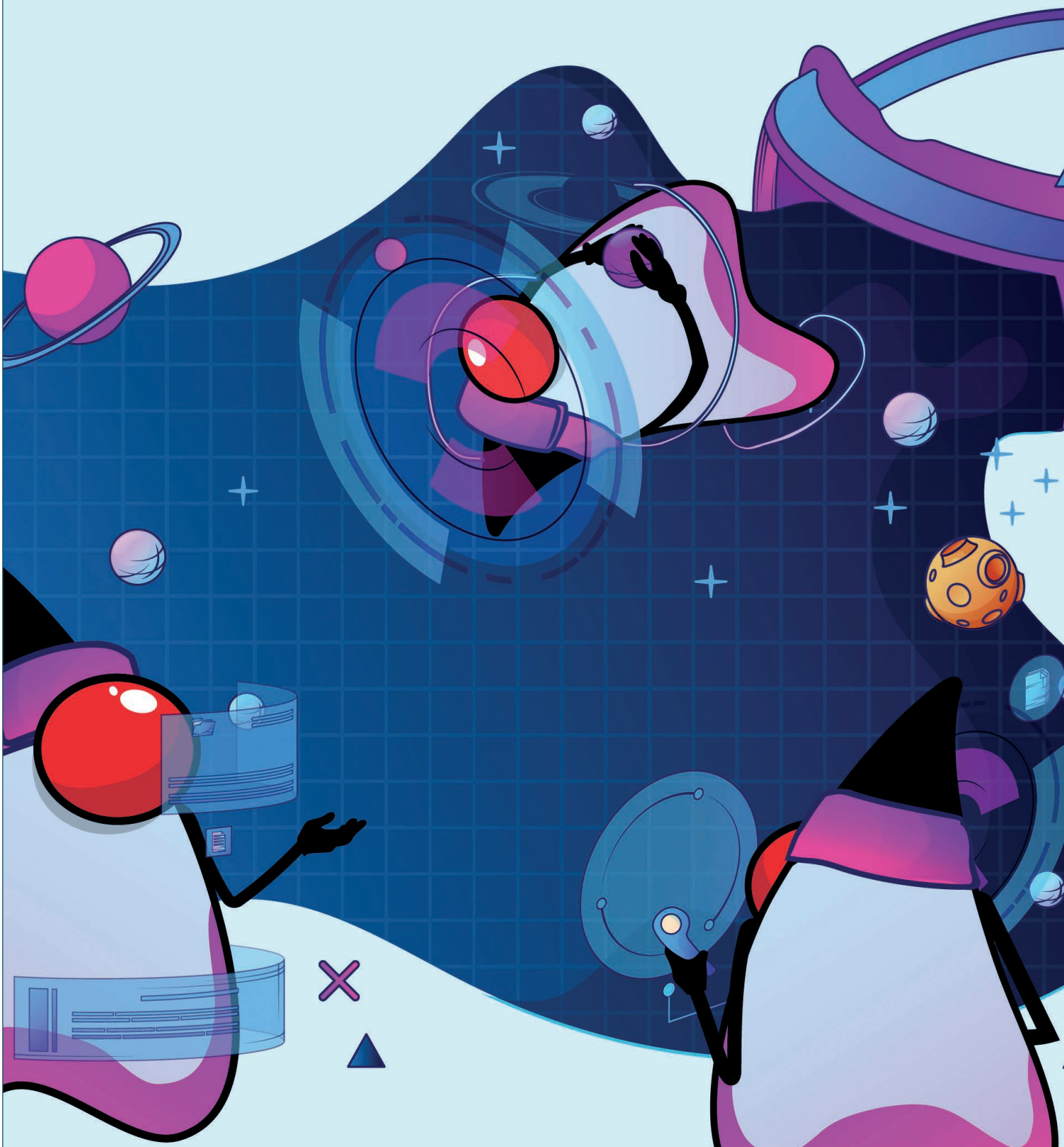
Java aktuell

iJUG Verbund
www.ijug.eu

Jakarta EE
Moderne Berechtigungssteuerung

Transformer-Architekturen
Was steckt hinter ChatGPT, DeepSeek und Co.?

