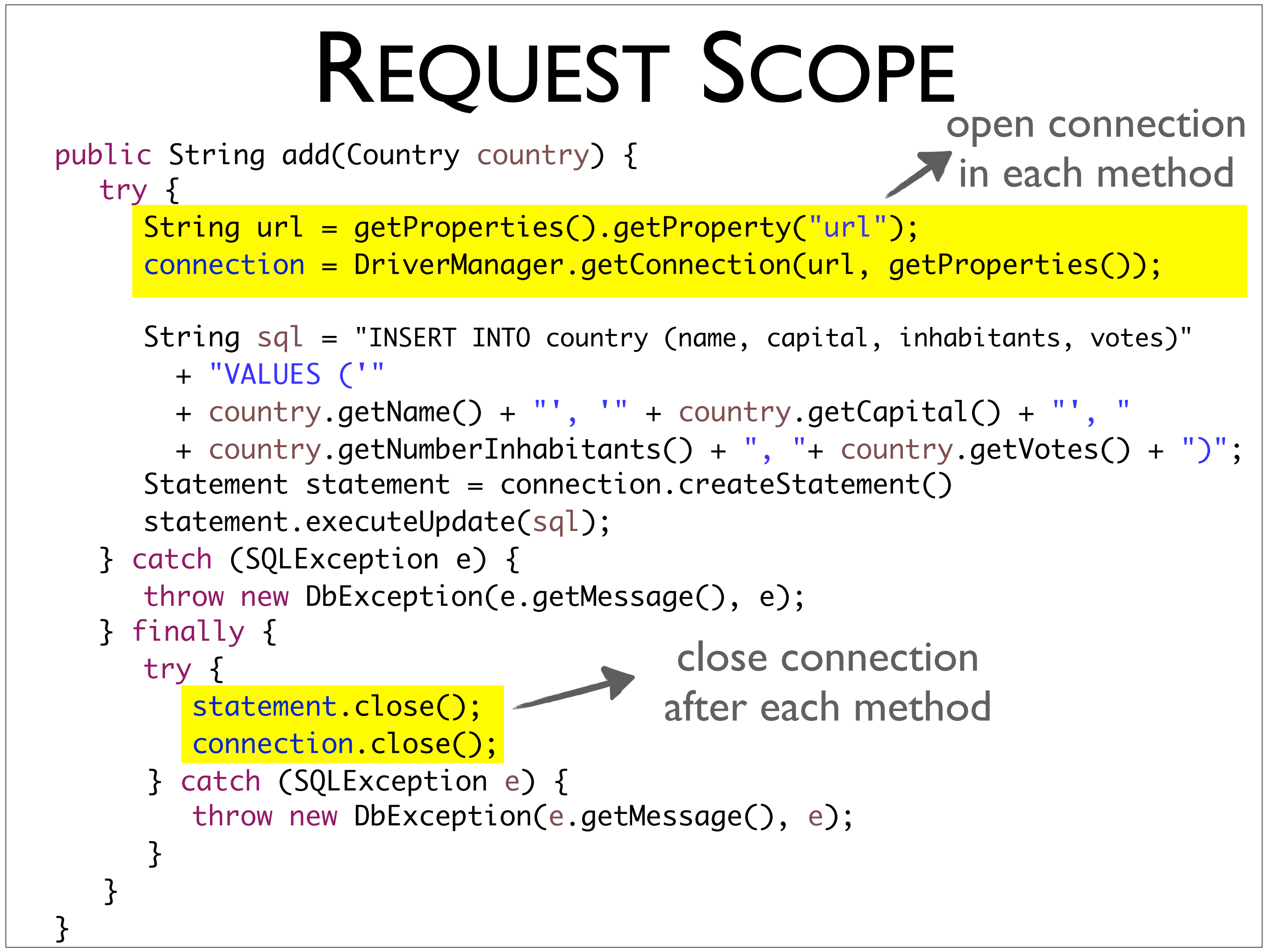
Application Scope

1 connection per application:

- OK for desktop application
- NOK for web application

Request Scope

- Connection is created in each method call (i.e. at the beginning of each method)
- Connection is closed in each method call (i.e. after the execution of the statement)



