Le Tour de

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

DART

Web-Technologien

Client- und Serverseitige Sprachen (Dart Teil II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat.
Nane Kratzke
Praktische Informatik und
betriebliche Informationssysteme

- **Raum: 17-0.10**
- **Tel.:** 0451 300 5549
- **Email: nane.kratzke@fh-luebeck.de**

Dart in a Nutshell ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

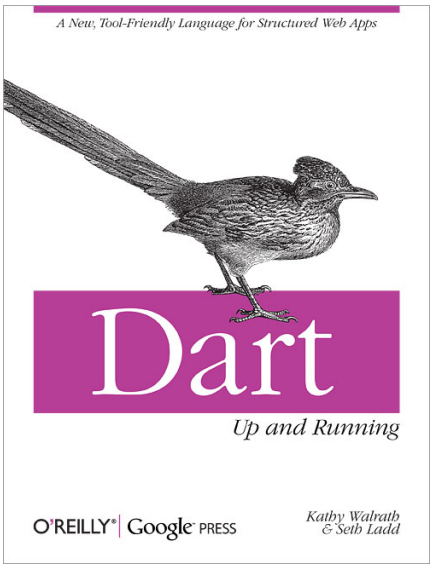
- Eine optional typisierte,
- Multiprogrammierparadigmen (imperativ, non dogmatic objektorientiert, funktional) unterstützende,
- Language VM (Virtual Machine) basierte Programmiersprache (die ein Cross Compilation nach Javascript ermöglicht)
- für Scalable Web App Engineering,
- die sowohl In-Browser als auch On-Server (in Dart VM) ausführbar ist.

https://www.dartlang.org/

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
3

Zum Nachlesen ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Chapter 1:
Quick Start

Chapter 2:
A Tour of the Dart Language

Chapter 3:
A Tour of the Dart Libraries

Chapter 4:
Tools

Chapter 5:
Walkthrough: Dart Chat

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
4

Le Grande Boucle

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Tour de Dart

- Optional Typed
- Multiparadigm
- Language VM (JavaScript Cross Compiled)
- In Browser and On Server

Tour de Dart Libraries

- Library System
- Asynchronous Programming
- Math and Random
- Browser-Based Apps
- I/O
- Decoding and Encoding

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
5

Tour de Dart (in Teil I behandelt)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Kern-Konzepte
- Variablen
- Built-In Types
- Kontrollfluss
- Funktionen (Methoden) und Typedefs
- Operatoren
- Klassen (OO)
- Generics

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
6

Tour de Dart Libraries (nun in Teil II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`dart:library`
Das Library System von Dart

`dart:async`
Asynchronous Programming

`dart:io`
Files, Directories

`dart:html`
Manipulating the DOM

`dart:servers`
HTTP Clients and Servers

`dart:convert`
Decoding and Encoding JSON, HTML and more

Und ja: Es ist noch weit bis Paris ☺

Tour de Dart Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`dart:library`
Das Library System von Dart

`dart:async`
Asynchronous Programming

`dart:io`
Files, Directories

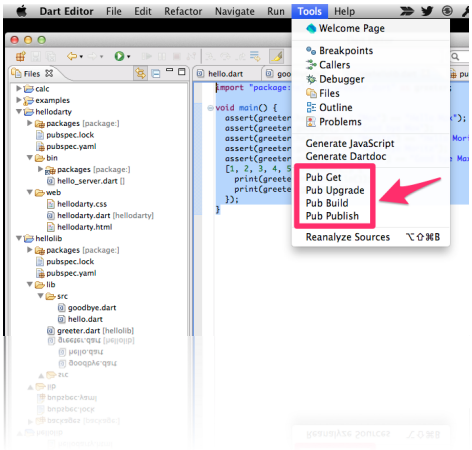
`dart:html`
Manipulating the DOM

`dart:servers`
HTTP Clients and Servers

`dart:convert`
Decoding and Encoding JSON, HTML and more

Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Dart hat einen Paketverwaltungssystem namens `pub`.

Mittels `pub get` ist es bspw. möglich, dokumentierte Abhängigkeiten zwischen Bibliotheken aufzulösen und erforderliche Bibliotheken aus unterschiedlichen Quellen automatisch nachzuladen.

Pub wertet hierzu eine `pubspec.yaml` Datei aus, um Abhängigkeiten zwischen Libraries (Packages) zu verwalten.

Eigene Bibliotheken können mittels `pub publish` auf Paketservern (der Allgemeinheit) zur Verfügung gestellt werden.

Für Client-Side Skripte bedeutet dies, dass nur eine Dart Datei in einem HTML Dokument angeben werden muss, alle weiteren Abhängigkeiten lädt Dart bei Bedarf.

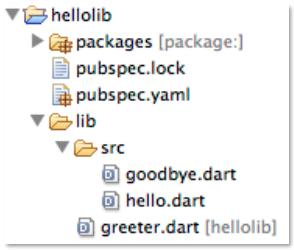
Eine Library definieren (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Wie in fast allen Sprachen *(bis auf JavaScript ;-)* ist es auch in Dart möglich Bibliotheken zu definieren, in denen wiederverwendbare Funktionalitäten bereitgestellt werden können.

Sourcen die als Libraries geladen werden können, befinden sich per Konvention in Dart Projekten in einem `lib` Unterordner.

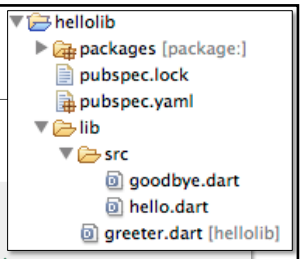
In einer `pubspec.yaml` Datei wird die Library für den Library Manager `pub` von Dart beschrieben. Abhängigkeiten zwischen Bibliotheken können dann durch das Dart Ökosystem einfach ermittelt und ggf. erforderliche Anteile nachgeladen werden.




```
name: hellolib
author: Nane Kratzke
version: 1.0.0
description: Library to demonstrate library features of Dart.
dependencies:
  math:
```

Datei: `pubspec.yaml`

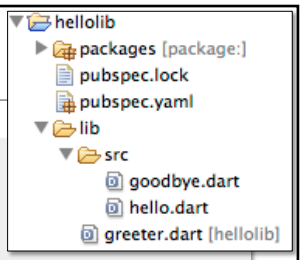
Eine Library definieren (II)




```dart
library hellolib; // Eine Library wird deklariert.

import "dart:math"; // Eine Library benötigt weitere Libraries.

part "src/hello.dart"; // Library kann aus Unterteilen bestehen.
part "src/goodbye.dart";

// Libraries koennen top level Variablen beinhalten.
String _subjectToGreet = "World";
var _subjects = ["Max", "Moritz", "Maya", "Tessa"];

// Libraries koennen auch top level Funktionen definieren und natuerlich auch Klassen.
String randomHello() {
  final r = new Random();
  return hello(name : _subjects[r.nextInt(_subjects.length)]);
}
```

Datei: `lib/greeter.dart`

Eine Library definieren (III)



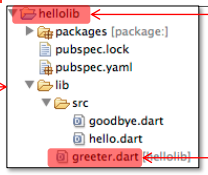

```dart
// Angabe zu welcher Library dieser Teil gehoert
// (Backlink).
part of hellolib;

String goodbye([String name]) {
  return "Good bye ${ name != null ? name : _subjectToGreet }";
}
```

Datei: `lib/goodbye.dart`

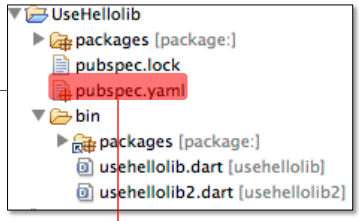
```dart
part of hellolib;

String hello({ String name : "World" }) {
  _subjectToGreet = name;
  return "Hello $_subjectToGreet";
}
```

Datei: `lib/hello.dart`

Eine Library nutzen (I)



Eine Import-Anweisung importiert eine Library. Sollen Libraries ggf. über `pub` automatisch bezogen werden, so müssen erforderliche Bibliotheken in der `pubspec.yaml` aufgeführt werden.



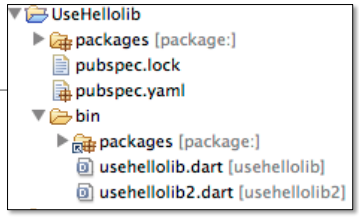

```dart
import "package:hellolib/greeter.dart" as greeter;

void main() {
  assert(greeter.hello(name : "Max") == "Hello Max");
  assert(greeter.goodbye() == "Good bye Max");
  assert(greeter.hello(name : "Moritz") == "Hello Moritz");
  assert(greeter.goodbye() == "Good bye Moritz");
  assert(greeter.goodbye("Max again") == "Good bye Max a
  [1, 2, 3, 4, 5].forEach((_) {
    print(greeter.randomHello());
    print(greeter.goodbye());
  });
}
```

Datei: `bin/usehellolib.dart`

```
name: UseHellolib
description: An application
dependencies:
  hellolib:
    path: ../hellolib
```

Datei: `pubspec.yaml`

Eine Library nutzen (II)



Ergeben sich bei einem Import keine Namenskollisionen so kann man auf `as name` verzichten. Die in der Library definierten Bezeichner sind dann direkt nutzbar.

```dart
import "package:hellolib/greeter.dart";

void main() {
  assert(hello(name : "Max") == "Hello Max");
  assert(goodbye() == "Good bye Max");
  assert(hello(name : "Moritz") == "Hello Moritz");
  assert(goodbye() == "Good bye Moritz");
  assert(goodbye("Max again") == "Good bye Max again");
  [1, 2, 3, 4, 5].forEach((_) {
    print(randomHello());
    print(goodbye());
  });
}
```

Datei: `bin/usehellolib.dart`

pubspec.yaml

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Pub unterscheidet drei Quellen, die in einer `pubspec.yaml` angegeben werden können, um von dort Packages zu laden.

- **Hosted** packages
- **Git provided** packages
- **Path provided** packages

Path provided packages:

```
dependencies:
  hellolib:
    path: /path/to/your/hellolib
```

Hosted packages:

```
dependencies:
  hellolib:
```

```
dependencies:
  hellolib:
    hosted:
      name: hellolib
      url: "http://ex.org"
    version: ">=1.0.0 <2.0.0“
```

Git provided packages:

```
dependencies:
  hellolib:
    git: "git:github.com/hellolib.git"
```

```
dependencies:
  hellolib:
    git:
      ref: some-branch
      url: "git:github.com/hellolib.git"
```


Weitere Informationen zu Dart pub

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Weitere Information zum Dart Package System `pub` finden sich hier:

```
http://pub.dartlang.org/doc/
http://pub.dartlang.org/doc/pubspec.html
http://pub.dartlang.org/doc/dependencies.html
```

Tour de Dart Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`dart:library`
Das Library System von Dart

`dart:async`
Asynchronous Programming

`dart:io`
Files, Directories

`dart:html`
Manipulating the DOM

`dart:servers`
HTTP Clients and Servers

`dart:convert`
Decoding and Encoding JSON, HTML and more

Asynchron programmieren

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Dart führt (wie jede andere imperative Programmiersprache auch) Anweisungen sequentiell aus. Die sequentielle Ausführung lassen sich mit Kontrollstrukturen (Schleifen, Bedingte Anweisungen, etc.) „umlenken“, jedoch werden immer zwei Anweisungen hintereinander ausgeführt und grundsätzlich existiert nur ein Ausführungsthread.

Insbesondere in der Webprogrammierung bedeutet sequentielle Ausführung von Anweisungen häufig „stockende Reaktionen“ auf Benutzerinteraktionen, da der Ausführungsthread gerade „anderes zu tun hat“.

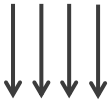
Asynchrone Programmierung bedeutet diese sequentielle Anweisungsabfolge zu durchbrechen und Anweisungen bspw. nur dann auszuführen, wenn Ereignisse eintreten oder Anweisungen tatsächlich parallel (d.h. gleichzeitig) auszuführen.

