Angular BehaviorSubject Explained: A Comprehensive Tutorial

Introduction:

In Angular, `BehaviorSubject` is a powerful tool for managing and sharing state across components. It extends the `Subject` class and holds the current value, making it accessible to subscribers even if they join late. This tutorial will guide you through creating and using `BehaviorSubject` with a practical example using Angular services and components.

1. Understanding BehaviorSubject:

What is BehaviorSubject?

`BehaviorSubject` is both an observable and an observer. It maintains the "current value" and emits it immediately to subscribers upon subscription. Subscribers receive the current value and subsequent changes.

2. Setting Up the Angular Project:

Install Angular CLI:

```bash
npm install -g @angular/cli
```

Create a New Angular Project:

```bash
```

```
ng new BehaviorSubjectExample
cd BehaviorSubjectExample
```

3. Creating a Service with BehaviorSubject:

Create a Service:

```bash
ng generate service data
```

Implement DataService (data.service.ts):

```typescript
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class DataService {
  private _dataSubject = new BehaviorSubject<string>('Initial Value');
  data$ = this._dataSubject.asObservable();

  updateData(newValue: string): void {
```

```
    this._dataSubject.next(newValue);
  }
}
```

4. Using BehaviorSubject in a Component:

Create a Component:

```bash
ng generate component sample
```

Implement SampleComponent (sample.component.ts):

```typescript
import { Component, OnDestroy } from '@angular/core';
import { DataService } from '../data.service';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-sample',
  templateUrl: './sample.component.html',
  styleUrls: ['./sample.component.css'],
})
export class SampleComponent implements OnDestroy {
```

```
  data: string;
  private dataSubscription: Subscription;

  constructor(private dataService: DataService) {
    this.dataSubscription = this.dataService.data$.subscribe((value) => {
      this.data = value;
    });
  }

  ngOnDestroy(): void {
    this.dataSubscription.unsubscribe();
  }

  updateData(newValue: string): void {
    this.dataService.updateData(newValue);
  }
}
```

Implement SampleComponent HTML (sample.component.html):

```html
<div>
  <h2>Data in Sample Component: {{ data }}</h2>
  <input
    type="text"
```

```
  [(ngModel)]="data"
  placeholder="Enter new value"
 />
 <button (click)="updateData(data)">Update Data</button>
</div>
```

5. Integrating the Component:

Update App Component HTML (app.component.html):

```html
<div style="text-align: center;">
 <h1>
  Welcome to BehaviorSubject Example!
 </h1>
 <app-sample></app-sample>
</div>
```

6. Run the Application:

```bash
ng serve
```

Visit `http://localhost:4200/` to see the application.

7. Explanation:

- The `DataService` contains a `BehaviorSubject` named `_dataSubject` initialized with 'Initial Value'.

- The `SampleComponent` subscribes to the `data$` observable in the `DataService` to receive updates.

- Changing the input value in `SampleComponent` triggers the `updateData` method, updating the `BehaviorSubject` value.

Now, your Angular application demonstrates the usage of `BehaviorSubject` to share and manage state across components efficiently. Feel free to extend this example to suit your specific application requirements.