Java

Les 6

jeroen.devos01@ap.be

AP HOGESCHOOL ANTWERPEN

AP.BE

Lesdoelen

- Annotation
- File
- Try with resources
- Logging

Wat is …

Met Java Annotation voorzie je extra informatie over een onderdeel van je programma : vorm van metadata.

Java Annotation is géén code commentaar (JavaDoc) aangezien de Java Compiler de annotations begrijpt én er iets mee kan doen.

Java Annotation begint steeds met een @ (bv @Override) en wordt aan een class, methode of variabele gekoppeld.

Studie wijst uit dat code die gebruik maakt van annotations minder fouten bevat !

Syntax

@Override

- Geen parameters
- “Marker annotation”

@Table(name = “mijntabelnaam”)

@Column(name = “mijnkolomnaam”, length = 32)

- Één of meer parameters volgens key – value patroon

Annotation @Override

```
@Override
public String toString() {
   return …
}
```

- Plaatsing bij een methode.
- Geeft aan dat deze methode een methode uit de super overschrijft.
- Zal fout geven als method signature wijzigt.

Annotation @Deprecated

```
@Deprecated(since = “episode 3”)
public class AnakinSkywalker extends StarWarsCharacter {
}
```

- Plaatsing bij een class, methode of variabele.
- Geeft aan dat deze class, methode of variabele best niet meer gebruikt wordt.
- Je krijgt geen fout als je deze toch gebruikt !
- IDE zal de class, methode of variabele doorstrepen.

Custom Annotation

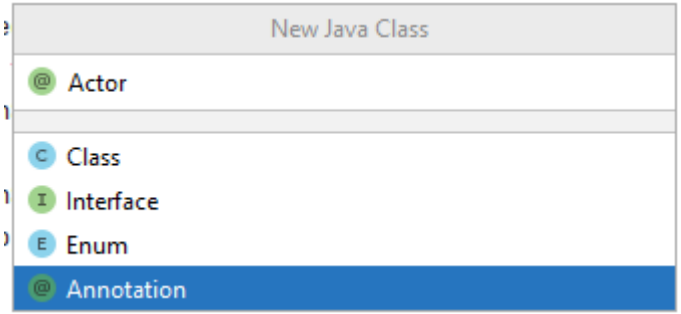
```
@Actor(name = “Harrison Ford”)
public class HanSolo extends StarWarsCharacter {
}
```

- Metadata toevoegen aan class.
- Niet nodig voor goede werking van het object.

Custom Annotation maken

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Actor {
   String name();
}
```

- = interface class met @ voor het keyword interface
- Volg zelfde regels als interface om methodes toe te voegen
- @Retention : geeft aan wanneer de annotation beschikbaar moet zijn : RUNTIME = tijdens uitvoeren programma



Custom Annotation waarde lezen

```
HanSolo hanSolo = new HanSolo();
Actor annotation = hanSolo.getClass().getAnnotation(Actor.class);
System.out.println(annotation.name());
```

- .getClass() : het class object opvragen
- .getAnnotation(…) = de annotation van dat type opvragen

File IO with Java 8 Streams

nextptr

java.io.File

File Object

- Beschrijft bestand
- Constructor parameter : path
  - Path = relatief tov project root

- Belangrijke operaties:
  - .exists() returns true/false
  - .createNewFile() returns true/false
  - .delete() returns true/false
  - .isFile() returns true/false

```
File file = new File("demo.txt");

File file = new File("src/main/resources/demo.txt");

file.exists();
file.createNewFile();
file.delete();
file.isFile();
```

java.io.File - continued

File Object

- Beschrijft **folder**
- Constructor parameter : path
  - Path = relatief tov project root

- Belangrijke operaties:
  - .exists() returns true/false
  - .mkDir() returns true/false
  - .delete() returns true/false
  - .isFolder() returns true/false
  - .listFiles() returns File[]

```
File folder = new File("demo");

File folder = new File("src/main/resources/demo");

folder.exists();
folder.mkDir();
folder.delete();
folder.isFolder();
folder.listFiles();
```

Bestanden lezen en schrijven zonder stream

- FileReader en BufferedReader
- Scanner
- FileOutputStream
- …

FileReader / BufferedReader

```
File file = new File("src/main/resources/starwars/movies.txt");
BufferedReader reader = null;
try {
    reader = new BufferedReader(new FileReader(file));
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if(reader != null) {
        reader.close();
    }
}
```

Scanner

```
File file = new File("src/main/resources/starwars/movies.txt");
Scanner scanner = null;
try {
   scanner = new Scanner(file);
   while (scanner.hasNext()) {
      System.out.println(scanner.nextLine());
   }
} catch (FileNotFoundException e) {
   e.printStackTrace();
} finally {
   if(scanner != null) {
      scanner.close();
   }
}
```

FileOutputStream