Dart sieht hierzu u.a. folgende Konzepte vor:

Streams

Futures

Isolates



Asynchron programmieren

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die folgenden Beispiele werden wir anhand der Fibonacci Funktion verdeutlichen. Diese ist nur ein Platzhalter für beliebige Verarbeitungen von Daten. Die Fibonacci Funktion hat für uns den Vorteil, dass sie bereits bei kleinen Zahlenwerten „spürbare“ Laufzeiten für ihr Berechnungsergebnis erzeugt.

```
// Eine klassische synchrone Methode.
num fib(num n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib(n-1) + fib(n-2);
}
```

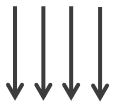
Wir werden die Fibonaccifolgen z.B. für die Zahlen von 1 bis 20 wie folgt berechnen (und im weiteren) mittels Streams, Futures und Isolates zunehmend asynchroner (und echt paralleler) implementieren.

```
main() {
  for (int i=0; i <= 20; i++) {
    print("fib($i) = ${fib(i)}");
  }
}
```

Streams

Futures

Isolates



Streams

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

In der Informatik versteht man unter einem Stream eine (kontinuierliche) Übertragung von Daten (gleichen Typs) von einer Quelle zu einer Senke.

In Dart können solche Streams mit einem `StreamController` erzeugt werden (viele Dart IO Funktionen/Methoden liefern Streams zurück, auf die dann – wie hier gezeigt – zugegriffen werden kann).

Mittels **`listen`** kann man einen Callback (anonyme Funktion) registieren, die immer dann aufgerufen wird, wenn ein neues Element in den Stream geschickt wurde.

```
import "dart:async";

main() {
  // Erzeugen eines Streams
  var broadcast = new StreamController<int>();
  var stream = broadcast.stream;

  // Berechne alles was im Stream kommt
  // ereignisbasiert (Senke)
  stream.listen((v) {
    print("fib($v) = ${fib(v)}");
  });

  // Den Stream befuellen (Quelle)
  for (int i=1; i <= 10; i++) broadcast.add(i);
  for (int i=10; i >= 1; i--) broadcast.add(i);
  print("Main finished.");
}
```

Streams

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

D.h. die Daten des Streams werden ausgeführt nachdem die `main()` Funktion beendet wurde!

Mit anderen Worten **asynchron**!

Man sieht aber auch: Die Ausgabe ist nicht beliebig (das wäre der Fall bei echter Parallelität).

- Dart Programme sind grundsätzlich **single threaded**! Es kann zu einem Zeitpunkt nur eine Anweisung ausgeführt werden.
- Das vermeidet aber auch alle Probleme der Multithread Programmierung (vgl. Programmieren II)

Erzeugt folgende Konsolenausgabe:

```
Main finished.
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
fib(10) = 55
fib(10) = 55
fib(9) = 34
fib(8) = 21
fib(7) = 13
fib(6) = 8

...
```


Broadcast Streams

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gezeigtes Beispiel verknüpft genau eine Quelle mit einer Senke.

Möchte man, dass eine Quelle mehrere (beliebig viele) Senken speist, muss man in Dart Broadcast Streams mittels `asBroadcastStream()` erzeugen.

```dart
main() {
  // Erzeugen eines Broadcaststreams
  var broadcast = new StreamController<int>();
  var stream = broadcast.stream.asBroadcastStream();

  // Senke 1
  stream.listen((v) {
    print("Listener 1: fib($v) = ${fib(v)}");
  });
  // Senke 2
  stream.listen((v) {
    print("Listener 2: fib($v) = ${fib(v)}");
  });

  [...] // wie vorher
}
```

Erzeugt folgende Konsolenausgabe:

```
Main finished.
Listener 1: fib(1) = 1
Listener 2: fib(1) = 1
Listener 1: fib(2) = 1
Listener 2: fib(2) = 1
Listener 1: fib(3) = 2
Listener 2: fib(3) = 2
Listener 1: fib(4) = 3
Listener 2: fib(4) = 3
Listener 1: fib(5) = 5
Listener 2: fib(5) = 5
Listener 1: fib(6) = 8

...
```

Streams filtern

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Es kann sein, dass Senken nicht auf alle Ereignisse in einem Stream reagieren sollen.

Dann kann man mittels Filter Methoden

- `where()`
- `takeWhile()`
- `skipWhile()`
- `take()`

Streams nur anteilig verarbeiten.

```dart
// Erzeugen eines Broadcaststreams
var broadcast = new StreamController<int>();
var stream =
broadcast.stream.asBroadcastStream();

// Berechne nur Fibonaccis von geraden Werten
stream.where((v) => v % 2 == 0).listen((v) {
  print("Listener 1: fib($v) = ${fib(v)}");
});

// Verabeite alles
stream.listen((v) {
  print("Listener 2: fib($v) = ${fib(v)}");
});

// Verarbeite nur bis fib(7).
stream.takeWhile((v) => v <= 7).listen((v) {
  print("Listener 3: fib($v) = ${fib(v)}");
});
```


Streams filtern mittels `where()`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Nur Ereignisse die einer Bedingung genügen verarbeiten.

```dart
// Erzeugen eines Streams
var broadcast = new StreamController<int>();
var stream = broadcast.stream;

// Berechne nur Fibonaccis von geraden Werten
stream.where((v) => v % 2 == 0).listen((v) {
  print("Listener 1: fib($v) = ${fib(v)}");
});

// Den Stream befuellen
for (int i=1; i <= 10; i++) broadcast.add(i);
for (int i=10; i >= 1; i--) broadcast.add(i);
print("Main finished.");
```

Erzeugt folgende Konsolenausgabe:

```
Main finished.
Listener 1: fib(2) = 1
Listener 1: fib(4) = 3
Listener 1: fib(6) = 8
Listener 1: fib(8) = 21
Listener 1: fib(10) = 55
Listener 1: fib(10) = 55
Listener 1: fib(8) = 21
Listener 1: fib(6) = 8
Listener 1: fib(4) = 3
Listener 1: fib(2) = 1
```

Streams filtern mittels `takeWhile()`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Nur Ereignisse verarbeiten, bis eine Bedingung nicht mehr erfüllt ist.

```
// Erzeugen eines Streams
var broadcast = new StreamController<int>();
var stream = broadcast.stream;

// Verarbeite nur bis fib(7).
stream.takeWhile((v) => v <= 7).listen((v) {
  print("Listener 2: fib($v) = ${fib(v)}");
});

// Den Stream befuellen
for (int i=1; i <= 10; i++) broadcast.add(i);
for (int i=10; i >= 1; i--) broadcast.add(i);
print("Main finished.");
```

Erzeugt folgende Konsolenausgabe:

```
Main finished.
Listener 2: fib(1) = 1
Listener 2: fib(2) = 1
Listener 2: fib(3) = 2
Listener 2: fib(4) = 3
Listener 2: fib(5) = 5
Listener 2: fib(6) = 8
Listener 2: fib(7) = 13
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
25

Streams filtern mittels `skipWhile()`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Ereignisse nicht verarbeiten, solange eine Bedingung nicht erfüllt ist.

```
// Erzeugen eines Streams
var broadcast = new StreamController<int>();
var stream = broadcast.stream;

// Verarbeite ab fib(7).
stream.skipWhile((v) => v <= 7).listen((v) {
  print("Listener 3: fib($v) = ${fib(v)}");
});

// Den Stream befuellen
for (int i=1; i <= 10; i++) broadcast.add(i);
for (int i=10; i >= 1; i--) broadcast.add(i);
print("Main finished.");
```

Erzeugt folgende Konsolenausgabe:

```
Main finished.
Listener 3: fib(8) = 21
Listener 3: fib(9) = 34
Listener 3: fib(10) = 55
Listener 3: fib(10) = 55
Listener 3: fib(9) = 34
Listener 3: fib(8) = 21
Listener 3: fib(7) = 13
Listener 3: fib(6) = 8
Listener 3: fib(5) = 5
Listener 3: fib(4) = 3
Listener 3: fib(3) = 2
Listener 3: fib(2) = 1
Listener 3: fib(1) = 1
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
26

Streams filtern mittels `take()`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Nur die ersten n Ereignisse verarbeiten.

```
// Erzeugen eines Streams
var broadcast = new StreamController<int>();
var stream = broadcast.stream;

// Verarbeite nur die ersten drei Elemente
// des Streams
stream.take(3).listen((v) {
  print("Listener 4: fib($v) = ${fib(v)}");
});

// Den Stream befuellen
for (int i=1; i <= 10; i++) broadcast.add(i);
for (int i=10; i >= 1; i--) broadcast.add(i);
print("Main finished.");
```

Erzeugt folgende Konsolenausgabe:

```
Main finished.
Listener 4: fib(1) = 1
Listener 4: fib(2) = 1
Listener 4: fib(3) = 2
```

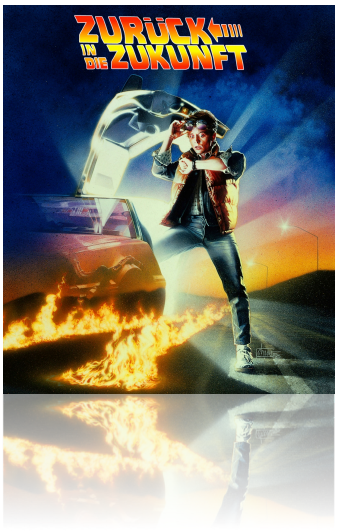

Futures

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

*„Eine **Future** […] bezeichnet in der Programmierung einen Platzhalter […] für ein Ergebnis, das noch nicht bekannt ist, meist weil seine Berechnung noch nicht abgeschlossen ist.*

*Eine Future ist meist das Ergebnis eines **asynchronen** Aufrufs einer Funktion oder einer Methode und kann verwendet werden, um auf das Ergebnis zuzugreifen, sobald es verfügbar ist. […] Das Konzept der Futures wurde 1977 […] von Henry G. Baker und Carl Hewitt vorgestellt.“*

Quelle: Wikipedia, Future (Programmierung)



Futures

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gezeigtes Beispiel demonstriert die Wirkungsweise von Futures. Die Bearbeitung der Fibonacci Methode erfolgt asynchron zur Anweisungssequenz (t1, t2, t3). Mit Aufruf der asynchronen Methode `fibAsync()` wird die Berechnung bereits angestoßen. Mit einem zeitlichen Verzug wird das Berechnungsergebnis mittels `then()` Methode einem Callback zur Verfügung gestellt (t2). Das Ende des Programms (t3) erfolgt aber vor Beendigung des Futurecallbacks (t2).

```
import "dart:async";
// Eine klassische sequentielle Methode.
num fib(num n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib(n-1) + fib(n-2);
}
// Man kann Completer und Streams nutzen,
// um Methoden asynchron auszufuehren.
Future<int> fibAsync(int n) {
  final completer = new Completer();
  new StreamController()..stream.listen(
    (n) => completer.complete(fib(n))
  )..add(n);
  return completer.future;
}
```

```
  final t1 = new DateTime.now();
  print("Zeit t1 vor fibAsync(): $t1");
  fibAsync(45).then((r) {
    final t2 = new DateTime.now();
    print("Zeit t2 nach fibAsync():
$t2");
    print("fib(45) = $r");
  });
  final t3 = new DateTime.now();
  print("Ende t3 um $t3");
```

Konsolenausgabe (vereinfacht):

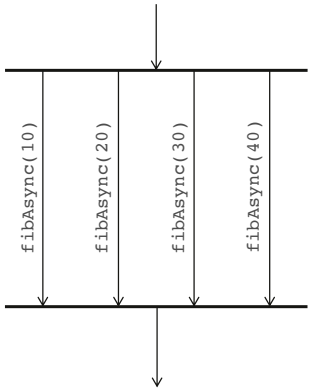
```
Zeit t1 vor fibAsync(): 14:30:33.590
Ende t3 um 14:30:33.786
Zeit t2 nach fibAsync(): 14:30:52.686
fib(45) = 1134903170
```


