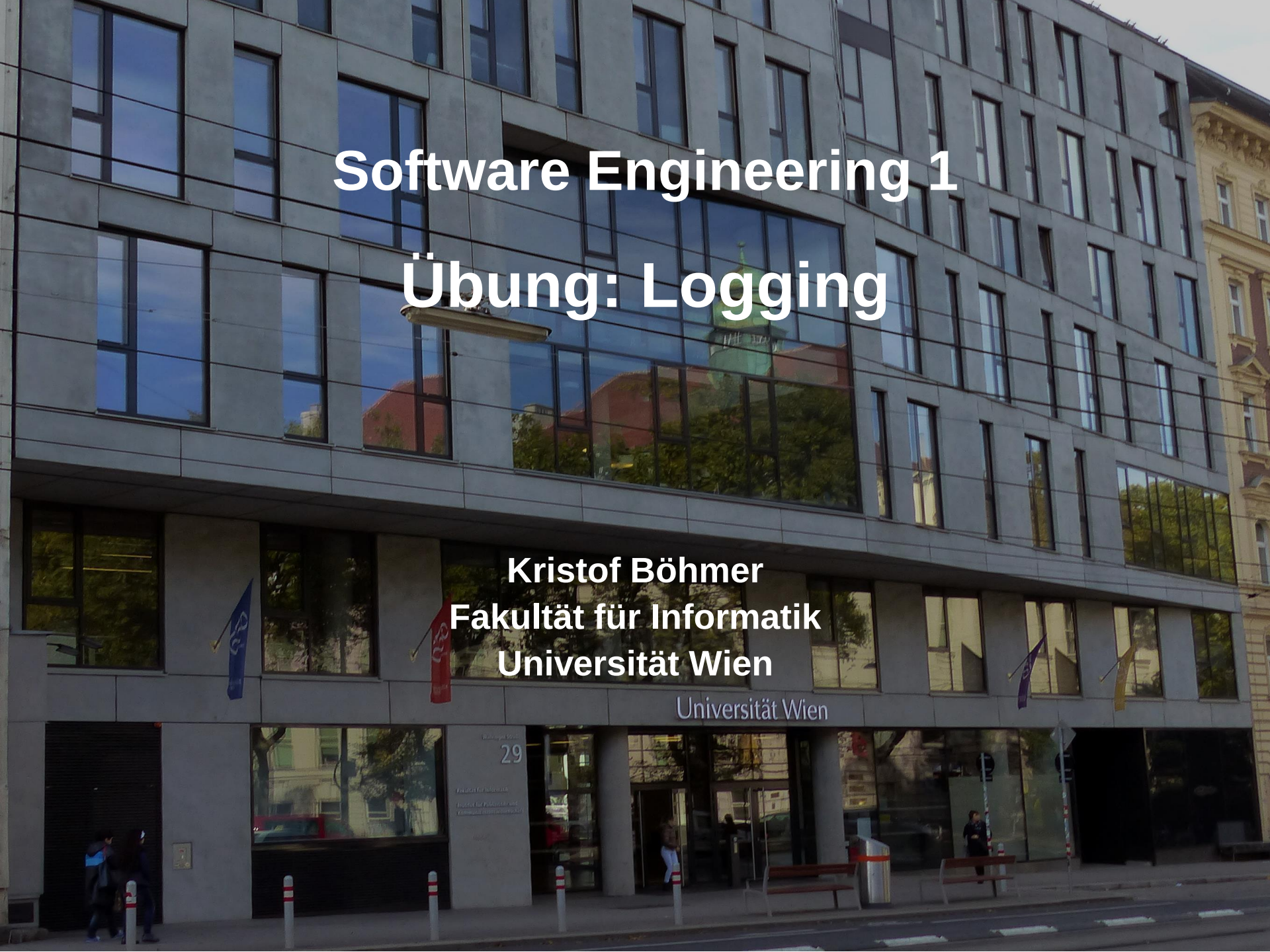
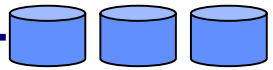


Software Engineering 1

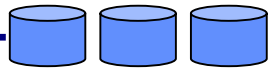
Übung: Logging

Kristof Böhmer
Fakultät für Informatik
Universität Wien

Inhalt

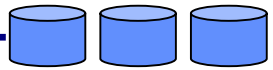


4.1 Logging: Definition



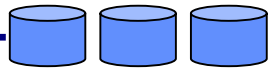
- **Was ist Logging?**
  - Mitschreiben des Verhaltens des entwickelten Programms während dessen Ausführung.
- **Logging kann für verschiedene Zwecke eingesetzt werden, wie:**
  - Fehler finden – Debugging während der Entwicklung.
    - Was sollte das Programm tun?
    - Was tut es tatsächlich?
  - Fehlerzustände während der Laufzeit analysieren und aufzeichnen.
  - Verhalten der Benutzer analysieren.
  - Dokumentation
    - Beispielsweise rechtliche Rahmenbedingungen (Datenschutz etc.) – wer hat auf eine Datei zugegriffen?
    - Absicherung bei Fehlverhalten und Problemen: Von Ihrer Software kann ein Leben abhängen mit entsprechenden (rechtlichen) Konsequenzen falls Fehler auftreten.

4.1 Logging: Ausgabemöglichkeiten



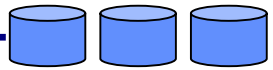
- **Es existieren verschiedene Ausgabemöglichkeiten**
  - Direkt in der Konsole
    - Vorteil: Kann vom Entwickler direkt während der Arbeit eingesehen werden.
    - Nachteil: Kann kaum für die Dokumentation von Fehlern herangezogen werden.
  - Ausgabe in eine Log-Datei
    - Kann von mehreren Entwicklern zur Fehlerbehebung herangezogen werden.
    - Kann zur Dokumentation der Fehler herangezogen werden.
    - Nachteil: Die Ausgabe erfolgt nicht direkt innerhalb der Entwicklungsumgebung.
  - Sammlung der Logs in einem zentralen Logging-System
    - Zentrale Stelle zur Speicherung, Analyse und Verwaltung
    - Gesamtüberblick über Systeme während des Betriebs möglich
    - Mit SOA leicht umsetzbar
    - Nachteil: Single Point of Failure

4.1 Logging: Debugging



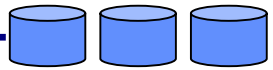
- **Debugging: Was kann geloggt werden?**
  - Aufrufe von Methoden und Programmblöcken
    - In welcher Reihenfolge werden die Methoden ausgeführt?
    - Passt diese Reihenfolge zu dem was ich mir erwarte bzw. wünsche?
  - Ausgabe von Variablenwerten
    - Was steht zu einem bestimmten Zeitpunkt in einer Variable?
    - Was sollte drin stehen?
  - Ausgabe des Systemzustandes
    - Wie viel Speicher wurde benötigt?
    - Welche Threads laufen gerade parallel?
    - Wie ist die Auslastung des Netzwerks oder der CPU?