Metaversum
Digitale Transformation im öffentlichen Sektor



Ausgabe 02/2025

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwicklerinnen und Entwickler

D: 7,95 EUR | CH: 11,99 CHF | A: 9,00 EUR | Benelux: 8,50 EUR

4 191978 307959 02

Unbekannte Kostbarkeiten des JDK – Heute: Source- und Byte-Code-Versionen

Bernd Müller, Ostfalia

Das JDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet werden, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir wollen in dieser Reihe derartige Features des JDK vorstellen: die unbekannten Kostbarkeiten.

Unsere heutige Ausgabe der unbekannten Kostbarkeiten widmet sich den beiden Enums `SourceVersion` und `ClassFileFormatVersion`. Für das Tagesgeschäft eines Java-Entwicklers eher weniger interessant, sind sie doch eine nützliche Unterstützung, wenn man – was beim Autor häufiger vorkommt – sich mal wieder nicht an die Major-Versionen des Byte-Codes oder versionsabhängige Syntaxänderungen erinnern kann.

Byte-Code-Versionen

Der Autor wird häufig von Studenten gefragt, was es wohl mit der Fehlermeldung

```
java.lang.UnsupportedClassVersionError: Main has been compiled by a more recent version of the Java Runtime (class file version <n>), this version of the Java Runtime only recognizes class file versions up to <k>
```

auf sich habe. Der erfahrene Java-Entwickler weiß, dass eine JVM sich geflissentlich weigert, eine neuere Byte-Code-Version zu verarbeiten als die, für die sie gemacht ist. Eine Möglichkeit, die oben genannten Byte-Code-Versionen <n> und <k> einer Java-Version zuzuordnen, ist etwa die Übersicht der Java-Releases mit jeweiliger Byte-Code-Versionsnummer des Java Version *Almanac [1]*, Wikipedia oder die Verwendung des LLMs des Vertrauens.

Sportlicher – und für Java-Enthusiasten befriedigender – ist es, Java selbst beziehungsweise das JDK als Lösungshilfe zu verwenden. Alle benötigten Informationen sind in dem Enum `ClassFileFormatVersion` im Package `java.lang.reflect` vorhanden und können abgefragt werden. Wir verwenden hierzu `JShell`, da das API sehr schlank ist und das Schreiben und Ausführen eines Programms mit einer IDE deutlich aufwendiger wäre. *Listing 1* zeigt eine `Jshell`-Sitzung, die mit den eingefügten Kommentaren hoffentlich selbsterklärend ist. Bitte gehen Sie die einzelnen Zeilen des Listings durch.

Quell-Code-Versionen

IDEs zeigen uns verlässlich Syntaxfehler an und empfehlen uns Code-Verbesserungen. Nichtsdestotrotz fragt sich der Autor manchmal: „War das schon immer so?", oder, „Seit wann ist das so?" Auch für diese tiefergehenden Fragen des Lebens gibt es ein API: das Enum `SourceVersion` im Package `javax.lang.model`. Dieses Enum modelliert die einzelnen Versionen identisch zum Enum `ClassFileFormatVersion`. Entstehungsgeschichtlich ist es jedoch so, dass `ClassFileFormatVersion` die Enums von `SourceVersion` übernommen hat, da `SourceVersion` seit Java 6, `ClassFileFormatVersion` aber erst seit Java 20 existiert.

```
// letzte Class-File-Version dieser JVM
jshell> ClassFileFormatVersion.latest()
$183 ==> RELEASE_23

// jede Version ist als Enum 'RELEASE_N' definiert
jshell> ClassFileFormatVersion.RELEASE_23
$184 ==> RELEASE_23

// kann auch über einen String erzeugt werden
jshell> ClassFileFormatVersion.valueOf("RELEASE_12")
$185 ==> RELEASE_12

// und die Major-Version als Integer
jshell> ClassFileFormatVersion.RELEASE_23.major()
$186 ==> 67

// Suche Release für Major-Version, hier 61
jshell> Arrays.stream(ClassFileFormatVersion.values())
             .filter(v -> v.major() == 61).findFirst()
$187 ==> Optional[RELEASE_17]
```

Listing 1

Bevor wir das API auch hier mit Beispielen in der JShell erläutern wollen, müssen wir uns aber zunächst mit Teilen der Java-Syntax beschäftigen, nämlich Bezeichner, Namen und Schlüsselwörter, da diese sich im API widerspiegeln.

