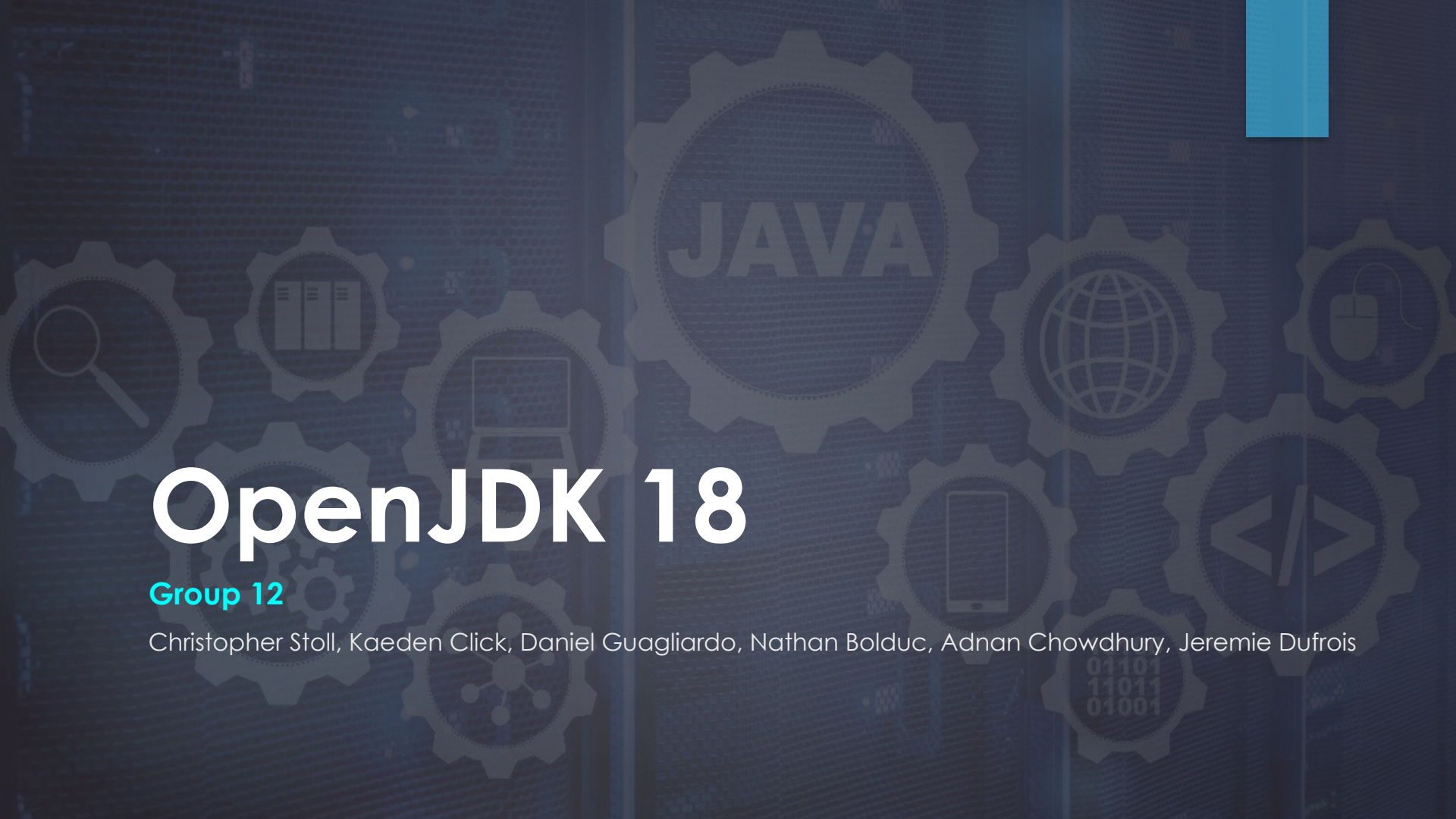


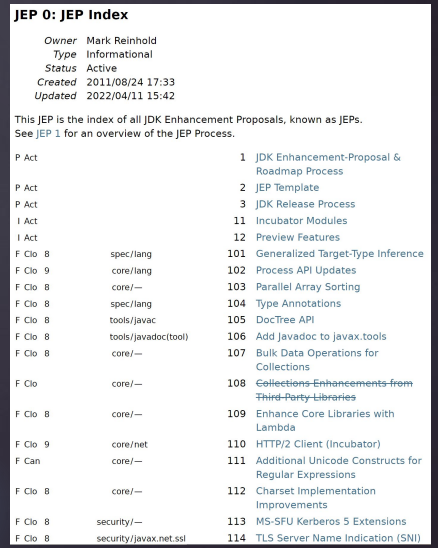

# OpenJDK 18

## Group 12

Christopher Stoll, Kaeden Click, Daniel Guagliardo, Nathan Bolduc, Adnan Chowdhury, Jeremie Dufrois

# Background & JEP

- JDK 18 released on March 22, 2022 as non-LTS release
- JDK Enhancement Proposal (JEP)
- Described by JEP 1
- Goal is to organize and define enhancement proposals for JDK release projects
- Create a roadmap based on proposals
- Central archive for changes and documentation



**JEP 0: JEP Index**

*Owner* Mark Reinhold
*Type* Informational
*Status* Active
*Created* 2011/08/24 17:33
*Updated* 2022/04/11 15:42

This JEP is the index of all JDK Enhancement Proposals, known as JEPs.
See JEP 1 for an overview of the JEP Process.

| | | | | |
|---|---|---|---|---|
| P | Act | | 1 | JDK Enhancement-Proposal & Roadmap Process |
| P | Act | | 2 | JEP Template |
| P | Act | | 3 | JDK Release Process |
| I | Act | | 11 | Incubator Modules |
| I | Act | | 12 | Preview Features |
| F | Clo | 8 | spec/lang | 101 | Generalized Target-Type Inference |
| F | Clo | 9 | core/lang | 102 | Process API Updates |
| F | Clo | 8 | core/— | 103 | Parallel Array Sorting |
| F | Clo | 8 | spec/lang | 104 | Type Annotations |
| F | Clo | 8 | tools/javac | 105 | DocTree API |
| F | Clo | 8 | tools/javadoc(tool) | 106 | Add Javadoc to javax.tools |
| F | Clo | 8 | core/— | 107 | Bulk Data Operations for Collections |
| F | Clo | | core/— | 108 | ~~Collections Enhancements from Third-Party Libraries~~ |
| F | Clo | 8 | core/— | 109 | Enhance Core Libraries with Lambda |
| F | Clo | 9 | core/net | 110 | HTTP/2 Client (Incubator) |
| F | Can | | core/— | 111 | Additional Unicode Constructs for Regular Expressions |
| F | Clo | 8 | core/— | 112 | Charset Implementation Improvements |
