1.4 Anmerkung zum Programmierstil

In diesem Abschnitt möchte ich vorab noch etwas zum Programmierstil sagen, weil bei Ihnen vielleicht ab und an einmal die Frage aufkommen mag, ob man gewisse Dinge nicht kompakter gestalten könnte oder sollte.

1.4.1 Gedanken zur Sourcecode-Kompaktheit

In der Regel sind mir beim Programmieren und insbesondere für die Implementierungen in diesem Buch vor allem eine leichte Nachvollziehbarkeit sowie eine übersichtliche Strukturierung und damit später eine vereinfachte Veränderbarkeit wichtig. Deshalb sind die gezeigten Implementierungen möglichst verständlich programmiert. Dadurch ist vielleicht nicht jedes Konstrukt maximal kompakt, dafür aber in der Regel gut nachvollziehbar. Diesem Aspekt möchte ich in diesem Buch den Vorrang geben. Auch in der Praxis kann man damit oftmals besser leben als mit einer schlechten Wartbarkeit, dafür aber einer kompakteren Programmierung.

Beispiel

Schauen wir uns zur Verdeutlichung ein kleines Beispiel an. Zunächst betrachten wir die lesbare, gut verständliche Variante zum Umdrehen des Inhalts eines Strings, die zudem sehr schön die beiden wichtigen Elemente des rekursiven Abbruchs und Abstiegs verdeutlicht:

```
static String reverseString(final String input)
{
    // rekursiver Abbruch
    if (input.length() <= 1)
        return input;

    final char firstChar = input.charAt(0);
    final String remaining = input.substring(1);

    // rekursiver Abstieg
    return reverseString(remaining) + firstChar;
}
```

Die folgende deutlich kompaktere Variante bietet diese Vorteile nicht:

```
static String reverseStringShort(final String input)
{
    return input.length() <= 1 ? input :
           reverseStringShort(input.substring(1)) + input.charAt(0);
}
```

Überlegen Sie kurz, in welcher der beiden Methoden Sie sich sicher fühlen, eine nachträgliche Änderung vorzunehmen. Und wie sieht es aus, wenn Sie noch Unit Tests ergänzen wollen: Wie finden Sie passende Wertebelegungen und Prüfungen?

Außerdem sollte man bedenken, dass die obere Variante entweder bereits während der Kompilierung (Umwandlung in den Bytecode) oder später während der Ausführung und Optimierung automatisch in etwas Ähnliches wie die untere Variante konvertiert wird – eben mit dem Vorteil der besseren Lesbarkeit beim Programmieren.

1.4.2 Gedanken zu `final` und `var`

Normalerweise bevorzuge ich, unveränderliche Variablen als `final` zu markieren. In diesem Buch verzichte ich mitunter darauf. Ein Grund ist, dass die JShell das Schlüsselwort `final` nicht überall unterstützt, glücklicherweise aber an den wichtigen Stellen, nämlich für Parameter und lokale Variablen.

Local Variable Type Inference – `var`

Seit Java 10 existiert die sogenannte Local Variable Type Inference, besser bekannt als `var`. Diese erlaubt es, auf die explizite Typangabe auf der linken Seite einer Variablendefinition zu verzichten, sofern sich der konkrete Typ für eine lokale Variable anhand der Definition auf der rechten Seite der Zuweisung vom Compiler ermitteln lässt:

```
var name = "Peter";                // var => String
var chars = name.toCharArray();    // var => char[]

var mike = new Person("Mike", 47); // var => Person
var hash = mike.hashCode();        // var => int
```

Insbesondere im Zusammenhang mit generischen Containern spielt die Local Variable Type Inference ihre Vorteile aus:

```
// var => ArrayList<String>
var names = new ArrayList<String>();
names.add("Tim");
names.add("Tom");
names.add("Jerry");

// var => Map<String, Long>
var personAgeMapping = Map.of("Tim", 47L, "Tom", 12L,
                              "Michael", 47L, "Max", 25L);
```

Konvention: `var`, falls lesbarer

Sofern die Verständlichkeit darunter nicht leidet, werde ich `var` an geeigneter Stelle verwenden, um den Sourcecode kürzer und klarer zu halten. Ist jedoch eine Typangabe für das Nachvollziehen von größerer Wichtigkeit, bevorzuge ich den konkreten Typ und vermeide `var` – die Grenzen sind aber fließend.

Konvention: `final` oder `var`

Eine weitere Anmerkung noch: Zwar kann man `final` und `var` kombinieren, ich finde dies jedoch stilistisch nicht schön und verwende entweder das eine oder das andere.

1.4.3 Blockkommentare in Listings

Beachten Sie bitte, dass sich in den Listings diverse Blockkommentare finden, die der Orientierung und dem besseren Verständnis dienen. In der Praxis sollte man derartige Kommentierungen mit Bedacht einsetzen und lieber einzelne Sourcecode-Abschnitte in Methoden auslagern. Für die Beispiele des Buchs dienen diese Kommentare aber als Anhaltspunkte, weil die eingeführten oder dargestellten Sachverhalte für Sie als Leser vermutlich noch neu und ungewohnt sind.

```
// Prozess erzeugen
final String command = "sleep 60s";
final String commandWin = "cmd timeout 60";
final Process sleeper = Runtime.getRuntime().exec(command);
//    ...

// Process => ProcessHandle
final ProcessHandle sleeperHandle = ProcessHandle.of(sleeper.pid()).
                                    orElseThrow(IllegalStateException::new);
//    ...
```

1.4.4 Gedanken zur Formatierung

Die in den Listings genutzte Formatierung weicht leicht von den Coding Conventions von Oracle[1] ab. Ich orientiere mich an denjenigen von Scott Ambler[2], der insbesondere (öffnende) Klammern in jeweils eigenen Zeilen vorschlägt. Dazu habe ich ein spezielles Format namens `Michaelis_CodeFormat` erstellt. Dieses ist im Projekt-Download integriert und wird nachfolgend für eine Klasse demonstriert:

```
import java.util.Arrays;
import org.apache.log4j.Logger;

public final class FormatingExample
{
    private static final Logger log = Logger.getLogger("FormatingExample");

    public static String asHex(final byte[] telegram)
    {
        log.info("asHex(" + Arrays.toString(telegram) + ")");

        final StringBuffer sb = new StringBuffer("0x");

        for (int i = 0; i < telegram.length; i++)
        {
            final String hex = Integer.toHexString(telegram[i]);
            sb.append(hex);
        }

        return sb.toString();
    }
    // ...
}
```

[1] https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html

[2] http://www.ambysoft.com/essays/javaCodingStandards.html