4.1 Logging: Log Level



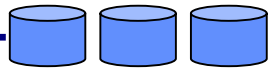
- **Log Level:** Informationen über ein Programm oder einen Programmablauf können auf verschiedenen Granularitätsebenen ausgegeben bzw. erzeugt werden.
  - **Debug:** Infos während des Debuggens (Variableninhalte etc.).
  - **Info:** Generelle Infos die nützlich sein können (start / stop des Services etc.).
  - **Warning:** Infos über Zustände die potentiell zu Problemen führen können.
  - **Error:** Jeder Fehler, der für die aktuelle Operation negative Auswirkungen hat.
  - **Fatal:** Jeder Fehler, der zum Absturz des kompletten Services führt - nur für die schlimmsten (Problem-)Fälle.

4.1 Logging: Best Practices – Was loggen?



- **Best Practices:** Was und wie viel geloggt werden soll, hängt von der aktuellen Situation bzw. Projektphase ab.
  - Entwicklung
    - Konkretes Problem: Möglichst viel, um Fehler einzugrenzen.
    - Nicht mehr benötigte Log Meldungen sollten immer entfernt werden.
    - Aspect Oriented Programming ermöglicht Programmabläufe detailliert aufzuzeichnen.
  - Testphase
    - Ergebnisse der Unit Tests ⇨ Was wurde bestanden / nicht bestanden.
    - Testen der Log-Meldungen im Betrieb.
  - Betrieb
    - Rechtliche Verpflichtungen: Zugriff auf gewisse Datensätze muss geklärt werden (siehe DSGVO).
    - Projektverwaltung: Von wann bis wann wurde woran gearbeitet?
    - Statistiken: Auf welche Module/Funktionen wurde besonders oft zugegriffen? ⇨ Diese können entsprechend skaliert bzw. erweitert werden.

4.1 Logging: Best Practices – Wie loggen?



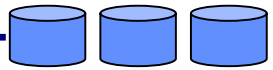
- **Best Practices:** Wie kann geloggt werden?
  - `System.out.println` ist nicht der optimale Weg um Programmzustände während der Entwicklung zu loggen, weil:
    - Unflexibel,
    - Kann „nur“ auf die Konsole ausgegeben werden und
    - Alle Log Level werden gleich behandelt – keine Unterscheidung möglich zwischen Debug, Info, Warning.
  - Stattdessen: Einsatz von Logging Frameworks!

4.1 Logging: Frameworks 1

- **Anforderungen an Logging Frameworks**
  - Logging sollte nicht aufwändig sein.
    - Leicht zu konfigurieren.
    - Geringe Auswirkung auf die Performance eines Programmes.
  - Mehrere Log Level müssen definierbar sein.
    - Debug, Info, Warning, Error, ...
  - Die Definition des jeweiligen Log Levels sollte individuell pro Software Komponente möglich sein (z.B. Info für Netzwerk- und Error für Datenbankkomponenten).
  - Zentrale Konfiguration sollte möglich sein.
  - Es gibt viele verschiedene Java Libraries für Logging wie beispielsweise:
    - java.util.logging (Standard Java Logging)
    - Logback
    - Log4J
  - Jede Library hat ihre Vorteile und Nachteile.

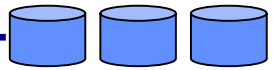
- **Logging Frameworks**
  - Bei externen Libraries können Sie zumeist nicht direkt festlegen bzw. selbst entscheiden welches Logging Framework diese verwenden.
    - Werden in verschiedenste Projekte integriert.
    - Jedes Projekt verwendet potentiell eine andere Art zu loggen.
    - Konfiguration und Einbindung wird unnötig komplex.
  - Lösung: **Simple Logging Facade for Java (SLF4J)**
    - Kann verwendet werden um verschiedenste Logging Frameworks zu integrieren.
    - Besonders geeignet für Libraries, die in anderen Applikationen eingebunden werden.
    - Externe Libraries können so nicht ihre eigenen Logging Frameworks, welche diese mitbringen, der eigentlichen von Ihnen entwickelnden Applikationen „aufdrängen“.
    - **Beispiel**: Sie können z.B. die Logausgaben des in der Übung verwendeten Spring und die Logausgaben in ihrem eigenen Code dank SLF4J gleichermaßen kontrollieren/konfigurieren (z.B. das zu loggende Loglevel).

4.1 Logging: Logback

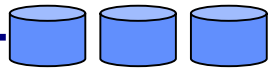


- **Logback:** Bietet eine umfangreichere und besser durchdachte Funktionalitäten als „normales“ Java Logging (`java.util.logging`).
  - Höhere Performance, selbst bei vielen Logeinträgen.
  - Zahlreiche nützliche Features:
    - Konfiguration in XML/Groovy möglich
    - Graceful Recovery from I/O Failures
    - Automatisches komprimieren/löschen alter Logdaten (Log-Rotation)
    - Automatisches versenden von Logs per E-Mail, HTTP, SMB, etc.
  - Logback ist ein Nachfolger von Log4J (Log4J ist einer der erfolgreichsten javabasierten Logger)
  - Konfiguration und Setup basiert auf SLF4J
  - Homepage: https://logback.qos.ch/

Inhalt

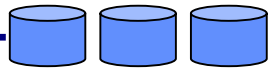


4.2 Einbinden von SLF4J mit Logback über Gradle

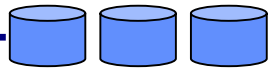


- **Verwendung von Logback und SLF4J mit Gradle**
  - **SLF4J:** Abstraktionsschicht um mehrere Logging-Frameworks mit der gleichen Konfiguration verwenden zu können.
  - Einfügen folgender Zeilen in die bereits erstellte Datei `build.gradle` unter `dependencies`
    - `compile group: 'org.slf4j', name:'slf4j-api', version: 'X.X.X'`
    - `compile group: 'ch.qos.logback', name:'logback-classic', version: 'X.X.X'`
    - `compile group: 'ch.qos.logback', name:'logback-core', version: 'X.X.X'`
  - Bedeutung der drei Zeilen:
    - Die erste Zeile lädt das Logging-Abstraktions-Framework SLF4J.
    - Die restlichen beiden Zeilen laden den eigentlichen Logger: Logback.
    - Unter `version: 'X.X.X'` muss `'X.X.X'` mit der jeweiligen Versionsnummer ersetzt werden
  - **Hinweis:** Das bereitgestellte Basic-Clientprojekt (siehe Tipps und Tricks in Moodle) umfasst bereits alle notwendigen Gradle-Konfigurationen mit Softwareversionen welche von uns erfolgreich getestet wurden. Es wurde dort auch ein Beispielprojekt eingestellt welches sich speziell auf Logging konzentriert.

