```
@Pipe({
  name: 'async',
  pure: false
})
export class AsyncPipe
  implements OnDestroy, PipeTransform {
  ...
```

INDEPTH
ANGULAR

It's an impure pipe, it means the transform method will be called on every change detection cycle

EVERYTHING YOU NEED TO KNOW ABOUT THE ASYNC PIPE IN ANGULAR

more content →

```
@Pipe({
  name: 'async',
  pure: false
})
export class AsyncPipe
  implements OnDestroy, PipeTransform {
  transform<T>(
  obj:
    Observable<T>|
    Subscribable<T>|
    Promise<T>|
    null|
    undefined
  ): T|null;
```

INDEPTH
ANGULAR

It's a built-in pipe in Angular, which accepts a promise or an observable

EVERYTHING YOU NEED TO KNOW ABOUT THE ASYNC PIPE IN ANGULAR

more content →

```
export class AsyncPipe
  implements OnDestroy, PipeTransform {
  private _obj = null;

  transform<T>(
    obj:
      Observable<T>|
      Subscribable<T>|
      Promise<T>|
      null|
      undefined
  ): T|null {
    if (!this._obj) {
      if (obj) {
        this._subscribe(obj); // ←
      }
      return this._latestValue;
    }
    ...
  }
```

INDEPTH
ANGULAR

It subscribes to the observable or promise and returns the latest emitted value

EVERYTHING YOU NEED TO KNOW ABOUT THE ASYNC PIPE IN ANGULAR

more content →

```
export class AsyncPipe
  implements OnDestroy, PipeTransform {
  private _obj = null;

  transform<T>(
    obj:
      Observable<T>|
      Subscribable<T>|
      Promise<T>|
      null|
      undefined
  ): T|null {
    ...
    if (obj !== this._obj) {
      this._dispose(); // ←
      return this.transform(obj);
    }
    ...
  }
```

INDEPTH
ANGULAR

When the source observable changes its reference, the async pipe will unsubscribe from the old observable and subscribe to the new one

EVERYTHING YOU NEED TO KNOW ABOUT THE ASYNC PIPE IN ANGULAR

more content →

```
export class AsyncPipe
  implements OnDestroy, PipeTransform {

  ngOnDestroy(): void { // ←
    if (this._subscription) {
      this._dispose();
    }
  }
```

INDEPTH
ANGULAR

It will automatically unsubscribe from observable when the component in which the pipe is used gets destroyed

EVERYTHING YOU NEED TO KNOW ABOUT THE ASYNC PIPE IN ANGULAR

more content →

```
export class AsyncPipe
  implements OnDestroy, PipeTransform {

  private _obj = null;
  private _latestValue: any = null;

  constructor(
    private _ref: ChangeDetectorRef
  ) {}

  private _updateLatestValue(async: any, va
lue: Object): void {
    if (async === this._obj) {
      this._latestValue = value;
      this._ref.markForCheck(); // ←
    }
  }
  ...
```

INDEPTH
ANGULAR

It will call markForCheck when new value from observable emitted

EVERYTHING YOU NEED TO KNOW ABOUT THE ASYNC PIPE IN ANGULAR

more content →

```
export class AsyncPipe implements OnDestroy, PipeTransform {
  private _obj = null;
  private _latestValue: any = null;

  transform<T>(
    obj:
      Observable<T>|
      Subscribable<T>|
      Promise<T>|
      null|
      undefined
  ): T|null {
    if (!this._obj) {
      if (obj) {
        this._subscribe(obj);
      }
      return this._latestValue; // ⬅ initial null value returned
    }
    ...
  }
```

INDEPTH
ANGULAR

Async pipe has the problem with initial null value, it means before the observable emits any value, the result of transform always return null initially

EVERYTHING YOU NEED TO KNOW ABOUT THE ASYNC PIPE IN ANGULAR

end of content