| F | Clo | 8 | security/— | 113 | MS-SFU Kerberos 5 Extensions |
| F | Clo | 8 | security/javax.net.ssl | 114 | TLS Server Name Indication (SNI) |

# LTS vs. Non-LTS (Long-Term Support)

- LTS (Long Term Support) releases every two years
- Stability, security, and performance updates every quarter
- LTS releases: Java 7, 8, 11, and 17
- Non-LTS releases are a cumulative set of enhancements of the most recent LTS release
- Once a new feature release is available, previous non-LTS releases are superseded

# Release Features

- 400: Uses UTF-8 by Default
- 408: Creates a Simple Web Server
- 413: Code Snippets are included in Java API Documentation
- 416: Reimplements Core Reflection with Method Handles
- 417: Has a Vector API (Third Incubator)
- 418: Introduces Internet-Address Resolution SPI
- 419: Introduces Foreign Function & Memory API (Second Incubator)
- 420: Introduces Pattern Matching for switch (Second Preview)
- 421: Deprecates Finalization for Removal

# Simple Web Server

- Provides alternative to Node for testing web servers
- Easy to use
  - Calls jwebserver to start up a server
  - Uses the SimpleFileServer class to create within an application
- Useful for testing basic UI
- Education & Testing Purposes
  - Not meant for production
  - Only handles GET and HEAD requests (not POST or others)

# Internet-Address Resolution SPI

- Define a service-provider interface (SPI) for host name and address resolution, so that java.net.InetAddress can make use of resolvers other than the platform's built-in resolver.
- API currently uses the OS's native resolver
- Combination of local host files and Domain Name System (DNS)
- Allows for more customization, and use of other resolution protocols like DNS over QUIC, TLS, and HTTPS
- API will use the built-in implementation if no resolver provider is specified.

