Angular 7

RxJS

> scalac

.mo.

Data flows

500 BC	Heraclitus of Ephesus
	“everything flows”
21st century	reactive programming
	asynchronous data streams

Rx - Reactive Extensions

implementations:

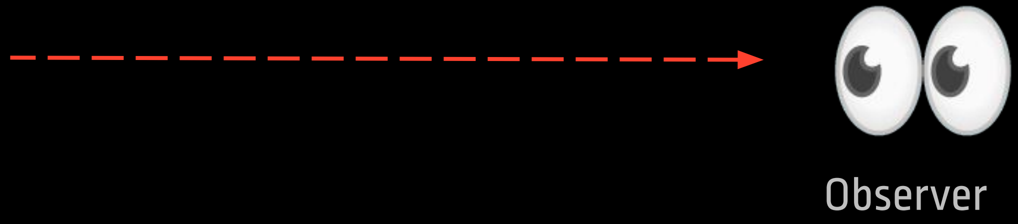
- JS, Scala, Java, .NET, Go, Python and more....
- RxJS 6

→ RxJS API

Stream sources

- HTTP responses / Websocket messages
- Promises callbacks
- Reactive forms changes
- DOM events (keyup, click, ...)
- Router navigation events
- timers, static values, and more...

Observable$



- subscribe starts emitting values from stream
- unsubscribe cancels emitting
- values can flow through pipe with operators

Subscription

Observer

```
s = observable$.subscribe({
        next(response) {},
        error(err) {},
        complete() {}
      });
```

😋 Consuming data stream

- subscribe

```
chatService.message$.subscribe(m => {
    this.messages.push(m);
})
```

- AsyncPipe - automatic subscribe and unsubscribe in template

```
<li *ngFor="let u of users$ | async">
```

Operators

standalone functions for creating Observable

● Creation / Join	from, fromEvent, of, interval, forkJoin, …

used inside pipe - return new Observable based on input Observable

● Transformation	map, switchMap, …
● Filtering	filter, debounceTime, distinctUntilChanged, …
● Multicasting	share, …
● Helpers	tap, delay, catchError, …

Using operators

```
chatService.message$.pipe(
    tap(m => console.log(m)),
    map(m => {
        m.text = m.text.replace(':)', '😀');
        return m;
    })
).subscribe(m => { … });
```

Operator - pure function

```
let variable;
dataStream$.pipe(
	operator1(x => {}),
	operator2(),
	...,
).subscribe(result => {});
```

☠️ variable = x

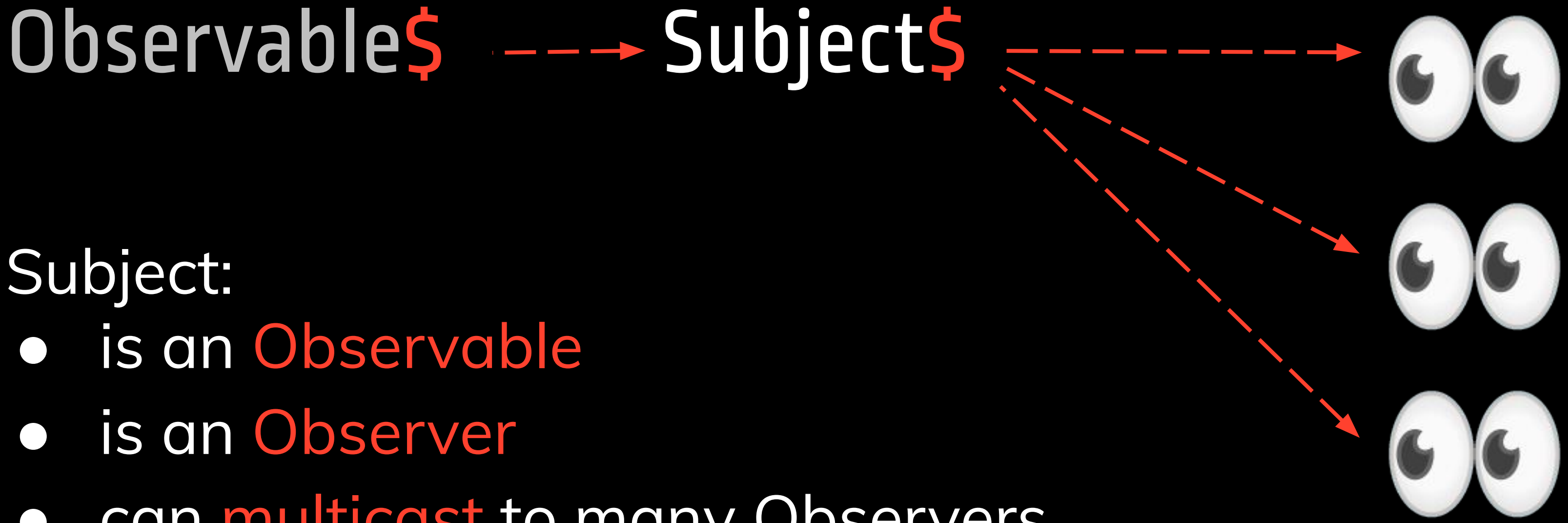
don’t perform side-effects inside operators!

unsubscribe

```
interval: Subscription;


this.interval = interval(1000).subscribe(x => {
    chatService.sendMessage(`spam ${x}`);
});


this.interval.unsubscribe();
```



Subject:

- is an Observable
- is an Observer
- can multicast to many Observers

🤔 Problems with promises	🤓 Rx solution
● cascade requests	switchMap / combineLatest
● parallel requests	forkJoin
● multiple requests errors	catch in observer
● result only in one place	Subject / share

Sharing data stream

- Subject - connection state change available in 3 components

```
connectionState$ = new BehaviorSubject<State>(State.none);
```

- share() - connection success and errors handled in 2 places

```
connect$ = combineLatest(user$, room$).pipe(
   share()
);
```

Cascade requests

switchMap

```
from(getUser()).pipe(
   switchMap(user => user.getRoom()
).subscribe(room => {} )
```

combineLatest

```
user$ = from(getUser())
room$ = user$.pipe(
   switchMap(user => user.getRoom()));
combineLatest(user$, room$)
   .subscribe([user, room] => {} )
```

Service, dependency injection

```
@Injectable({ providedIn: 'root' })
export class ChatService { }


export class MessagesComponent  {
  constructor(private chatService: ChatService) { }
}
```

NgRx - Reactive State for Angular

inspired by Redux

- actions, reducers, selectors, store
- isolated side-effects

→ [ngrx.io]

Custom operator

```
const emoji = () => (source: Observable<Message>) =>
  new Observable(observer => {
    return source.subscribe({
      next(m) { m.text = m.text.replace(':)', '😀'); observer.next(m); },
      error(err) { … },
      complete() { … }
    });
  });
```

Rx and DOM events in Angular

```
@ViewChild('chatInput') chatInput: ElementRef;
ngAfterViewInit() {
    const keyStream$ = fromEvent(this.chatInput.nativeElement, 'keydown');
    keyStream$.pipe(throttleTime(600)).subscribe(() => {
        chatService.sendTypingNotification();
    });
}
```

Homework

- discover samples on StackBlitz from RxJS API and experiment!
- show typing indicator in users component

Thank You