Workshop
Getting started with Lab 1

Getting the Servlet up and running

Prepare

- Download the files from the link provided by the teachers
- Create a directory where you will be working - name it something useful, like tig058-lab1
- Put the files in this directory
- Verify the directory structure

Directory structure

```
.
|-- clean.sh
|-- deploy_systemet_jar.sh
|-- download_sqlite_driver.sh
|-- download_winstone.sh
|-- run_servlet.sh
|-- webroot
|   |-- api.html
|   |-- search.html
|   |-- style.css
|   `-- WEB-INF
|       |-- classes
|       |   `-- servlets
|       |       |-- FormatterFactory.java
|       |       |-- Formatter.java
|       |       |-- JsonFormatter.java
|       |       |-- ParameterParser.java
|       |       `-- SystemetWebAPI.java
|       |-- db
|       |   `-- bolaget.db
|       |-- lib
|       |   |-- sqlite-jdbc.jar
|       |   `-- systemet.jar
|       `-- web.xml
```

Directory structure, continued (systemet-api)

```
|-- systemet-api
|   |-- build_jar.sh
|   |-- build.sh
|   |-- generate_csv.sh
|   |-- products.csv
|   |-- examples
|   |   |-- ApiExample.java
|   |   `-- ProductsFromCSV.java
|   |-- se
|   |   `-- itu
|   |       `-- systemet
|   |           |-- domain
|   |           |   |-- package-info.java
|   |           |   `-- Product.java
|   |           `-- storage
|   |               |-- FakeProductLine.java
|   |               |-- JsonExporter.java
|   |               |-- package-info.java
|   |               |-- ProductLineFactory.java
|   |               |-- ProductLine.java
|   |               `-- SQLBasedProductLine.java
|   `-- systemet.jar
`-- winstone.jar
```

Investigate the scripts

These are scripts provided for you:

- `clean.sh` - deletes all class files recursively (in current and all sub directories)
- `download_sqlite_driver.sh`
  - downloads the Java driver for SQLite3 (for Lab3 - ignore for now)
- `download_winstone.sh` - downloads the Servlet engine Winstone
- `run_servlet.sh` - compiles the Servlet classes and runs the Servlet engine Winstone
- `Deploy_systemet_jar.sh` – see below

Make the scripts executable:

```
chmod u+x script_name.sh
```

Run a script:

```
./script_name.sh
```

Investigate the classes for the servlet and API

The servlet uses a few classes included under `webroot/WEB-INF/classes/`

The servlet also uses a JAR-file with the Systemet API (we've provided for you)

- This JAR-file should be placed in `webroot/WEB-INF/lib/` (done by a script)
- The source code for the Systemet API is put in the directory `systemet-api/`
- To build and put the JAR file in the right place, you use the script
- `./deploy_systemet_jar.sh` which will compile, create the jar file and put it in the correct place (`webroot/WEB-INF/lib/`) so that the servlet can use it.

Overview

Servlet code:
webroot/WEB-INF/classes/

<<uses>>

systemet.jar:
placed in
webroot/WEB-INF/lib/

Classes in webroot/WEB-INF/classes/

servlets.SystemetWebAPI	(The servlet)
servlets.ParameterParser	(Parses the request)
servlets.FormatterFactory	(Creates the Formatter)
servlets.JsonFormatter	(Formats Product:s)

Classes in systemet.jar:

se.itu.systemet.domain.Product	(Represents a Product)
se.itu.systemet.storage.ProductLineFactory	(Creates a product line)
se.itu.systemet.storage.ProductLine	(Represents all products)
se.itu.systemet.storage.JsonExporter	(Creates Json from Product)
se.itu.systemet.storage.FakeProductLine	(Creates products from fake db)

Reading the Javadoc for the “systemet” api

You must read and understand the following:

How is a new `Product` created? (Using the `Product.Builder` class)

What is a `ProductLine`?

The `ProductLineFactory` will return a new `FakeProductLine` object to the servlet. It’s methods only return an empty list of products (no products).

