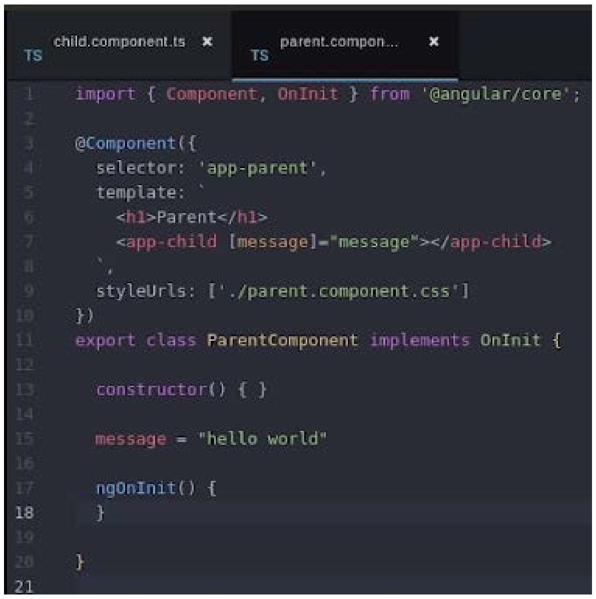
Angular Communication:

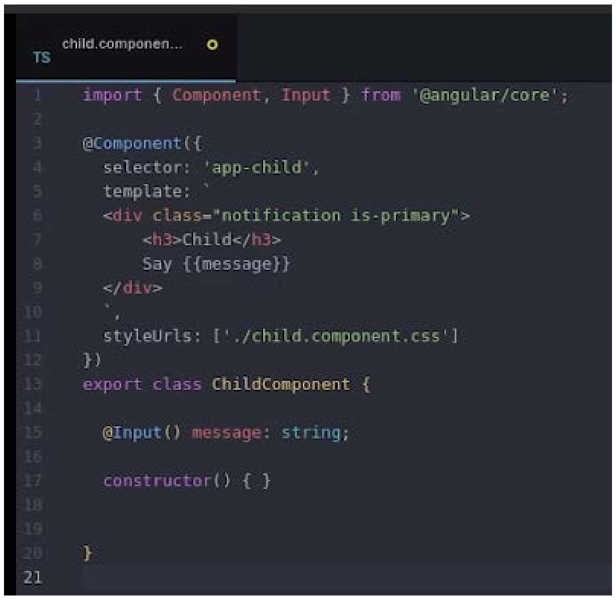
1. Using @Input (Parent ----->>>>>Child)

Parent Component



```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    <app-child [message]="message"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent implements OnInit {

  constructor() { }

  message = "hello world"

  ngOnInit() {
  }

}
```

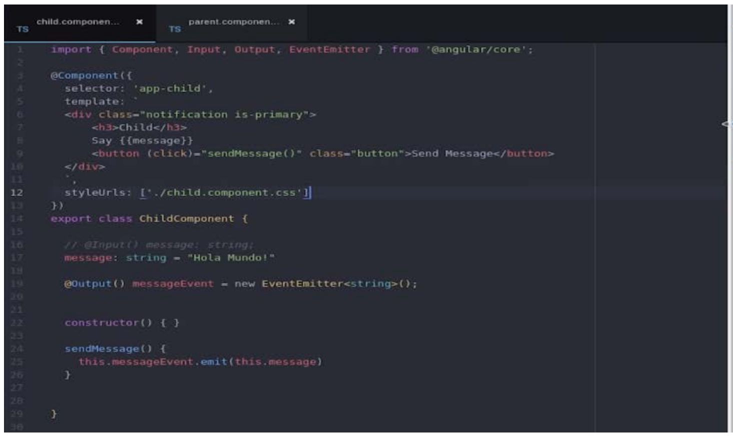
Child Component



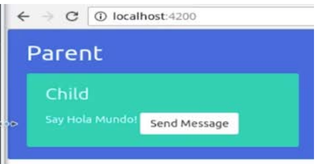
```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `
  <div class="notification is-primary">
      <h3>Child</h3>
      Say {{message}}
  </div>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  @Input() message: string;

  constructor() { }


}
```

