

# Observables & Reactive Programming



# Agenda

- REST Calls: Observables vs. Promises
- Observable & Observer
- Introduction to Reactive Extensions
- Creating Observables
- Using Operators

# Promises vs Observables

# Promises vs Observables

## Promise

- One Time Result
- Not Cancable
- Executed through method ie. `http.get(...)`
- Processed using `.then()`
- Need reference to original function for retry

## Observable

- One Time Result or Stream
- Cancable
- Executed through subscription
- Processed using `[.map() and] .subscribe()`
- Build in Support for retry (`retry`, `retryWhen`)
-

# Promises vs Observables - Code

## Service implementation

```
@Injectable()
export class VouchersService {
  constructor(private http: HttpClient) { }

  getVouchers() : Promise<any> {
    return this.http.get('/api/vouchers').toPromise();
  }

  getVouchersObs() : Observable<Voucher[]> {
    return this.http.get<Voucher[]>('/api/vouchers');
  }
}
```

## Service consumption

```
export class VouchersComponent implements OnInit {
  vouchers: Voucher[];
  constructor(private router: Router, private vs: VouchersService) { }
  ngOnInit() {

    //promise
    this.vs.getVouchers().then(data => this.vouchers = data)

    //observable
    this.vs.getVouchersObs()
      .subscribe((responseData)=>{
        this.vouchers = responseData;
      })
  }
}
```

# Observables & Observer

# What is an Observable

- *Observable can be:*
  - *A one-time response (of http operations)*
  - *A Sequence of items (using WebSockets, or Streams)*
  - *Events, triggered by Code or User input*
  - *...*
- *Alternative to using:*
  - *Callbacks or*
  - *Promises*

```
@Injectable()
export class VouchersService {
  constructor(private http: HttpClient) { }

  getVouchers() : Observable<Voucher[]> {
    return this.http.get<Voucher[]>('/api/vouchers');
  }
}
```

```
export class VouchersComponent implements OnInit {
  vouchers: Voucher[];
  constructor(private router: Router, private vs: VouchersService) { }
  ngOnInit() {
    this.vs.getVouchers()
      .subscribe((responseData)=>{this.vouchers = responseData;})
  }
}
```

# What is an Observer

- Observer subscribes to an Observable (... observes Observable)
- Reacts to whatever item or sequence of items the Observable emits:
  - Handle Data: `next()`
  - Handle Error: `error()`
  - Handle Completion: `complete()`
- Might use Operators to deal with Observables – ie Filter

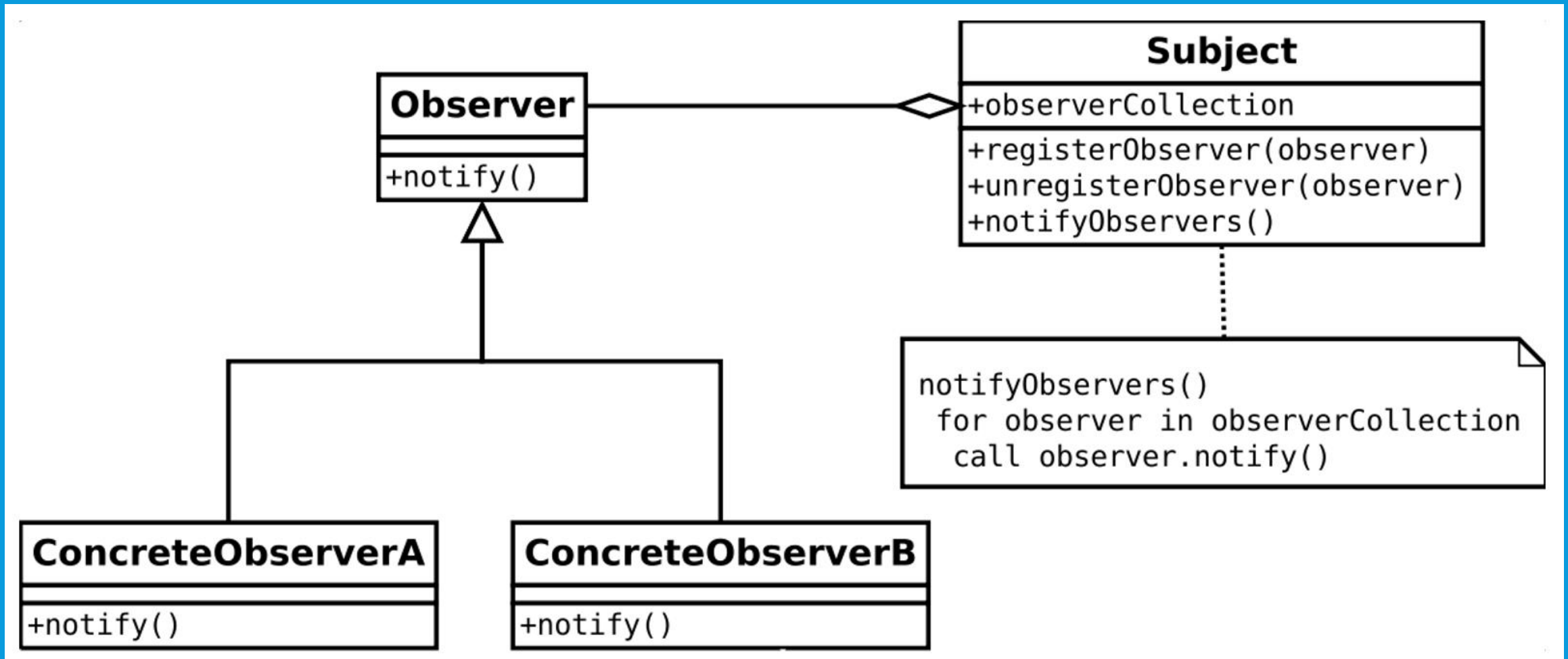
```
myObservable.subscribe(myOnNext, myError, myComplete);
```