Your task will be to implement the `FakeProductLine` class so that it returns a list with quite a few `Product` objects, so that you can test the servlet.

Systemet API walk-through

The purpose of the API is to provide an API for accessing and representing a product line and a products.

Main classes/interfaces:

- `se.itu.systemet.domain.Product` (represents a `Product`)
- `se.itu.systemet.storage.ProductLineFactory` (creates a `ProductLine`)
- `se.itu.systemet.storage.ProductLine` (represents the whole product line)
- `se.itu.systemet.storage.JsonExporter` (exports a Product to a JSON object)

Systemet API walk-through

Your Lab1 is about creating a product line (a class implementing ProductLine) which provides a list of products.

Since we don’t know databases yet, you will work with the class `se.itu.systemet.storage.FakeProductLine` which is the class the `se.itu.systemet.storage.ProductLineFactory` will use, the way the Servlet is configured.

More on this later.

Systemet API walk-through

The way client code (e.g. a Servlet or stand-alone program) uses the API:

```
package examples;

import java.util.function.Predicate;
import java.util.List;
import se.itu.systemet.domain.Product;
import se.itu.systemet.storage.ProductLine;
import se.itu.systemet.storage.ProductLineFactory;

public class ApiExample {
  public static void main(String[] args) {
    ProductLine productLine = ProductLineFactory.getProductLine();
    Predicate<Product> maxPriceFilter = p -> p.price() <= 20;
    List<Product> products = productLine.getProductsFilteredBy(maxPriceFilter);
    // Do something with the products...
  }
}
```

Don’t worry - The Servlet is written for you

The Servlet is already written, so the parsing of the GET parameters are translated to a Predicate<Product>, and a filtered List<Product> is retrieved and converted to JSON and returned. The servlet has 6 responsibilities:

1. Setting the correct content-type application/json
2. Parsing the GET parameters (e.g. min_price=50)
3. Creating a Predicate<Product> to use as a filter
4. Getting the filtered products from a product line
5. Formatting the products as JSON
6. Writing the response (JSON) to the client requesting products

Servlet source code excerpt

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException{
  request.setCharacterEncoding(UTF_8.name());
  response.setContentType("application/json;charset="+UTF_8.name());
  PrintWriter out =
    new PrintWriter(new OutputStreamWriter(response.getOutputStream(),
                                           UTF_8), true);
  ParameterParser paramParser = new ParameterParser(request.getQueryString().split("&"));
  Predicate<Product> filter = paramParser.filter();
  ProductLine productLine = ProductLineFactory.getProductLine();
  List<Product> products = productLine.getProductsFilteredBy(filter);

  Formatter formatter = FormatterFactory.getFormatter();
  String json = formatter.format(products);
  StringBuilder sb = new StringBuilder(json);
  out.println(sb.toString());
  out.close();
}
```

Servlet source code excerpt - 1. content type etc

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException{
  request.setCharacterEncoding(UTF_8.name());
  response.setContentType("application/json;charset="+UTF_8.name());
  PrintWriter out =
    new PrintWriter(new OutputStreamWriter(response.getOutputStream(),
                                           UTF_8), true);
  ParameterParser paramParser = new ParameterParser(request.getQueryString().split("&"));
  Predicate<Product> filter = paramParser.filter();
  ProductLine productLine = ProductLineFactory.getProductLine();
  List<Product> products = productLine.getProductsFilteredBy(filter);

  Formatter formatter = FormatterFactory.getFormatter();
  String json = formatter.format(products);
  StringBuilder sb = new StringBuilder(json);
  out.println(sb.toString());
  out.close();
}
```

Servlet source code excerpt - 2.parsing the params

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException{
  request.setCharacterEncoding(UTF_8.name());
  response.setContentType("application/json;charset="+UTF_8.name());
  PrintWriter out =
    new PrintWriter(new OutputStreamWriter(response.getOutputStream(),
                                           UTF_8), true);
  ParameterParser paramParser = new ParameterParser(request.getQueryString().split("&"));
  Predicate<Product> filter = paramParser.filter();
  ProductLine productLine = ProductLineFactory.getProductLine();
  List<Product> products = productLine.getProductsFilteredBy(filter);