Waiting for Futures

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die Berechnung von Futures erfolgt also asynchron zur sequentiellen Abfolge von Anweisungen. Dies ist kein Problem, solange zwischen Beendigung von Futures und weiteren Schritten keine Abhängigkeit besteht. Dies ist aber nicht immer der Fall. So kann man sich bspw. vorstellen, dass vier Fibonacci Zahlen berechnet werden sollen, um die Summe dieser vier Zahlen zu bestimmen. Die Summe kann aber erst bestimmt werden, wenn die vier Fibonaccizahlen vorliegen. Somit muss die Summenbildung auf die Beendigung der vier Futures warten. Hierzu gibt es die **`wait()`** Methode.

```
final f10 = fibAsync(10);
final f20 = fibAsync(20);
final f30 = fibAsync(30);
final f40 = fibAsync(40);

Future.wait([f10, f20, f30, f40]).then((r) {
  // r enthaelt die Berechnungsergebnisse
  // der Futures auf die gewartet werden soll.
  final sum = r.reduce((a, b) => a + b);
  print("$sum");
});
```



await / async seit Dart 1.9 (Teil I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Seit Dart 1.9 sind die Schlüsselworte **async** und **await** hinzugekommen. Diese vereinfachen es Methoden mit langer Laufzeit asynchron zu definieren und auf Methodenergebnisse einzelner asynchroner Methoden zu warten.

```dart
// Eine klassische synchrone Methode.
num fib(num n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib(n-1) + fib(n-2);
}
```

```dart
// Asynchrone Variante (async/await)
Future<num> fibAsync(num n) async {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return await fib(n-1) + await fib(n-2);
}
```

```dart
// Klassischer synchroner Aufruf
int n = fib(10);
```

```dart
// Klassischer Aufruf (vor 1.9)
fibAsync(10).then((result) {
  int n = result;
});
```

```dart
// Neuer Aufruf (mit await, seit Dart 1.9)
int n = await fib(10);
```


await / async seit Dart 1.9 (Teil II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Mit den Schlüsselworten **async** und **await** kann auch auf mehrere Futures gewartet werden. Hierzu kombiniert man einfach das bereits bekannte `Future.wait` mit `await`.

```dart
// Asynchrone Variante (async/await)
Future<num> fibAsync(num n) async {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return await fib(n-1) + await fib(n-2);
}
```

```dart
// Alles asynchrone Aufrufe!
final f1 = fibAsync(1);
final f2 = fibAsync(2);
final f4 = fibAsync(4);
final f8 = fibAsync(8);

// Warten auf deren Ergebnisse
List<num> results = await Future.wait([f1, f2, f4, f8]);
print("Summe: ${results.reduce((a, b) => a + b) }");
```

Exception Handling bei Futures (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Wir schreiben unsere `fib()` Methode nun etwas um. Und zwar derart, dass diese eine Exception erzeugt, wenn sie mit negativen Werten aufgerufen wird.

```
num fib(num n) {
  if (n < 0) throw new Exception("fib ist nicht für < 0 definiert");
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib(n-1) + fib(n-2);
}
```

```
Future<int> fibAsync(int n) {
  final completer = new Completer();
  new StreamController()..stream.listen((n) {
    try {
      completer.complete(fib(n));
    } catch (err, stacktrace) {
      completer.completeError(err,
stacktrace);
    }
  }
  )..add(n);
  return completer.future;
}
```

Unsere `fibAsync()` Methode wird dadurch etwas komplexer. Die Fibonacci Berechnung sollte nun unter Exception Kontrolle laufen.

Sollte eine Exception auftreten so ist diese über `completeError()` an den `Completer` weiterzugeben.

Die asynchrone Berechnung endet in diesem Fall mit einem Fehler und nicht mit einem Berechnungsergebnis.

Fehler dieser Art, können mit `catchError()` gefangen und vom Aufrufer behandelt werden.

Exception Handling bei Futures (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
fibAsync(-20).then((r) {
  print("fib(20) == $r");
  return r;
}).catchError((err) {
  // Hier ist die ausloesende Future klar. Wuerde es
  // keinen catchError geben, wuerde der Error propagiert
  print("$err");
  return null;
});
```

Fehler können nun direkt beim asynchronen Aufruf erkannt und behandelt werden.

Oder auch erst bei nachfolgenden Verarbeitungsschritten.

```
Future.wait([fibAsync(10), fibAsync(20), fibAsync(-10), fibAsync(30)])
  .then((r) {
    final sum = r.reduce((a, b) => a + b);
    print("$sum");
  })
  .catchError((err) {
    // Future Error werden also propagiert. Dafuer kann nicht mehr festgestellt,
    // welche Future der Ausloeser war.
    print("$err");
  });
```

Werden Futures parallel ausgeführt?

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
final start = new DateTime.now();
final f1 = fibAsync(45).then((r) {
  print("Finished f1"); return r;
});
final f2 = fibAsync(45).then((r) {
  print("Finished f2"); return r;
});
final f3 = fibAsync(45).then((r) {
  print("Finished f3"); return r;
});
final f4 = fibAsync(45).then((r) {
  print("Finished f4"); return r;
});

Future.wait([f1, f2, f3, f4]).then((r) {
  final sum = r.reduce((a, b) => a + b);
  final runTime = new DateTime.now().difference(start);
  print("Berechnungsdauer von 4 x fib(45): $runTime");
});
```

Unser kleines Experiment sieht wie folgt aus:

Wir starten 4 mal die asynchrone Berechnung von fib(45) und messen die `runTime`.

Jedesmal wenn fib(45) berechnet wurde, geben wir dies aus.

```
Finished f1
Finished f2
Finished f3
Finished f4
Berechnungsdauer von 4 x fib(45): 0:01:14.845000
```

Wir stellen fest, die Reihenfolge der Ausgabe f1, f2, f3, f4 bleibt erhalten (spricht gegen echte Parallelität).

Werden Futures parallel ausgeführt?

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
final start = new DateTime.now();
fibAsync(45).then((r) {
  final runTime = new DateTime.now().difference(start);
  print("Finished fib(45) in $runTime");
});
```

Bestimmen wir die Laufzeit nur einer Berechnung

```
Finished fib(45) in 0:00:18.858000
```

dauert diese etwa 18.85 Sekunden.

D.h. unser kleines Experiment zeigt. Die viermalige Berechnung von fib(45) dauert etwa viermal so lange wie die einmalige Berechnung von fib(45).

Die Ausgabe asynchron ausgeführter Berechnungen von fib(45) erhält die Reihenfolge.

Beide Effekte zusammen genommen, lassen die Vermutung zu, dass sich Futures einen Ausführungsthread teilen. Und genau dies ist der Fall in Dart. DartsFutures und Streams sind per se nicht Multithreaded ausgelegt. Nur mittels Futures und Streams lassen sich also Multiprozessorsysteme nicht optimal ausnutzen (d.h. Berechnungen tatsächlich auf mehrere Kernen zeitgleich berechnen).

Hierzu benötigen wir **Isolates!**

Isolates

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Der Unterschied zwischen einem Prozess und einem Thread ist, dass Threads gemeinsam auf Daten eines Prozesses zugreifen können.

Dies führt jedoch ggf. zu Synchronisierungsproblemen (vgl. Thread Safeness, Programmieren II aber auch Betriebssysteme).

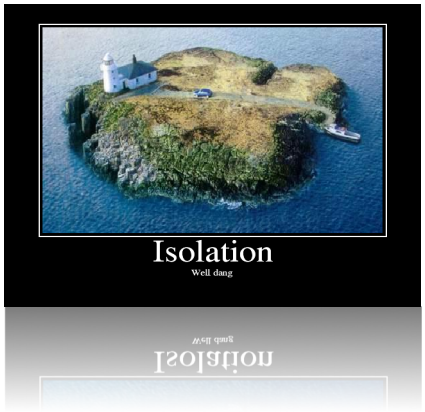
So betrachtet sind Threads heikel, weil sie konzeptionell schwer zu beherrschen sind. Der Mensch ist meist nicht gut darin, in echt parallelen statusbehafteten Abläufen zu denken.

Dart kennt daher Isolates. Isolates sind Threads (BS-Sicht), jedoch erlaubt die Dart VM keinen gemeinsamen Datenzugriff zwischen Threads (isolierter Status pro Thread, daher der Name).

Isolates können sich nur Nachrichten schicken, um miteinander zu kommunizieren.

Dies vermeidet konzeptionell viele Probleme, die Java EntwicklerInnen bedenken müssen, wenn sie bspw. das `synchronized` Schlüsselwort einsetzen.

Dart kennt kein `synchronized`. Dart kennt **Isolates**!



The Downfall of Imperative Programming

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

„If programmers were electricians, parallel programmers would be bomb disposal experts. Both cut wires [...]“

Bartosz Milewski, „The Downfall of Imperative Programming“

Quelle (letzter Zugriff am 22.12.2013):

https://www.fpcomplete.com/business/blog/the-downfall-of-imperative-programming/

Isolatekonzept in Dart

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Dart hat keine Shared-Memory Threads.

Dart Code läuft in Isolates, die miteinander mittels Nachrichten kommunizieren.

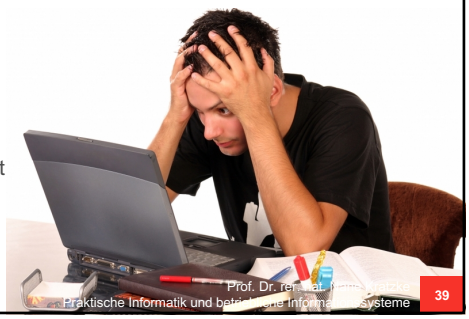
Nachrichten werden kopiert, bevor sie von anderen Isolates verarbeitet werden.

So wird sichergestellt, dass verschiedene Isolates nicht gegenseitig ihren Status manipulieren können.

Da jedes Isolate einen eigenen Status hat, benötigt man keine Locks, keine Mutuxes, etc.

Dies erleichtert konzeptionell Concurrent Programming.

Ja: Wir sind alle daran gewöhnt, statusbehaftet zu programmieren! Spätestens wenn zwei Threads im Spiel sind, sollten wir aber anfangen nervös zu werden.



Isolates (is like programming the hell)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die Programmierung von Isolates ist in Dart alles andere als ein Vergnügen.

Weder ist die Isolates API gut dokumentiert, noch ist ersichtlich, wieso das Spawning von Isolates derart kompliziert sein muss.

Wir werden daher Isolates mit einem Dart Package `worker` erzeugen.

```
http://pub.dartlang.org/packages/worker
```

Das kapselt die etwas „gewöhnungsbedürftige“ Isolate API von Dart und macht ziemlich genau das, was wir wollen (mehrere Kerne eines Systems ausnützen).

Hinweis:

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Mit der Version 1.9 haben sich Isolates nun etwas geändert. Man kann u.a. Top-Level Funktionen Isolates zur Bearbeitung zuweisen.

Zudem gibt es das Isolates Package was das Isolate Handling etwas vereinfachen soll.

`http://pub.dartlang.org/packages/isolate`

Da ich diese Features aber noch nicht ausprobiert habe, bleiben wir erst einmal beim Worker Package...

*Dank an **Jos Hirth** für diesen Hinweis ...*

Isolates (`worker` Package I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Das `worker` Package führt Tasks in sogenannten Workern aus. `Worker` werden durch das Package auf Isolates abgebildet.

```
abstract class Task<T> {
  T execute ();
}
```

Um eine durch `Worker` auszuführende `Task` definieren zu können, muss die abstrakte `execute()` Methode implementiert werden. Für unsere Fibonacci Methode könnte das wie folgt aussehen:

```
class Fibonacci extends Task<num> {
  final int _n;
  Fibonacci(this._n);
  num execute() => fib(_n);
}
```

Schaffen wir das? Jo, wir schaffen das!

Isolates (`worker` Package II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
// Eintrag in pubspec.yaml nicht
// vergessen, damit pub funktioniert
import "package:worker/worker.dart";

main() {
  // Wir sehen maximal 4 Isolates vor.
  final isolates = new Worker(poolSize: 4);

  // Wir instantiieren unser Problem.
  final fib45 = new Fibonacci(45);
  // Wir starten die Isolates und warten auf das
Ergebnis.
  isolates.handle(fib45).then((result) {
    print("Iso 1: fib(45) = $result");
  });

  [...]

  isolates.handle(fib45).then((result) {
    print("Iso 4: fib(45) = $result");
  });
}
```

Wir können dann isolates wie folgt mittels `handle()` starten (und auf Multicore Prozessoren echt parallel ablaufen lassen).

```
Iso 2: fib(45) = 1134903170
Iso 4: fib(45) = 1134903170
Iso 1: fib(45) = 1134903170
Iso 3: fib(45) = 1134903170
```

Gut zu erkennen, die Ausgabereihenfolge der Berechnungsergebnisse ist nun nicht mehr identisch mit den Vorkommen im Code.