Die Java-Sprachbeschreibung *[2]* nutzt hierzu die folgenden Abschnitte: 3.8 Identifiers, 3.9 Keywords und 6.2 Names and Identifiers, wobei aber auch noch andere Abschnitte zumindest teilweise weitere Informationen beisteuern. Da sich die genannten Abschnitte über mehrere Seiten erstrecken, wollen wir hier nur eine erste Näherung zusammengefasst wiedergeben. Ein Bezeichner besteht aus ASCII-Zeichenfolgen, die auch als Unicode geschrieben werden dürfen und darf kein Schlüsselwort, `true`, `false` oder `null` darstellen. Bei den Schlüsselwörtern gibt es 51 reservierte und 17 kontextuelle Schlüsselwörter (Stand Java 23). Reservierte Schlüsselwörter dürfen nicht als Bezeichner verwendet werden. Kontextuelle Schlüsselwörter dürfen in bestimmten Kontexten als Schlüsselwörter verwendet werden, in anderen Kontexten nicht. Beispiele für reservierte Schlüsselwörter sind etwa `class`, `for` und `new`. Beispiele für kontextuelle Schlüsselwörter sind etwa `module`, `record` und `sealed`. Namen existieren als einfache und als qualifizierte Namen, wobei qualifizierte Namen die Namensbestandteile (Bezeichner) durch einen Punkt trennen.

Bitte, lieber Leser, nageln Sie uns nicht auf diese simplifizierenden Definitionen fest. Sie erlauben es uns aber, dass wir uns näher mit dem API des Enums `SourceVersion` beschäftigen können, was wir wiederum mit der JShell tun wollen und in *Listing 2* dargestellt haben. Wir beginnen mit Schlüsselwörtern, die nicht von Anfang an in Java enthalten waren, sondern später eingeführt wurden: `strictfp` in 1.2, `assert` in 1.4 und `enum` in 5. Dann versuchen wir noch den Unterschied zwischen reservierten und kontextuellen Schlüsselwörtern herauszuarbeiten und widmen uns zuletzt dem Unterstrich. Da sich die Semantik der Methoden, insbesondere von `isIdentifier()`, nicht sofort erschließt, zeigt *Abbildung 1* das JavaDoc der verwendeten Methoden.

Bei den Recherchen zu diesem Artikel haben wir uns auch den Quellcode des Enums `SourceVersion` angeschaut. Die Dokumentation der einzelnen Enum-Werte ist in *Listing 3* dargestellt. Es ist die kom-

```
jshell> SourceVersion.isKeyword("class")
$183 ==> true

// ein Schlüsselwort ist (leider) auch ein Bezeichner
jshell> SourceVersion.isIdentifier("class")
$184 ==> true

jshell> SourceVersion.isKeyword("strictfp", SourceVersion.RELEASE_1)
$185 ==> false

// In Java 1.2 eingeführt für exakte Gleitkommaarithmetik
// mit JEP 306 [3] obsolet
jshell> SourceVersion.isKeyword("strictfp", SourceVersion.RELEASE_2)
$186 ==> true

jshell> SourceVersion.isKeyword("assert", SourceVersion.RELEASE_3)
$187 ==> false

// In Java 1.4 als Schlüsselwort eingeführt
jshell> SourceVersion.isKeyword("assert", SourceVersion.RELEASE_4)
$188 ==> true

jshell> SourceVersion.isKeyword("enum", SourceVersion.RELEASE_4)
$189 ==> false

// In Java 5 als Schlüsselwort eingeführt
jshell> SourceVersion.isKeyword("enum", SourceVersion.RELEASE_5)
$190 ==> true

// Identifier, reservierte und kontextuelle Schlüsselwörter
jshell> SourceVersion.isIdentifier("sealed")
$191 ==> true

// gilt nicht für kontextuelle Schlüsselwörter
jshell> SourceVersion.isKeyword("sealed")
$192 ==> false

jshell> SourceVersion.isName("_", SourceVersion.RELEASE_8)
$193 ==> true

// Unterstrich mit JEP 213 [4], Java 9, kein Bezeichner mehr
jshell> SourceVersion.isName("_", SourceVersion.RELEASE_9)
$194 ==> false

jshell> SourceVersion.isKeyword("_", SourceVersion.RELEASE_8)
$195 ==> false

jshell> SourceVersion.isKeyword("_", SourceVersion.RELEASE_9)
$196 ==> true

jshell> SourceVersion.isIdentifier("System.out")
$197 ==> false

// Unterschied zwischen Namen und Bezeichnern
jshell> SourceVersion.isName("System.out")
$198 ==> true
```

