Java Selenium Tests als standalone jar ausführen

https://nissel.it/index.php/2020/03/30/java-selenium-tests-als-standalone-jar-ausfuehren/

Selenium Tests werden normalerweise in der IDE oder über einen Jenkins ausgeführt. Möchte man die Test aber von einem Benutzer lokal ohne Java Kenntnisse ausführen lassen, so ergeben sich einige Hürden.
Voraussetzung ist der Artikel Einrichten von Selenium.

Junit Tests über Konsole starten

Dazu gibt es für Junit5 ein Modul welches in der build.gradle unter dependencies eingebunden werden muss.

```
implementation group: 'org.junit.platform', name: 'junit-platform-
console-standalone', version: '1.6.1'
```

Danach kann entweder die main() Methode der Klasse ConsoleLauncher direkt aufgerufen werden oder über eine eigene Main Methode gestartet werden.

```
public static void main(String[] args) throws IOException {
    System.setOut(outputFile("result.txt"));
    ConsoleLauncher.main("--select-package=it.nissel.selenium");
}
private static PrintStream outputFile(String name) throws FileNotFound
Exception {
    return new PrintStream(new BufferedOutputStream(new FileOutputStre
am(name)), true);
}
```

Welche Test ausgeführt werden kann man über verschiedene Wege mitgeben. Hier wurde dies anhand des package "it.nissel.sekenium" festgelegt. Weitere Informationen.
Die Standard-Ausgabe wird in eine result.txt geschrieben. Da der Benutzer später das Programm nur ausführen soll und am Ende keine Konsolenausgabe sieht, wird das Ergebnis in eine Datei geschrieben. In der Error-Ausgabe zeigt Selenium allerlei Müll an, was man über System.*setErr*() z.B: in eine logfile schreiben könnte.

Download des Treibers

Um einen Seleniumtest auszuführen ist ein entsprechender Treiber notwendig. Dieser muss sich je nach Plattform heruntergeladen werden. Damit dies nicht mitgeliefert werde muss, kann man sich die Datei über Java herunter laden.

```
@SneakyThrows
private static File downloadDriverZip(Driver driver, String version) {
    String urlString = driver.getDownloadUrl(OS.getOs(), version);
    log.info("download driver from " + urlString);
    URL url = new URL(urlString);
    URLConnection connection = url.openConnection();
    InputStream driverInputStream = connection.getInputStream();
    File targetFile = File.createTempFile("Selenium-driver", ".zip");
    targetFile.deleteOnExit();

    java.nio.file.Files.copy(
            driverInputStream,
            targetFile.toPath(),
            StandardCopyOption.REPLACE_EXISTING);

    driverInputStream.close();
    return targetFile;
}
```

In dem Enum Driver ist die URL zum Download enthalten und über OS.getOS() wird das aktuelle

Betriebssystem ermittel.

Betriebssystem ermitteln

```
public enum OS {
    WINDOWS,
    MAC,
    LINUX;

    public static OS getOs() {
        String osString = System.getProperty("os.name").toLowerCase();

        if(osString.contains("win")) {
            return OS.WINDOWS;
        } else if (osString.contains("mac")) {
            return OS.MAC;
        } else if(osString.contains("nix") || osString.contains("nux")
 || osString.contains("aix")) {
            return OS.LINUX;
        }
        throw new RuntimeException("Unknown os " + osString);

    }
}
```

URL generieren

```
public enum Driver {
    CHROME("https://chromedriver.storage.googleapis.com/", "chromedriv
er");

    private final String serverUrl;
    private final String driverFileName;

    Driver(String serverUrl, String driverFileName) {
```

```
        this.serverUrl = serverUrl;
        this.driverFileName = driverFileName;
    }

    public String getDownloadUrl(OS os, String version) {
        String fileName;
        switch (os) {
            case WINDOWS:
                fileName = "chromedriver_win32.zip";
                break;
            case MAC:
                fileName = "chromedriver_mac64.zip";
                break;
            case LINUX:
                fileName = "chromedriver_linux64.zip";
                break;
            default:
                throw new RuntimeException("Unknown OS" + os);
        }
        return serverUrl + version + "/" + fileName;
    }

    public String getDriverFileName(OS os) {
        String fileName = driverFileName;
        if(OS.WINDOWS.equals(os)) {
            fileName += ".exe";
        }
        return fileName;
    }

}
```

Hier habe ich mir den Aufbau der Downloadseite des Google Chrome Treiber angeschaut und mit getDownloadUrl() einen URL simplen Generator gebaut. getDriverFileName() wird später zum cachen benötigt.