Isolates (`worker` Package III)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import "package:worker/worker.dart";
import "dart:async";
import "dart:io";

main() {
  final start = new DateTime.now();
  final isolates = new Worker(poolSize: 4);
  final ends = new StreamController<DateTime>();

  final fib45 = new Fibonacci(45);

  Future.wait(
    [isolates.handle(fib45), isolates.handle(fib45),
     isolates.handle(fib45),isolates.handle(fib45),
     ]
  ).then((results) {
    final actual = new DateTime.now();
    final duration = actual.difference(start);
    print("Berechnungsdauer von 4 x fib(45): $duration");
    isolates.close();
  });
}
```

`handle()` liefert uns immer ein Future auf das zu erwartende Ergebnis des Isolates, dass einen Task bearbeitet.

Mittels `Future.wait()` können wir also auf mehrere echt parallel angestoßene Berechnungen warten, um bspw. Den Zeitpunkt zu messen, wenn alle Berechnungen beendet wurden.

```
Berechnungsdauer von 4 x fib(45):
0:00:22.776000
```

Im Gegensatz zu dem rein Future basierten Ansatz nutzen wir mit Isolates also die Multicore Kapazitäten aus.

- **Isolates**: ca. 22,7 Sekunden
- **Futures**: ca. 74,8 Sekunden
- Speedup von etwa 3,3 (gemessen auf einem 4 Core Rechner [8 Threads max. parallel])

More about Isolates

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`http://pub.dartlang.org/packages/worker`

Convienence Package zur komfortablen Nutzung von Isolates. Wird in dieser Präsentation genutzt, um die parallele Berechnung von Fibonacci Zahlen mittels Isolates zu realisieren. Das `worker` Package hat allerdings die Einschränkung, dass es momentan nur serverseitig einsetzbar ist (Stand 04.02.2014).

`https://gist.github.com/mitsuoka/3102095`

Der Fibonacci Test, mit reinen „Bordmitteln“ von Dart. Der Komplexitätsunterschied sollte deutlich werden.

Seth Ladd

Developer Advocate at Google

According to Seth Ladd:

„There‘s talk what to do“ with isolates.

An der Isolates API wird sich daher voraussichtlich (und hoffentlich) noch einiges ändern.

So far: Use the worker package.

Tour de Dart Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`dart:library`
Das Library System von Dart

`dart:async`
Asynchronous Programming

`dart:io`
Files, Directories

`dart:html`
Manipulating the DOM

`dart:servers`
HTTP Clients and Servers

`dart:convert`
Decoding and Encoding JSON, HTML and more

Dateien all at once lesen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:io'; import 'dart:convert';

void main() {
  final f = new File('hello_world.txt');

  // Man kann Dateien als String Future basiert auslesen
  f.readAsString(encoding: const Utf8Codec()).then((contents) {
    print(contents);
  });
  // oder auch synchron
  print(f.readAsStringSync(encoding: const Utf8Codec()));
  // Man kann Dateien auch in binary Form auslesen
  f.readAsBytes().then((List<int> bytes) {
    print("Länge der Datei ${f.path}: ${bytes.length} byte");
  });
  // Auch das geht natuerlich wieder synchron
  // Oder auch synchron
  List<int> bytes = f.readAsBytesSync();
  print("Länge der Datei ${f.path}: ${bytes.length} byte");
}
```

Dart:IO ermöglicht command line apps Dateien lesen und schreiben sowie Verzeichnisse auswerten zu können.

Dateien können dabei entweder

- synchron
- mittels Futures
- als String oder
- als binary

verarbeitet werden.

Hinweis: *Auf Random Access wird hier nicht eingegangen (weil man das selten benötigt).*

Dateien streambasiert lesen (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:io'; import 'dart:async'; import 'dart:convert';

main() {
  final f = new File('hello_world.txt');

  // Wir koennen Dateien auch streambasiert verarbeiten und werden dann
  // informiert, wenn etwas der Datei hinzugefuegt wurde.
  // Dies geht binary
   f.openRead().listen((List<int> data) => print(data.length));

  // Aber natuerlich auch (mit einem transform) String basiert
  var c = 1;
  f.openRead().transform(UTF8.decoder).listen((String data) {
    print("Chunk ${c++}: $data");
  });
}
```

Dateien streambasiert lesen (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:io';
import 'dart:async';
import 'dart:convert';

main() {
  final f = new File('hello_world.txt');
  // Dateien kann man mittels mehrerer transforms auch
  // streambasiert zeilenweise lesen
  var l = 1;
  f.openRead().transform(UTF8.decoder)       // Byte to String
              .transform(new LineSplitter()) // Zeilenweise
              .listen((String data) {
                print("Line ${l++}: $data");
              });
```


Dateien schreiben

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:io';
import 'dart:convert';

main() {
  final toWrite = ["Hallo,", "mein Name ist", "Hase!"];

  // Dateien lassen sich all at once String basiert schreiben
  final f = new File("/Users/Nane/Desktop/rabbit.txt");
  f.writeAsString(toWrite.join("\n"));

  // Und natuerlich auch haeppchenweise Stream basiert
  final g = new File("/Users/Nane/Desktop/rabbit-stream.txt");
  final IOSink out = g.openWrite(encoding: const Utf8Codec());
  for (var s in toWrite) out.write("$s\n");
  out.close();
}
```

Directories auslesen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:io';

main() {
  // Ein Directory Object bietet die Moeglichkeit
  // alle in ihm abgelegten Eintraege zu durchlaufen
  // Hier Stream basiert
  final home = new Directory("/Users/Nane/Sync/websites");
  home.list(recursive: true).listen((dirOrFile) {
    if (dirOrFile is Directory) print("${dirOrFile.path}");
  });

  // Das ganze geht auch synchron
  home.listSync(recursive: true).forEach((dirOrFile) {
    if (dirOrFile is Directory) print("${dirOrFile.path}");
  });
}
```


Weitere Informationen zu Datei IO

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Sie können natürlich auch Dateien kopieren, löschen, Vereichnisse anlegen, löschen etc.

https://api.dartlang.org/apidocs/channels/be/#dart-io

https://api.dartlang.org/apidocs/channels/be/#dart-io.FileSystemEntity

https://api.dartlang.org/apidocs/channels/be/#dart-io.File

https://api.dartlang.org/apidocs/channels/be/#dart-io.Directory

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Bestimmen Sie die Dateigröße aller ihrer Dateien im Home-Verzeichnis.

```
final String home = Platform.environment['HOME'];
```


Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Bestimmen Sie wie viele Verzeichnisse (inkl. Unterverzeichnisse, Unter-/Unterverzeichnisse, etc.) sich in ihrem Home-Verzeichnis befinden.

```
final home = Platform.environment['HOME'];
```

Tour de Dart Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`dart:library`
Das Library System von Dart

`dart:async`
Asynchronous Programming

`dart:io`
Files, Directories

`dart:html`
Manipulating the DOM

`dart:servers`
HTTP Clients and Servers

`dart:convert`
Decoding and Encoding JSON, HTML and more

Elemente im DOM selektieren (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

JavaScript Programme können (wenn im Browser ausgeführt) auf das aktuell dargestellte Dokument in seiner DOM-Form zugreifen. Dies ist in der JS Standard-API etwas umständlich, deswegen hat sich jQuery etabliert, dass es ermöglicht Elemente des DOM-Trees mittels (CSS) Selektoren zu selektieren.

Dart stellt hierzu die Library `dart:html` zur Verfügung, die jQuery vergleichbare Funktionalitäten anbietet.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Dom examples</title>
    <link rel="stylesheet" href="dom_examples.css">
  </head>
  <body>
    <h1>Dom examples</h1>
    <p class="intro">Hello world from Dart!</p>
    <p id="output">Here you will see examples.</p>
    <p>You could <em>click me</em> for example.</p>
    <script type="application/dart"
            src="dom_examples.dart"></script>
    <script src="packages/browser/dart.js"></script>
  </body>
</html>
```

Elemente im DOM selektieren (II)

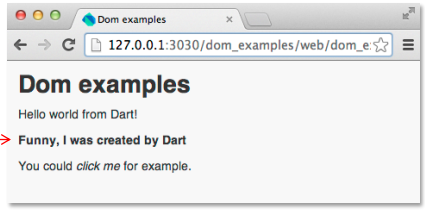
FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:html';

void main() {
  querySelector("#output").innerHtml = "<strong>Funny, I was created by Dart</strong>";
}
```

Oben stehendes Dart Programm selektiert das Element mit der Id `output` und ersetzt es mit clientseitig erzeugtem HTML.

Ohne clientseitige Verarbeitung

Mit clientseitiger Verarbeitung



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
57

Elemente im DOM selektieren (III)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

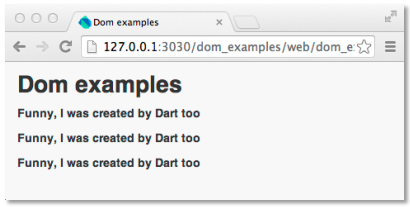
```
import 'dart:html';

void main() {
  querySelector("body p").innerHtml =
    "<strong>Funny, I was created by
     Dart</strong>";
}
```

```
void main() {
  querySelectorAll("p").forEach(
      (HtmlElement e) => e.innerHtml =
        "<strong>
         Funny, I was created by Dart too
         </strong>"
  );
}
```

Mittels `querySelector` können Sie ein Element selektieren (und anschließend auslesen/verarbeiten). Werden mehrere Elemente selektiert, wird nur das erste davon ausgewählt. `querySelectorAll` wählt hingegen immer alle Elemente aus (das entspricht dem $ in jQuery).

`querySelect`

`querySelectAll`



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
58

Elemente im DOM manipulieren (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

https://api.dartlang.org/apidocs/channels/stable/#dart-dom-html.HtmlElement

Selektoren selektieren Elemente des DOM-Tree und stellen sie im Datentyp `HtmlElement` zur Verfügung. `HtmlElement` stellt Attribute und Methoden zur Verfügung um ein Element des DOM zu ändern (hier ein Auszug der „wichtigsten“ Methoden/Attribute).

