Java 9 (2017)

Module (*) Upate
HTTP/2
JShell
Reactive Streams (in Java "Flow")
Logging(*)
ProcessAPI(*)
Collection/StreamAPI(*)

JShell

JShell --> JavaShell --> Java Sprache verwenden
jsh-Dateien
jshell script.jsh. sofort ausführen

/help = Hilfe aufrufen
/vars = Definierte Variablen anzuzeigen
/types = Definierte Typen anzuzeigen
/list = Snippets anzuzeigen
/env = Klassen und Modulpfad defineiren
/save = Historie speichern (*,jsh)
/exit = JShell verlassen

Module

Vorgehensweise:
- Definition eines Moduls
- Beschrfeibung in module-info,.java
- Starten von Java mit --module-path
- Hinzufügen des Modules mit --add-modules (Einstiegspunkt/Package)

01) Projekt erzeugen
02) Module 1 (start) erzeugen
- Package "cmd" erzeugt
- Klasse "Start()" erzeugt
- Methode "main()" erzeugt
- Ausgabe

03) Module 2 (hello) erzeugen
- Package "hello" erzeugt
- Klasse "HelloProvider()" erzeugt
- Methode "pubic static String hello() return "Hallo World" erzeugt

04) Module 1 : Abhängigkeit zu Module 2 definieren
05) Methode main() --> System.out.print(HelloProvider.hello());
06) Um richtige Module zu programmieren, müssen "module-info.java" Klassen erzeugt werden
Module 2:

```
        module hello {          <---- Eindeutiger Module-Name
          exports hello;        <---- Bereitgestellte Paket(e)
        }
      Module 1:
        module cmd {            <---- Eindeutiger Module-Name
          require hello;        <---- Module Abhängigkeit (Statische Abhängigkeit)
        }
  07) Jar-Dateien pro Module erzeugen
    Unter IntelliJ --> Projekt -> Articats -> Jar (hello.jar mitt Output von Hello - Module 1)
    Unter IntelliJ --> Projekt -> Articats -> Jar (commandline.jar mitt Output von cmd -
Module 2)
    --> Output-Order -> 2 Jar-Dateien
  08) Module starten
    java -cp ./commandline/cmd.jar;./hello/hello.jar
       cmd.Start
  09) Java 9 Alternative zu ClasasPath, weil Fehler (ClassNotFound) zur Laufzeit erzeugt
werden
    java --module-path ./commandline/cmd.jar;./hello/hello.jar
       --add-modules cmd.
       cmd.Start

  module-info.java

    Pakete Bereitstellen (EXPORT)
    - Exports: Bereitstellen eines Pakets
    - Optional mit Besc hränkung für Zielmodul
    - Name des Pakets
    Module importieren (REQUIRE)
    - Requires : Importieren eines Moduls
    - Name des Moduls
    - Zuzätzliche Optionen:
      - transitive : Exportiere das importierte Module für andere Module
      - static : Abhängigkeit nur zur Kompilierzeit

    require MODULE;        ---> Beim Start der Anwendung wird gerpüft, ob das Module
vorhanden ist
    require static MODULE;  ---> Es wird zur Laufzeit geprüft, ob das Module vorhanden ist

    export howdy to hello;  ----> Das bereitgestellt Module ist nur für das angegebene
Module erlaubt

    Modulsyntax:

      [open]module MODULENAME
      {
        [exports|opens] PACKAGE (to PACKAGE2)
        .....
        requires (transitive)(static)MODUL
```

}

export ; Alle Packages sind für Module freigegeben
open ; Es ist für Reflection freigegeben (auch private Felder) DEFAULT : AUSGESCHALTET

java --list-modules ----> Liste alle Module die beim Start der Virtuellen Maschine geladen werden

Reactive Streams (in Java "Flow")

* Implementierung im JDK
* Erzeugung auf Basis von ThreadPool und Buffer
* Buffer-Kapazität pro Subscriber
* Bereitstellen von Daten
  - Submit() : Blockierend
  - Offer(): Nicht blockierend mit Timeout

```
public class ReactiveStreams {
   static class LogSubscriber implements Flow.Subscriber<String> {
      @Override
      public void onSubscribe(Flow.Subscription subscription) {
         subscription.request(Long.MAX_VALUE);
      }

      @Override
      public void onNext(String item) {
         System.out.println(item);
      }

      @Override
      public void onError(Throwable throwable) {
         System.out.println("Error occured");
         throwable.printStackTrace();
      }

      @Override
      public void onComplete() {
         System.out.println("Completed");
      }
   }

   static class LogProcessor extends SubmissionPublisher<String> implements Flow.Processor<String, String>, Flow.Subscriber<String> {

      @Override
      public void onSubscribe(Flow.Subscription subscription) {
```

```
            subscription.request(Long.MAX_VALUE);
        }

        @Override
        public void onNext(String item) {
            this.submit("[" + new Date().toString() + "] " + item);
        }

        @Override
        public void onError(Throwable throwable) {
            this.getExecutor().execute(() -> this.getSubscribers().forEach(s ->
s.onError(throwable)));
        }

        @Override
        public void onComplete() {
            this.close();
        }
    }

    public static void main(String[] args) {
        SubmissionPublisher<String> logPublisher = new
SubmissionPublisher<>(Executors.newCachedThreadPool(), 256);
        LogSubscriber logSubscriber = new LogSubscriber();
        LogProcessor logProcessor = new LogProcessor();
        logPublisher.subscribe(logProcessor);
        logProcessor.consume(System.out::println);
        //logPublisher.subscribe(logSubscriber);
        //logPublisher.offer("Text", 5000, TimeUnit.MILLISECONDS, (s, el) -> true);
        //logPublisher.close();
        logPublisher.submit("Text");

        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Collectiom-API

Neue Schreibweise für das erzeugen von unveränderlichen Datentypen (of()-Methode)

```
List<String> lst = List.of("Hello", "World");

Set.of("Test");
```

```
Map.of("key", "value");

Map.ofEntries(new AbstractMap.SimpleImmutableEntry<String, String>("key", "value"));
```

Prozess-API

```
public static void main(String[] args)
{
    ProcessHandle currentProcess = ProcessHandle.current();
    System.out.println(currentProcess.info().command());

    //-------

    ProcessBuilder builder = new ProcessBuilder().command("cmd"); // Programm starten
    try {
       Process p = builder.start();
       ProcessHandle processHandle = p.toHandle();
       processHandle.onExit().whenComplete((h, t) -> System.out.println("Process
exited"));
       processHandle.destroy();
    } catch (IOException e) {
       e.printStackTrace();
    }

    //-------

    ProcessHandle
          .allProcesses()
          .filter(p -> p.info().command().map(cmd ->
cmd.contains("notepad")).orElse(false))
          .findFirst()
          .ifPresent(ProcessHandle::destroy);

    try {
       Thread.sleep(5000);
    } catch (InterruptedException e) {
       e.printStackTrace();
    }
 }
```

Logging-API

JVM-Logging

VM-Options

```
-Xlog:gc=debug:file=gc.txt:time:filesize=5M
```

Nähere Informationen auf der Java Homepage
http://openjdk.java.net/jeps/158

Spring Data (beispiel Hibernate)

H2-Datenbank ???