2. Using @Output + EventEmitter<string> (Child--->>>Parent)

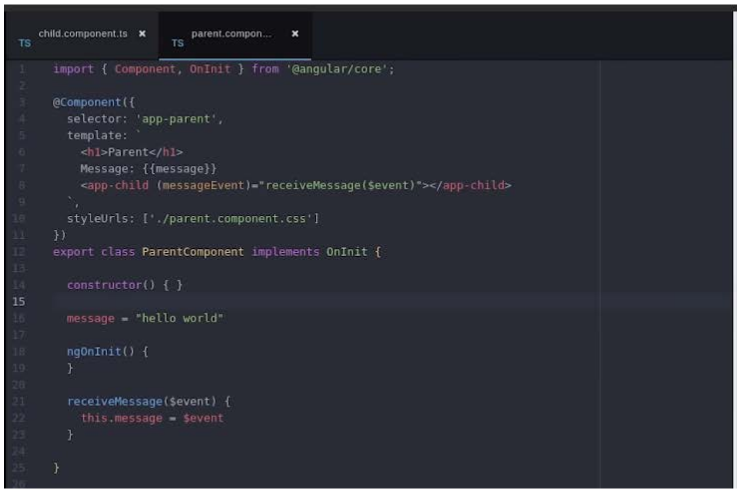
a) Child Component



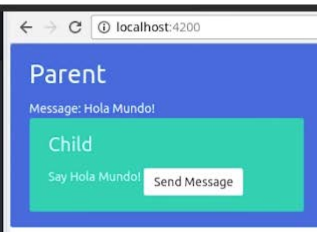
```
import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `
  <div class="notification is-primary">
      <h3>Child</h3>
      Say {{message}}
      <button (click)="sendMessage()" class="button">Send Message</button>
  </div>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  // @Input() message: string;
  message: string = "Hola Mundo!"

  @Output() messageEvent = new EventEmitter<string>();


  constructor() { }

  sendMessage() {
    this.messageEvent.emit(this.message)
  }


}
```



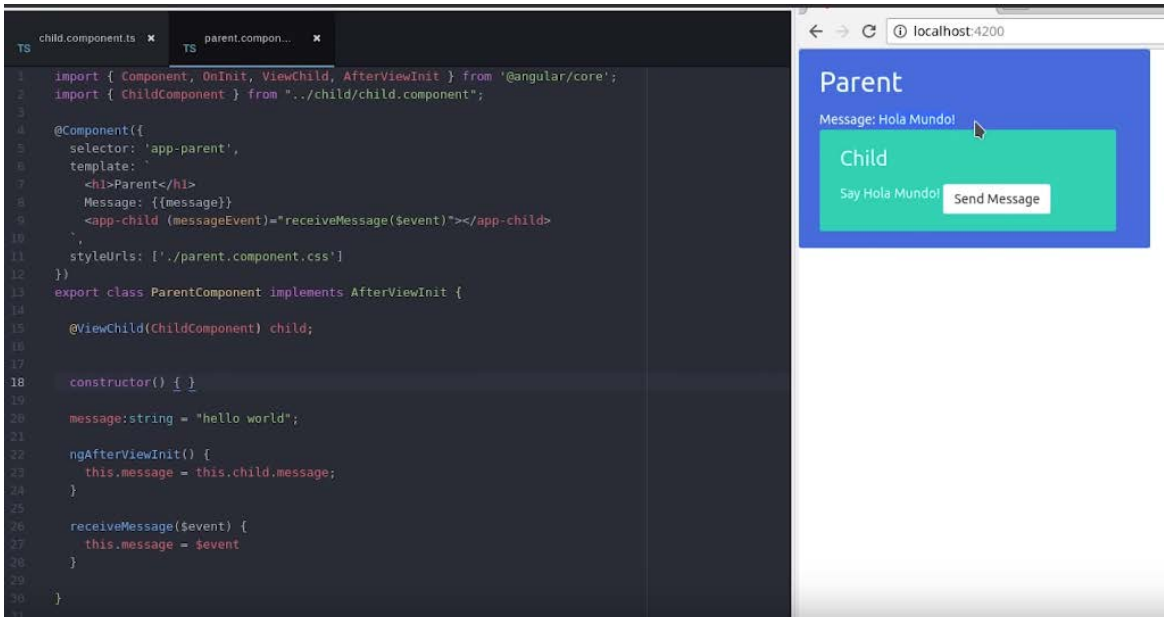
b) Parent Component



```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    Message: {{message}}
    <app-child (messageEvent)="receiveMessage($event)"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent implements OnInit {

  constructor() { }

  message = "hello world"

  ngOnInit() {
  }

  receiveMessage($event) {
    this.message = $event
  }

}
```