Listing 2

Modifier and Type	Method	Description
static boolean	isIdentifier(CharSequence name)	Returns whether or not name is a syntactically valid identifier (simple name) or keyword in the latest source version.
static boolean	isKeyword(CharSequence s)	Returns whether or not s is a keyword, boolean literal, or null literal in the latest source version.
static boolean	isKeyword(CharSequence s, SourceVersion version)	Returns whether or not s is a keyword, boolean literal, or null literal in the given source version.
static boolean	isName(CharSequence name)	Returns whether or not name is a syntactically valid qualified name in the latest source version.
static boolean	isName(CharSequence name, SourceVersion version)	Returns whether or not name is a syntactically valid qualified name in the given source version.

Abbildung 1

```
/*
 * Summary of language evolution
 * 1.1: nested classes
 * 1.2: strictfp
 * 1.3: no changes
 * 1.4: assert
 * 1.5: annotations, generics, autoboxing, var-args...
 * 1.6: no changes
 * 1.7: diamond syntax, try-with-resources, etc.
 * 1.8: lambda expressions and default methods
 *   9: modules, small cleanups to 1.7 and 1.8 changes
 *  10: local-variable type inference (var)
 *  11: local-variable syntax for lambda parameters
 *  12: no changes (switch expressions in preview)
 *  13: no changes (text blocks in preview; switch expressions in
 *      second preview)
 *  14: switch expressions (pattern matching and records in
 *      preview; text blocks in second preview)
 *  15: text blocks (sealed classes in preview; records and pattern
 *      matching in second preview)
 *  16: records and pattern matching (sealed classes in second preview)
 *  17: sealed classes, floating-point always strict (pattern
 *      matching for switch in preview)
 *  18: no changes (pattern matching for switch in second preview)
 *  19: no changes (pattern matching for switch in third preview,
 *      record patterns in preview)
 *  20: no changes (pattern matching for switch in fourth preview,
 *      record patterns in second preview)
 *  21: pattern matching for switch and record patterns (string
 *      templates in preview, unnamed patterns and variables in
 *      preview, unnamed classes and instance main methods in preview)
 *  22: unnamed variables & patterns (statements before super(...)
 *      in preview, string templates in second preview, implicitly
 *      declared classes and instance main methods in second preview)
 *  23: no changes (primitive Types in Patterns, instanceof, and
 *      switch in preview, module Import Declarations in preview,
 *      implicitly declared classes and instance main in third
 *      preview, flexible constructor bodies in second preview)
 *  24: tbd
 */
```

Listing 3

pakteste Darstellung der syntaktischen Entwicklung von Java, die wir kennen, sodass wir sie dem Leser nicht vorenthalten wollen.

Zusammenfassung

Das JDK besitzt mit den Enums `SourceVersion` und `ClassFileFormatVersion` zwei Informationsquellen, die Entwickler für verschiedene Fragestellungen zur Entwicklungsgeschichte von Java auf Quell- und Byte-Code-Ebene nutzen können. Für den Entwickler-Alltag nicht von zentralem Interesse, da typischerweise durch IDE-Funktionalitäten abgedeckt, ist es aber eventuell doch hilfreich, diese Enums zu kennen, da sie insbesondere Diskussionen bezüglich der Einführungszeitpunkte verschiedener Sprach-Features sowie verschiedener Major-Versionen schnell beenden können.