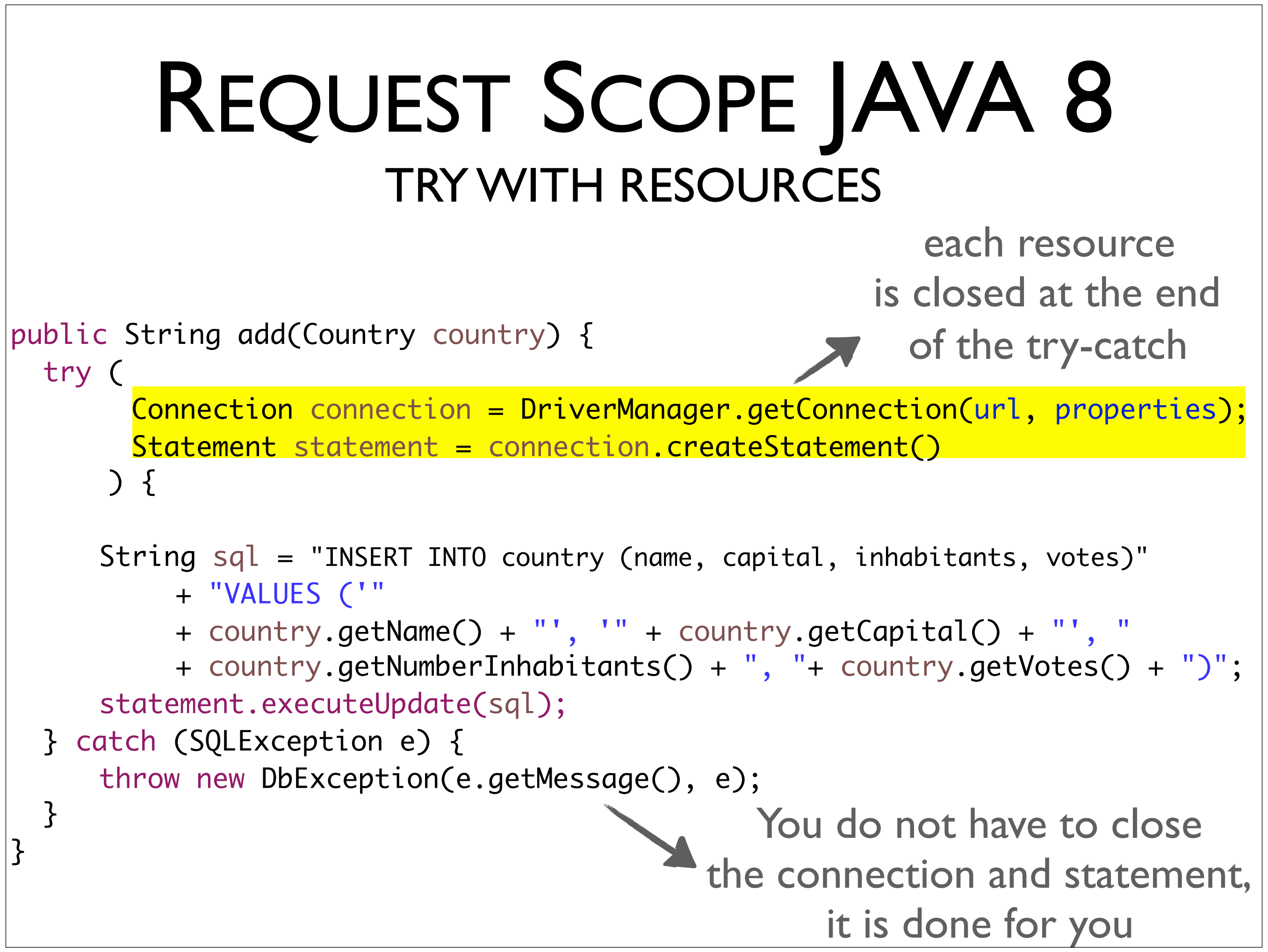
REQUEST SCOPE

open connection in each method

```
public String add(Country country) {
   try {
      String url = getProperties().getProperty("url");
      connection = DriverManager.getConnection(url, getProperties());

      String sql = "INSERT INTO country (name, capital, inhabitants, votes)"
        + "VALUES ('"
        + country.getName() + "', '" + country.getCapital() + "', "
        + country.getNumberInhabitants() + ", "+ country.getVotes() + ")";
      Statement statement = connection.createStatement()
      statement.executeUpdate(sql);
   } catch (SQLException e) {
      throw new DbException(e.getMessage(), e);
   } finally {
      try {
         statement.close();
         connection.close();
      } catch (SQLException e) {
         throw new DbException(e.getMessage(), e);
      }
   }
}
```

close connection after each method

What about the URL and the properties?



