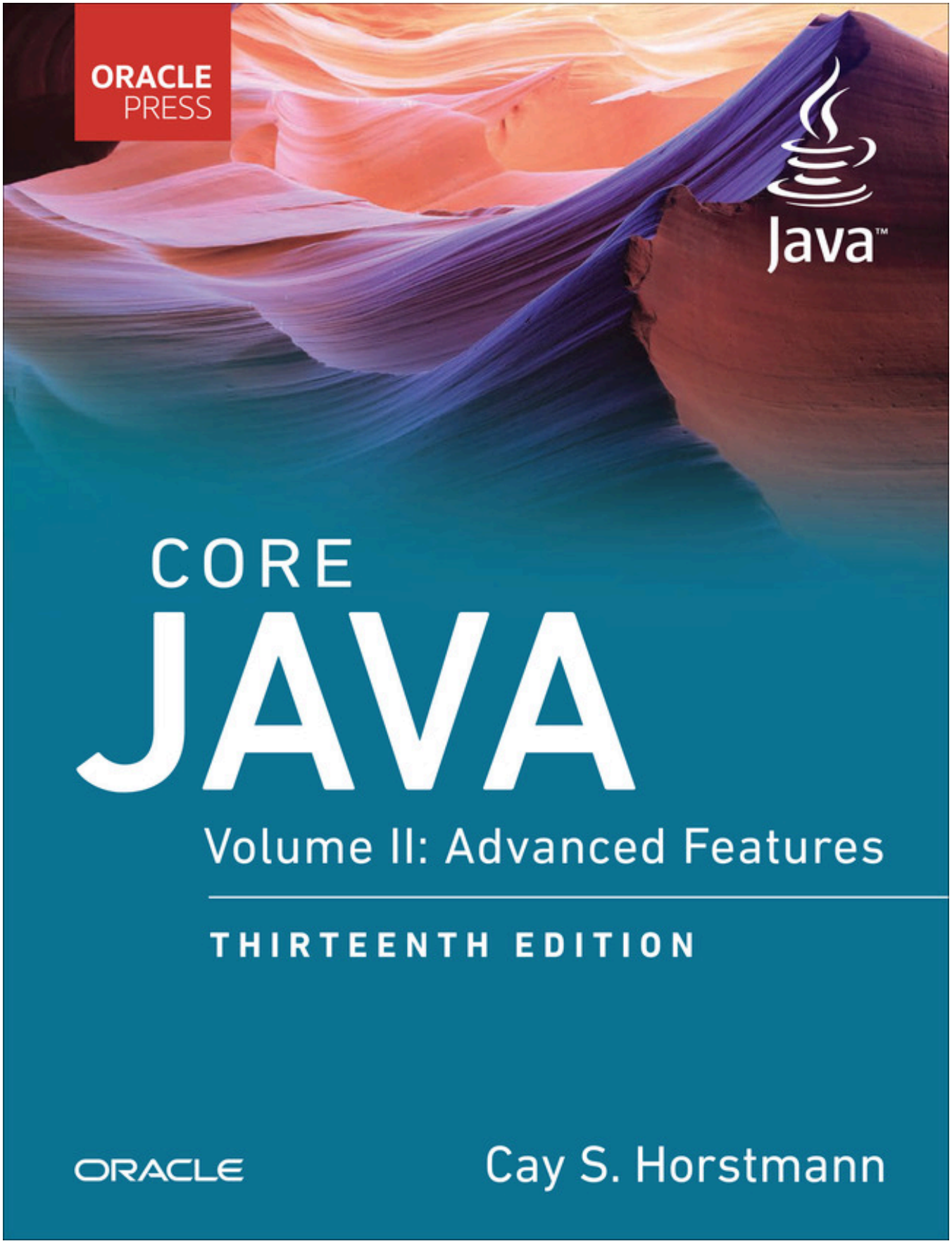
ORACLE
PRESS
Java™
CORE
JAVA
Volume II: Advanced Features
THIRTEENTH EDITION
ORACLE
Cay S. Horstmann

Java 16 adds a builder for filtering the headers of an existing `HttpRequest`. You provide the request and a function that receives the header names and values, returning `true` for those that should be retained. For example, here we modify the content type:

```
HttpRequest request2 = HttpRequest.newBuilder(request,
      (name, value) -> !name.equalsIgnoreCase("Content-Type")) // Remove old content type
   .header("Content-Type", "application/xml") // Add new content type
   .build();
```

4.4.3. The `HttpResponse` Interface and Body Handlers

When sending the request, you have to tell the client how to handle the response. If you just want the body as a string, send the request with a `HttpResponse.BodyHandlers.ofString()`, like this:

```
HttpResponse<String> response
   = client.send(request, HttpResponse.BodyHandlers.ofString());
```

`HttpResponse` is a generic interface whose type parameter denotes the type of the response body. You get the response body string simply as

```
String bodyString = response.body();
```

There are other response body handlers that get the response as a byte array or input stream. `BodyHandlers.ofFile(filePath)` yields a handler that saves the response to the given file, and `BodyHandlers.ofFileDownload(directoryPath)` saves the response in the given directory, using the file name from the `Content-Disposition` header. Finally, the handler obtained from `BodyHandlers.discarding()` simply discards the response.

