8 Übungen zu den JVM-Neuerungen in JDK 11 bis 17

Die folgenden Aufgaben sollen helfen, die Neuerungen in der JVM aus den JDKs 11 bis 17 anhand von Übungen in ihrer Handhabung zu vertiefen.

8.1 Aufgaben

✍ Aufgabe 1 – JMH ✍

Erzeugen Sie mit dem Maven-Kommando aus Abschnitt 7.4.2 ein JMH-Projekt. Erweitern Sie dieses Projekt und erstellen Sie einen einfachen Benchmark (als Ideengeber: Öffnen Sie das bestehende Projekt `jmh-java-profi-performance-test` und kopieren Sie eine der Benchmark-Klassen aus dem Verzeichnis `benchmark-templates`). Bauen Sie das Projekt und führen Sie den oder die Benchmark(s) aus.

✍ Aufgabe 2 – JPackage ✍

Experimentieren Sie mit dem Projekt `PackagingDemoWithGuava` und wandeln Sie dieses so ab, dass noch eine weitere Abhängigkeit verwendet wird, etwa auf Apache Commons. Erstellen Sie dann eine Distribution mit `jpackage` (vgl. Abschnitt 7.5).

✍ Aufgabe 3 – JShell-API ✍

Führen Sie mit dem JShell-API ein paar dynamische Berechnungen aus. Beginnen Sie beispielsweise mit folgenden Variablendefinitionen:

```
int result = x * y;
var today = LocalDate.now();
var values = List.of(1, 2, 3, 4);
```

Für einige der obigen Variablendefinitionen bedarf es zuvor anderer Variablendefinitionen bzw. Imports. Erstellen Sie geeignete Aufrufe mit `eval()` und erzeugen Sie zuvor eine JShell-Instanz.

Geben Sie zum Abschluss alle Variablen und deren Werte auf der Konsole aus. Verwenden Sie dafür die Methode `variables()`.

✍ Aufgabe 4 – Direkte Kompilierung und Ausführung ✍

Schreiben Sie eine Klasse `HelloWorld` im Package `direct.compilation` und speichern Sie diese in einer gleichnamigen Java-Datei. Führen Sie diese direkt mit dem Kommando `java` aus.

8.2 Lösungen

✍ Lösung 1 – JMH ✍

Erstellen Sie ein eigenes JMH-Projekt oder nutzen Sie das mitgelieferte Projekt `jmh-java-profi-performance-test` als Basis, um dort die Aktionen und Messungen auszuführen.

✍ Lösung 2 – JPackage ✍

Führen Sie die Aktionen im mitgelieferten Projekt `PackagingDemoWithGuava` aus. Erstellen Sie dann eine Distribution mit Kommandos wie dem folgenden:

```
jpackage --input target/ --name JPackageDemoApp
         --main-jar PackagingDemoExamples-1.0.0-SNAPSHOT.jar
         --main-class de.java17.ApplicationExample
         --type <dmg / msi / ...>
```

Sofern Sie mit den neuen Preview-Features aus Java 17 experimentieren, ergänzen Sie bitte noch folgende Option:

```
--java-options '--enable-preview'
```

Installieren Sie die zuvor erzeugte Distribution und führen Sie das Programm aus.

✍ Lösung 3 – JShell-API ✍

Zunächst einmal erzeugen wir uns mit `JShell.create()` eine JShell-Instanz. Nun führen wir zwei Variablendefinitionen mit `eval()` durch. Weil wir mit dem Resultat nicht im Java-Sourcecode weiterarbeiten, können wir auch die von `eval()` gelieferte `List<SnippetEvent>` ignorieren. Nur für die Berechnung der Variablen `result` nutzen wir diese Liste, um ein paar mehr Informationen zum Snippet auszugeben.

Für die weiteren beiden Variablendefinitionen müssen wir an den Tipp aus der Aufgabenstellung denken und jeweils einen passenden Import per `eval()` ausführen.

Danach rufen wir die Methode `variables()` auf, um alle zuvor definierten Variablen samt ihrer Werte auszugeben:

```
import java.util.List;
import java.util.function.Consumer;

import jdk.jshell.JShell;
import jdk.jshell.SnippetEvent;
import jdk.jshell.VarSnippet;

public class Exercise03_JShellCalculation
{
    public static void main(final String args[])
    {
        try (JShell jshell = JShell.create())
        {
            jshell.eval("int x = 7;");
            jshell.eval("int y = 6;");
```

```
            List<SnippetEvent> snippetEvents =
                                jshell.eval("int result = x * y;");
            System.out.println("Size of list: " + snippetEvents.size());
            System.out.println("Value of the expression is " +
                                snippetEvents.get(0).value());

            jshell.eval("import java.time.*;");
            jshell.eval("var today = LocalDate.now();");

            jshell.eval("import java.util.*;");
            jshell.eval("var values = List.of(1, 2, 3, 4);");

            jshell.variables().forEach(printVarInfo(jshell));
            snippetEvents = jshell.eval("""
                    for (var val : values) {
                         System.out.println("val: " + val);
                    }
                    """);
            System.out.println("source: " + snippetEvents.get(0).
                                            snippet().source());
        }
    }

    private static Consumer<VarSnippet> printVarInfo(JShell jshell)
    {
        return x -> System.out.println("var:" + x.name() +
                                       "=" + jshell.varValue(x));
    }
}
```

✍ Lösung 4 – Direkte Kompilierung und Ausführung ✍

Führen Sie die Aktionen im mitgelieferten Verzeichnis `directcompilation` mit den Beispielen aus. Vergleichen Sie die Resultate mit denen in Abschnitt 7.3 beschriebenen, etwa der Palindrom-Prüfung:

```
$ java ./PalindromeChecker.java OTTO
'OTTO' is a palindrome? => true
$
$ java ./PalindromeChecker.java SOPHIE
'SOPHIE' is a palindrome? => false
```