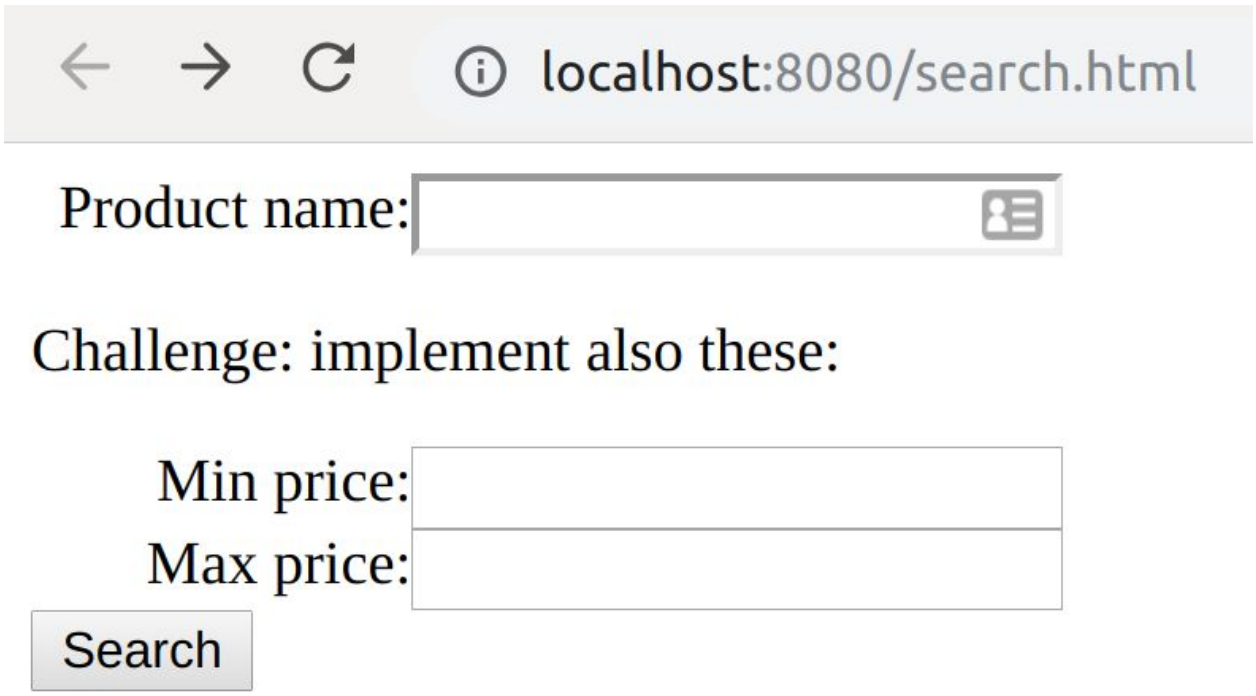
Övning i JDBC och Servlet - sök i Systembolaget

Vi ger er en databas (med lite gammal data från våren 2018 eller så) med produkter från systembolaget.

Ni får en Servlet som klarar av att söka efter produktnamn (namnet söks upp i databasen med hjälp av LIKE ‘%namnet%’) och returnerar JSON med data för de produkter som matchar namnet.

Exempel



Endast namn-sök fungerar i koden ni får.

Söker man efter namnet “Loch Lomond”, så får man en JSON-lista:


```
[
    {
    "name": "Loch Lomond Inchmurrin 12 Years",
    "price": 559.00,
    "volume": 700,
    "alcohol": 46.00,
    "nr": 8550801,
    "product_group": "Whisky"
  },
```

```
    {
    "name": "Loch Lomond Càrn Mòr 6 Years",
    "price": 628.00,
    "volume": 700,
    "alcohol": 46.00,
    "nr": 8567401,
    "product_group": "Whisky"
  },
    {
    "name": "Loch Lomond Single Malt",
    "price": 299.00,
    "volume": 700,
    "alcohol": 40.00,
    "nr": 59701,
    "product_group": "Whisky"
  },
    {
    "name": "Loch Lomond Single Grain",
    "price": 369.00,
    "volume": 700,
    "alcohol": 46.00,
    "nr": 1011801,
    "product_group": "Whisky"
  }
]
```


Sökfunktionen är känslig för SQL-injection, så ni kan öva på att byta från `Statement` till `PreparedStatement`.

Er uppgift går ut på att utöka `Servleten`s förmågor med att kunna agera på värden i fälten för max price och min price och kombinera detta med name.