Request Scope Java 8

TRY WITH RESOURCES

each resource
is closed at the end
of the try-catch

```java
public String add(Country country) {
  try (
        Connection connection = DriverManager.getConnection(url, properties);
        Statement statement = connection.createStatement()
      ) {

      String sql = "INSERT INTO country (name, capital, inhabitants, votes)"
           + "VALUES ('"
           + country.getName() + "', '" + country.getCapital() + "', "
           + country.getNumberInhabitants() + ", "+ country.getVotes() + ")";
      statement.executeUpdate(sql);
  } catch (SQLException e) {
      throw new DbException(e.getMessage(), e);
  }
}
```

You do not have to close
the connection and statement,
it is done for you

Alternative for opening and closing connections in Java 8: try-with-resources

REQUEST SCOPE

TODO

store properties
as instance variables

```
public class CountryDbSql {
      private Properties properties = new Properties();
      private String url = "jdbc:postgresql://gegevensbanken.khleuven.be:…";

      public CountryDbSql() {
         properties.setProperty("user", "u0015529");
         properties.setProperty("password", "XXX");
         properties.setProperty("ssl", "true");
         properties.setProperty("sslfactory",
                                 "org.postgresql.ssl.NonValidatingFactory");
         try {
            Class.forName("org.postgresql.Driver");     }
         catch (ClassNotFoundException e) {
            throw new DbException(e.getMessage(), e);
         }
      }
      …
}
```

to be continued…

Properties have to be instance variables now, because we need to access them in each method.
In a few weeks, we will see how to use a property file.

Request Scope

- 1 connection per method (thread)
- 100 simultaneous users:
  - how many connections ? $\rightarrow$ 100
  - time lost creating connections ? $\rightarrow$ a lot !
  - how many statements per connection ? $\rightarrow$ 1
  - scalable ? $\rightarrow$ Yes

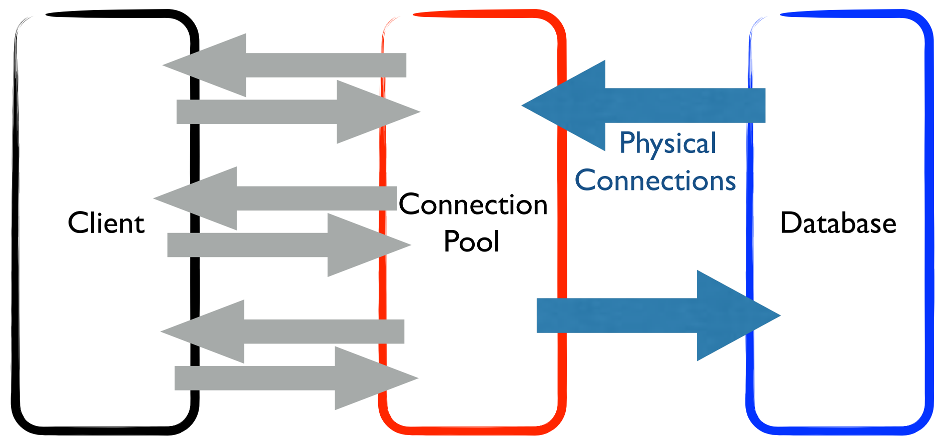
REQUEST SCOPE

1 connection per thread:

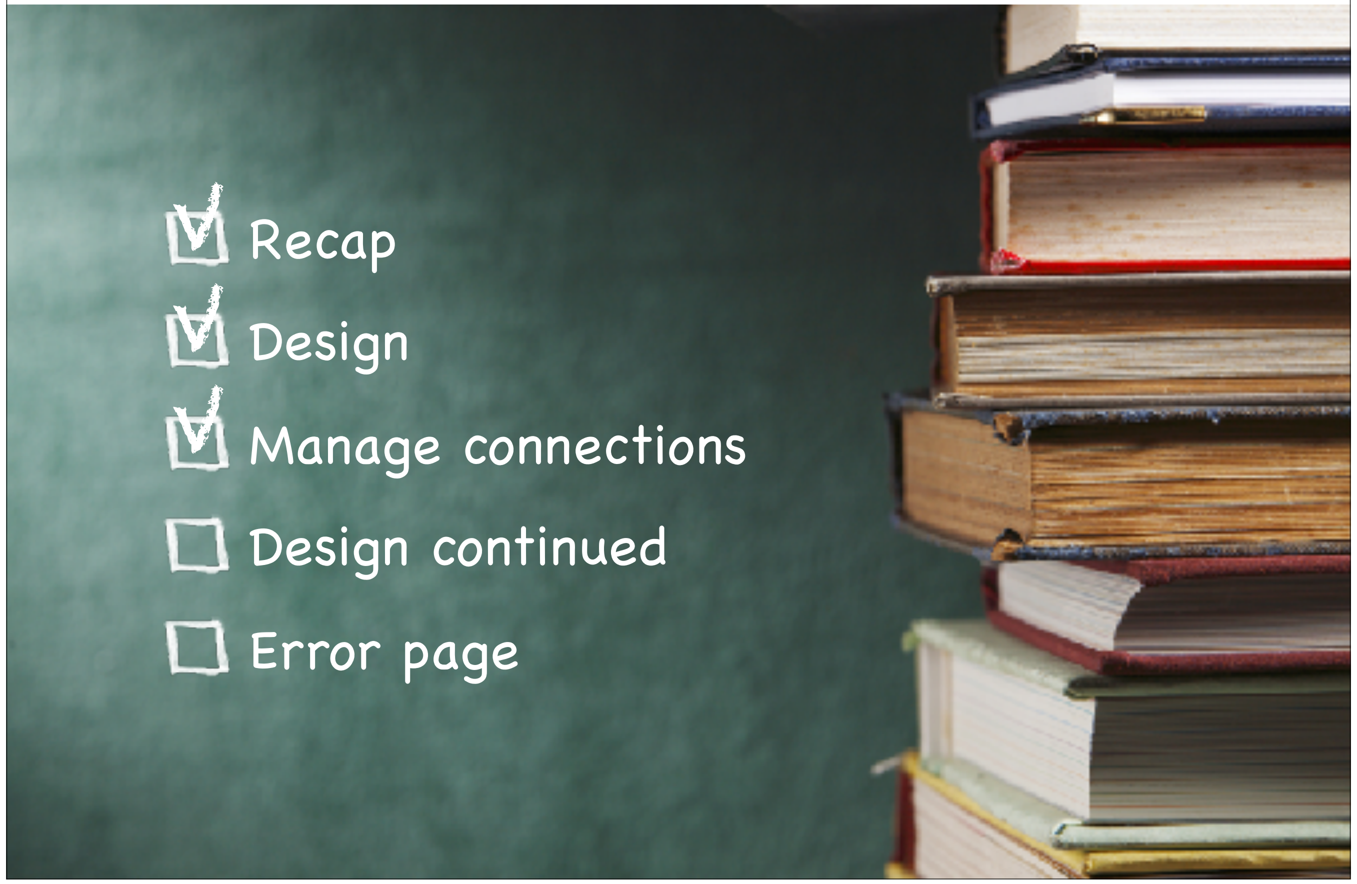
- better for web application
- performance issue !

Connection Pooling

- OK for web application
- to be continued…



Agenda



- ✓ Recap
- ✓ Design
- ✓ Manage connections
- ☐ Design continued
- ☐ Error page

Where does CountryDbSql belong?

- [✓ Model]
- ☐ View
- ☐ Controller

Who uses CountryDbSql?