Inhalt



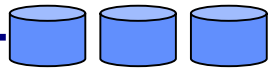
4.3 Logback: Setup 1 – Einstieg mit `logback.xml`



- **Logback: Setup**
  - Konfiguration wird über `logback.xml` gesteuert.
  - Muss über den Klassenpfad des Projektes erreichbar sein.
  - In diesem Ordner nun `logback.xml` mit folgendem Inhalt anlegen:

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

4.3 Logback: Setup 2 – Ausgabe in Konsole

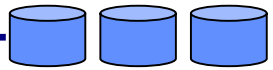


- **Logback: Setup (Ausgabe in Konsole)**
  - Konfiguration wird über `logback.xml` gesteuert.
  - Muss über den Klassenpfad des Projektes erreichbar sein.
  - In diesem Ordner nun `logback.xml` mit folgendem Inhalt anlegen:

Ausgabe in Konsole

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

4.3 Logback: Setup 3 - Formatierung

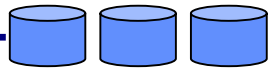


- **Logback: Setup (Format)**
  - Konfiguration wird über `logback.xml` gesteuert.
  - Muss über den Klassenpfad des Projektes erreichbar sein.
  - In diesem Ordner nun `logback.xml` mit folgendem Inhalt anlegen;

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

Format der Ausgabe

4.3 Logback: Setup 4 – Ausgabe in Datei (einfach)



- **Logback: Setup (Ausgabe in Datei)**
  - Konfiguration wird über `logback.xml` gesteuert.
  - Muss über den Klassenpfad des Projektes erreichbar sein.
  - In diesem Ordner nun `logback.xml` mit folgendem Inhalt anlegen:

```
<configuration>
    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

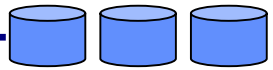
Ausgabe in Datei

4.3 Logback: Setup 5 – Ausgabe in Datei (realistisch)

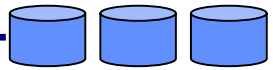
- **Logback: Komplexeres realistisches Beispiel (File Rolling)**
  - Auszug aus `logback.xml`, gegeben ist nur der `appender`.
  - File Rolling Beispiel, stückelt Logs in einzelne Dateien mit maximal 10MB pro Log-Datei, Log-Dateien älter als 10 Tage werden gelöscht.

```
<appender name="FileRolling" class="ch.qos.logback.core.rolling.RollingFileAppender">
   <file>./logs/SE1.log</file>
   <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>
   </encoder>
   <!-- only log messages with log level INFO or more significant are retained -->
   <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
      <level>INFO</level>
   </filter>
   <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
      <!-- start a new archived log file daily (rollover) -->
      <fileNamePattern>
      ./logs/archived/SE1-%d{yyyy-MM-dd}.%i.log
      </fileNamePattern>
      <!-- or when the current log file reaches 10MB or more -->
      <maxFileSize>10MB</maxFileSize>
      <!-- overall keep logs for at most 10 days, older ones are deleted -->
      <maxHistory>10</maxHistory>
   </rollingPolicy>
</appender>
```

Inhalt



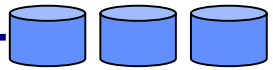
4.4 Codebeispiele: Hello World 1



- **Ein erstes Logger Hello World**

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


public class HelloWorld {
  public static void main(String [ ] args) {
    Logger logger = LoggerFactory.getLogger(HelloWorld.class);
    logger.info("Hello World");
  }
}
```

4.4 Codebeispiele: Hello World 2

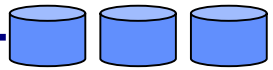


- **Ein erstes Logger Hello World**

Die notwendigen Libraries müssen importiert werden

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


public class HelloWorld {
  public static void main(String [ ] args) {
    Logger logger = LoggerFactory.getLogger(HelloWorld.class);
    logger.info("Hello World");
  }
}
```

4.4 Codebeispiele: Hello World 3

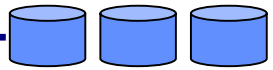


Ein erstes Logger Hello World

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


public class HelloWorld {
  public static void main(String [ ] args) {
    Logger logger = LoggerFactory.getLogger(HelloWorld.class);
    logger.info("Hello World");
  }
}
```

Neue Instanz der Klasse Logger wird erzeugt

4.4 Codebeispiele: Hello World 4

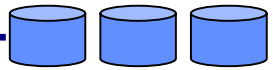


Ein erstes Logger Hello World

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


public class HelloWorld {
  public static void main(String [ ] args) {
    Logger logger = LoggerFactory.getLogger(HelloWorld.class);
    logger.info("Hello World");
  }
}
```

Logzeile wird ausgegeben

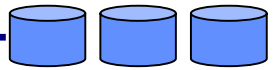
4.4 Codebeispiele: Rekursive Funktion 1



- **Ausgabe einer rekursiven Funktion**

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloWorld {
  static Logger logger = LoggerFactory.getLogger(HelloWorld.class);
  public static void main(String [ ] args) {
    logger.info("Hello World");
    int x = rekursiv(3);
    logger.info("Ergebnis: " + x);
  }
  static int rekursiv(int n) {
    logger.info("calling rekursiv with value {}", n);
    if(n>0) {
      logger.info("not there yet - we need to go deeper");
      return n + rekursiv(n-1);
    } else {
      logger.warn("final call");
      return n;
    }
  }
}
```