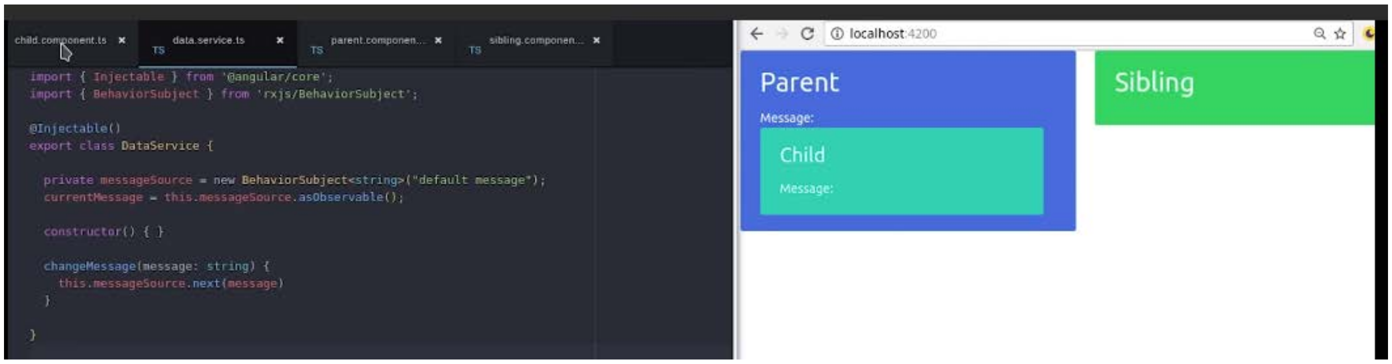
3. Using @ViewChild (Child--->>>Parent)



```typescript
import { Component, OnInit, ViewChild, AfterViewInit } from '@angular/core';
import { ChildComponent } from "../child/child.component";

@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    Message: {{message}}
    <app-child (messageEvent)="receiveMessage($event)"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent implements AfterViewInit {

  @ViewChild(ChildComponent) child;


  constructor() { }

  message:string = "hello world";

  ngAfterViewInit() {
    this.message = this.child.message;
  }

  receiveMessage($event) {
    this.message = $event
  }

}
```

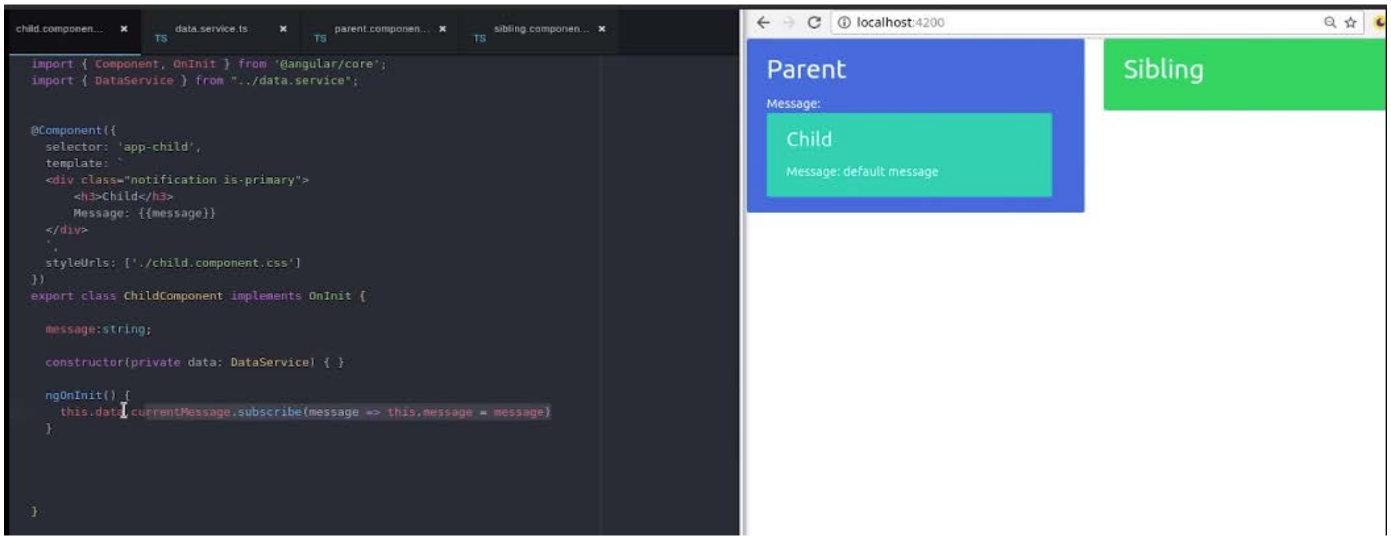
4. Using BehaviourSubject<string> (Comp1 -->> Comp2) or (Comp2 -->> Comp1)