  Formatter formatter = FormatterFactory.getFormatter();
  String json = formatter.format(products);
  StringBuilder sb = new StringBuilder(json);
  out.println(sb.toString());
  out.close();
}
```

Servlet source code excerpt - 3. getting the filter

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException{
  request.setCharacterEncoding(UTF_8.name());
  response.setContentType("application/json;charset="+UTF_8.name());
  PrintWriter out =
    new PrintWriter(new OutputStreamWriter(response.getOutputStream(),
                                           UTF_8), true);
  ParameterParser paramParser = new ParameterParser(request.getQueryString().split("&"));
  Predicate<Product> filter = paramParser.filter();
  ProductLine productLine = ProductLineFactory.getProductLine();
  List<Product> products = productLine.getProductsFilteredBy(filter);

  Formatter formatter = FormatterFactory.getFormatter();
  String json = formatter.format(products);
  StringBuilder sb = new StringBuilder(json);
  out.println(sb.toString());
  out.close();
}
```

Servlet source code excerpt - 4. fetching products

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException{
  request.setCharacterEncoding(UTF_8.name());
  response.setContentType("application/json;charset="+UTF_8.name());
  PrintWriter out =
    new PrintWriter(new OutputStreamWriter(response.getOutputStream(),
                                           UTF_8), true);
  ParameterParser paramParser = new ParameterParser(request.getQueryString().split("&"));
  Predicate<Product> filter = paramParser.filter();
  ProductLine productLine = ProductLineFactory.getProductLine();
  List<Product> products = productLine.getProductsFilteredBy(filter);

  Formatter formatter = FormatterFactory.getFormatter();
  String json = formatter.format(products);
  StringBuilder sb = new StringBuilder(json);
  out.println(sb.toString());
  out.close();
}
```

Servlet source code excerpt - 5. Products to JSON

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException{
  request.setCharacterEncoding(UTF_8.name());
  response.setContentType("application/json;charset="+UTF_8.name());
  PrintWriter out =
    new PrintWriter(new OutputStreamWriter(response.getOutputStream(),
                                           UTF_8), true);
  ParameterParser paramParser = new ParameterParser(request.getQueryString().split("&"));
  Predicate<Product> filter = paramParser.filter();
  ProductLine productLine = ProductLineFactory.getProductLine();
  List<Product> products = productLine.getProductsFilteredBy(filter);

  Formatter formatter = FormatterFactory.getFormatter();
  String json = formatter.format(products);
  StringBuilder sb = new StringBuilder(json);
  out.println(sb.toString());
  out.close();
}
```

Servlet source code excerpt - 6. The response

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException{
  request.setCharacterEncoding(UTF_8.name());
  response.setContentType("application/json;charset="+UTF_8.name());
  PrintWriter out =
    new PrintWriter(new OutputStreamWriter(response.getOutputStream(),
                                           UTF_8), true);
  ParameterParser paramParser = new ParameterParser(request.getQueryString().split("&"));
  Predicate<Product> filter = paramParser.filter();
  ProductLine productLine = ProductLineFactory.getProductLine();
  List<Product> products = productLine.getProductsFilteredBy(filter);