4.4 Codebeispiele: Rekursive Funktion 2



Ausgabe einer rekursiven Funktion

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloWorld {
  static Logger logger = LoggerFactory.getLogger(HelloWorld.class);
  public static void main(String [ ] args) {
    logger.info("Hello World");
    int x = rekursiv(3);
    logger.info("Ergebnis: " + x);
  }
  static int rekursiv(int n) {
    logger.info("calling rekursiv with value {}", n);
    if(n>0) {
      logger.info("not there yet - we need to go deeper");
      return n + rekursiv(n-1);
    } else {
      logger.warn("final call");
      return n;
    }
  }
}
```

4.4 Codebeispiele: Rekursive Funktion 3

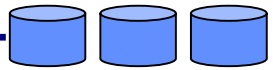
- **Ausgabe einer rekursiven Funktion**

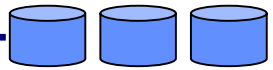
```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloWorld {
  static Logger logger = LoggerFactory.getLogger(HelloWorld.class);
  public static void main(String [ ] args) {
    logger.info("Hello World");
    int x = rekursiv(3);
    logger.info("Ergebnis: " + x);
```

Problems | Javadoc | Declaration | Console

```
<terminated> HelloWorld [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Conte
19:24:29.306 [main] INFO HelloWorld - Hello World
19:24:29.309 [main] INFO HelloWorld - calling rekursiv with value 3
19:24:29.310 [main] INFO HelloWorld - not there yet - we need to go deeper
19:24:29.310 [main] INFO HelloWorld - calling rekursiv with value 2
19:24:29.310 [main] INFO HelloWorld - not there yet - we need to go deeper
19:24:29.310 [main] INFO HelloWorld - calling rekursiv with value 1
19:24:29.310 [main] INFO HelloWorld - not there yet - we need to go deeper
19:24:29.310 [main] INFO HelloWorld - calling rekursiv with value 0
19:24:29.310 [main] WARN HelloWorld - final call
19:24:29.310 [main] INFO HelloWorld - Ergebnis: 6
```

Inhalt

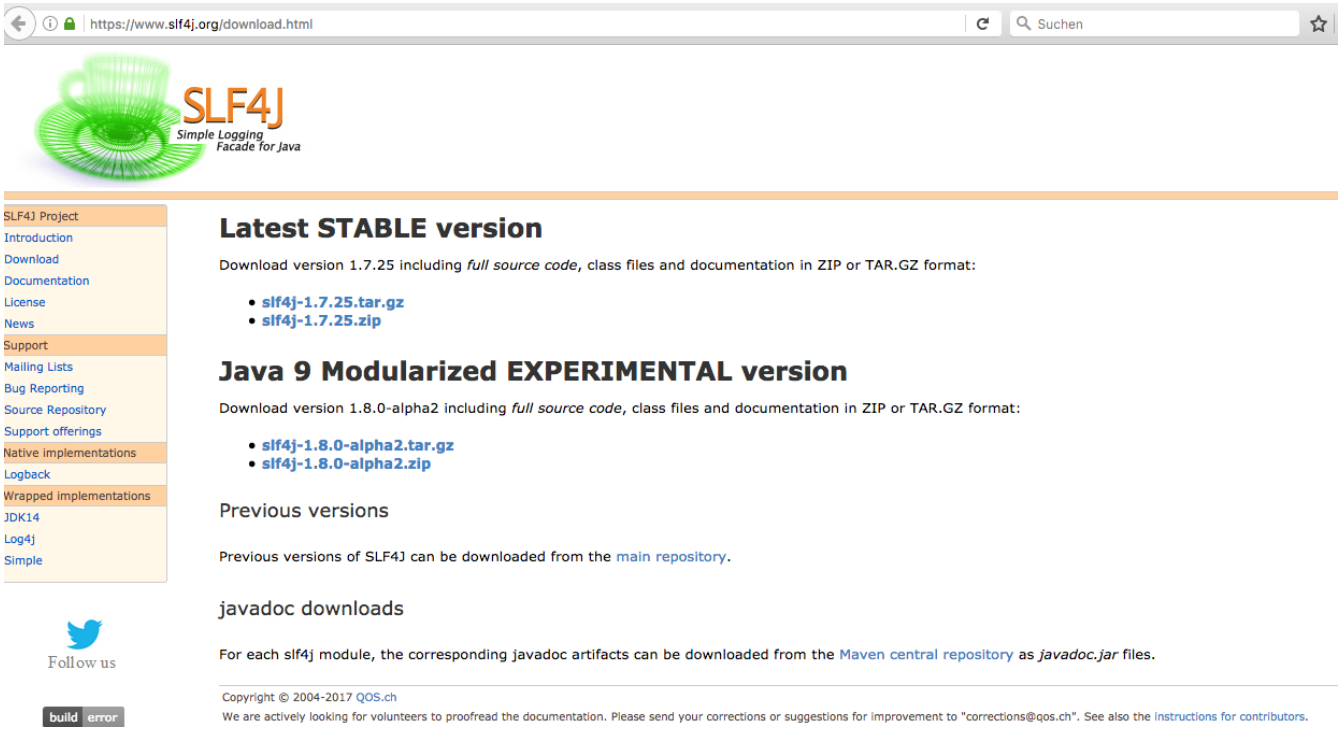




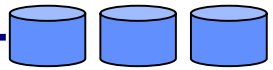
EINBINDEN VON SLF4J OHNE GRADLE

4.5 Einbinden von SLF4J 1 ohne Gradle 1

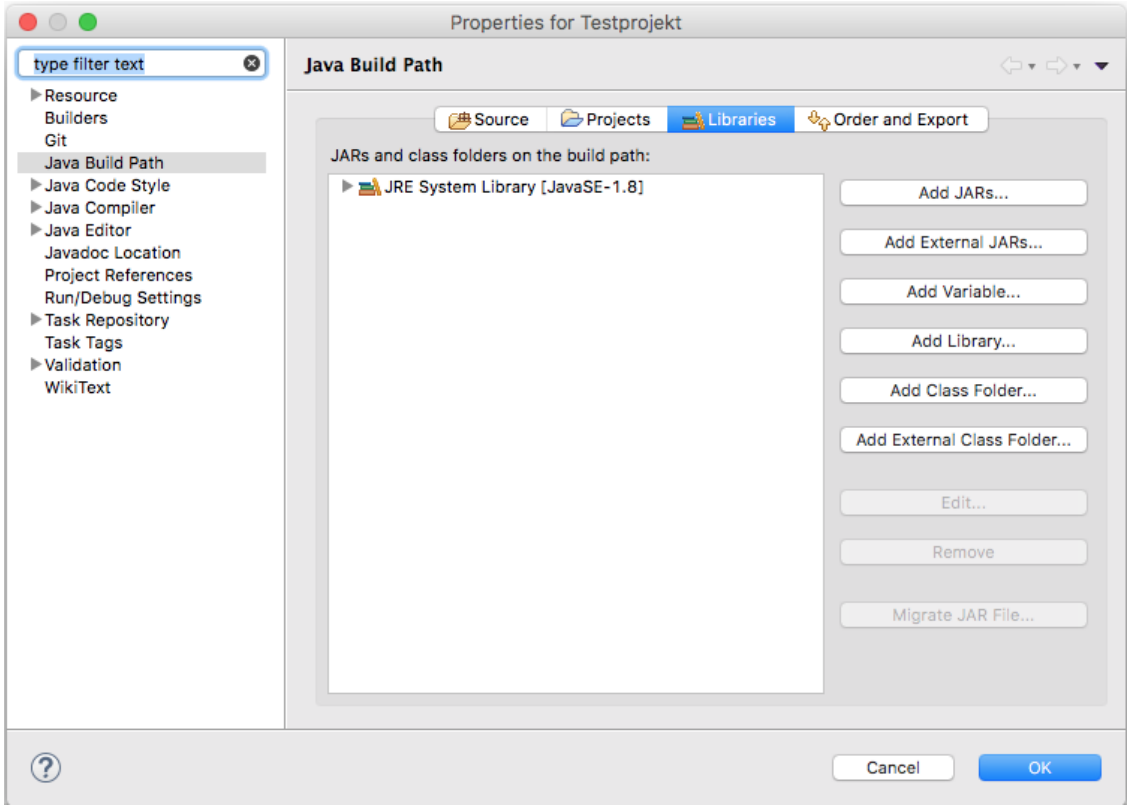
- **Setup der Simple Logging Facade for Java (SLF4J)**