Methode/Attribute	
`void append{Html|Text}(String html)`	Fügt einem Element weiteren HTML/Text an.
`Element querySelector[All](String selector)`	Selektiert relativ zum Element Unterelemente gem. des angegebenen Selectors
`void remove()`	Entfernt das Element aus dem DOM-Tree
`Node replaceWith(Node other)`	Ersetzt das Element durch ein anderes Element
`Node clone(bool deep)`	Erzeugt eine (Tiefen-)Kopie eines Knotens.
`innerHtml`	Hiermit kann das innere HTML eines Elements gesetzt oder ausgelesen werden.
`attributes`	Liest/setzt die Attribute eines HTML Elements
`style`	Liest/setzt CSS Style Attribute eines HTML Elements (bspw. background oder padding/margin Eigenschaften)
`id`	Liest/setzt die Id eines HTML Elements (`<p id="me"></p>`)
`classes`	Liest/setzt die Klassen eines HTML Elements (`<p class="important error"></p>`)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
59

Elemente im DOM manipulieren (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```html
<h1>Dom examples</h1>
<p class="intro">Hello world from Dart!</p>
<p id="output">Here you will see examples.</p>
<p>You could <em>click me</em> for example.</p>
```

```css
.highlight {
  background: rgba(255, 0, 0, 0.25);
}
```

```
querySelector("p em").classes.add("highlight");
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
60

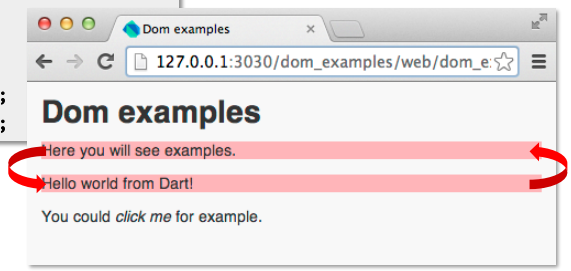
Elemente im DOM manipulieren (III)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
<h1>Dom examples</h1>
<p class="intro">Hello world from Dart!</p>
<p id="output">Here you will see examples.</p>
<p>You could <em>click me</em> for example.</p>
```

```
.highlight {
  background: rgba(255, 0, 0, 0.25);
}
```

```
final n1 = querySelector(".intro");
final n2 = querySelector("#output");

n1.classes.add("highlight");
n2.classes.add("highlight");

n1.replaceWith(n2.clone(true));
n2.replaceWith(n1.clone(true));
```



Auf Ereignisse des DOM reagieren

https://api.dartlang.org/apidocs/channels/stable/#dart-dom-html.HtmlElement

Elemente des DOM-Trees können mit sogenannten Event Handlern (Auszug aller möglichen, siehe rechts) versehen werden.

Hierzu wird ein Closure mit einem Event Stream verknüpft. Jedes mal, wenn nun ein Event durch den Browser ausgelöst wird, wird dieses an die registrierte Closure weitergereicht und kann dort verarbeitet werden.

EventStream

```
querySelectorAll("p").onMouseEnter.listen((MouseEvent e) {
  (e.target as HtmlElement).classes.toggle("highlight");
});
```

Event Handler (closure)

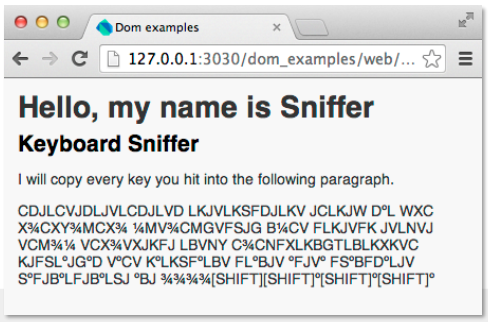
```
querySelectorAll("p").onMouseLeave.listen((MouseEvent e) {
  (e.target as HtmlElement).classes.toggle("highlight");
});
```

```
on
onAbort
onBeforeCopy
onBeforeCut
onBeforePaste
onBlur
onChange
onClick
onContextMenu
onCopy
onCut
onDoubleClick
onDrag
onDragEnd
onDragEnter
onDragLeave
onDragOver
onDragStart
onDrop
onError
onFocus
onFullscreenChange
onFullscreenError
onInput
onInvalid
onKeyDown
onKeyPress
onKeyUp
onLoad
onMouseDown
onMouseEnter
onMouseLeave
onMouseMove
onMouseOut
onMouseOver
onMouseUp
onMouseWheel
```

Beispiel: Keyboard Sniffer (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Ein Keyboard Sniffer, der alle Tastendrucke, die innerhalb des Browserfensters „mitschneidet“, lässt sich hiermit recht einfach entwickeln.



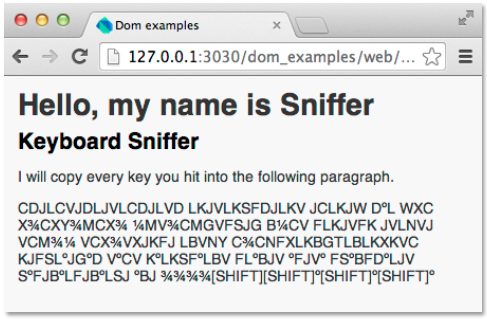

```
<h1>Hello, my name is Sniffer</h1>
<h2>Keyboard Sniffer</h2>

<p>I will copy every key you hit into the following paragraph.</p>

<p id="sniffer“></p>

<script type="application/dart" src="dom_examples_keyboard.dart"></script>
<script src="packages/browser/dart.js"></script>
```


Beispiel: Keyboard Sniffer (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Und so funktioniert er ... (ein Vierzeiler).

```
final e = querySelector("#sniffer");
window.onKeyDown.listen((KeyboardEvent ev) {
  e.appendText(new String.fromCharCode(ev.keyCode));
});
```



Wie Sie (gesniffte) Daten an einen anderen Server bekommen, dazu gleich mehr ...

Zusammenfassung
Die wichtigsten Methoden/Klassen aus `dart:html`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Mittels `querySelector` oder `querySelectorAll` können Sie `HtmlElement`e des DOM-Trees mittels CSS Selektoren selektieren und über folgende Methoden/Attribute und EventStreams auswerten/verändern..

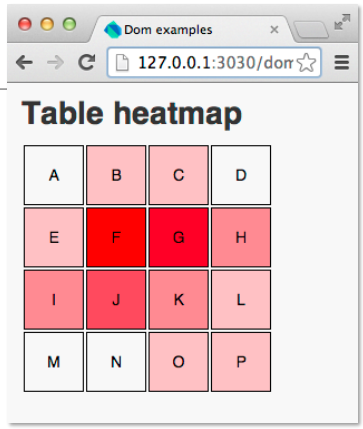
Methoden	Attribute	EventStreams
appendHtml()	innerHtml	onMouse...
appendText()	id	onKey...
remove()	classes	onChange
clone()	style	onClick
replaceWith()	attributes	

https://api.dartlang.org/apidocs/channels/stable/#dart-dom-html.HtmlElement

Miniübung:

Gegeben sei folgendes HTML Dokument:

```
<h1>Table heatmap</h1>

<table class="heatmap">
  <tr>
    <td>A</td><td>B</td><td>C</td><td>D</td>
  </tr>
  <tr>
    <td>E</td><td>F</td><td>G</td><td>H</td>
  </tr>
  <tr>
    <td>I</td><td>J</td><td>K</td><td>L</td>
  </tr>
  <tr>
    <td>M</td><td>N</td><td>O</td><td>P</td>
  </tr>
</table>
```



Verfolgen Sie bitte MouseOver Events derart, dass diejenigen Tabellenelemente einen umso roteren Hintergrund bekommen, je mehr MouseOver Events sie erhalten haben (desto häufiger die Maus darüber war).

Die Farbdarstellung soll relativ sein. Das Element mit den meisten MouseOver Events soll die Farbe rgba(255, 0, 0, 1.0) erhalten.

Miniübung:

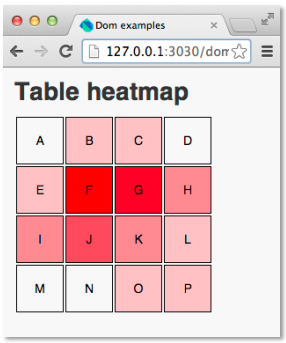


FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die Farbdarstellung soll relativ sein. Das Element mit den meisten MouseOver Events soll die Farbe rgba(255, 0, 0, 1.0) erhalten.

Alle anderen Tabellenelemente sollen eine dazu relative Transparenz bekommen (und dadurch heller erscheinen).

D.h. hat das Element E_{max} mit den meisten MouseOver Events z.B. 17 gezählte Events und ein anderes Element E_X drei gezählte MouseOver Events, so soll E_X die Farbe rgba(255, 0, 0, 0.176) erhalten (0.176 == 3/17).

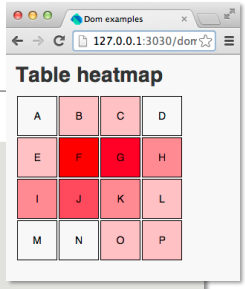


Sie können über die Properties `attributes` und `style` auf Attribute und Styles von `HtmlElement`en lesend und schreibend zugreifen.

```
HtmlElement elem;
[...]
final i = int.parse(elem.attributes['n']);
e.style.background = "rgba(255, 0, 0, 0.15)"
```


Miniübung:





Tour de Dart Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`dart:library`
Das Library System von Dart

`dart:async`
Asynchronous Programming

`dart:io`
Files, Directories

`dart:html`
Manipulating the DOM

`dart:servers`
HTTP Clients and Servers

`dart:convert`
Decoding and Encoding JSON, HTML and more

Ein sehr einfacher `HttpServer` mit `dart:io`

FACH HOCHSCHULE LÜBECK

```
import 'dart:io';

void main() {
  int i = 0;                                          // Requestzaehler (seit Serverstart)
  HttpServer.bind("0.0.0.0", 8080).then((server) {  // Erstellen eines HttpServers
    server.listen((HttpRequest request) {
      final answer =                                  // Antwort in Form eines HTML Dok.
          "<html>"
          "  <head><title>HTML Hello form Dart Server</title></head>"
          "  <body>"
          "    <h1>HTML Hello form Dart Server</h1>"
          "    <p>This is my ${++i} request.</p>"
          "  </body>"
          "</html>";
      request.response.headers.add('Content-Type', 'text/html; charset=UTF-8');
      request.response.writeln(answer);
      request.response.close();                       // Request bearbeitet
    });
  });
}
```

Ein sehr einfacher `HttpServer` mit `dart:io`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Da die Entwicklung serverbasierter Applikationen mit der Klasse `HttpServer` stellenweise nicht immer pragmatisch ist, bietet es sich an, auf Frameworks zurückzugreifen.

In der Ruby Welt gibt es dafür bspw.

- **Rails** (`http://rubyonrails.org/`)
- bzw. das leichtgewichtigere **Sinatra** (`http://http://www.sinatrarb.com/`).

Beides populäre Frameworks zum Entwickeln von serverbasierten Applikationen.

Ein an Sinatra angelehntes Framework in der Dart Welt ist **`start`**. Daher werden die folgenden serverbasierten Aspekte am Beispiel dieses Frameworks erläutert.

`https://pub.dartlang.org/packages/start`

Hinweis: *Mit* ***Rikulo Stream*** *steht ein mächtigeres aber auch komplexeres Framework zur Verfügung.*

`http://rikulo.org/projects/stream`

Ein statischer HTTP Server mit `start`

Ein Zweizeiler (den Rest erledigt `pub`)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

start 0.2.0

README.md | CHANGELOG.md | Installing | Versions

1. Depend on it

Add this to your package's pubspec.yaml file:

```
dependencies:
  start: ">=0.2.0 <0.3.0"
```

If your package is an application package you should use `any` as

2. Install it

If you're using the Dart Editor, choose:

```
Menu > Tools > Pub Install
```

Or if you want to install from the command line, run:

```
$ pub install
```

3. Import it

Now in your Dart code, you can use:

```
import 'package:start/start.dart';
```

```dart
import 'package:start/start.dart';

main() {
  start(port: 8080).then((Server app) {
    // Starte Server auf Port 8080
    app.static('../web');
    // Directory aus dem statische
    // Inhalte bereitgestellt werden
    // sollen
  });
}
```

Unser Eingangsbeispiel mit `start`

FACH HOCHSCHULE LÜBECK

```
import 'package:start/start.dart';

main() {
  start(port: 8080).then((Server app) {  // Starte Server auf Port 8080
    var i = 0;                           // Requestzaehler seit Serverstart
    app.get("/").listen((Request req) {  // Handler für alle get Requests auf /
      final answer =
          "<html>"
          "  <head><title>Hello form Start Server</title></head>"
          "  <body>"
          "    <h1>HTML Hello form Start Server</h1>"
          "    <p>This is my ${i++} request.</p>"
          "  </body>"
          "</html>";
      req.response.header('Content-Type', 'text/html; charset=UTF-8');
      req.response.send(answer);         // Antwort senden
    });
  });
}
```


Formulare auswerten mit `start`
Method: GET

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'package:start/start.dart';

main() {
 start(port: 8080).then((Server app) {
    app.get("/formular").listen((Request req) {

      // Parameter aus dem Request auslesen
      final name = req.param("first name");

      final answer =
          [...]
          "<h1>Hello $name, please insert another name:</h1>"
          "<form action='/formular' method='get'>"
          "  First name: <input type='text' name='first name'>"
          "  <input type='submit'>"
          "</form>"
          [...]
       req.response.header('Content-Type', 'text/html; charset=UTF-8');
       req.response.send(answer);
    });
  });
}
```

Formulare auswerten mit `start`
Method: POST (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Unsere App benötigt nun zwei Handler. Einen get Handler (für die erstmalige Darstellung des Formulars und einen post Handler für die eigentliche Verarbeitung der Formulardaten).

