Polyglott

Wie wir gelernt haben, kann GraalVM aber noch sehr viel mehr, als einfach nur JavaCode auszuführen. Dazu zählt das native-image building und der Support vieler anderer Programmiersprachen. Native-images schauen wir uns am Ende noch mal genauer an und fokussieren uns erstmal auf die anderen Programmiersprachen. Ziel dieser Übung ist es, andere Sprachen als Java mit GraalVM auszuführen.

Um andere Programmiersprachen verwenden zu können müssen wir den entsprechenden Sprachenadapter für Truffle installieren. Dabei werden auch andere Tools, die mit einer normalen Installation der Programmiersprache benötigt werden mit installiert (z.B. npm im Falle von NodeJS).

Wir starten mit einem Beispiel in Python:

```
gu install python
which python
echo 'print("Hi :)"' > hello.py
python hello.py
```

Probiert das ganze Mal selbst aus. Hier z. B. JavaScript bzw. NodeJS

```
gu install nodejs
which js
echo 'console.log("Hi :)"' > hello.js
js hello.js
```

Jetzt haben wir Python oder Javascript Code über die GraalVM ausgeführt, wir können aber noch einen Schritt weiter gehen und die Sprachen miteinander mixen. Graal erlaubt es uns, beliebig Sprachen zu mixen. Also können wir auch eine Sprache mit einer anderen kombinieren.

Dafür schreiben wir zunächst den gewünschten Code und speichern uns den Quellcode als String zwischen. Anschließend erstellen wir einen Context. Mit einem Context können wir die Einstellungen der eingebetteten Sprache setzen. In diesem Fall haben wir keine Besonderen, also rufen wir einfach create() auf. Mit eval() können wir dann den Code auswerten. In diesem Aufruf wertet Graal den Quellcode aus und führt ihn einmal aus.

```
import org.graalvm.polyglot.*;
import org.graalvm.polyglot.proxy.*;

 public class HelloPolyglot {

     static String JS_CODE = "console.log('hello TestValue')";

     public static void main(String[] args) {
         System.out.println("Hello Java!");
         try (Context context = Context.create()) {
             context.eval("js", JS_CODE);
         }
     }
 }
```

In der Konsole sehen wir nun am Ende unser „hello TestValue“.

Wir können auch ganze Berechnungen in eine andere Sprache auslagern. Ist vielleicht bei Data Science Aspekten hilfreich, die üblicherweise in Python besser aufgehoben sind, aber wir anderen Code in Java haben. Gerade wenn Drittanbieter-Software gebraucht wird, dann sind diese nicht immer in allen Sprachen vorhanden.

In dem folgenden Beispiel definieren wir eine neue Funktion „myFun“, die Daten erhält um eine Ausgabe zu erhalten und „Woop“ an Java zurückgibt. Wir sehen, dass aus dem eval() Aufruf nun „value“ herauskommt. Value entspricht der definierten myFun und kann mit execute beliebig oft ausgeführt werden, ohne dass jedes Mal der Javascript-Code neu ausgewertet werden muss. „returnedValue“ ist dann unser „Woop“.

```
import org.graalvm.polyglot.*;
import org.graalvm.polyglot.proxy.*;

public class HelloPolyglot {

    static String JS_CODE = "(function myFun(param){console.log('hello
'+param); return 'WOOP'})";

    public static void main(String[] args) {
        System.out.println("Hello Java!");
        try (Context context = Context.create()) {
            var value = context.eval("js", JS_CODE);
            var returnedValue = value.execute("TestValue");
            System.out.println(returnedValue);
        }
    }
}
```

Auch hier können wir noch einen Schritt weiter gehen und können sogar andere Abhängigkeiten z. B. von NPM verwenden.

npm i cowsayjs
npx cowsayjs Hello

Wir sehen eine Kuh auf der Kommandozeile. Genau das gleiche Verhalten wollen wir in Java abbilden. Wir erstellen also wieder einen Context, müssen aber noch einige Parameter setzen. Die beiden allow-Optionen erlauben es, dass GraalVM auf die Festplatte zugreifen kann. Die js.commonjs-require definieren sprachspezifische Einstellungen, in diesem Fall js für JavaScript. Mit den Optionen sagen wir Graal, dass es okay ist commonjs-require Abhängigkeiten zu laden (das die das Format, wie NodeJS Abhängigkeiten liest) und in welchem Verzeichnis diese abgelegt sind.

Anschließend verhält sich alle wie wir es aus Javascript und aus den Vorherigen Übungen kennengelernt haben.

```java
import java.util.HashMap;

import org.graalvm.polyglot.*;
import org.graalvm.polyglot.proxy.*;

public class HelloWorld {

    public static void main(String[] args) {
        var options = new HashMap<String, String>();
        options.put("js.commonjs-require", "true");
        options.put("js.commonjs-require-cwd", ".");

        var cx = Context.newBuilder("js")
                .allowExperimentalOptions(true)
                .allowIO(true)
                .options(options)
                .build();

        cx.eval("js", """
            var cowsayjs = require("cowsayjs");

            console.log(cowsayjs.moo("Hello"));
                """);
    }
}
```