  - Eine Anleitung ist hier verfügbar: https://www.slf4j.org/manual.html
  - Download der SLF4J Distribution möglich unter:
    - https://www.slf4j.org/download.html



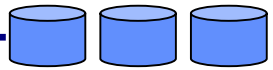
4.5 Einbinden von SLF4J 1 ohne Gradle 2



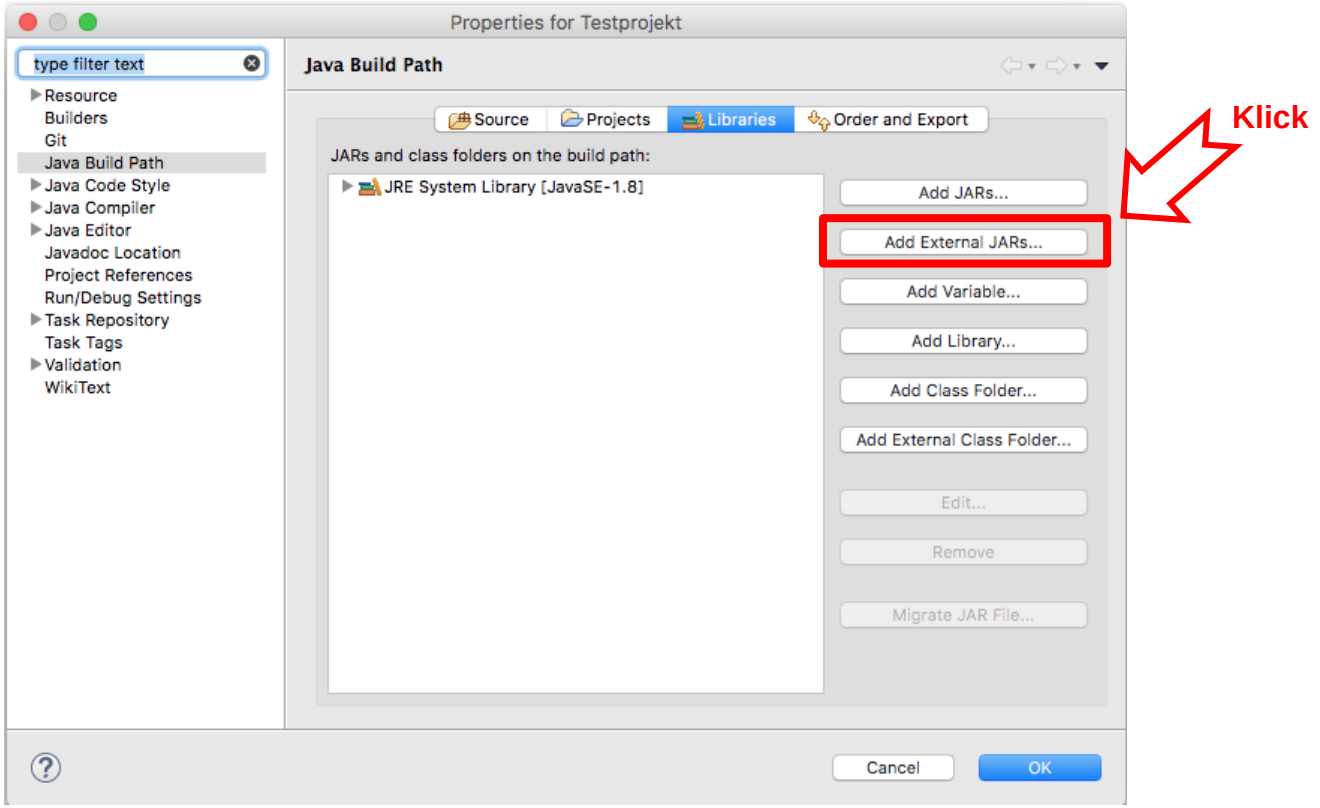
- **Setup von Simple Logging Facade for Java (SLF4J)**
  - Hinzufügen zum Klassenpfad des Projektes
    - Rechte Maustaste ➔ Properties ➔ Java Build Path



4.5 Einbinden von SLF4J 1 ohne Gradle 3

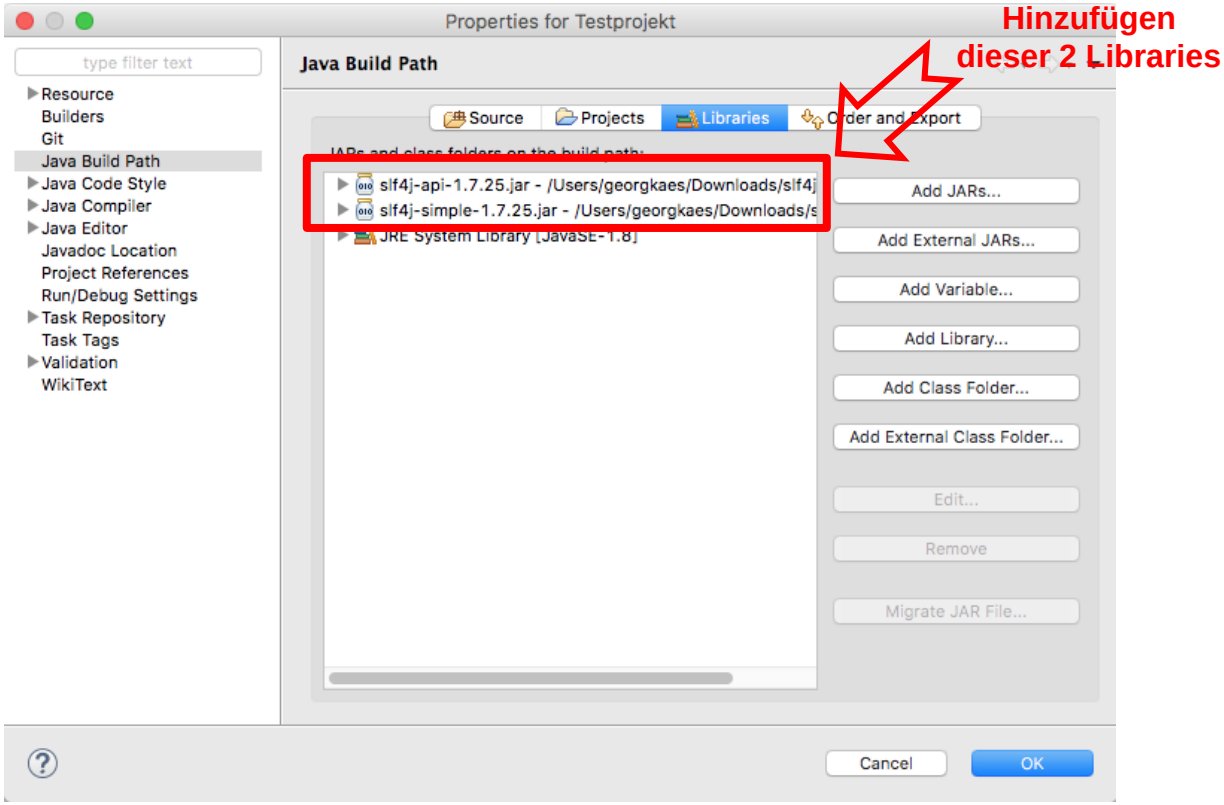


- **Setup von Simple Logging Facade for Java (SLF4J)**
  - Hinzufügen zum Klassenpfad des Projektes
    - Rechte Maustaste ➔ Properties ➔ Java Build Path

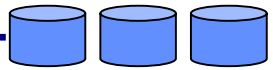


4.5 Einbinden von SLF4J 1 ohne Gradle 4

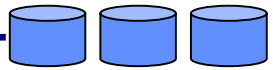
- **Setup von Simple Logging Facade for Java (SLF4J)**
  - Hinzufügen zum Klassenpfad des Projektes
    - Rechte Maustaste ➔ Properties ➔ Java Build Path



4.5 Einbinden von SLF4J 1 ohne Gradle 5

