Pure and Impure Custom Pipe and Change Detection

Angular provides pure and impure pipes on the basis of change detection. Here we will discuss pure and impure pipes with examples.   
**1. Change Detection**  
  
Angular uses a change detection process for changes in data-bound values. Change detection runs after every keystroke, mouse move, timer tick, and server response. In case of pipe, angular picks a simpler, faster change detection algorithm.   
  
**2. Pure Pipes**  
  
By default all pipes are pure pipe. Pure pipes are those pipes that have pure: true in @Pipe decorator while creating pipe class. If we have not used pure metadata then its default value will be **true** in @Pipe decorator. Pure pipes executes only for pure changes in its input values.   
Find the pure changes.   
**a.** Change to a primitive input values such as String, Number, Boolean.   
**b.** Change to object reference of Date, Array, Function, Object.   
  
Above changes are pure changes. If the input values used with pure pipe, comes under the pure changes then pipe will run again to give output accordingly. Pure pipe is created as follows.

@Pipe({

name: 'companyone'

})

By default @Pipe decorator is using pure: true.   
  
**3. Impure Pipes**  
  
Impure pipes will run for every component change detection cycle. So it is obvious that impure pipes will also run for pure changes. Impure pipe will run for every keystroke or mouse move. So the conclusion is that impure pipe will run a lot and hence we should take care while using impure pipe because it may reduce performance of the application and can destroy user experience. Impure pipe is created as follows.

@Pipe({

name: 'companytwo',

pure: false

})

Make sure that metadata pure has been assigned with **false** boolean value.   
  
**4. Example of Pure and Impure Pipes**  
  
Now we will discuss an example of pure and impure pipe. We will see that pure pipe will run only for pure changes whereas impure pipe will run for every type of changes in component properties. We will start by creating pure and impure pipes whose transform() method will use Company class as parameter type and string as return type.   
Find the pure pipe named as **companyone**   
**companyone.pipe.ts**

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

import {Company} from './company';

@Pipe({

name: 'companyone'

})

export class CompanyOnePipe implements PipeTransform {

transform(obj: Company): string {

let output = obj.cname+' : '+ obj.location;

return output;

}

}

Now find the impure pipe named as **companytwo**   
**companytwo.pipe.ts**

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

import {Company} from './company';

@Pipe({

name: 'companytwo',

pure: false

})

export class CompanyTwoPipe implements PipeTransform {

transform(obj: Company): string {

let output = obj.cname+' : '+ obj.location;

return output;

}

}

We will observe that **companyone** and **companytwo** both pipes are doing same task. The only difference is that **companyone** is a pure pipe and **companytwo** is an impure pipe.   
Now we will code scenarios for pure and impure changes in Company object. Suppose we have created Companyobject as follows.

compName:string = "ABCD LTD";

compLocation:string = "Varanasi";

company = new Company(this.compName, this.compLocation);

**a.** Pure change in company object by changing its **reference**: To change the reference we will create a new object of Company class and assign it to component property company as follows.

createCompany() {

this.company = new Company(this.compName, this.compLocation);

}

When we call the above method then pure and impure both pipes will run. It means **companyone** and **companytwo** both pipe will run again.   
  
**b.** Impure change in company object by updating its property values: To generate the scenario of impure change, we will update the property value of our company object as follows.

updateCompany() {

this.company.cname = this.compName;

this.company.location = this.compLocation;

}

When we call the above method then only impure pipe will run. It means **companyone** pipe will not run again but **companytwo** pipe will run again for the changes in company object.   
  
Now find the below code from custompipe.component.ts file.

Company Name: <input [(ngModel)] ="compName"/> {{compName}}

<br/>Location: <input [(ngModel)] ="compLocation"/> {{compLocation}}

<br/><br/><button (click)="updateCompany()">Update Existing</button>

<button (click)="createCompany()">Create New </button>

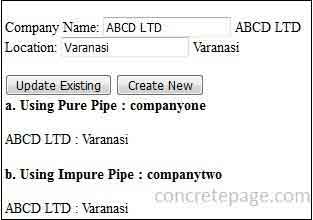
<br/><b>a. Using Pure Pipe : companyone</b><br/><br/>

{{company | companyone}}

<br/><br/><b>b. Using Impure Pipe : companytwo</b><br/><br/>

{{company | companytwo}}

**Output**  
  
Initially both text box will be populated with following values.   
Company Name : ABCD LTD   
Location: Varanasi



**Case 1**: Change the text box values as follows.   
Company Name : ABCD LTD11   
Location: Varanasi11   
  
Now click on **Update Existing** button. Output of **companyone** pipe will be as follows.

ABCD LTD : Varanasi

And output of **companytwo** pipe will be as follows.

ABCD LTD11 : Varanasi11

Let us understand what is happening now. When we click on **Update Existing** button then updateCompany() will execute and performs impure change in company object. So only impure pipe **companytwo** will run again and change its output. There will be no change in pure pipe **companyone** output because it has not run again due to impure change in company object.   
  
**Case 2**: Change the text box values as follows.   
Company Name : ABCD LTD1111   
Location: Varanasi1111   
  
Now click on **Create New** button. Output of **companyone** pipe will be as follows.

ABCD LTD1111 : Varanasi1111

And output of **companytwo** pipe will be as follows.

ABCD LTD1111 : Varanasi1111

Let us understand what is happening. When we click on **Create New** button then createCompany() will execute and performs pure change in company object. So this time pure and impure pipe both will run again. It means **companyone** and **companytwo** both pipe will run again and change its output.