**Example 1**

1. **Define the Pipe Logic**  
   Open windows.pipe.ts and modify it to process the array of window objects.

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'windows'

})

export class WindowsPipe implements PipeTransform {

transform(windows: any[], filterActive: boolean = false): any[] {

if (!windows) {

return [];

}

// Filter windows based on the 'filterActive' flag

return windows.filter(window => filterActive ? window.isActive : true);

}

}

In this example, the pipe:

* + Takes an array of windows as input.
  + Optionally filters the windows based on an isActive flag, which is passed as filterActive (default is false, meaning no filtering).

**Step 2: Use the Pipe in a Component**

Now, let's assume you have a component where you want to use this pipe to filter and display windows.

1. **Component Setup**

In your app.component.ts (or any other component), you might have an array of windows:

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

windows = [

{ title: 'Window 1', content: 'Content of Window 1', isActive: true },

{ title: 'Window 2', content: 'Content of Window 2', isActive: false },

{ title: 'Window 3', content: 'Content of Window 3', isActive: true },

{ title: 'Window 4', content: 'Content of Window 4', isActive: false }

];

}

1. **Using the Pipe in the Template**

In your app.component.html, you can now use the windows pipe to filter the array or display all windows.

To display only active windows:

html

Copy code

<div \*ngFor="let window of windows | windows: true">

<h3>{{ window.title }}</h3>

<p>{{ window.content }}</p>

</div>

To display all windows (active and inactive):

html

Copy code

<div \*ngFor="let window of windows | windows">

<h3>{{ window.title }}</h3>

<p>{{ window.content }}</p>

</div>

**Step 3: Style and Enhance (Optional)**

You can extend this pipe or the component to add more functionality, such as:

* Sorting the windows by title or content.
* Modifying the display logic based on window properties.
* Adding animations or interactive behavior for active/inactive windows.

For example, adding a class to the active windows in the template:

html

Copy code

<div \*ngFor="let window of windows | windows: true" [class.active]="window.isActive">

<h3>{{ window.title }}</h3>

<p>{{ window.content }}</p>

</div>

You can define the .active class in your CSS:

css

Copy code

.active {

background-color: lightgreen;

border: 2px solid green;

}

**Final Notes**

* **Performance Considerations**: Custom pipes are re-evaluated whenever Angular's change detection runs. If your array is large or if the transformation is complex, consider optimizing the pipe, e.g., by implementing pure: false in the pipe decorator or using memoization.
* **Advanced Scenarios**: You can extend this pipe to support more complex transformations, like sorting, pagination, or more advanced filtering based on multiple criteria.

This basic structure should help you get started with a custom Angular pipe that processes an array of objects representing windows or similar entities.

Example 2

**1. Create the Pipe**

First, let's create a custom pipe using the Angular CLI.

bash

Copy code

ng generate pipe user-filter

This will create a new pipe file called user-filter.pipe.ts.

**2. Implement the Pipe**

Now, edit the generated pipe file (user-filter.pipe.ts) to implement your custom logic using array methods like filter and map.

typescript

Copy code

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'userFilter'

})

export class UserFilterPipe implements PipeTransform {

transform(users: any[], role: string): any[] {

if (!users || !role) {

return users;

}

// Filter the users based on role

const filteredUsers = users.filter(user => user.role === role);

// Transform user names to uppercase

return filteredUsers.map(user => ({

...user,

name: user.name.toUpperCase()

}));

}

}

**Explanation**

1. **transform(users: any[], role: string)**: The transform method is where we apply our logic. It takes the users array and a role string as parameters.
2. **filter(user => user.role === role)**: We filter the users to only include those whose role matches the provided role parameter.
3. **map(user => ({ ...user, name: user.name.toUpperCase() }))**: After filtering, we use map to create a new array with transformed user objects, where each user's name is converted to uppercase.

**3. Register the Pipe**

Ensure that the pipe is declared in your Angular module (app.module.ts).

typescript

Copy code

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import { UserFilterPipe } from './user-filter.pipe'; // Import the pipe

@NgModule({

declarations: [

AppComponent,

UserFilterPipe, // Declare the pipe

],

imports: [BrowserModule],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule {}

**4. Use the Pipe in a Component**

Now you can use this custom pipe in your component's template. Let's assume you have a component where the users' data is available.

typescript

Copy code

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

template: `

<h1>Filtered and Transformed Users</h1>

<ul>

<li \*ngFor="let user of users | userFilter: 'admin'">

{{ user.name }} ({{ user.role }})

</li>

</ul>

`

})

export class AppComponent {

users = [

{ name: 'Alice', role: 'admin' },

{ name: 'Bob', role: 'user' },

{ name: 'Charlie', role: 'admin' },

{ name: 'David', role: 'user' }

];

}

**Explanation of the Template:**

* **users | userFilter: 'admin'**: The pipe filters the users array to only include users with the role 'admin', and then it transforms their names to uppercase.
* **\*ngFor="let user of users | userFilter: 'admin'"**: This loops over the transformed list of users and displays them in an unordered list.

**5. Output**

For the given users array, the output in the browser will look like this:

diff

Copy code

Filtered and Transformed Users

- ALICE (admin)

- CHARLIE (admin)

**Notes**

* **Handling Null or Undefined Values**: The pipe checks if the users array is provided. If it’s null or undefined, it returns the original array.
* **Optimization**: If the data changes frequently, consider using Pure pipes (default) or memoization techniques to avoid unnecessary recalculations.

**Conclusion**

This is a simple example of using array methods like filter and map inside a custom Angular pipe. You can customize this pipe to include additional logic depending on your use case, such as sorting, reducing, or chaining multiple array methods.

Top of Form

Bottom of Form