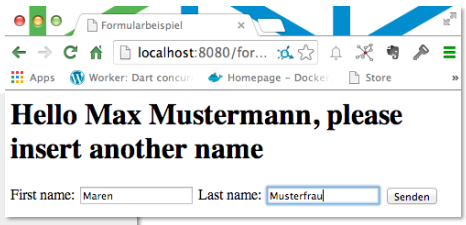
```
start(port: 8080).then((Server app) {
   app.post("/formular").listen((Request req) {    // Formulardaten per POST
     req.response.header('Content-Type', 'text/html; charset=UTF-8');
     req.payload().then((params) {                 // Parameter aus Req. Body holen
       final fn = params['firstname'];
       final ln = params['lastname'];
       req.response.send(renderForm("$fn $ln"));
     });
   });
   app.get("/formular").listen((Request req) {      // Formulardaten per GET
     req.response.header('Content-Type', 'text/html; charset=UTF-8');
     req.response.send(renderForm());
   });
});
```

Die Formulargenerierung benötigen wir nun an zwei Stellen. Gem. DRY lagern wir sie daher besser in eine eigene Funktion aus.

Formulare auswerten mit `start`
Method: POST (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die Formulargenerierung kann bspw. so gekapselt werden.




```
String renderForm([String name = ""]) {
  return
      "<html>"
      "<head><title>Formularbeispiel</title>"
      "<meta charset='UTF8'>"
      "<body>"
      "<h1>Hello $name, please insert another name</h1>"
      "<form action='/formular' method='post'>"
      "  First name: <input type='text' name='firstname'>"
      "  Last name: <input type='text' name='lastname'>"
      "  <input type='submit'>"
      "</form>"
      "</body>"
      "</html>";
}
```

Zustände clientseitig speichern und serverseitig auslesen mit Cookies

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
start(port: 8080).then((Server app) {
  app.get("/formular").listen((Request req) {
    [...]
    final cookies = new Map.fromIterable(                    // Alle gesendeten Cookies des
                                                             // Req. auslesen
        req.cookies.map((Cookie c) => [c.name, c.value]),
        key:   (kv) => kv[0]),
        value: (kv) => kv[1])
    );

    final prevFirstName = cookies['firstname'];              // Relevante Cookies auslesen
    final prevLastName = cookies['lastname'];

    [...]

                                                             // Cookies beim Client setzen
    req.response.cookie('firstname', '$fname');
    req.response.cookie('lastname', '$lname');

    [...]
  });
});
```


Zustände serverseitig speichern und auslesen mit Sessions (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Sessions können in Dart mittels `HttpSession` Objekten gespeichert und verwaltet werden (letztlich `Maps`). Diese können auf folgende Art aus ***Start*** heraus ausgelesen und gesetzt werden.

```
start(port: 3000).then((Server server) {
  server.get("/form").listen((Request req) {

    final actualFirstName     = req.param("firstname");          // GET Parameter auslesen
    final actualLastName      = req.param("lastname");

    final previousFirstName   = req.session["first name"];       // Session data auslesen
    final previousLastName    = req.session["last name"];

    req.session["first name"] = actualFirstName;                 // Session data setzen
    req.session["last name"]  = actualLastName;

    req.response.header('Content-Type', 'text/html; charset=UTF-8');
    req.response.send(
        form(actualFirstName, actualLastName, previousFirstName, previousLastName)
    );
  });
});
```

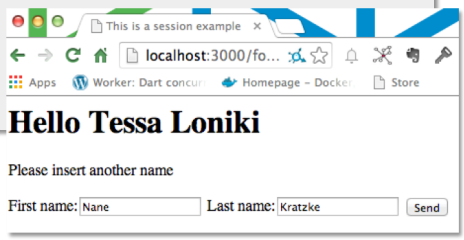
https://api.dartlang.org/apidocs/channels/stable/#dart-io.HttpSession

Zustände serverseitig speichern und auslesen mit Sessions (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die Implementierung der `form()` Methode.

```
String form([String fn = "", String ln = "", String pfn = "", String pln = ""]) {
  return
      "<html>"
      "<head><title>This is a session example</title></head>"
      "<body>"
      "<h1>Hello $fn $ln</h1>"
      "<p>Please insert another name</p>"
      "<form action='/form' method='get'>"
      " <label>First name:</label><input type='text' name='firstname' value='$pfn'>"
      " <label>Last name:</label><input type='text' name='lastname' value='$pln'>"
      " <input type='submit' value='Send'>"
      "</form>"
      "</body>"
      "</html>";
}
```



Zusammenfassung

Die wesentlichen Klassen und ihre Methoden zur serverseitigen Programmierung mit *Start*

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Server

```
listen(host, port) // start the server (it's performed by the start function)
stop() // stops the server
get|post|put|delete(String route) // adds a handler, returns a Stream<Request>
ws(String route) // adds WebSocket handler, returns a Stream
static(String path) // serves static files from `path`
```

Request

```
header(String name) // get header
accepts(String type) // check accept header
input // raw HttpRequest stream
path // requested URI path
uri // requested URI
cookies // provided cookies
param(String name) // get query param by name
payload(Encoding enc) // get a promise with a Map of request body params
```

Response

```
header(String name, [value]) // get or set header
get(String name) // get header
set(String name) // set header
type(contentType) // set Content-Type
cache(String cacheType) // set Cache-Control
status(code) // sets response status code
cookie(name, val) // sets cookie
add(string) // add a string to response
close() // closes response
send(string) // sends string and closes response
json(Map data) // stringifies map and sends it
jsonp(String name, Map data) // stringifies map and sends it in callback as `name(data)`
render(viewName, [Map params]) // renders server view
```

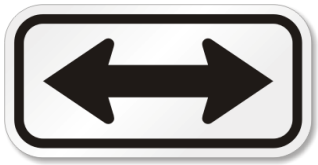
Details: `https://github.com/lvivski/start`

Web Sockets

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

*„Das WebSocket-Protokoll ist ein auf TCP basierendes Netzwerkprotokoll, das entworfen wurde, um eine **bidirektionale** Verbindung zwischen einer Webanwendung und einem WebSocket-Server bzw. einem Web-Server, der auch WebSockets unterstützt, herzustellen. [...]*

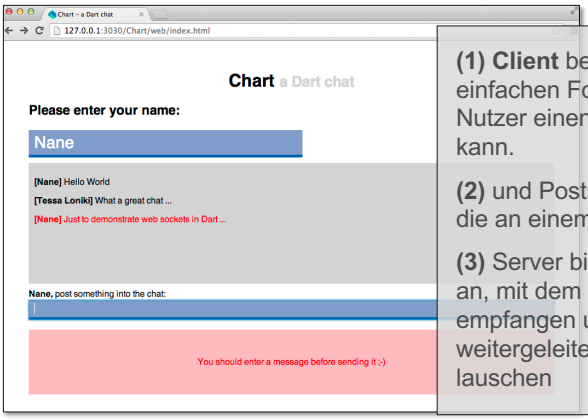
Zudem entfallen bei WebSockets die durch den HTTP-Header verursachten zusätzlichen Daten, die bei jeder Anfrage einige Hundert Bytes umfassen können.“



Wikipedia:
`https://de.wikipedia.org/wiki/WebSocket`

Beispiel: Ein einfacher Chat

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



(1) Client besteht aus einem einfachen Formular, mit dem sich ein Nutzer einen Chatnamen wählen kann.

(2) und Posts an alle senden kann, die an einem Websocket lauschen.

(3) Server bietet einen **Websocket** an, mit dem Messages von Chatusern empfangen und alle Clients weitergeleitet werden, die dem Chat lauschen

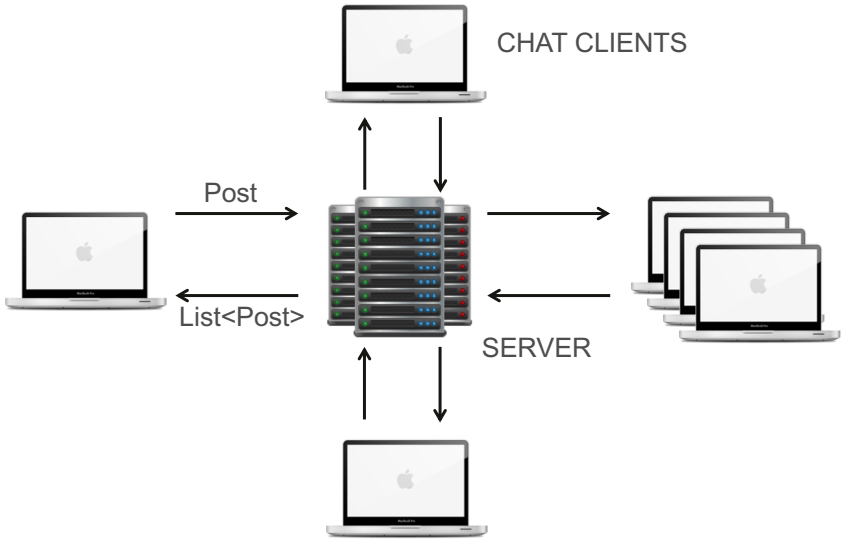
Beispiel zu finden unter:

`https://github.com/nkratzke/dartchat`

`http://pub.dartlang.org/packages/dartchat`

Beispiel: Ein einfacher Chat auf Basis von WebSockets

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



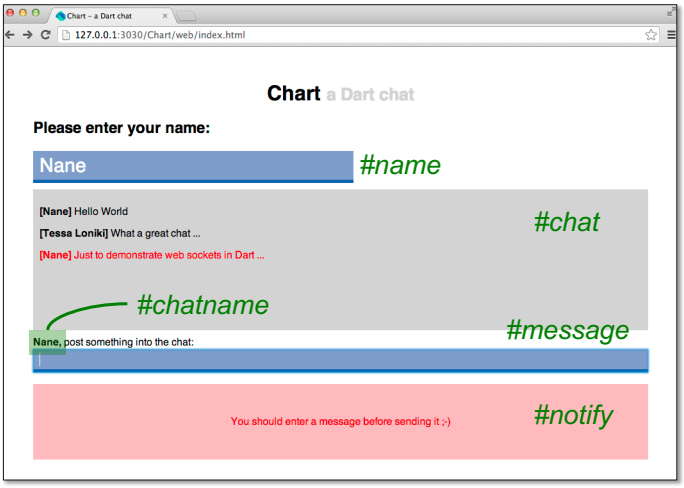
Beispiel: Ein einfacher Chat Hier: Server

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'package:start/start.dart'; import 'dart:io';

main() {
  start(port: 3000).then((Server app) {
    // Alle Verbindungen in einer Liste verwalten ...
    final sockets = <WebSocket>[];
    // Wir nutzen start, um statischen Content (z.B. unseren
    // Client, ist ja nur ein HTML Dokument) auszuliefern
    app.static("build/web/");
    // Hier nehmen wir eingehende Socketanfragen an
    app.get("/messages").listen((Request req) {
      // wandeln sie in einen WebSocket
      WebSocketTransformer.upgrade(req.input).then((ws) {
        sockets.add(ws); // speichern sie uns,
        // und leiten eingehende Messages an alle anderen
        // WebSockets weiter.
        ws.listen((msg) => sockets.forEach((ws) => ws.add(msg)));
      });
    });
  });
}
```

Beispiel: Ein einfacher Chat
Hier: Client

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Beispiel: Ein einfacher Chat
Hier: Client (HTML Definition)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
<body>
  <div class="container">
    <h1>
      Chart <small>a Dart chat</small>
    </h1>
    <h2>Please enter your name:</h2>
    <input id='name' type='text'>
    <div id="chat"></div>
    <label><span id='chatname'></span> post something into the chat: </label>
    <input id='message' type='text'>
    <div id='notify'></div>
  </div>
  <script type='application/dart' src='chartclient.dart'></script>
  <script src='js/dart.js'></script>
</body>
```

Beispiel: Ein einfacher Chat
Hier: Client (Dart Logik, Elemente)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:html';
import 'package:chart/chart.dart';

main() {
  // Die Interaktions-Elemente mit dem DOM-Tree
  final HtmlElement  chatname = querySelector("#chatname");
  final InputElement input    = querySelector("#message");
  final InputElement name     = querySelector("#name");
  final DivElement   chat     = querySelector("#chat");
  final DivElement   notify   = querySelector("#notify");

  // Wir bauen uns einen Socket auf.
  final WebSocket chatSocket = new WebSocket(
    "ws://${window.location.host}/messages"
  );

  // Hier speichern wir uns den Namen des/der Chatters/in
  var chatter = "";

  [...]
}
```

Beispiel: Ein einfacher Chat
Hier: Client (Dart Logik, Event Handler I)

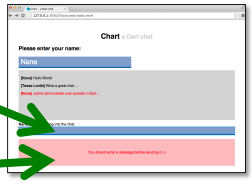
FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'dart:html';
import 'package:chart/chart.dart';

main() {
  [...]
  // Wann immer etwas in das Namensfeld eingegeben wird
  name.onInput.listen((_) {
    chatter = name.value;
    chatname.innerHtml = chatter;
  });

  [...]