- ☐ countryOverview.jsp
- ☐ countryForm.jsp
- ☐ Controller.java
- ☐ Country.java
- ☑ CountryService.java
- ☐ CountryRepositoryInMemory.java

COUNTRYSERVICE.JAVA

```
public class CountryService {
   private CountryDbSql db = new CountryDbSql();

   public void addCountry(Country country){
      db.add(country);
   }

   public List<Country> getCountries(){
      return db.getAll();
   }
```

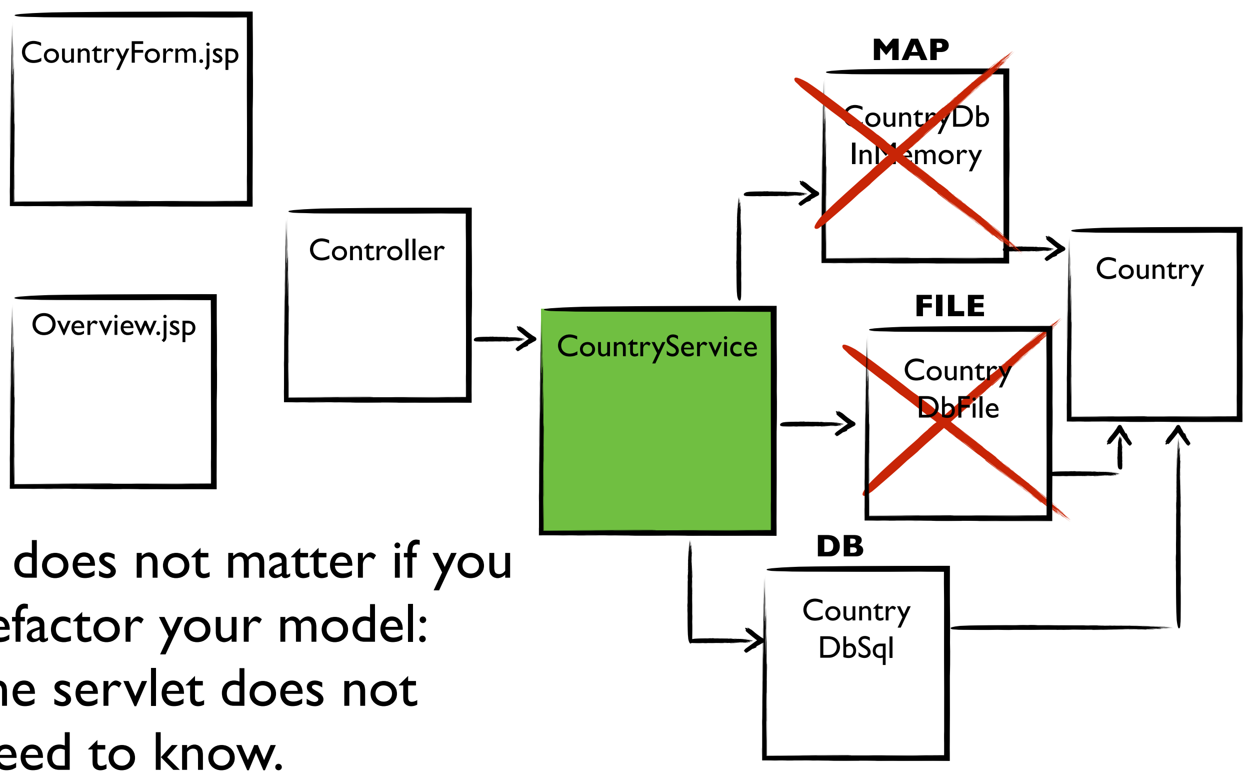
- ☑ order of addition does not matter
- ☑ we want no doubles
- ☑ we want to use an identifier to lookup elements
- ☑ **data are stored permanently**

OK?

```java
public class CountryService {
  private CountryDbSql db = new CountryDbSql();

  public void addCountry(Country country){
    db.add(country);
  }

  public List<Country> getCountries(){
    return db.getAll();
  }
```