# Java Native Interface(JNI)

- Generate header file from java file
- Any changes to native function definitions you must regenerate header file
- Slow to convert c structures or c++ classes into java objects
- SLOW

# Java Memory APIs

- ByteBuffer API
  - Safety checks
  - Slow
- sun.misc.Unsafe API
  - No safety checks
  - Fast
- Using JNI to call other memory allocators(malloc/free)
  - No safety checks
  - Slow

# Memory API

- Memory Segment
  - Size of allocation
  - Scope
- Segment Allocator
  - Scope

# Foreign Function Interface

- Symbol Lookup (CLinker)
  - CLinker.systemLinker()
- MethodHandle
  - Function
    - Use the CLinker for lookup of functions
  - FunctionDescriptor
    - Describes function arguments and return value
- Describing Types
  - Use MemoryLayout to describe structs

# Foreign Function Interface Use

- Reversed model from JNI
  - Use jextract on C/C++ header files to generate java file
- Use any C/C++ function already written
- Allows practical interoperability between many languages

# Switch Case Improvements

*The Original*

```
switch (str) {
        case "option_a":
            flag = true;
            break;
        case "option_b":
        case "option_c":
            // do something else
            break;
        default:
            // do something else different
            break;
}
```

```
switch (i) {
        case 1:
            flag = true;
            break;
        case 2:
        case 3:
            // do something else
            break;
        default:
            // do something else different
            break;
}
```

# Switch Case Improvements

*New Syntax*

```
switch (str) {
        case "option_a":
            flag = true;
            break;
        case "option_b":
        case "option_c":
            // do something else
            break;
        default:
            // do something else different
            break;
}
```

```
switch (str) {
    case "option_a" -> flag = true;
    case "option_b", "option_c" -> {
        // do something else
    }
    default -> {
        // do something else different
    }
}
```

# Switch Case Improvements

*Variables*

```
String description;
switch (day) {
    case SATURDAY:
    case SUNDAY:
        description = “Weekend”;
        break;
    default:
        description = “Weekday”;
        break;
}
```

```
String description = switch (day) {
    case SATURDAY, SUNDAY -> "Weekend";
    default -> "Weekday";
};
```

```
String description = switch (day) {
    case SATURDAY, SUNDAY -> "Weekend";
    default -> {
        yield "Weekday";
    }
};
```

# Switch Case Improvements

*Pattern Matching*

```
double result;
if (o instanceof Integer) {
    result = ((Integer) o).doubleValue();
} else if (o instanceof Double) {
    result = (Double) o;
} else if (o instanceof String) {
    result = Double.parseDouble((String) o);
} else {
    result = 0d;
}
```

```
double result = switch (o) {
    case Integer i -> i.doubleValue();
    case Double d -> d;
    case String s -> Double.parseDouble(s);
    default -> 0d;
};
```

# Questions

- How do you see the new switch statement changes impacting their usage? How about their readability?
- With greatly reduced overhead what new applications will the new foreign function interface and new memory interface allow Java developers to create?

# References


- https://openjdk.java.net/projects/jdk/18/
- https://www.oracle.com/java/technologies/java-se-support-roadmap.html
- https://res.infoq.com/news/2022/03/java-18-so-far/en/headerimage/java-istock-image-01-1646072147555.jpg
- https://www.baeldung.com/java-switch-pattern-matching
- https://blogs.oracle.com/javamagazine/post/java-long-term-support-lts
- http://openjdk.java.net/jeps/1