```
FileOutputStream fos = null;
try {
   fos = new FileOutputStream(“output.txt”);
   String text = “Hello world”;
   byte[] arr = tekst.getBytes();
   fos.write(arr);
} catch (IOException e) {
   e.printStackTrace();
} finally {
   if(fos != null) {
      fos.close();
   }
}
```


STREAM?
STREAM?
memegenerator.net

Tekst file lezen met Files & Streams

```
Stream<String> lines = Files.lines(file.toPath());

lines.forEach(line -> System.out.println(line));
```

- Files.lines maakt een Stream van String objecten aan
- Elke ‘line’ is een regel in je tekstbestand

- Opgelet werkt enkel met tekst bestanden !

Use the power of Streams

```
File file = new File("src/main/resources/starwars/movies.txt");

Stream<String> lines = Files.lines(file.toPath());

List<Movies> movies = lines
   .map(line -> createMovie(line))
   .collect(Collectors.toList());

private Movie createMovie(String line) {
   //parse line using String operations
   //bv line.split(";")
   return new Movie(… insert parameters from parsed line … );
}
```

Tekst file schrijven met Files & Streams

```
List<String> lines = new ArrayList<>();

Files.write(file.toPath(), lines);
```

- Files.write heeft als parameter een lijst van String object
- Elke ‘line’ van de List is een regel in je tekstbestand

- Opgelet werkt enkel met tekst bestanden !

Use the power of Streams

```
File file = new File(“src/main/resources/starwars/movies.txt”);

List<String> lines = movies
   .map(movie -> createLine(movie))
   .collect(Collectors.toList());

Files.write(file.toPath(), lines)

private String createLine(Movie movie) {
   return movie.getEpisode() + “;” + movie.getTitle();
}
```

Wat met “niet tekst” bestanden?

- Image
- PDF
- …

Als je hier meer over wilt weten, stuur me een bericht…


try-catch or try-finally
Joshua Bloch
Effective Java
Third Edition
There is no try.

Try-with-resources

- Eerst geïntroduceerd in Java 7 , verbeterd in Java 9
- Doel : verminder Java boilerplate code
- Wat doet het: -> sluit automatisch een resource zonder dat je een finally block moet schrijven

Voorbeeld zonder try-with-resources

```
File file = new File("src/main/resources/starwars/movies.txt");
Scanner scanner = null;
try {
    scanner = new Scanner(file);
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} finally {
    if(scanner != null) {
        scanner.close();
    }
}
```

Voorbeeld mét try-with-resources

```
File file = new File("src/main/resources/starwars/movies.txt");
try (Scanner scanner = new Scanner(file)) {
   while (scanner.hasNext()) {
      System.out.println(scanner.nextLine());
   }
} catch (FileNotFoundException e) {
   e.printStackTrace();
}
```

- Resource wordt aangemaakt samen met de try
- De .close() method van resource wordt automatisch uitgevoerd, inclusief null check.

AutoCloseable interface

- Voorziet de .close() method

- Een object dat AutoCloseable implementeert is typisch een resource die een IO stream opent:
  - File
  - Socket
  - Reader
  - …

- Een resource met AutoCloseable kan door try-with-resources automatisch afgesloten worden

Zelf Autocloseable gebruiken

```
public class StarWarsMovieReader implements AutoCloseable {

   private int index;

   @Override
   public void close() {
      this.index = 0;
   }
}
```

- Doe dit enkel als je zelf een IO resource aanmaakt of een class die IO simuleert !

Try with multiple resources

```
StarWarsMovieReader reader1 = new StarWarsMovieReader();
StarWarsMovieReader reader2 = new StarWarsMovieReader();

try (reader1; reader2) {
   …
}
```

- Met ‘;’ kan je meerdere AutoCloseable resources tegelijk toevoegen
- Opgelet! De .close() method wordt van achter naar voor uitgevoerd!
  1. reader2.close();
  2. reader1.close();

LOG4J

Log4j

- Java Logging Framework

- Open Source
- Apache

- Heel populair!

Log4j

- Makkelijk in gebruik
- Log level aanpasbaar ‘at run time’
- Veel mogelijke plugins,
  - om een output in te stellen: File, console, socket, system log, …
  - om logs te formatteren
  - …
- Enorm snel : 18 miljoen logberichten per seconde

Log4j – waarom logging?

Wat is het nut van logging toevoegen?

→Debugging !

- Goede log geeft inzicht in het gedrag van je toepassing.
- Handig bij:
  - Fouten
  - Vreemde resultaten
  - Vertraging

Log4j gebruiken

```
<dependency>
   <groupId>org.apache.logging.log4j</groupId>
   <artifactId>log4j-api</artifactId>
   <version>2.20.0</version>
</dependency>
<dependency>
   <groupId>org.apache.logging.log4j</groupId>
   <artifactId>log4j-core</artifactId>
   <version>2.20.0</version>
</dependency>
```

Log4j gebruiken