Zipdatei entpacken

```
public static File downloadDriver(Driver driver,String version) {
    List<File> fileList = ZipUtils.unzip(downloadDriverZip(driver, ver
sion).getAbsolutePath());
    if(fileList.size() != 1) {
        String filenames = fileList.stream().map(File::getName).collec
t(Collectors.joining(", "));
        throw new RuntimeException("The driver file not found: " + fil
enames);
    }
    File driverFile = fileList.get(0);
    if(!driverFile.setExecutable(true)) {
        log.warn("The driver file " + driverFile.getAbsolutePath() + "
 cannot be set executable");
    }
    return driverFile;
}
```

Die herunter geladene Datei muss entpackt werden und nur wenn eine Datei enthalten war wird dies der Treiber sein. Dies ist sehr primitiv aber funktioniert. Die Klasse ZipUtils ist eine selbst geschriebene Klasse um einfach Zip Dateien zu entpacken. Danach muss die Datei (auf jeden Fall unter Linux) ausführbar gemacht werden.

```
public class ZipUtils {

    public static List<File> unzip(String zipFile) {
        return unzip(zipFile, ".");
    }

    public static List<File> unzip(String zipFile, String destination)
 {
        List<File> unzippedFiles = new ArrayList<>();
        try {
            ZipInputStream zipInputStream = new ZipInputStream(new Fil
eInputStream(zipFile));
            ZipEntry zipEntry = zipInputStream.getNextEntry();
            while (zipEntry != null) {
                File unzippedFile = new File(destination, zipEntry.get
Name());
                if(zipEntry.isDirectory()) {
                    if(!unzippedFile.mkdirs()) {
```

```
                        throw new RuntimeException("Cannot create dire
ctory " + unzippedFile.getAbsolutePath());
                    }
                }

                unzippedFiles.add(unzippedFile);
                if(!unzippedFile.isDirectory()) {
                    zipInputStream.transferTo(new FileOutputStream(unz
ippedFile));
                }
                zipEntry = zipInputStream.getNextEntry();
            }
            zipInputStream.closeEntry();
            zipInputStream.close();
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage(), e);
        }
        return unzippedFiles;
    }

}
```

Download Cache

Anhand des Dateinamens wird der Donwloadvorgang gecached.

```
public static File cachedDownloadDriver(Driver driver,String version)
{
    String osFileName = driver.getDriverFileName(OS.getOs());
    File cachedFile = new File(osFileName);
    if (!cachedFile.exists()) {
        return downloadDriver(driver, version);
    } else {
        return cachedFile;
    }
}
```

Hier könnte man sich auch einen intelligenteren Cache vorstellen der z.B: die Versionsnummer berücksichtigt.

Ausführbares Jar File erzeugen

Da die Testfälle unter src/main/test liegen werden sie normalerweise nicht mit in ein jar File aufgenommen. Dazu muss in der build.gradle folgender Abschnitt hinzugefügt werden.

```
jar {
    manifest {
        attributes "Main-Class": "it.nissel.selenium.TestRunner"
    }

    from {
        configurations.testRuntimeClasspath.collect { it.isDirectory()
 ? it : zipTree(it) }
    }
    from {
        configurations.runtimeClasspath.collect { it.isDirectory() ? i
t : zipTree(it) }
    }
    from sourceSets.test.output
    from sourceSets.main.output
}
```

Damit wird eine Jar Datei erzeugt die alle Abhängigkeiten aus dem Test und der Implementierung und die Klassen aus dem Test und der Implementierung in eine Jar packen. Zusätzlich wird die Klasse it.nissel.selenium.TestRunner als Startpunkt für die Jar festgelegt. Die Datei wird durch den Gradle Build unter Build/libs erzeugt.

Ausführbare .exe für Windows Benutzer erzeugen

Eigentlich kann bereits jetzt die jar Datei mit folgenden Befehl gestartet werden:

```
java -jar MeineJar.jar
```

Wem das nicht reicht, der kann eine Java Runtime >= 11 herunter bei sich lokal installieren und den Ordner unter c:\Programme\java\jre11 (genauer Pfad ist mir gerade unbekannt) kopieren und mit in den Ordner der Jar legen. Dann kann mit Launch4J eine .exe Datei erzeugt werden die, die JRE benutzt und die Jar Datei startet.
Hier gibt es glaube ich modernere Möglichkeiten, in die ich mich noch nicht eingearbeitet habe.

Links

https://junit.org/junit5/docs/current/user-guide/#running-tests-console-launcher

http://launch4j.sourceforge.net/