Processing the contents of the response is not considered part of the API. For example, if you receive JSON data, you need a JSON library to parse the contents.

The `HttpResponse` object also yields the status code and the response headers.

```
int status = response.statusCode();
HttpHeaders responseHeaders = response.headers();
```

You can turn the `HttpHeaders` object into a map:

```
Map<String, List<String>> headerMap = responseHeaders.map();
```

The map values are lists since in HTTP, each key can have multiple values.

If you just want the value of a particular key, and you know that there won't be multiple values, call the `firstValue` method:

```
Optional<String> lastModified = responseHeaders.firstValue("Last-Modified");
```

You get the response value or an empty optional if none was supplied.

Note: The `HttpHeaders` class is aware that keys for HTTP parameters are case-insensitive. If you call `responseHeaders.firstValue("Last-Modified")`, you get the desired header value even if it was sent as `last-modified`.

4.4.4. Asynchronous Processing

You can process the response asynchronously. When building the client, provide an executor:

```
ExecutorService executor = Executors.newCachedThreadPool();
HttpClient client = HttpClient.newBuilder().executor(executor).build();
```

Build a request and then invoke the `sendAsync` method on the client. You receive a `CompletableFuture<HttpResponse<T>>`, where `T` is the type of the body handler. Use the `CompletableFuture` API as described in Chapter 10 of Volume I:

```
HttpRequest request = HttpRequest.newBuilder().uri(uri).GET().build();
client.sendAsync(request, HttpResponse.BodyHandlers.ofString())
   thenAccept(response -> . . .);
```

Tip: To enable logging for the `HttpClient`, add this line to `net.properties` in your JDK:

```
jdk.httpclient.HttpClient.log=all
```

Instead of `all`, you can specify a comma-separated list of the following event types:

- `channel`
- `content`
- `errors`
- `frames`, optionally followed by one or more of `:control`, `:data`, `:window`, or `:all`
- `headers`
- `requests`
- `ssl`
- `trace`

Don't use any spaces:

```
jdk.httpclient.HttpClient.log=headers,requests,frames:control:data
```

Then, set the logging level for the logger with name `jdk.httpclient.HttpClient` to `INFO`, for example by adding this line to the `logging.properties` file in your JDK:

```
jdk.httpclient.HttpClient.level=INFO
```

The program in Listing 4.10 shows how to make synchronous requests to different services. Run it as:

```
java httpClient.HttpClientTest form.properties
java httpClient.HttpClientTest fileupload.properties
java httpClient.HttpClientTest json.properties
```

With each run, a "Hello, World" program is sent to the CodeCheck service, and the result of running it is displayed. That service can accept form posts, file uploads, and JSON posts. Note the logging output from the `HttpClient`.

