Simple Web Server

Simple Web Server is a minimal HTTP static file server that was added in JEP 408 to the jdk.httpserver module. It serves a single directory hierarchy, and it serves only static files over HTTP/1.1; dynamic content and other HTTP versions are not supported

Code:

```
import com.sun.net.httpserver.*;
import java.net.*;
import java.io.*;
import java.util.concurrent.Executors;

/**
 * Run: `java --enable-preview --source 19 WebServerThreads.java`
 */
public class WebServerThreads {
 public static void main(String[] args) throws IOException {
 var server = HttpServer.create(new InetSocketAddress(8000), 0);
 server.createContext("/", HttpHandlers.of(200, new Headers(), "Handled!"));

 // server.setExecutor(Executors.newCachedThreadPool());

server.setExecutor(Executors.newThreadPerTaskExecutor(Thread.ofVirtual().facto
ry()));

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

Output:

```
Server started at /[0:0:0:0:0:0:0:0]:8000
```

Switch With Pattern Matching Third Preview

This preview changed ‘&&’ to “when” in guarded clause.

Code:

```
import java.time.LocalDate;
import java.time.YearMonth;
import java.time.ZoneOffset;
import java.time.temporal.ChronoUnit;
import java.util.stream.Stream;

/**
 * This preview changed `&&` to `when` in guarded clause.
 *
 * Run: `java --enable-preview --source 19
SwitchWithPatternMatchingThirdPreview.java`
 */
public class SwitchWithPatternMatchingThirdPreview {
 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 when i == 42 -> "42 is the answer";
 case Integer i when i > 0 -> "positive number";
 case Integer i when i < 0 -> "negative number";
 // this must be after because it will match all integers
 case Integer i -> "should be 0";

 case String s when s.isEmpty() -> "empty string";
 case String s when 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
```

Record Pattern

Record pattern to test whether a value is an instance of a record class type (see Record Classes) and, if it is, to recursively perform pattern matching on its component values.

Code:

```
/**
 * Run: `java --enable-preview --source 19 RecordPatternsTest.java`
 */
public class RecordPatternsTest {
 public static void main(String[] args) {
 simpleExampleOfDeconstruct();

 var p1 = new Point(10, 10);
 var p2 = new ColoredPoint(new Point(10, 10), Color.GREEN);

 System.out.println("Has collision: " + checkCollisionIfs(p1, p2));

 System.out.println("Has collision: " + checkCollisionSwitches(p1, p2));
 }
```

```
static void simpleExampleOfDeconstruct() {
record ComputedPoint (int x, int y) {
// the accessor method is used is the deconstruction
public int y() {
return this.y + 10;
}
}

Object point = new ComputedPoint(42, 42);

// `int x` will receive the result of implitcy accessor `Point.x()`
// `int y` will receive the result of explicity accesor `Point.y()`
if (point instanceof ComputedPoint(int x, var y)) {
System.out.printf("Deconstruction: x=%d, y=%d%n%n", x, y);
} else {
System.out.println("Not a instance of ComputedPoint");
}
}

static boolean checkCollisionIfs(Object p1, Object p2) {
if (p1 instanceof Point(int x1, int y1) && p2 instanceof Point(int x2, int y2)) {
return x1 == x2 && y1 == y2;
}
if (p1 instanceof Point(int x1, int y1) && p2 instanceof ColoredPoint(Point(int x2,
int y2), Color c)) {
return x1 == x2 && y1 == y2;
}
if (p1 instanceof ColoredPoint(Point(int x1, int y1), Color c) && p2 instanceof
Point(int x2, int y2)) {
return x1 == x2 && y1 == y2;
}
if (p1 instanceof ColoredPoint(Point(int x1, int y1), Color c1)
&& p2 instanceof ColoredPoint(Point(int x2, int y2), Color c2)) {
return x1 == x2 && y1 == y2;
}
throw new IllegalArgumentException("Invalid type");
}
```

```
static boolean checkCollisionSwitches(Object p1, Object p2) {
int x1, y1, x2, y2;

// Record pattern on switch should be exhaustive (careful with generics on record)
switch (p1) {
case Point(int px1, int py1) -> {
x1 = px1;
y1 = py1;
}
case ColoredPoint(Point(int px1, int py1), Color c) -> {
x1 = px1;
y1 = py1;
}
case null, default -> throw new IllegalArgumentException("Invalid type");
}

switch (p2) {
case Point(int px2, int py2) -> {
x2 = px2;
y2 = py2;
}
case ColoredPoint(Point(int px2, int py2), Color c) -> {
x2 = px2;
y2 = py2;
}
case null, default -> throw new IllegalArgumentException("Invalid type");
}
return x1 == x2 && y1 == y2;
}
}

record Point(int x, int y) {}

enum Color { RED, GREEN, BLUE }

record ColoredPoint(Point p, Color c) {}
```

Output:

```
Deconstruction: x=42, y=52

Has collision: true
Has collision: true
```

Stress Virtual Thread

Java virtual threads are a new thread construct added to Java from Java 19. *Java virtual threads* are different from the original platform threads in that virtual threads are much more lightweight in terms of how many resources (RAM) they demand from the system to run. Thus, you can have far more virtual threads running in your applications than platform threads.

Code:

```
import java.util.concurrent.Executors;
import java.util.stream.IntStream;

/**
 * Run: `java --enable-preview --source 19 <FileName.java>`
 */
public class StressVirtualThread {
 public static void main(String[] args) {
 var executor =
Executors.newThreadPerTaskExecutor(Thread.ofVirtual().factory());

 IntStream.range(0, 10_000).forEach(i -> {
 executor.submit(() -> {
 System.out.printf("VirtualThread %d - %s%n", i,
Thread.currentThread().getName());
 try {
 Thread.sleep(500);
 } catch (InterruptedException e) {
 System.out.printf("VirtualThread %d - %s interrupted%n", i,
Thread.currentThread().getName());
 } finally {
 System.out.printf("VirtualThread %d - %s woke up%n", i,
Thread.currentThread().getName());
```

```
 }
 });
 });

 while (!executor.isTerminated()) {
 ;
 }
 }
}
```

Output:

```
VirtualThread 0 -

VirtualThread 6281 -

VirtualThread 6378 -

VirtualThread 6440 -

…......

VirtualThread 9994 -  woke up

VirtualThread 9997 -  woke up

VirtualThread 9901 -  woke up
```