Angular: Observables und Promises, async und await

Observables am Beispiel des ItemServices

```
@Injectable()
export class ItemService {
  constructor(private http: HttpClient) { }
  getAllItems(): Observable<Item[]> {
    return this.http.get<Item[]>(`${URL}/items`);
  }
  deletItem(id: string): Observable<HttpResponse<Item>> {
    id = id.toUpperCase();
    return this.http.delete<Item>(`${URL}/items/${id}`,
      { observe: 'response' });
  }
  …
}
```

Verwendung als Observable

```
is.getAllItems()
  .subscribe({
    next: result => this.items = result,
    error: error => this.error = error
  });
```

Verwendung als Promise

```
import { lastValueFrom } from 'rxjs';

…
lastValueFrom(is.getAllItems())
  .then(result => this.items = result)
  .catch(error => this.error = error);
```

- `lastValueFrom()` liefert Promise zurück
- `then()` können geschachtelt werden. Bei Fehler wird äußerstes `catch()` ausgeführt
- Auch `finally()` möglich

Unterschied Observable – Promise

Observable kann während seiner Aktivität mehrere Werte in zeitlichen Abständen liefern, *Promise* liefert immer nur einen Wert, deshalb entweder `firstValueFrom` oder `lastValueFrom`

Promises am Beispiel des ItemServices

```
@Injectable()
export class ItemService {

  constructor(private http: HttpClient) { }

  getAllItems(): Promise<Item[]> {
    return lastValueFor(this.http.get<Item[]>(`${URL}/items`));
  }

  deleteItem(id: string): Promise <HttpResponse<Item>> {
    id = id.toUpperCase();
    return lastValueFrom(this.http.delete<Item>(
      `${URL}/items/${id}`, { observe: 'response' }));
  }
  …
}
```

Verwendung als Promise

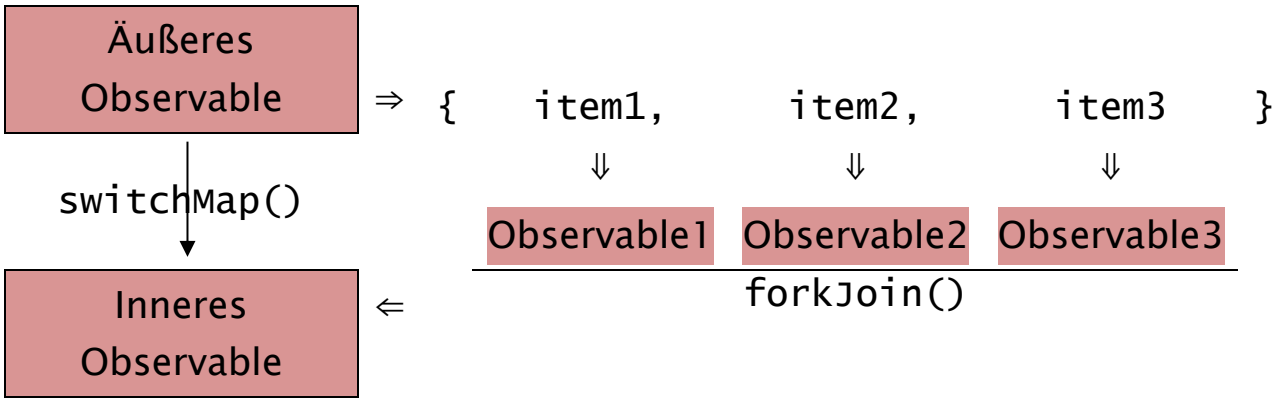
```
is.getAllItems()
  .then(result => this.items = result)
  .catch(error => this.error = error);
```

PROBLEM: Alle Artikel sollen über den Web-Service gelöscht werden

Innerhalb eines Observables müssen weitere gestartet werden

Zuerst müssen alle Artikel gesucht und dann jeder Einzelne davon gelöscht werden

```
deleteAllItems(): Observable<any> {
  return this.getAllItems()
    .pipe(
      switchMap(items => {
        if (items != null && items.length > 0) {
          return forkJoin(items.map(
            item => this.deleteItem(item.id)));
        } else {
          return new Observable(obs => {
            obs.next(null);
            obs.complete();
          });
        }
      })
    );
}
```



Mit `switchMap()` wird ein *inneres* Observable gestartet und dieses anstelle des *äußeren* Observable zurück geliefert. Da in Wirklichkeit mehrere *innere* Observables gestartet werden, müssen diese mit `forkJoin()` zu einem Observable zusammengefasst werden

Promises

```
async deleteAllItems(): Promise<number> {
  const items: Item[] = await this.getAllItems();
  let deleted = 0;
  for (const item of items) {
    await this.deleteItem(item.id).then(_ => deleted++);
  }
  return deleted;
}
```

- `await` wartet bis Promise Ergebnis liefert. Damit werden Methodenaufrufe serialisiert
- `await` kann nur in mit async gekennzeichneter Methode verwendet werden
- Eine mit async versehene Methode kapselt die Rückgabe in einem Promise

HINWEIS: Würde zweites `await` von oben fehlen würde Promise den Wert 0 zurück liefern

PROBLEM der obigen Lösung: Artikel werden nacheinander gelöscht

Das Löschen wird für alle Artikel gleichzeitig gestartet*1), und dann wird auf alle Antworten der Promises gewartet*2)

```
async deleteAllItems(): Promise<number> {
  let deleted: number = 0;
  const deletePromises: Promise<number>[] =
    (await this.getAllItems()).map(
      item => this.deleteItem(item.id).then(_ => deleted++)*1)
    );
  for(const deletePromise of deletePromises) {
    await deletePromise;*2)
  }
  return deleted;
}
```