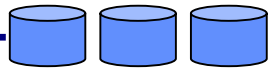


- **Setup der Simple Logging Facade for Java (SLF4J)**
  - Nun können wir SLF4J mit Logback (oder anderen Loggingframeworks) kombinieren.
  - Andere Logging Frameworks können ganz einfach eingebunden werden.
    - Dafür müssen die entsprechenden Bindings hinzugefügt werden.
    - Sind im Prinzip nur andere .jar Dateien.
    - Genaue Erklärung in SLF4J Dokumentation: https://www.slf4j.org/manual.html
- **Vorteil:** Kein Ändern des Codes notwendig wenn das Logging Framework ausgetauscht wird – SLF4J Funktionen können immer aufgerufen werden!

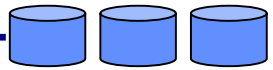


EINBINDEN VON LOGBACK OHNE GRADLE

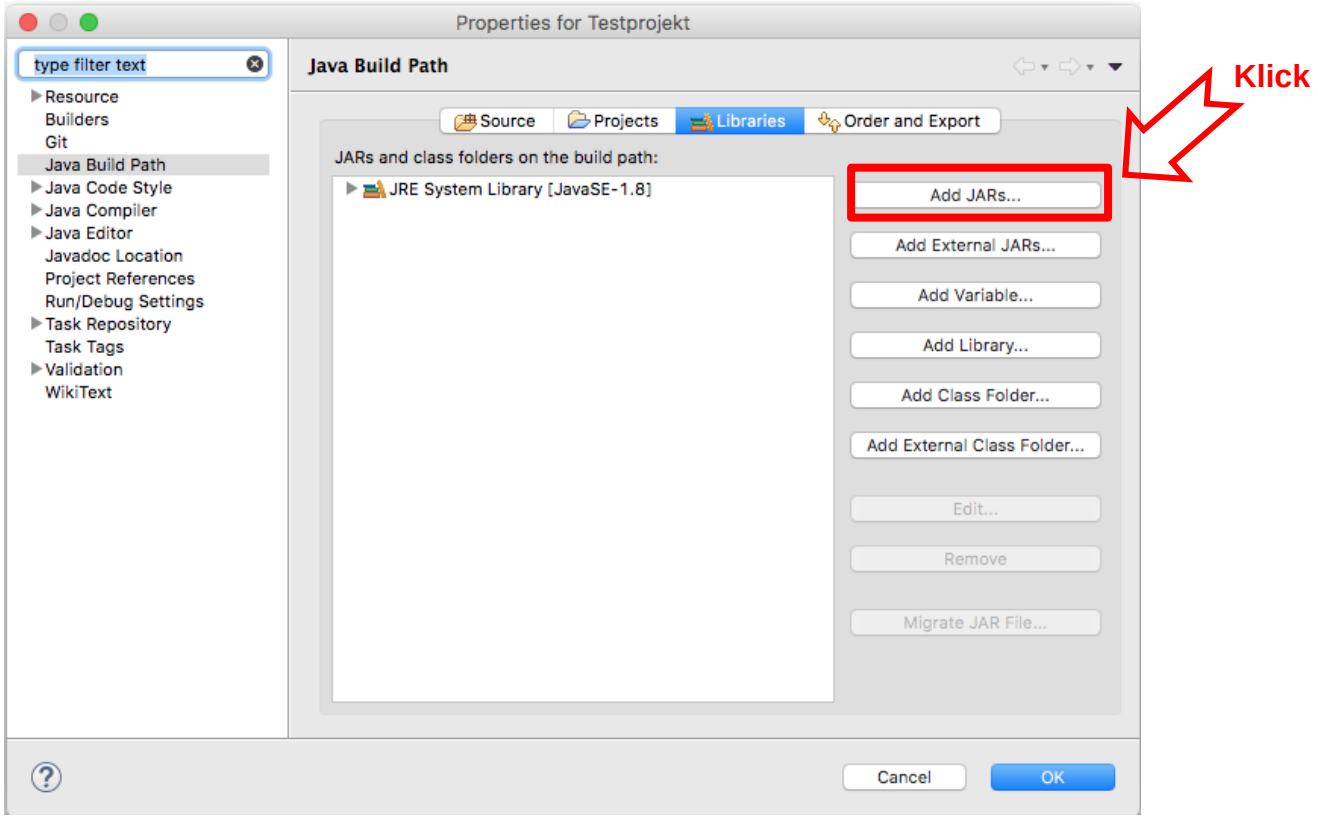
4.5 Einbinden von Logback ohne Gradle 1



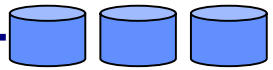
- **Logback – Setup**
  - Download der Logback Library: https://logback.qos.ch/download.html
  - Download der SLF4J Library: https://www.slf4j.org/download.html
  - Erstellen eines neuen Java Projekts in Eclipse.
  - Hinzufügen der .jar Dateien zum Klassenpfad des Projekts.