Listing 4.10 httpClient/HttpClientTest.java

```
1  package httpClient;
2
3  import java.io.*;
4  import java.net.*;
5  import java.nio.charset.*;
6  import java.nio.file.*;
7  import java.util.*;
8  import java.util.stream.Stream;
9  import java.net.http.*;
10 import java.net.http.HttpRequest.*;
11
12 /**
13  * This program demonstrates the HTTP client
14  * @version 1.02 2023-08-16
15  * @author Cay Horstmann
16  */
17 class MoreBodyPublishers
18 {
19    public static BodyPublisher ofFormData(Map<Object, Object> data)
20    {
21       boolean first = true;
22       var builder = new StringBuilder();
23       for (Map.Entry<Object, Object> entry : data.entrySet())
24       {
25          if (first) first = false;
26          else builder.append("&");
27          builder.append(URLEncoder.encode(entry.getKey().toString(),
28             StandardCharsets.UTF_8));
29          builder.append("=");
30          builder.append(URLEncoder.encode(entry.getValue().toString(),
31             StandardCharsets.UTF_8));
32       }
33       return BodyPublishers.ofString(builder.toString());
```

```
34      }
35
36      public static BodyPublisher ofMimeMultipartData(Map<String, List<?>> data, String boundary)
37            throws IOException
38      {
39         var bps = new ArrayList<BodyPublisher>();
40         var header = new StringBuilder();
41         for (Map.Entry<String, List<?>> entry : data.entrySet())
42         {
43            for (Object value : entry.getValue())
44            {
45               header.append("--%s\r\nContent-Disposition: form-data; name=%s".formatted(
46                     boundary, entry.getKey()));
47
48               if (value instanceof Path path)
49               {
50                  header.append("; filename=\"%s\"\r\n".formatted(path.getFileName()));
51                  String mimeType = Files.probeContentType(path);
52                  if (mimeType != null) header.append("Content-Type: %s\r\n".formatted(mimeType));
53                  header.append("\r\n");
54                  bps.add(BodyPublishers.ofString(header.toString()));
55                  bps.add(BodyPublishers.ofFile(path));
56               }
57               else
58               {
59                  header.append("\r\n\r\n%s\r\n".formatted(value));
60                  header.append("\r\n");
61                  bps.add(BodyPublishers.ofString(header.toString()));
62               }
63               header = new StringBuilder("\r\n");
64            }
65         }
66         bps.add(BodyPublishers.ofString("\r\n--%s--\r\n".formatted(boundary)));
67         return BodyPublishers.concat(bps.toArray(BodyPublisher[]::new));
68      }
69
70      public static BodyPublisher ofSimpleJSON(Map<Object, Object> data)
71      {
72         var builder = new StringBuilder();
73         builder.append("{");
74         var first = true;
75         for (Map.Entry<Object, Object> entry : data.entrySet())
76         {
77            if (first) first = false;
78            else
79               builder.append(",");
80            builder.append(jsonEscape(entry.getKey().toString())).append(": ")
81               .append(jsonEscape(entry.getValue().toString()));
82         }
83         builder.append("}");
84         return BodyPublishers.ofString(builder.toString());
85      }
```

```
86
87    private static Map<Character, String> replacements = Map.of('\b', "\\b", '\f', "\\f",
88       '\n', "\\n", '\r', "\\r", '\t', "\\t", '"', "\\\"", '\\', "\\\\");
89
90    private static StringBuilder jsonEscape(String str)
91    {
92       var result = new StringBuilder("\"");
93       for (int i = 0; i < str.length(); i++)
94       {
95          char ch = str.charAt(i);
96          String replacement = replacements.get(ch);
97          if (replacement == null) result.append(ch);
98          else result.append(replacement);
99       }
100      result.append("\"");
101      return result;
102   }
103}
104
105public class HttpClientTest
106{
107   public static void main(String[] args)
108         throws IOException, URISyntaxException, InterruptedException
109   {
110      System.setProperty("jdk.httpclient.HttpClient.log", "headers,requests,content");
111      String propsFilename = args.length > 0 ? args[0] : "post.properties";
112      Path propsPath = Path.of(propsFilename);
113      var props = new Properties();
114      try (Reader in = Files.newBufferedReader(propsPath))
115      {
116         props.load(in);
117      }
118      String urlString = "" + props.remove("url");
119      String contentType = "" + props.remove("Content-Type");
120      BodyPublisher publisher = null;
121      if (contentType.equals("application/x-www-form-urlencoded"))
122         publisher = MoreBodyPublishers.ofFormData(props);
123      else if (contentType.equals("multipart/form-data"))
124      {
125         // Split each value along commas, replace strings starting with
126         // file:// with Path objects
127         var data = new HashMap<String, List<?>>();
128         for (Map.Entry<Object, Object> entry : props.entrySet())
129         {
130            data.put(entry.getKey().toString(),
131                  Stream.of(entry.getValue().toString().split("\\s*,\\s*"))
132                     .map(s -> s.startsWith("file://")
133                        ? propsPath.getParent().resolve(Path.of(s.substring(7)))
134                        : s)
135                     .toList());
136         }
137         String boundary = UUID.randomUUID().toString().replace("-", "");
```

```
138            contentType += "; boundary=" + boundary;
139            publisher = MoreBodyPublishers.ofMimeMultipartData(data, boundary);
140         }
141         else
142         {
143            contentType = "application/json";
144            publisher = MoreBodyPublishers.ofSimpleJSON(props);
145         }
146
147         String result = doPost(urlString, contentType, publisher);
148         System.out.println(result);
149      }
150
151      public static String doPost(String url, String contentType, BodyPublisher publisher)
152            throws IOException, URISyntaxException, InterruptedException
153      {
154         try (HttpClient client = HttpClient.newBuilder()
155            .followRedirects(HttpClient.Redirect.ALWAYS).build())
156         {
157
158            HttpRequest request  = HttpRequest.newBuilder()
159               .uri(new URI(url))
160               .header("Content-Type", contentType)
161               .POST(publisher)
162               .build();
163
164            HttpResponse<String> response
165               = client.send(request, HttpResponse.BodyHandlers.ofString());
166            return response.body();
167         }
168      }
169   }
```

java.net.http.HttpClient 11

- `static HttpClient newHttpClient()`
  yields an `HttpClient` with a default configuration.
- `static HttpClient.Builder newBuilder()`
  yields a builder for building an `HttpClient`.
- `<T> HttpResponse<T> send(HttpRequest request, HttpResponse.BodyHandler<T> responseBodyHandler)`
- `<T> CompletableFuture<HttpResponse<T>> sendAsync(HttpRequest request, HttpResponse.BodyHandler<T> responseBodyHandler)`
  make a synchronous or asynchronous request and process the response body with the given handler.