Värdena `min_price` respektive `max_price` skickas via HTTP som en del av URL:en. I servleten kan de hämtas ut med t ex

```
String max = request.getParameter("max_price");
```

Filer som ni får av oss

Här är en överblick över de filer ni får av oss:

```
.
|-- clean.sh
|-- compile_servlet.sh
|-- download_sqlite_driver.sh
|-- download_winstone.sh
|-- start_winstone.sh
|-- webroot
|   |-- search.html
|   |-- style.css
|   `-- WEB-INF
|       |-- classes
|       |   `-- servlets
|       |       |-- DB.java
|       |       |-- FormatterFactory.java
|       |       |-- Formatter.java
|       |       |-- JsonExporter.java
|       |       |-- JsonFormatter.java
|       |       |-- Product.java
|       |       `-- SystemetWebAPI.java
|       |-- db
|       |   `-- bolaget.db
|       |-- lib
|       |   `-- sqlite-jdbc.jar
|       `-- web.xml
`-- winstone.jar

6 directories, 18 files
```

Det borde inte vara några konstigheter för er.

Vi börjar med Bash-skripten:

- `clean.sh` - tar bort class-filer så ni kan kompilera från scratch
- `compile_servlet.sh` - kompilerar servleten (och alla filer den behöver)
- `download_sqlite_driver.sh` - laddar ned sqlite-drivrutin till `webroot/WEB-INF/lib` (om ni vill ha en nyare version)
- `download_winstone.sh` - laddar ned `winstone.jar` (behövs för att kompilera servleten, samt för att köra webbservern och servlet-containern)
- `start_winstone.sh` - startar servletcontainern Winstone och lyssnar på localhost:8080

Länk till koden

https://github.com/progund/java-extra-lectures/raw/master/jdbc-servlet-systembolaget/systembolaget-jdbc.zip

Klasser ni fått av oss

- `servlets/SystemetWebAPI.java` - Servleten
- `servlets/Product.java` - Representerar en produkt från databasen
- `servlets/JsonExporter.java` - Hjälpklass för att göra om en Product till JSON
- `servlets/DB.java` - En klass med JDBC-anrop mot databasen
- `servlets/FormatterFactory.java` - Factory för att få tag i en `Formatter`
- `servlets/Formatter.java` - Ett interface som kan formatera en `List<Product>`
- `servlets/JsonFormatter.java` - En formatter som kan göra om en `List<Product>` till JSON

Filer i webroot

- `search.html` - formuläret för att söka efter produkter
  - `style.css` - lite stil för formuläret
  - `WEB-INF/web.xml` - konfiguration av servleten
  - `db/bolaget.db` - en SQLite3-databas med systembolagets produkter

Kom igång med övningen

När ni laddat ned filerna, så ska ni köra:

- `./download_winstone.sh`
- `./download_sqlite_driver.sh`
- `./compile_servlet.sh`
- `./start_winstone.sh`

Gå till http://localhost:8080/search.html och prova att sök efter Carlsberg . OBS! min price och max price fungerar alltså inte - det är det ni ska lägga till funktionalitet för i Servlet:en.

Vad gör Servleten

Servleten ligger i katalogen `webroot/WEB-INF/classes/servlets/SystemetWebAPI.java` och har klassnamnet `servlets.SystemetWebAPI`.

I doGet() så sker följande:

- Response content-type sätts till `application/JSON`
- En `PrintWriter out`, hämtas för att skriva svaret (via HTTP)
- Parametern product_name hämtas från GET
    - Om parametern är tom skrivs ett JSON-error
- En `List<Product> products` hämtas från klassen DB via den statiska metoden `getProductsByName(productName)`
- En `Formatter` fås från `FormatterFactory`
    - I dagsläget finns bara formatter för JSON
- En String `json` fås från formattern via metodanropet `formatter.format(products)`
- Resultatet skrivs ut på `out`

Databasen

Databasen ligger i `./webroot/WEB-INF/db/bolaget.db` och har följande schema:

```
$ sqlite3 ./webroot/WEB-INF/db/bolaget.db '.schema'
CREATE TABLE productGroup(id INT PRIMARY KEY NOT NULL, name text);
CREATE TABLE product(nr INT primary key not null,
                     name text, price REAL, alcohol REAL,
                     volume INT, productGroupId INT, type text,
foreign key(productGroupId) references productGroup(id));
```

Här listas alla name, price, alcohol, volume för produkter med namn som innehåller ‘Loch Lomond’ (utan att blanda in tabellen productGroup):

```
$ sqlite3 ./webroot/WEB-INF/db/bolaget.db "SELECT name, price,
alcohol, volume FROM product WHERE name LIKE '%loch lomond%'"
Loch Lomond Inchmurrin 12 Years|559.0|46.0|700
Loch Lomond Càrn Mòr 6 Years|628.0|46.0|700
Loch Lomond Single Malt|299.0|40.0|700
Loch Lomond Single Grain|369.0|46.0|700
```

Vad ni ska göra

Ni ska alltså lägga till att servleten läser parametrarna `max_price` och `min_price` och om de inte är null eller är tomma, så ska ni använda dessa för att hämta data med namn och antingen max-pris eller min-pris.