  Formatter formatter = FormatterFactory.getFormatter();
  String json = formatter.format(products);
  StringBuilder sb = new StringBuilder(json);
  out.println(sb.toString());
  out.close();
}
```

Back to the task for Lab1

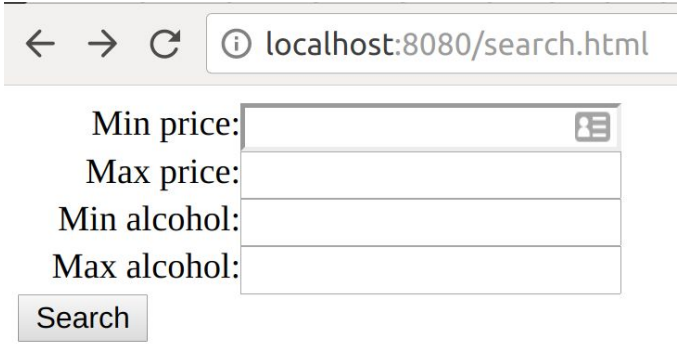
You should create the FakeProductLine in the Systemet API, compile the whole API, package the API as a JAR file, and, deploy the JAR file to the Servlet.

Let’s first see how the Servlet is started.

How to start the servlet container

Use the ./`run_servlet.sh` script to start Winstone running your servlet. If everything compiles, you will then be able to point your browser to:

http://localhost:8080/search.html which is a static HTML page with a form for searching for products, so that you can test your FakeProductLine



How to start the servlet container

As with all shell scripts, you need to make them executable:

```
chmod u+x run_servlet.sh
```

For trouble shooting bash scrips etc, please see our FAQ

http://wiki.juneday.se/mediawiki/index.php/FAQ

Your first task - Write a fake product line

What is a `ProductLine` ?

A a `ProductLine` is an interface in the Systemet-API which represents all the products in Systembolaget's catalog. The simple interface defines just a few methods for retrieving a `List<Product>` from the product line, either all products or products that satisfy some filter criteria.

We chose to represent this functionality as an interface, since it is not important for e.g. the Servlet *how* the products are fetched, just that they *can* be fetched.

Your first task - Write a fake product line

The idea is to let the servlet get a list of Product objects, just the same way it would from the database. But we haven't learned enough database stuff yet, so does this mean we can't write and test the servlet?

No. The servlet uses a Factory class to get a `ProductLine` (which is an interface) object. The servlet is configured to make the factory return a `FakeProductLine` (which implements `ProductLine`, so *it is a* `ProductLine`).

You will add products manually to a `List<Product>` in this class, and the servlet won't know that it didn't come from the database.

Add products wisely

The search page lets you search for products filtered on four criteria:

- minimum price
- maximum price
- minimum alcohol by volume
- maximum alcohol by volume

So it would be wise to add a lot of products which differ a lot on those factors (price and alcohol). Also it is good to add products of different product group (like “Öl” (beer) “ We’ll give you some suggestions on the next page.

Example fake products

Name, price, alcohol, volume, nr, product group, type (type may be empty)
"Johanneshof Reinisch Pinot Noir","143.0","13.0","750","7440201","Rött vin",""
"Dobogó Tokaji Furmint","187.0","13.5","750","7598701","Vitt vin",""
"Château de Cazeneuve Carignan","250.0","13.5","750","7105201","Rött vin",""
"Engelholms Pacific Pilsner","21.0","5.0","330","3067603","Öl","Ljus lager"
"Speyside Sherry Cask 21 Years","1537.0","55.1","700","8591801","Whisky","Malt"
"Giró i Giró Montaner Brut Nature Gran Reserva","151.0","11.5","750","7723101","Mousserande vin","Vitt Torrt"
"Clynelish No 4051 17 Years","1583.0","55.0","700","8515601","Whisky","Malt"
"Albarossa","252.0","14.0","750","7321901","Rött vin",""
"Los Frailes Monastrell Garnacha","115.0","13.5","750","7444301","Rött vin",""
"Auchentoshan Maltbarn Bourbon Cask 23 Years","1735.0","52.0","700","8537101","Whisky","Malt"
"Albin Jacumin Châteauneuf-du-Pape","246.0","14.5","750","7551101","Rött vin",""
"Bibbiano Chianti Classico","147.0","13.5","750","7539001","Rött vin",""
"Chapter 7 Irish Whiskey Bourbon Hogshead 14 Years","1113.0","56.7","700","8771001","Whisky","Malt"
"Amour de Deutz","796.0","12.0","375","9602502","Mousserande vin","Vitt Torrt"
"Deutz Exclusive Gift Box","2397.0","12.0","1125","9696209","Mousserande vin",""
"Villa Spinosa Valpolicella Classico","130.0","12.0","750","7249001","Rött vin",""
"D'Aria Sauvignon Blanc","146.0","13.0","750","7337501","Vitt vin",""
"Fairtransport Tres Hombres 21 Años","885.0","41.6","700","8712101","Rom","Mörk"
"Donatella Cinelli Colombini","1104.0","14.0","4500","7475209","Rött vin",""
"Abrigo Giovanni Piemonte Mix 1","732.0","13.5","4500","7096809","Rött vin",""