  // Wann immer auf unserem Socket eine Nachricht eingeht
  chatSocket.onMessage.listen((msg) {
    final message = new ChartMessage.fromJSON(msg.data);
    chat.appendHtml("<p>${message.html}</p>");
    chat.scrollByLines(1000);
  });
}
```

Beispiel: Ein einfacher Chat
Hier: Client (Dart Logik, Event Handler II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

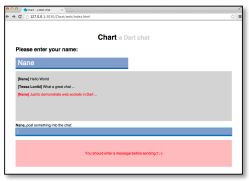
```
main() {
 // Wann immer etwas in #message eingegeben wird
  input.onKeyPress.listen((KeyboardEvent ev) {
    if (new KeyEvent.wrap(ev).keyCode != KeyCode.ENTER) return;
    if (chatter == "") {
      notify.innerHtml = error("Select a name ...");
      name.focus();
      return;
    }
    if (input.value == "") {
      notify.innerHtml = error("No message ...");
      return;
    }
    if (chatSocket.readyState != WebSocket.OPEN) {
      notify.innerHtml = error("Sorry, no connection");
      return;
    }
    final msg = new ChartMessage(chatter, input.value);
    input.value = "";
    chatSocket.send(msg.json);
    notify.innerHtml = "";
  });
}
```



Beispiel: Ein einfacher Chat
Hier: Client (Dart Logik, „Datenmodel“)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

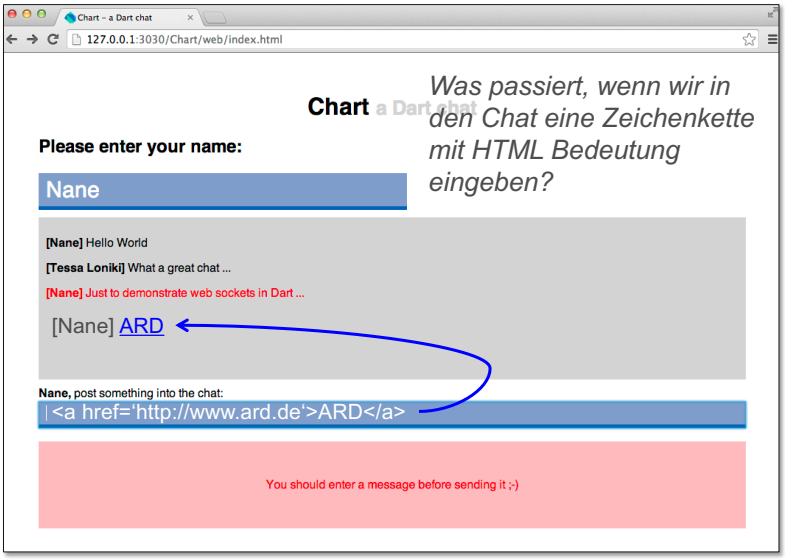
```
library chart;

import 'dart:convert';

class ChartMessage {
  String _from;
  String _msg;

  ChartMessage.fromJSON(msgAsJSON) {
    final map = JSON.decode(msgAsJSON);
    _from = map['from'];
    _msg = map['msg'];
  }

  ChartMessage(this._from, this._msg);

  String get json => '{ "from" : "$_from", "msg" : "$_msg" }';

  String get html => "<strong>[$_from]</strong> $_msg";
}
```



Eine **ChartMessage** definiert eine Chatnachricht (Zeile im Chatfenster).

Do you know: Cross Site Scripting?

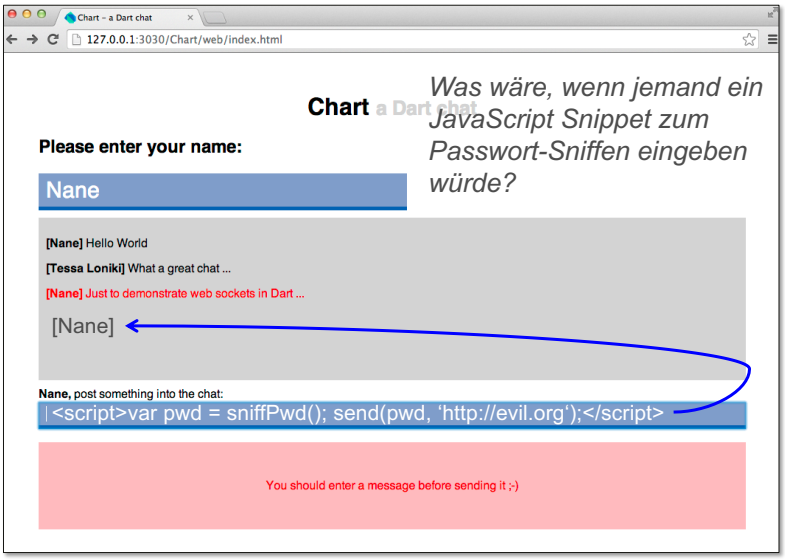
FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Was passiert, wenn wir in den Chat eine Zeichenkette mit HTML Bedeutung eingeben?

Do you know: Cross Site Scripting?

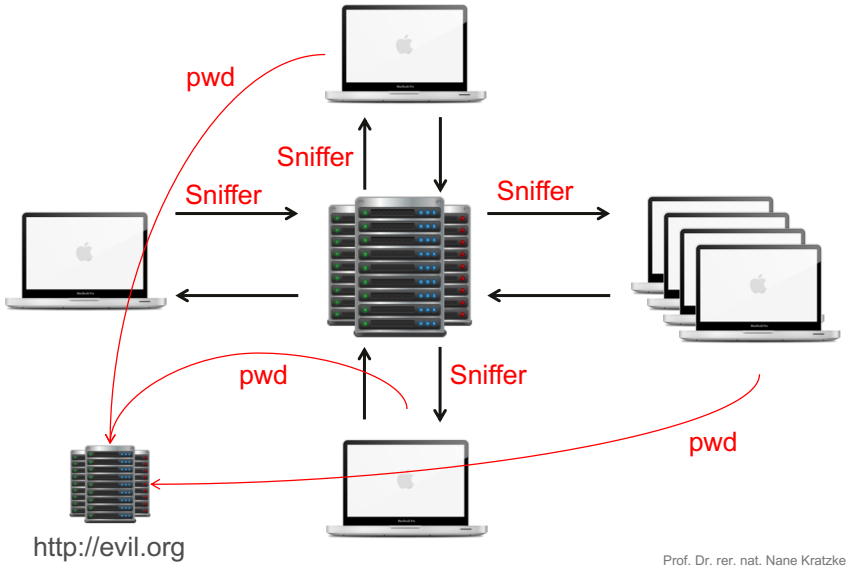
FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Was wäre, wenn jemand ein JavaScript Snippet zum Passwort-Sniffen eingeben würde?

Beispiel: Ein einfacher XSS Angriff

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Dieser Effekt wird auch Cross Site Scripting (XSS) genannt

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Es lässt sich so über Formulare
  - beliebiger Script Code (z.B. JavaScript Code)
    - Adressbuch durchsuchen?
    - Tastatureingaben protokollieren (um Ihre Passwörter auszuspähen)
  - beliebiger HTML Code
    - z.B. Links auf Phishing Seiten, um Sie unbemerkt von Ihrer Online Banking Site auf eine Seite umzulenken, die nur so aussieht, wie die der vertrauenswürdigen Deutschen Bank.
    - z.B. weitere Formulare zur Abfrage Ihrer Kontodaten mit action Handlern auf ganz anderen Servern
- einschleusen (per Chat an andere weiterverteilen).
- Wird eine Formular-Eingabe gar in einer Datenbank gespeichert, ist sie sogar persistent und kann durch andere mit zeitlichem Abstand aufgerufen werden (z.B. in Forenbeiträgen)

HTML Injection

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Es gilt also zu vermeiden, dass ein Chat Nutzer Texte mit HTML Bedeutung in das System „injizieren“ kann.

HTML Injection
Wo liegt hier die Problemstelle?

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
library chart;

import 'dart:convert';

class ChartMessage {
  String _from;
  String _msg;

  ChartMessage.fromJSON(msgAsJSON) {
    final map = JSON.decode(msgAsJSON);
    _from = map['from'];
    _msg = map['msg'];
  }

  ChartMessage(this._from, this._msg);

  String get json => '{ "from" : "$_from", "msg" : "$_msg" }';

  String get html => "<strong>[$_from]</strong>: $_msg";
}
```

Diese Stellen sind problematisch, da hier HTML erzeugt wird, dass auf nicht geprüften Nutzereingaben beruht.

Messages kommen von den Nutzern! Und Nutzer sind böse!

Arrrrgh ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Wie bekommen wir denn so etwas in den Griff?

Dazu gibt es `dart:convert` ...

Tour de Dart Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

`dart:library`
Das Library System von Dart

`dart:async`
Asynchronous Programming

`dart:io`
Files, Directories

`dart:html`
Manipulating the DOM

`dart:servers`
HTTP Clients and Servers

`dart:convert`
Decoding and Encoding HTML, JSON and more

Encoding und Decoding

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Escapen von **HTML** (Cross Site Scripting)
- Encoding und Decoding von **JSON**
- **Base64** Encoding und Decoding (inkl. **UTF8** Encoding und Decoding)
- Hashsummen (**SHA256**)

HtmlEscape

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```dart
import 'dart:convert';

/**
 * Methode zum Escapen von Zeichen mit HTML-Bedeutung.
 * Liefert eine Zeichenkette die in einem HTML DOM-Tree
 * nur noch eine Bedeutung als Textnode haben kann.
 */
String escape(String unsafe) {
  final sanitize = const HtmlEscape();
  return sanitize.convert(unsafe);
}
```

```dart
// Ein normaler String
final normal = "Ein normaler String.";

// Ein String mit HTML Bedeutung (Tags.)
final html = "Ein <em>normaler</em> String.";

assert(escape(normal) == normal);
assert(escape(html) == "Ein &lt;em&gt;normaler&lt;&#x2F;em&gt;
String.");
```

Mittels der Funktion `escape()` können sie also Eingaben von einem Nutzer (nie als vertrauens-würdig zu behandeln) so umwandeln, dass sie nur noch eine Textbedeutung, aber keine HTML Bedeutung haben.

Dadurch unterbinden sie Cross Site Scripting Attacken (da auch JavaScript dadurch seine Bedeutung verliert).

Ein sicherer Chat

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Das Problem in unserer clientseitigen Chatimplementierung war folgender Eventhandler. Dieser führt dazu, dass Posts von Nutzern ungefiltert in den DOM-Tree eingebaut werden. Posts mit HTML Bedeutung werden so ggf. umgesetzt (JavaScript evtl. ausgeführt!).


```
// Wann immer auf unserem Socket eine Nachricht eingeht
chatSocket.onMessage.listen((msg) {
  final message = new ChartMessage.fromJSON(msg.data);
  chat.appendHtml("<p>${message.html}</p>");
  chat.scrollByLines(1000);
});
```

Wir können das Problem bspw. In der `ChartMessage` Klasse lösen und Posts mit ggf. vorhandenem „Schadcode“ *entschärfen*.