```
private Logger logger = LogManager.getLogger();

public String createNewSquad(SquadDTO squad) {
   logger.log(Level.DEBUG, “Entering method createNewSquad”);
   //your code
}
```

- Maak een Logger class aan via LogManager als class variabele.
- Gebruik de log methode om een log lijn weg te schrijven
  - Bepaal welke logging level
  - Geef een log bericht mee

Log4j – log level

- Verschillende log levels beschikbaar:
  - FATAL
  - ERROR
  - WARN
  - INFO
  - DEBUG
  - TRACE
- Altijd zichtbaar : FATAL / ERROR / WARN / INFO
- Moet specifiek aangezet worden : DEBUG / TRACE

Log4j – configuratie

In resources folder : log4j2.xml

log4j2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <!-- Console appender configuration -->
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <!-- Root logger referring to console appender -->
        <Root level="info">
            <AppenderRef ref="console" />
        </Root>
    </Loggers>
</Configuration>
```

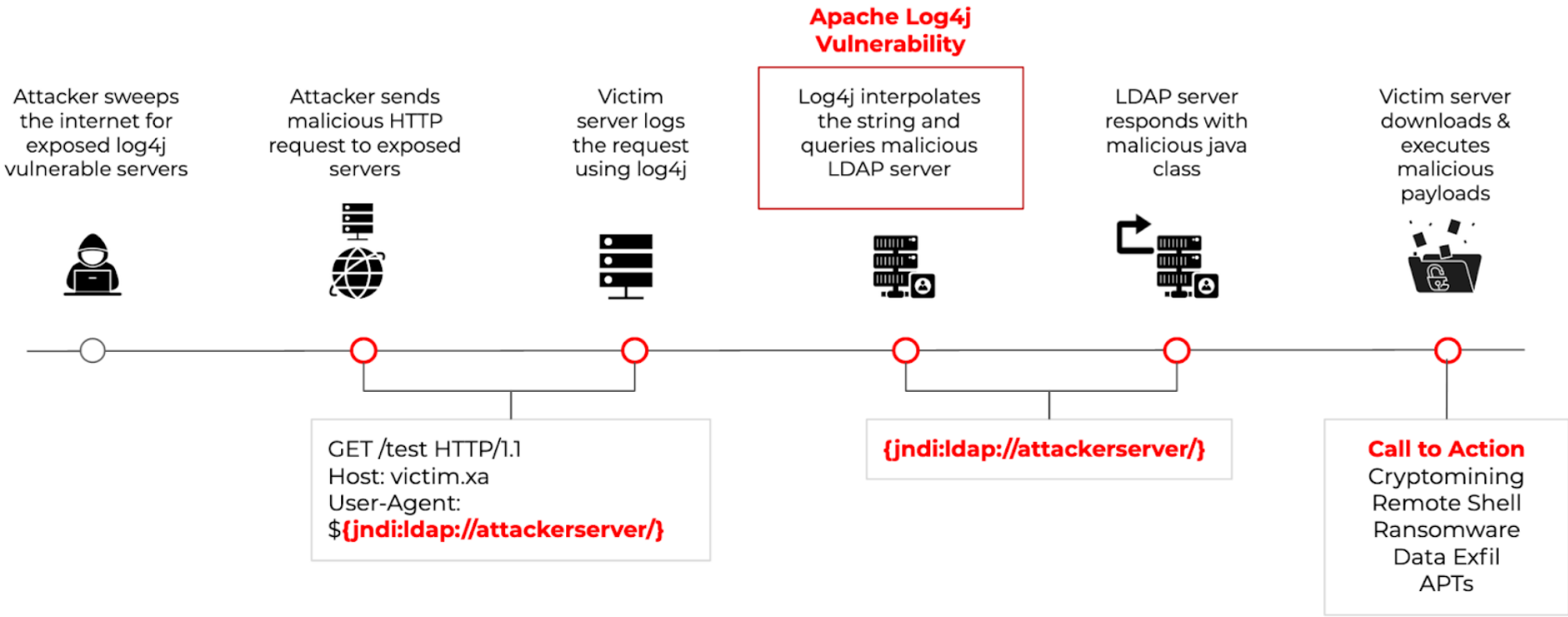
Log4j – configuratie

Instellingen:

- Soorten output (system out / file / …)
- Patroon van het logbericht
- Level koppelen aan output


LOG4J

Log4j attack lifecycle



Log4j : zero day vulnerability

- Wie is (was) geïnfecteerd:
  - AWS
  - Twitter
  - iCloud
  - Steam
  - Minecraft Java Edition
  - VMWare
  - … en miljoenen andere servers
  - … 8% van alle Java libs op Maven Central
  - … 1800+ GIT repositories
- Ben je zeker dat je niet geïnfecteerd bent? → NEEN


FOR TODAY
ANYWAY...