# Subject

- Is simply an Observer and Observable at the same time
- You can push new values as well as subscribe to it
- Many Subject implementations:
  - Subject
  - Behavior Subject
  - ...

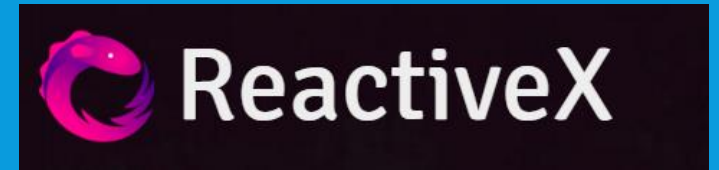
# Observable Pattern - Big Picture



# Introduction to Reactive Extensions

# What are Reactive Extensions?

- API for async programming with observables  
Streams
- Available for JavaScript, .NET, Swift, Java, ...
- An implementation of the Observable Pattern
- Reactive Programming provides a collection of operators to manipulate response (Observables)
- Most common used operator is Map – transform response into a subset (... matching the Data Model)
- Operator documented @ <http://reactivex.io/documentation/operators.html>



# Getting Started

- RxJS is used internally by Angular -> No need to install anything
- Just import Objects and Operators you want to use
- For production make sure you just import what you need to reduce download size

```
import { Observable } from 'rxjs/Rx';
```

```
import 'rxjs/add/operator/map';  
import 'rxjs/add/operator/toPromise';
```

# Operators

- Operators allow us to deal with / manipulate Observables
- Operators can be chained
- Can be grouped into Operators that:
  - Create Observables (Create, From, ...)
  - Transform Observables (Map, GroupBy, ...)
  - Filter Observables (Filter, Take, Distinct, ...)
  - Combine (And / Then / When, ...)
  - Error Handling & Utility (Catch, Retry, Subscribe, ...)
- Documented @ <http://reactivex.io/documentation/operators.html>

# Using Operators

# .map()

- Transform the items emitted by an Observable by applying a function to each item
- Often used when working with Http to extract Data from Response

```
getVouchersHttp(){  
  this.http.get('http://localhost:5000/api/vouchers')  
    .map(response => response.json()).subscribe((data)=>{  
      this.result = data;  
    })  
}
```



# .filter()

- Emit only those items from an Observable that pass a predicate test

```
getVouchersFilter(){  
  this.http.get('http://localhost:5000/api/vouchers')  
    .map(response => response.json())  
    .filter(data => data.json().deleted == false)  
    .subscribe((data)=>{  
      this.result = data;  
    })  
}
```