```
String get html => "<strong>[$_from]</strong> $_msg";
```

indem wir Nutzereingaben `_from` und `_msg` einer Message bevor wir sie in HTML wandeln mittels `escape()` entschärfen.

```
String get html => "<strong>[${escape(_from)}]</strong> ${escape(_msg)}";
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
101

Security First (be Dr. House)
Paranoia ist nicht immer schlecht ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Frei nach Dr. House:**
  - Alle Nutzer sind böse, lügen, betrügen und wollen Deinem Server etwas Böses.
  - Traue keiner Nutzereingabe,
  - die nicht auf dem eigenen Server geprüft, gefiltert
  - und für unschädlich befunden wurde
  - bzw. unschädlich gemacht wurde.

Hinweis: *Dart nutzt zwar clientseitig `sanitize` automatisch bei allen Methoden, die im DOM Tree etwas ändern,* ***es sei denn*** *sie wollen explizit HTML hinzufügen (bspw. mit `appendHtml()`), dann sind sie selber verantwortlich für das escapen.*

Serverseitig müssen sie jedoch immer escapen, wenn sie HTML erzeugen und bspw. Daten die von Nutzern eingegeben wurden aus ***Cookies***, ***Sessions*** *oder* ***Queryparametern*** *auslesen!*

Ansonsten bauen Sie XSS-Sicherheitslücken!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
102

JSON Notation

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die **JavaScript Object Notation**, kurz JSON, ist ein kompaktes Datenformat in für Mensch und Maschine einfach lesbarer Textform zum Zweck des Datenaustauschs zwischen Anwendungen.

JSON kennt

- Listen und
- Objekte (Mappings) sowie die primitiven Datentypen
- null (Nullwert),
- ganzzahlige Werte und Fließkommawerte (in der üblichen Notation)
- und Zeichenketten.

JSON ist unabhängig von der Programmiersprache JavaScript. Parser existieren in praktisch allen verbreiteten Sprachen (z.B. für Dart ;-).

Da dies so ist, ist es mittlerweile ein beliebtes Format auch komplex strukturierte Daten zwischen Web Information Systems auszutauschen. Im Gegensatz zu XML (was für einen vergleichbaren Zweck konzipiert wurde) ist es aber kürzer, effizienter und übersichtlicher als XML (allerdings nicht so mächtig).

Da es aber die zwei wesentlichen Datenstrukturen der Informatik (Listen und Mappings) unterstützt, ist es für die meisten Anwendungsfälle ausreichend und zumeist hinsichtlich der Komplexität besser als XML beherrschbar. Zudem bleibt es für den Menschen lesbar und verbraucht weniger Datenvolumen bei Datenübertragungen als XML.

JSON Notation in a Nutshell

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

In Anlehnung an Wikipedia:
http://de.wikipedia.org/wiki/JavaScript_Object_Notation

Datentyp	Literal	Erläuterung
Nullwert	`null`	Wird durch das Schlüsselwort `null` dargestellt.
Boolescher Wert	`true`, `false`	Wird durch die Schlüsselwörter `true` und `false` dargestellt. Dies sind keine Zeichenketten. Sie werden daher, wie null, nicht in Anführungszeichen gesetzt.
Zahl	`0–9`	Ist eine Folge der Ziffern 0–9. Diese Folge kann durch ein negatives Vorzeichen - eingeleitet und einen Dezimalpunkt . unterbrochen sein. Die Zahl kann durch die Angabe eines Exponenten e oder E ergänzt werden, dem ein Vorzeichen + oder - und eine Folge der Ziffern 0–9 folgt.
Zeichenkette	`“ “`	Beginnt und endet mit doppelten geraden Anführungszeichen ("). Sie kann Unicode-Zeichen und Escape-Sequenzen enthalten.
Liste	`[ ]`	Beginnt mit [und endet mit]. Es enthält eine durch Komma getrennte, geordnete Liste von Werten, gleichen oder verschiedenen Typs. Leere Listen sind zulässig.
Objekt (Mapping)	`{ }`	Beginnt mit { und endet mit }. Es enthält eine durch Komma getrennte, ungeordnete Liste von Eigenschaften. Objekte ohne Eigenschaften ("leere Objekte") sind zulässig.
Eigenschaft	`key:val`	Besteht aus einem Schlüssel und einem Wert, getrennt durch einen Doppelpunkt (key:val). Die Schlüssel aller Eigenschaften in einem Objekt müssen eindeutig, also paarweise verschieden sein. • Der Schlüssel (`key`) ist eine Zeichenkette. • Der Wert (`val`) ist ein Objekt, eine Liste, eine Zeichenkette, eine Zahl oder einer der Literale `true`, `false` oder `null`.

JSON.decode, JSON.encode

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
// Dies ist einfach nur ein String
final json =
  '['
    '{"from":"Nane","msg":"Hello World"},'
    '{"from":"Tessa","msg":"Hello again"},'
    '{"from":"Nane","msg":"Nice to meet you"},'
    '{"from":"Tessa","msg":"It is a pleasure"},'
    '{"from":"Nane","msg":"You are welcome"}'
  ']';

// Dies ist eine Dart Datenstruktur
final dart =
   [
     { "from": "Nane", "msg" : "Hello World" },
     { "from": "Tessa", "msg" : "Hello again" },
     { "from": "Nane", "msg" : "Nice to meet you" },
     { "from": "Tessa", "msg" : "It is a pleasure" },
     { "from": "Nane", "msg" : "You are welcome" }
   ];
```

JSONSTRING => Dart Datastructure

Mittels **`JSON.decode()`** können wir eine valide JSON Zeichenkette in eine aus Mappings und Listen basierende Dart-Datenstruktur wandeln.

```
assert("$dart" == "${JSON.decode(json)}");
```

Dart Datastructure => JSONSTRING

Mittels **`JSON.decode()`** können wir eine aus Mappings und Listen basierende Dart-Datenstruktur in eine valide JSON Zeichenkette wandeln.

```
assert(json == JSON.encode(dart));
```


Base64

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

*„**Base64** beschreibt ein Verfahren zur Kodierung von 8-Bit-Binärdaten (z. B. ausführbare Programme, ZIP-Dateien oder Bilder) in eine Zeichenfolge, die nur aus lesbaren, Codepage-unabhängigen ASCII-Zeichen besteht.“*

Quelle: Wikipedia, https://de.wikipedia.org/wiki/Base64

Da im Internetumfeld häufig rein textbasierte (7-Bit ASCII) Protokolle existieren (z.B. EMail) kann Base64 dazu genutzt werden, Binärdaten textbasiert zu übertragen.

Nachteil: Der Datenverbrauch steigt dabei um etwa ein Drittel.

CryptoUtils.bytesToBase64(), UTF8.encode()

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Aus Gründen der Einfachheit zeigen wir die Umwandlung in die Base64 Codierung am Beispiel eine Zeichenkette (auch wenn Base64 für Binärdaten) gedacht ist. Hierzu müssen wir eine Zeichenkette erst in bytes und dann in das Base64 Format umwandeln.

```
import 'dart:convert';
import 'package:crypto/crypto.dart';

main() {
  final bytes = UTF8.encode("Hello World");           // String to bytes
  final base64 = CryptoUtils.bytesToBase64(bytes);
  print("Hello World => $base64");
}
```

Dies ergibt auf der Konsole:

```
Hello World => SGVsbG8gV29ybGQ=
```


CryptoUtils.base64StringToBytes(), UTF8.decode

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Auch der umgekehrte Weg geht natürlich.

```
import 'dart:convert';
import 'package:crypto/crypto.dart';

main() {
  final BASE64 = "SGVsbG8gV29ybGQ=";
  final BYTES = CryptoUtils.base64StringToBytes("SGVsbG8gV29ybGQ=");
  final helloWorld = UTF8.decode(BYTES);          // bytes to String
  print("$BASE64 => $helloWorld");
}
```

Dies ergibt auf der Konsole:

```
SGVsbG8gV29ybGQ= => Hello World
```

Hashsummen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Eine kryptografische Hashfunktion ist eine Funktion, die eine Input-Zeichenfolge beliebiger Länge auf eine Output-Zeichenfolge mit fester Länge abbildet. **Es ist sehr aufwändig von der Output-Zeichenfolge auf die Input-Zeichenfolge zurückzurechnen (idealerweise nur per Brute Force).**

Dadurch eignen sich Hashfunktionen sehr gut, um „Fingerabdrücke“ von Texten zu bestimmen. Wird ein Text nur in einem Zeichen geändert, hat es sofort einen anderen „Fingerabdruck“ (eine andere Hashsumme). Mehr dazu in der Wahlpflichtveranstaltung Kryptologie.

Dart bietet mehrere Hashfunktionen in der `CryptoUtils` Bibliothek. Das Prinzip soll am Beispiel der SHA256 Hashfunktion verdeutlicht werden.

SHA256 Hashsummen (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
import 'package:crypto/crypto.dart';
import 'dart:convert';

String sha256(String input) {
  final sha256 = new SHA256();
  sha256.add(UTF8.encode(input));
  return CryptoUtils.bytesToHex(sha256.close());
}
```

Eine **SHA256 Hashsummen** Funktion lässt sich in Dart bspw. wie folgt definieren (leider gibt es diese Hilfsfunktion nicht in `CryptoUtils`, da würde sie eigentlich hinein gehören).

```
String LoremIpsum = """
Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et dolore
magna aliquyam erat, sed diam voluptua. At vero eos et accusam
et justo duo dolores et ea rebum. Stet clita kasd gubergren,
no sea takimata sanctus est Lorem ipsum dolor sit amet.
""";

String loremIpsum = """
lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et dolore
magna aliquyam erat, sed diam voluptua. At vero eos et accusam
et justo duo dolores et ea rebum. Stet clita kasd gubergren,
no sea takimata sanctus est Lorem ipsum dolor sit amet.
""";
```

Seien ferner diese beiden nahezu identischen Texte gegeben (unterscheiden sich nur im ersten Buchstaben ‚l‘ und ‚L‘).

SHA256 Hashsummen (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

So ergibt

```
print("SHA256 von LoremIpsum: ${sha256(LoremIpsum)}");
print("SHA256 von loremIpsum: ${sha256(loremIpsum)}");
```

die folgende Konsolenausgabe

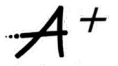
```
SHA256 von LoremIpsum:
7cb5f3ae204cbc19cdbef0b30735f1e6be4e5e4fcb93c072b3bc4a2574df0969
SHA256 von loremIpsum:
e6c3e2814c72099269a633d3e6a94cc614465c7cc65814138292f784e684acc8
```

D.h. zwei vollkommen unterschiedliche (aber gleichlange) Zeichenketten (Fingerprints) zweier nahezu identischer Zeichenketten (nur in einem Zeichen ein Unterschied).

Von diesen Zeichenketten kann nicht auf die ursprünglichen Texte zurückgerechnet werden.

Hashsummen eignen sich daher zum Beispiel dazu, Passwörter zu speichern. Unterschiedliche Passwörter haben unterschiedliche Hashsummen. Von den Hashsummen (sollte sie jemand in die Finger bekommen) kann jedoch nicht auf das Passwort zurückgerechnet werden.

Zusammenfassung



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Das Dart Library System**
  - Package Manager pub
  - Packages definieren, `pubspec.yaml`
- **`dart:async` (Streams, Futures und Isolates)**
- **`dart:io` (Files und Directories, streambasiert oder all at once)**
- **`dart:html` (DOM-Tree lesen und manipulieren, Selektoren, Event-Handler)**
- **Serverside and Clientside Programming**
  - `HttpServer`, `start` Web Framework
  - `WebSockets`, HTML Chat
- **`dart:convert`**
  - HTML Escapen (Cross Site Scripting!)
  - JSON, Base64 und Hashsummen (SHA256)

Tour de Dart Libraries

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

PARIS! Endlich sind wir da!

Beweisen Sie sich nun mit Dart auf der Vuelta ☺