OK: Facade

SEE OOO

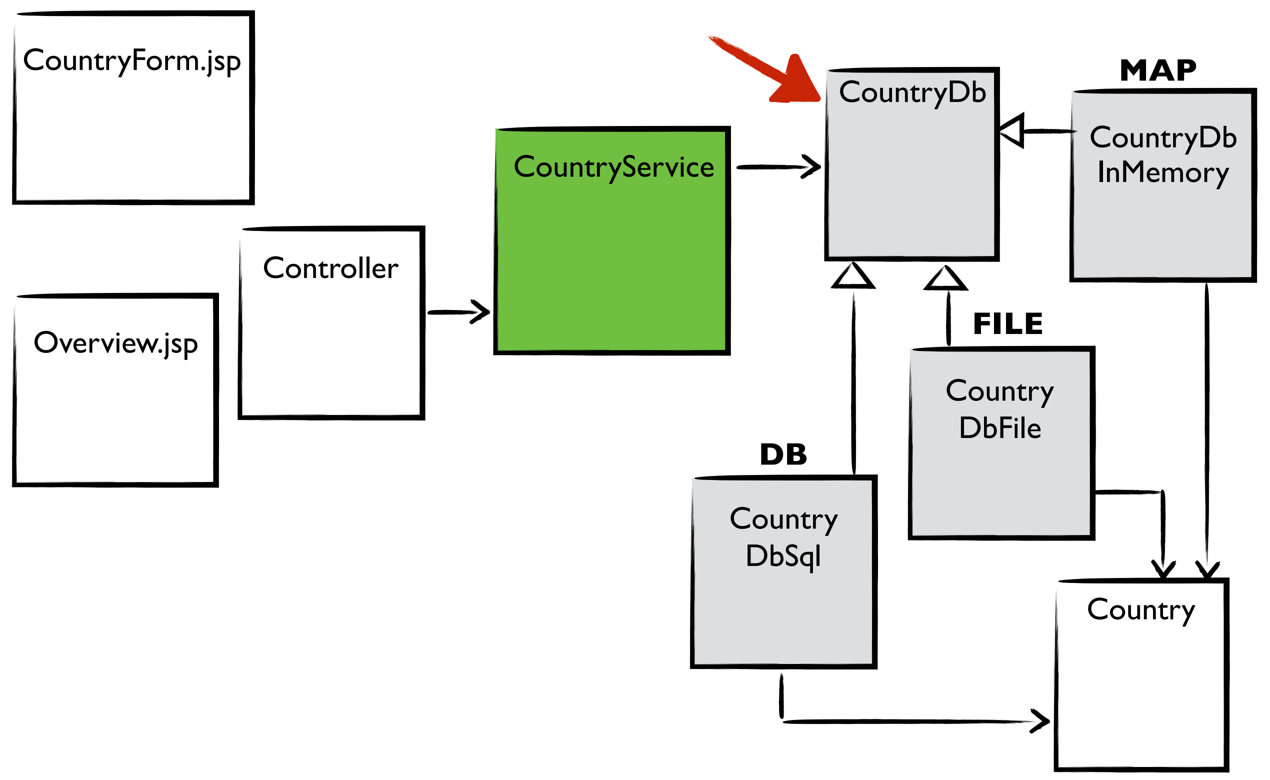


It does not matter if you refactor your model: the servlet does not need to know.