Quellen

[1] JDK Releases (Java Version Almanac), *https://javaalmanac.io/jdk/*

[2] The Java Language Specification – Java SE 23 Edition, *https://docs.oracle.com/javase/specs/jls/se23/jls23.pdf*

[3] JEP 306: Restore Always-Strict Floating-Point Semantics

[4] JEP 219: Milling Project Coin

Bernd Müller
Ostfalia
bernd.mueller@ostfalia.de

Nach seinem Studium der Informatik und der Promotion arbeitete Bernd Müller für die IBM und die HDI Informationssysteme. Er ist Professor, Autor mehrerer Bücher zu den Themen JSF und JPA sowie Speaker auf nationalen und internationalen Konferenzen. Er engagiert sich im iJUG und speziell in der JUG Ostfalen.

Mitglieder des iJUG

01 BED-Con e.V.
02 Clojure User Group Düsseldorf
03 DOAG e.V.
04 EuregJUG Maas-Rhine
05 JUG Augsburg
06 JUG Berlin-Brandenburg
07 JUG Bremen
08 JUG Bielefeld
09 JUG Bonn
10 JUG Darmstadt
11 JUG Deutschland e.V.
12 JUG Dortmund
13 JUG Düsseldorf rheinjug
14 JUG Erlangen-Nürnberg
15 JUG Freiburg
16 JUG Goldstadt
17 JUG Görlitz
18 JUG Hannover
19 JUG Hessen
20 JUG HH
21 JUG Ingolstadt e.V.
22 JUG Kaiserslautern
23 JUG Karlsruhe
24 JUG Köln
25 Kotlin User Group Düsseldorf
26 JUG Mainz
27 JUG Mannheim
28 JUG München
29 JUG Münster
30 JUG Oberland
31 JUG Ostfalen
32 JUG Paderborn
33 JUG Saxony
34 JUG Stuttgart e.V.
35 JUG Switzerland
36 JSUG
37 Lightweight JUG München
38 SUG Deutschland e.V.
39 JUG Thüringen
40 JUG Saarland
41 JUG Duisburg
42 JUG Frankfurt
43 JUG Oberpfalz

iJUG Verbund
www.ijug.eu

Impressum

Java aktuell wird vom Interessenverbund der Java User Groups e.V. (iJUG) (Tempelhofer Weg 64, 12347 Berlin, *www.ijug.eu*) herausgegeben. Es ist das User-Magazin rund um die Programmiersprache Java im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Java-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

Java aktuell wird verlegt von der DOAG Dienstleistungen GmbH, Tempelhofer Weg 64, 12347 Berlin, Deutschland, gesetzlich vertreten durch den Geschäftsführer Fried Saacke, deren Unternehmensgegenstand Vereinsmanagement, Veranstaltungsorganisation und Publishing ist.

DOAG e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. DOAG e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Björn Bröhl. DOAG e.V. informiert kompetent über alle Oracle-Themen, setzt sich für die Interessen der Mitglieder ein und führen einen konstruktiv-kritischen Dialog mit Oracle.

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
ViSdP: Fried Saacke
Redaktionsleitung: Lisa Damerow
Kontakt: redaktion@ijug.eu

Redaktionsbeirat:
Andreas Badelt, Marcus Fihlon, Markus Karg, Manuel Mauky, Bernd Müller, Benjamin Nothdurft, Daniel van Ross, Bennet Schulz

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH

Bildnachweis:
Titel: Bild © Designed by macrovector
https://freepik.com
S. 2: Bild © DOAG
https://doag.org
S. 10 + 11: Sloth McSloth
https://stock.adobe.com
S. 14 + 15: Bild © U2M Brand
https://stock.adobe.com
S. 18 + 19: Bild © Paper Trident
https://stock.adobe.com
S. 24 + 25: Bild © Designed by freepik
https://freepik.com
S. 30 + 31: Bild © Suriyo
https://stock.adobe.com
S. 36 + 37: Bild © Srinard
https://stock.adobe.com
S. 42 + 43: Bild © Designed by bunny
https://freepik.com
S. 50 + 51: Bild © yourapechkin
https://stock.adobe.com
S. 58 + 59: Bild © Designed by freepik
https://freepik.com
S. 64 + 65: Bild © Christian Horz
https://stock.adobe.com

Anzeigen:
DOAG Dienstleistungen GmbH
Kontakt: sponsoring@doag.org
Mediadaten und Preise:
www.doag.org/go/mediadaten

Druck:
WIRmachenDRUCK GmbH
www.wir-machen-druck.de

Inserentenverzeichnis