Simple Web Server

The easiest way to start the provided webserver is the jwebserver command. It starts the server on localhost:8000 and provides a file browser for the current directory.

Code:

```
import com.sun.net.httpserver.*;
import java.net.*;
import java.io.*;
import java.nio.file.*;
import java.util.function.Predicate;

/**
 * To run from the terminal we can use the utility command-line tool:
 * `jwebserver -b 0.0.0.0 -p 8080`
 *
 * The API only supports GET and HEAD requests, but we can added manual
support with `HttpHandlers.handleOrElse`.
 *
 * Doc:
https://download.java.net/java/early_access/jdk18/docs/api/jdk.httpserver/com/sun/net/httpserver/SimpleFileServer.html
 */
public class SimpleWebServer {
 private static final String SERVER_DIR = System.getProperty("user.dir");

 public static void main(String[] args) {
 var server = SimpleFileServer.createFileServer(
 new InetSocketAddress(8080),
 Path.of(SERVER_DIR),
 SimpleFileServer.OutputLevel.VERBOSE
 );

 var defaultHandler = HttpHandlers.of(404, new Headers(), "Not found");

 Predicate<Request> IS_POST = req -> "POST".equals(req.getRequestMethod());

 var postHandler = HttpHandlers.handleOrElse(
 IS_POST,
 (exchange) -> {
 System.out.println("Handling POST request");

 System.out.println("Headers:");
 exchange.getRequestHeaders().forEach((k, v) -> System.out.printf("\t%s: %s\n",
k, v));

 InputStream requestBody = exchange.getRequestBody();
 try (var reader = new BufferedReader(new InputStreamReader(requestBody))) {
 System.out.println("\nBody:");
 reader.lines().forEach(line -> System.out.printf("\t%s", line));
 } catch (RuntimeException | IOException ex) {
 System.err.println("Error reading request body: " + ex.getMessage());
 }

 var responseBody = "POST handled successfully".getBytes();
 exchange.sendResponseHeaders(200, responseBody.length);
 exchange.getResponseBody().write(responseBody);
 },
 defaultHandler
 );
 server.createContext("/post", postHandler);

 server.start();
 System.out.println("Server started at " + server.getAddress());
 }
}
```

Output:

```
Server started at /[0:0:0:0:0:0:0:0]:8080
```

Switch Pattern Matching Second Preview

JDK Enhancement Proposal 420 introduced two changes in Java 18 – one in dominance checking and one related to exhaustiveness analysis in combination with sealed types.

Code:

```
public class SwitchWithPatternMatchingSecondPreview {
 public static void main(String[] args) {
 System.out.println(stringify(42));
 System.out.println(stringify(-42));
 System.out.println(stringify("Some text"));
 System.out.println(stringify(""));
 System.out.println(stringify(null));
 }

 static String stringify(Object value) {
 return switch (value) {
 // the constant must be before the guarded pattern (otherwise it will never hit)
 case Integer i && i == 42 -> "42 is the answer";
 case Integer i && i > 0 -> "positive number";
 case Integer i && i < 0 -> "negative number";
 // this must be after because it will match all integers
 case Integer i -> "should be 0";

 case String s && s.isEmpty() -> "empty string";
 case String s && s.length() > 50 -> "long string";
 // this must be after because it will match all strings
 case String s -> "non-empty string";
 // same here
 case CharSequence cs -> "any other CharSequence";

 case null -> "null =s";
 default -> "unhandled type";
 };
 }
}
```

Output:

```
42 is the answer
negative number
non-empty string
empty string
null =s
```