    - logback-core-1.2.3.jar
    - logback-classic-1.2.3.jar
    - slf4j-api-1.7.25.jar



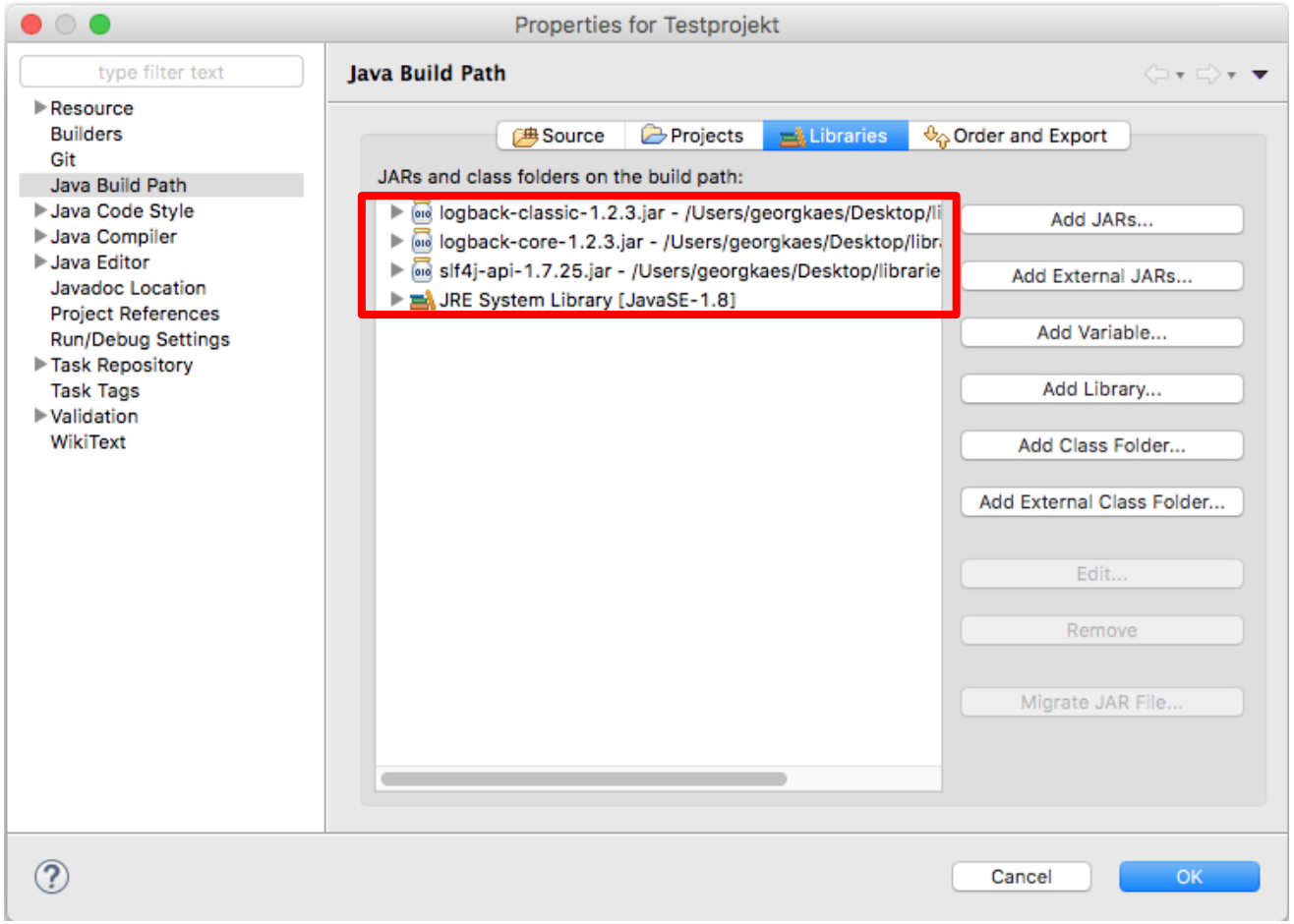
- **Logback – Setup**
  - Hinzufügen zum Klassenpfad des Projektes
    - Rechte Maustaste ➔ Properties ➔ Java Build Path

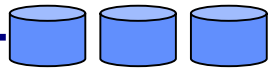


4.5 Einbinden von Logback ohne Gradle 3

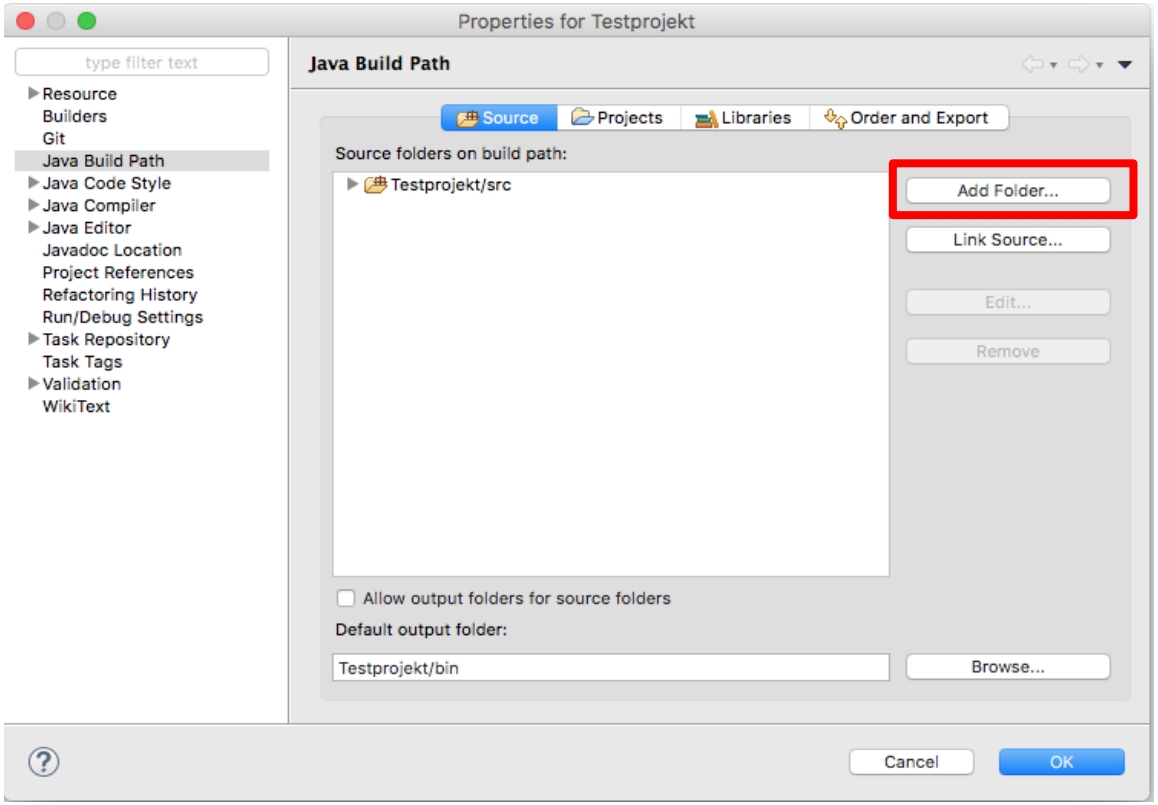


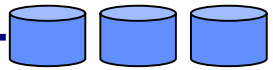
- **Logback – Setup**



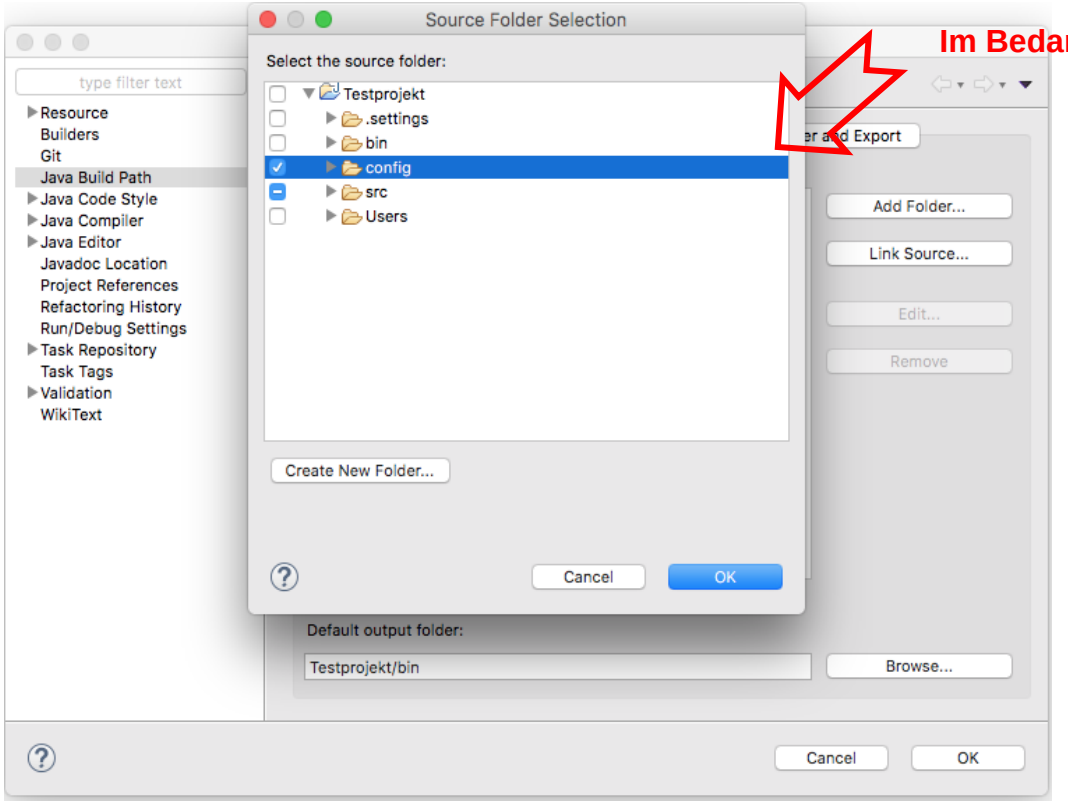


- **Logback – Setup**
  - Konfiguration wird über `logback.xml` gesteuert.
  - Muss über den Klassenpfad des Projektes erreichbar sein.



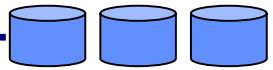


- **Logback – Setup**
  - Konfiguration wird über `logback.xml` gesteuert.
  - Muss über den Klassenpfad des Projektes erreichbar sein.



Im Bedarfsfall Ordner anlegen

4.5 Einbinden von Logback ohne Gradle 6



- **Logback – Setup**
  - Konfiguration wird über `logback.xml` gesteuert.
  - Muss über den Klassenpfad des Projektes erreichbar sein.