Exposing data lab Terms, and concepts

Revisiting terms, tech, design, classes and more

Contents

HTTP and Web server

- Servers are programs which run continuously, web servers too
- Web servers
    - Serve files to clients
        - Clients are typically web browsers but can be any application
    - “Talk” HTTP - a protocol defining how to communicate
- HTTP
    - Defines how to request a file or resource from a web server
    - What the response from the server will look like
- Resources can be HTML, images, text, data (json/xml)
- JSON usually is parsed by the client

HTTP communication example

Client requests https://www.gu.se/digitalAssets/1314/1314204_google.html

Server sends the file 1314204_google.html (which contains HTML code)

HTML is targeted at web browsers, who can make sense of it and present it as a web page

The browser looks at the Header-part of the response and finds that the file is HTML code, and knows what to do (parse and present it)

```
Content-Type: text/html; charset=UTF-8
```

When the HTTP response contains JSON

Server should send content-type: application/json

The client now knows that it is not HTML or an image, but text with data coded in JSON

Winstone

- Java application in a JAR file
- Is a web server (but with special capabilities)
- Talks HTTP with clients, typically requests for files
- Can send files from its webroot

If client sends, e.g. `GET /search.html` then winstone looks in its webroot directory and finds search.html and sends it with content-type text/html

Your client used: http://localhost:8080/search.html which in HTTP means:

contact localhost:8080 and send `GET /search.html`

Winstone

The GET /search.html request means that the file search.html must exist in the servlets webroot - / means “top directory” or “root directory” (from winstones perspective.

A http://localhost:8080/images/Henrik.jpg URL would thus mean

`GET /images/Henrik.jpg` and that the file must be in a directory images/ and that images/ in turn is in webroot: `/ images / Henrik.jpg`

This time winstone would use content-type: `image/jpeg` and the browser would know it’s an image and display the image for you.

Winstone’s special powers

- We can configure Winstone so it knows certain URL paths are not files
- Such paths should instead be handed over to a Servlet (Java program)
    - in web.xml we said `/search/*` was such a path
- Winstone sends the whole Request to the servlet as an argument to e.g. doGet
- Type is `HTTPServletRequest`
- Servlet responds with `HTTPServletResponse` to Winstone
- Winstone sends the HTTP response to the client (who knows nothing of what went on there)

When to use a Servlet

- images, html and other files in the webroot can be handled by any web server
- Other data must be produced and packaged before Winstone can send it
- Think about a search on a web page - the page with the results can’t exist in the webroot - we don’t know what people will search for!
- A Servlet can read the search phrase from the request and look up all pages containing the phrase and create HTML with links to those pages
- The browser creates a parameter with the search phrase and sends it to the web server (who forwards it to the Servlet)

Web page (HTML) with a search form and button

```
<html>
<body>
<form action="/search" method=”GET”>
Search: <input type="text" name="search_word" />
<input type="submit" value="Search!">
</form>
</body>
</html>
```

What the browser does when you click “Search”

If you type “Sport” in the search field and click the button, the browser will use the HTML in the page, particularly these parts:

- `action` (with the value `/search`)
  - Tells the browser to ask for the resource /search on the same server as the form
- `method` (with the value `GET`)
  - Tells the browser to use GET in the request
- `name` (with the value `search_word`) together with what you typed (“Sport”)
  - Will be encoded in the URL as `/search?`**`search_word=Sport`**

What happens on the server-side

- Winstone sees /seach in the URL path and knows to forward to Servlet
- Servlet reads the GET parameter `search_word` and finds the String Sport
- The Servlet uses “Sport” to find a list of each HTML file containing that string and creates an HTML response with links to those pages

Example HTML link (in the full HTML response):

```
<a href="/sport/sportnews.html">Read all about Henrik’s new record</a>
```

The Servlet can create HTML which to the client will look like any HTML page from a normal HTML file on the server. It can’t tell the difference.

