NON - HEAP

VIDEO

STREAMING

MOVIE.JAVA

```java
record Movie(ByteBuffer movieByteBuffer, Integer size)
    {}
```

- utilizes Java records as they are ideal for simple, straightforward data carriers

STORAGE.JAVA

```java
@Slf4j
class Storage {
```

- generates a logger field named "log" for the class, used to output log messages
- need to include Logback or Log4j dependency

```java
private final Map<String, Movie> movieLibrary = new HashMap<>();
```

- would hold stores movie names and movie object

```java
public void put(String name, MultipartFile file) throws IOException {
    var length = (int) file.getSize();
    var byteBuffer = ByteBuffer.allocateDirect(length);

    try (var channel = file.getResource().readableChannel()) {
        IOUtils.readFully(channel, byteBuffer);
    }

    byteBuffer.position(0);
    movieLibrary.put(name, new Movie(byteBuffer, length));
    log.info(format("Added a new movie '%s' with size %s",
        name, formatSize(length)));
}
```

- returns size of uploaded file and assigns it to variable 'length'
- creates a 'ByteBuffer' with capacity of 'length' bytes
- opens a 'ReadableByteChannel' from file resource and reads contents of file into ByteBuffer
- resets position of ByteBuffer to the beginning
- creates a new 'Mem'
- logs addition of new mem including its name and size

- line 6 reads bytes from channel into ByteBuffer until buffer is full

```
1 public long getTotalNoHeapMemoryUsage () {
2     return MemHolder. values(). stream()
3                 . map ( Mem :: size)
4                 . map ( Long :: valueOf )
5                 . reduce ( 0L, Long :: sum );
6             }
```

- line 5 calculates sum of all sizes of the mems, '0L' is the initial value of sum

ByteBufferBackedInputStream. JAVA

- purpose of this class is to wrap a ByteBuffer with an InputStream
- class contains a private ByteBuffer which holds a sequence of integer values to be used in an I/O operation

```
@ Override
public int read() throws IOException {
        this. ensureStreamAvailable ();
        return this. backendBuffer. hasRemaining() ? this. backendBuffer. get()
                & 0xFF : -1;
        }
```

- if there are remaining bytes, reads the byte
- expression 0xFF is used to convert the signed byte to an unsigned integer, to ensure that values are correctly represented for bytes in range 128 - 255

```
1  @Override
2  public int read(@NonNull byte[] buffer, int offset, int length) throws IOException {
3      this.ensureStreamNotNull();
4      Objects.requireNonNull(buffer, "Given buffer cannot be null!");
5      if (offset >= 0 && length >= 0 && length <= buffer.length - offset) {
6          if (length == 0) {
7              return 0;
8          } else {
9              int remainingSize = Math.min(this.bufferedBuffer.remaining(), length);
10             if (remainingSize == 0) {
11                 return -1;
12             } else {
13                 this.bufferedBuffer.get(buffer, offset, remainingSize);
14                 return remainingSize;
15             }
16         }
17     } else {
18         throw new IndexOutOfBoundsException();
19     }
20 }
```

- line 2 : buffer is where data will be read into, offset specifies at starting position in buffer and length specifying maximum number of bytes to read
- line 9 : calculate actual number of bytes that can be read from input stream
- line 13 : if there are remaining bytes, reads bytes from input stream into the 'buffer' array, starting from specified offset and up to remaining size

@ Override

```
public long skip(long n) throws IOException {
    this.ensureStreamAvailable();
    if (n <= 0L) {
        return 0L;
    }
    int lengths = (int) n;
    int remainingSize = Math.min(this.bufferedBuffer.remaining(), lengths);
    this.bufferedBuffer.position(this.bufferedBuffer.position() + remainingSize);
    return lengths;
}
```

- This method is used to skip over and discard a specified number of bytes from input stream, return actual number of bytes skipped
- line 9: move position with IS input stream

Information. JAVA

- purpose of this class is to display the memory usage

```
public static String formatSize(long v) {
    if (v < 1024) return v + " B";
    int z = (63 - Long.numberOfLeadingZeros(v)) / 10;
    return format("%.1f %sB", (double)v / (1L << (z*10)),
                  "KMGTPE".charAt(z-1));
}
```

- This method takes a byte value and return a formatted string representing size with an appropriate prefix (Kilo, Mega, Giga, etc)

Controller.JAVA

```
1  @RestController
2  @CrossOrigin
3  @RequestMapping("/mars")
4  @RequiredArgsConstructor
5  @Slf4j
6  class Controller {
```

- line 1: methods in class will automatically serialize return values to JSON or XML
- line 2: enable Cross-Origin Resource Sharing (CORS) for this particular controller. CORS is a security feature implemented by web browsers to restrict JavaScript from running on a webpage from making requests to a different domain. @CrossOrigin allows you to relax this restriction by specifying which origins are allowed to access the resources from the server. Can specify origin, HTTP method, request headers and many other attributes of the annotation.
- line 3: specifies that request URL path /mars will be handled by methods in the annotated controller
- line 4: generate a constructor for the annotated class, that includes all final fields with constructor parameters

```
1  @GetMapping("{name}")
2  ResponseEntity<Resource> getMarsByName(@PathVariable String name) {
3      return storage.pull(name)
4          .map(Mars::new).map(MarsByteBuffer::slice)
5          .map(ByteBufferBackedInputStream::new)
6          .map(InputStreamResource::new)
7          .<ResponseEntity<Resource>>map(resource ->
8              ResponseEntity.ok()
9                  .contentType(MediaType.APPLICATION_OCTET_STREAM)
10                 .body(resource))
11         .orElseGet(() -> ResponseEntity.notFound().build());
12 }
```

- line 2: @PathVariable is used to bind value of the path variable name from URL to method parameter 'name'.
- line 5: maps ByteBuffer object to a shared version of the buffer or a slice of a buffer shares its content with original buffer but has its own position, limit and capacity.
- line 7: InputStreamResource is a Spring class that wraps an InputStream and provides resource related functionality. Often used to represent a file or stream resource in Spring applications
- line 10: indicates content is a binary stream