Ni kan lösa detta på många sätt. Här är ett enkelt förslag.

Enkel lösning 1

Tillåt bara en av max_price och min_price tillsammans med product_name. Skapa två metoder i DB:

```
public static List<Product> getProductsByNameAndMaxPrice(String pName, double price)
public static List<Product> getProductsByNameAndMinPrice(String pName, double price)
```

Utgå från den SELECT som redan finns i getProductsByName och använd price för att skapa rätt kriterium (where-klausul).

Här är ett mer komplicerat förslag:

Lite svårare lösning

Skapa en metod:

```
public static List<Product>
 getProductsByNameInPriceRange(String pName, double from, double to)
```

Anropa alltid den här metoden. Om bara name skickats med till servleten, använd -1 som `from` och 9999999 som `to` . Om bara name och min skickats med, använd min_price som from och 9999999 som to . Om bara name och max skickats med, använd -1 som from och max_price som to.

Bygg upp en SQL som sätter rätt kriterium för att få produkter inom prisspannet. Tips: funktionen BETWEEN i SQL kan vara användbar.

Överkurs, när ni är klara - ***gör första delen först***!

Här är ett annat sätt som är svårare Java men gör att ni inte behöver fler metoder med SQL/JDBC. Se till att DB alltid hämtar alla produkter om alla produkter inte redan hämtats en gång (tar lite minne, men det har vi så det räcker).

Skapa en metod i DB som hämtar alla (eller en cache:ad version av alla) produkter. Skapa ett filter någonstans (i DB eller i en ny klass) som kan ta en lista med produkter och ett Predicate för produkter, och returnerar en ny lista med alla produkter som uppfyller predikatet.

Servleten får bygga upp ett predikat utifrån parametrarna:

- bara name
- name + min price
- name + max price
- name + min price + max price

Servleten ber då att få en filtrerad lista genom att skicka in predikatet till filtermetoden.

Denna lösning gör tyvärr att ni inte över på JDBC utan på något annat, så gör en version med JDBC innan ni försöker er på detta.

OBS! Denna lösning gör att ni kan filtrera på vad som helst (till exempel lägga till fält för min alcohol, max alcohol osv) utan att ni behöver ändra i er databaskod. Ni behöver bara bygga upp ett predikat som har rätt kriterier och skicka in det till filtermetoden. Nackdelen är att ni måste läsa in alla produkter i minnet och cachea dem där, så databasen blir ju gammal så fort ni ändrar i den riktiga databasen. Man kan lösa detta genom att skapa funktionalitet för att kasta den cacheade versionen av “alla produkter” och läsa in en ny.

Här är lite exempelkod för hur ni skulle kunna bygga predikat:

Exempel på att skapa Predicate<Product>

```
import java.util.function.Predicate;
import java.util.List;
///////////////
void someMethod() {
  // ett predikat för max_price = 20
  Predicate<Product> maxPriceFilter = p -> p.price() <= 20;
  List<Product> products =
       DB.getProductsFilteredBy(maxPriceFilter);
  products.stream().forEach(System.out::println);
  }
}
```

Hur getProductsFilteredBy(Predicate<Product>) fungerar får ni lista ut själva, men några tips är:

- man kan få en stream from en collection
- som man kan applicera ett filter på med predikat som argument
- och så kan man samla ihop en ny collection med metoden collect och en collector som argument - t ex Collectors.toList()

Här är ett enkelt exempel:

```
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.List;
import java.util.Arrays;

public class FilterExample {
  public static void main(String[] args) {

    Integer[] numbers = {1,2,3,4,5,6,7,8,9};
    List<Integer> list = Arrays.asList(numbers);
    Predicate<Integer> even = i -> i % 2 == 0;
    System.out.println(filter(list, even));
  }
  static List<Integer>filter(List<Integer> list, Predicate<Integer> p) {
    return list.stream().filter(p).collect(Collectors.toList());
  }
}
$ javac FilterExample.java && java FilterExample
[2, 4, 6, 8]
```