Servlet

- Object of a class implementing interfacet `javax.servlet.Servlet`
- Inheriting the class `javax.servlet.http.HttpServlet` does the trick
- Think of it as a Java program called by a web server (like Winstone)
- Knows how to speak HTTP
- Two methods are overridden in the lab:
    - `void doGet(HttpServletRequest req, HttpServletResponse resp)` and
    - `void init()`
- doGet is what Winstone calls when it forwards the request
- The Servlet writes its response to the response object, which Winstone then notices, and uses to send the HTTP response to the client
- init() is for intialization of the Servlet (is run once per servlet)

localhost:8080

- localhost is the hostname for your own computer (always)
- IP address 127.0.0.1
- 8080 is called a port number and is usually used for web servers run by non-superusers (non-administrator accounts on some computer) but it’s not a standard port
- Ports allow us to run more than one server program on the same IP-number (same computer)

JSON

- A standard for formatting text with data, like a language for data
- Looks like JavaScript (and is actually JavaScript)
- Has arrays and objects
- Uses key-value pairs for data
- Uses { and } for an object, [and] for an array (with objects)

```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "phoneNumbers": [
      { "Mobile": "111-111-1111" },
      { "Home": "222-222-2222" }
   ]
}    /* borrowed from Oracle’s tutorial */
```

JSON

- Makes it possible to transfer data from one application to another
- Is like a neutral intermediary format for data
- Can be produced and interpreted by various programming languages
- Sender and receivers don’t need to be written in the same language
- Can be read by (some) humans, since it’s only text and very structured

Parsing

- In a program, to read some text from a known format and syntax and check it
- Check that it conforms with the format rules
- Convert the text to the programming language’s own data types
  - e.g List<Product>
- Using the previous JSON example, we could in Java check that it’s valid JSON, and read in one object of type Person
  - String firstName, lastName, street, city
  - int age
  - List<Phone> (where Phone is a class)

Double.parseDouble vs. Double.valueOf

- A form of parsing text to a Java data type
- Both are static methods in the class java.lang.Double
- parseDouble and valueOf differ in the return type:
    - parseDouble: primitive double return type
    - valueOf: A reference to a Double object
- You can use either if you want a double of primitive type or a Double reference - Java does “autoboxing” and “unboxing” (translates between the two)
- Similar methods exists for the classes Integer, Boolean, Long etc

Systembolaget’s XML file

- Never mind it! It was just some background information
- It was used by the teachers in order to get data to put in your database
- It has nothing to do with your system, it has already happened
- We parsed it to Java and inserted the data into a database with two tables
  - product
  - product_group
- You got the database and don’t have to explain anything about the XML file, other than that it is all that Systembolaget offers, but we have a database instead, and a Web API
- *If you really need to mention it at all, that is!*

Java Interfaces and their uses

The servlet uses a few interfaces written by the teachers. ProductLine which provides the methods getAllProducts() and getProductsFilteredBy(Predicate).

Having ProductLine being an interface:

- Allows us to switch out the FakeProductLine for an SqlBasedProductLine in lab3, without changing one line of code in the Servlet
- Interface protect us from detail changes
- The servlet shouldn’t care about where products come from, just about how to get products (what method to call and with what arguments)

Predicate

- An interface in java.util.function (in the standard API since Java 8)
- Declares only one abstract method, which makes it a so called functional interface
- `public abstract boolean test(T t)`
- Predicate is generic, so T stands for any class, just like List<T>
- If we write a class implementing Predicate<Product>, the test method will take a Product as the argument
- We can now test claims about a Product, e.g. The price is between 100 and 200 sek
- The method getProductsFilteredBy(Predicate<Product>) uses this

ParameterParser

Java code often consists of methods calling methods in objects. That’s because Java is an OO language where

- Objects are first class citizens, capable of performing specialized tasks
- We use objects in order to keep code dealing with such a specialized class hidden from code dealing with the larger picture
- We prefer abstractions
  - Let the ParameterParser object deal with the GET parameters and create a Predicate for us!

ParameterParser

- Performs the parsing of the GET parameters, which are text key-value pairs
- Parses the key-value pairs to a single Predicate<Product>
- Servlet uses this to get a List<Product> with only the products for which the predicate tests true
- Turns e.g. min_price=100&max_price=200 to a Predicate<Product> such that if we test any Product against it, we only get true if the product's price really is between 100 and 200 (inclusive)

With such a predicate, it can't be hard for the ProductLine to filter out products that fits the criteria, once expressed in the GET parameters.

ParameterParser

One (horrible) alternative to using an object to do the parsing would be to do it in the doGet() method itself. But that would mean some 45 lines of code just for ***one single task***. The servlet has a few tasks only:

1. Understand the GET parameters - what is the criteria for the products the client wants?
   a. E.g. `min_price=100&max_price=150` must be understood, so that only such products are returned (with a price between 100 and 150)
2. Respond with a JSON String with the correct products
   a. Create a Predikate object for Products corresponding to the GET parameters meanin
   b. Fech all such Products from a `ProductLine` using `productsFilteredBy(Predicate<Product>)`
   c. Convert the List<Product> to a JSON response

These few tasks, if done in doGet() would make the code grow and hard to maintain.