CountryService hides the complexity of your model for the view

Better: Facade + Strategy

see OOO

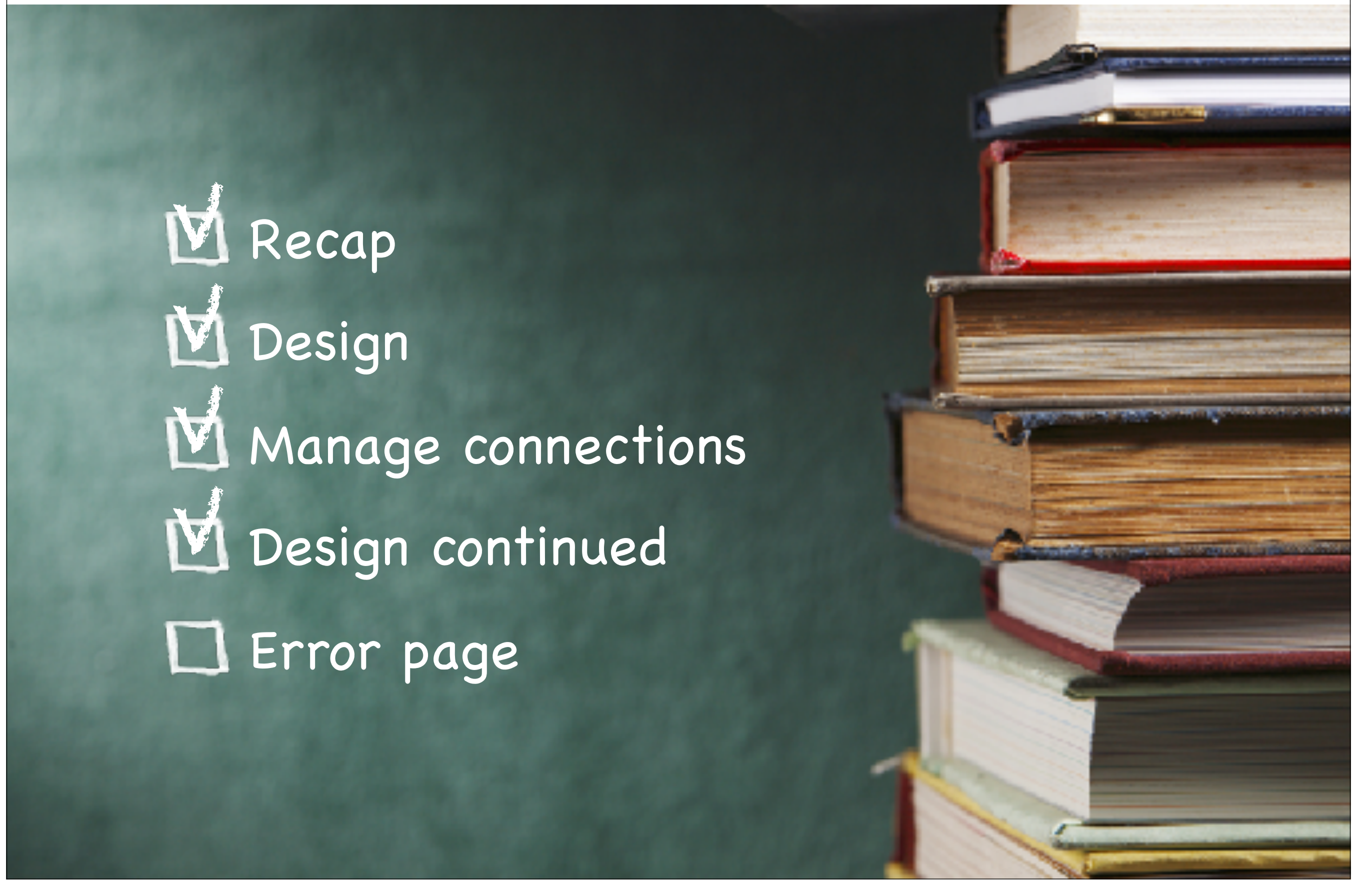


Using strategy, you can easily switch between different types of database.

BETTER...

```
public class CountryService {
   private CountryDb db = new CountryDbSql();

   public void addCountry(Country country){
      db.add(country);
   }

   public List<Country> getCountries(){
      return db.getAll();
   }
```

AGENDA



- ☑ Recap
- ☑ Design
- ☑ Manage connections
- ☑ Design continued
- ☐ Error page

Error page

1. Create error page
2. Navigate to error page
   - in case of an error
   - if the user cannot help it

1. ERROR.JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%@page isErrorPage="true"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Something wrong</title>
<link rel="stylesheet" href="css/sample.css">
</head>
<body>
   <main>
   <article>
      <h1>Oh dear !</h1>
      <p>You caused a ${pageContext.exception} on the server!</p>
      <p>
         <a href="Controller">Home</a>
      </p>
   </article>
   </main>
</body>
</html>
```

2. WEB.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://
java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd" version="3.0">
  <welcome-file-list>
    <welcome-file>Controller</welcome-file>
  </welcome-file-list>
  <error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error.jsp</location>
  </error-page>
  <context-param>
    <param-name>url</param-name>
    <param-value>
        jdbc:postgresql://gegevensbanken.khleuven.be:51415/webontwerp
    </param-value>
  </context-param>
  <context-param>
    <param-name>user</param-name>
    <param-value>u0082726</param-value>
  </context-param>
  <context-param>
    <param-name>password</param-name>
```