You can generate a CSV using a script

In `systemet-api/` there's a script `generate_csv.sh` which you can use to generate a CSV with products from the database.

Don't forget to `chmod u+x` the script, and run it with `./generate_csv.sh`

How to deploy the systemet API to the Servlet

You have a script `deploy_systemet_jar.sh` which will compile all classes, create a Jar-file (`systemet.jar`) and put it in a directory where the servlet can access it (`webroot/WEB-INF/lib/`).

The servlet will import some classes from the Systemet API jar file, so it needs to exist in that directory.

Using a Jar file for the Systemet API with the `Product` and `ProductLine` etc, allows you to work on that api on one computer, and run the servlet using it on another computer. You can even have one team working on the Systemet API and another team working on the web API (servlet).

Feel free to add more (we encourage that)

The keen student can use a comma separated values file in the FakeProductLine class for an easy way to create products. Remember to remove the quotes from the previous page, those are not part of the actual values, but rather there to show you where fields start and end.

We've included an example command line application showing how you could parse a CSV file to a List<Product> if you'd be interested in doing that.

Challenges

The following parts are optional challenges only for those who want something harder:

- Add a valid parameter to the servlet ParameterParser (e.g. product_group)
  - Update the API documentation file in webroot/api.html if you do this!
- Use a csv file for creating products in the FakeProductLine (see above)
- Test your servlet to see that it filters out the correct products and gives you a correct Json (Teachers can provide an example)
- Validate the Json you get from the servlet using jq or some other validating parser for Json

Challenges

Some more challenges for those who want something harder could be:

- Make the Servlet use correct HTTP response codes (see the Web API documentation) for cases such as
  - Couldn’t find any products (the filter was too narrow or wrong)
  - Couldn’t understand the filter
  - General failure (some kind of 500 error, the servlet failed to to its job)
- Add a JSON response explaining errors

Please note that you should always finish the mandatory part and be prepared to send it in before the deadline before starting on the challenges, so that you don’t ruin a working solution to the required tasks.

Challenges

- Add a JSON response explaining errors

Such a response could be sent from the servlet so that a client could see why something went wrong and what went wrong. E.g. A request results in no products matching the criteria, bad criteria etc:

```
{ error: "No products matched the criteria" }

{ error: "The server didn’t understand the request" }

{ error: "The server encountered technical problems" }
```

Write a diary - You must document what you do

Each member of your team must write a personal diary, documented what parts you wrote, what tasks you did and most importantly, what you have learned.

The diary should list what code you have written and how that code works.

You should also write down an explanation of what the Lab1 was all about, and how you solved it, what technologies were involved and how the over all web api works.

This diary will be the base for us when we check that all members of a group has contributed to the project code.

Write a diary - You must document what you do

The diary you hand in, should in the “technologies” section explain briefly the following:

- Servlets
- HTTP
- JSON
- Web API
- JAR file (e.g. the one with Systemet API)

Who does what in a group?

We strongly suggest that you have one master version of the system which you will hand in to the teachers. This should be a working system which produces the correct JSON and filters products correctly.

However, we strongly suggest that all members of the team writes a personal copy of the system, which will not be handed in. This is to make sure that you all understand and learn from all tasks in the assignment.

We will read the diaries to see what everybody has contributed to the group, and what everybody has learned. You should be able to explain your parts and how they fit into the system as a whole.