In case of an exception navigate to error.jsp

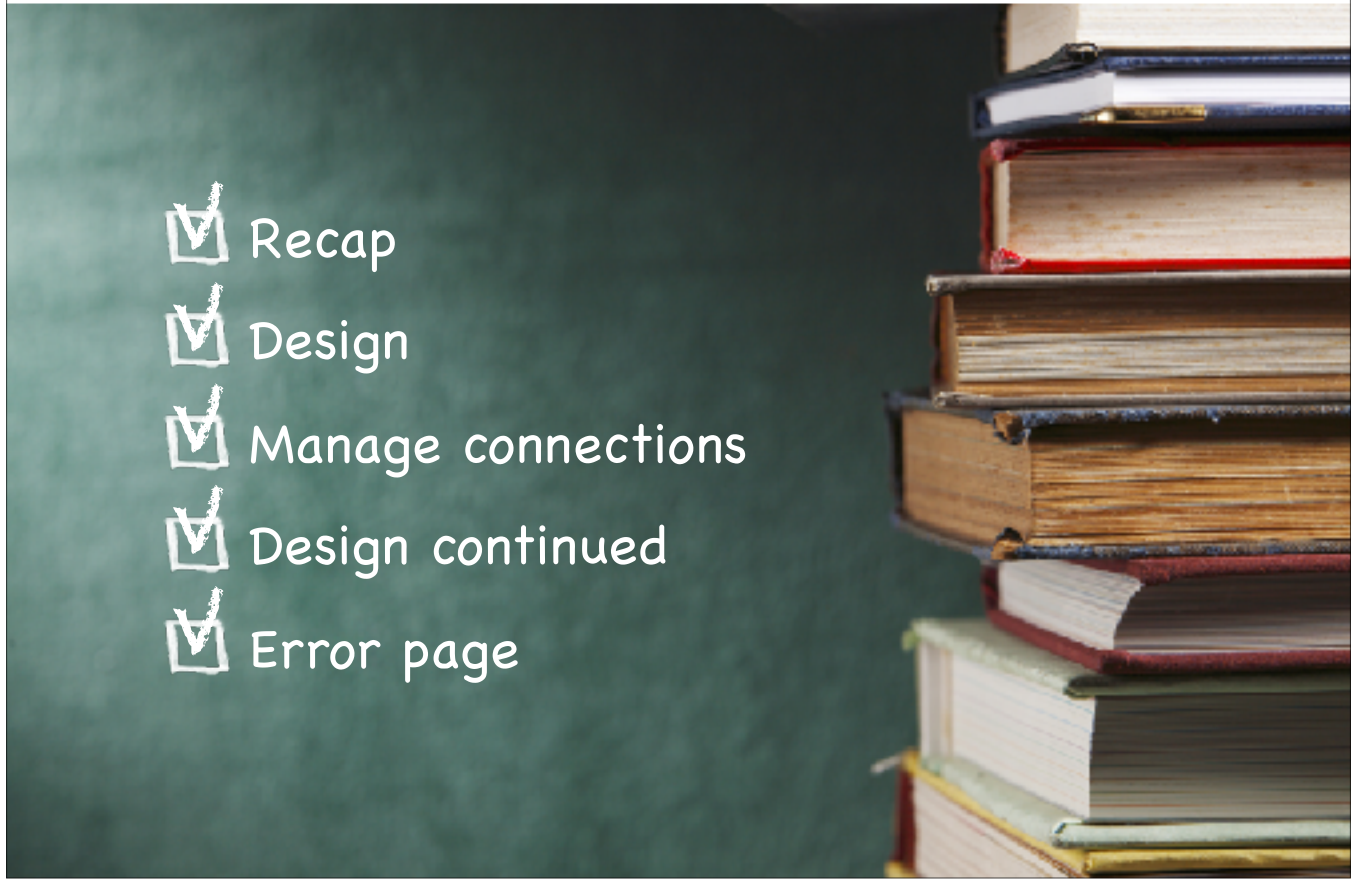
REMARK

2 WAYS OF EXCEPTION HANDLING

- Validation
  - user did something wrong
  - catch exceptions from model
  - show a message on the same page

- Other
  - something unexpected went wrong
  - show error page

AGENDA



- ☑ Recap
- ☑ Design
- ☑ Manage connections
- ☑ Design continued
- ☑ Error page