We make common service of BehaviourSubject<string>



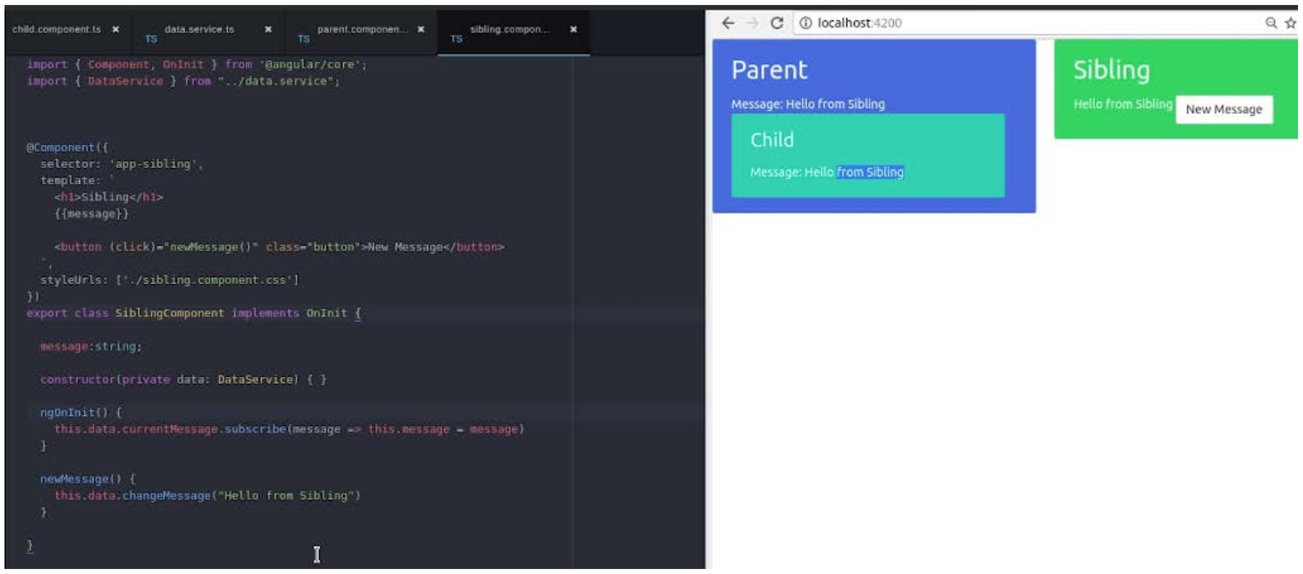
```typescript
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs/BehaviorSubject';

@Injectable()
export class DataService {

  private messageSource = new BehaviorSubject<string>("default message");
  currentMessage = this.messageSource.asObservable();

  constructor() { }

  changeMessage(message: string) {
    this.messageSource.next(message)
  }

}
```

Child Component



```typescript
import { Component, OnInit } from '@angular/core';
import { DataService } from "../data.service";


@Component({
  selector: 'app-child',
  template: `
  <div class="notification is-primary">
      <h3>Child</h3>
      Message: {{message}}
  </div>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnInit {

  message:string;

  constructor(private data: DataService) { }

  ngOnInit() {
    this.data.currentMessage.subscribe(message => this.message = message)
  }




}
```

Parent Component:



```
import { Component, OnInit } from '@angular/core';
import { DataService } from "../data.service";



@Component({
  selector: 'app-sibling',
  template: `
    <h1>Sibling</h1>
    {{message}}

    <button (click)="newMessage()" class="button">New Message</button>
  `,
  styleUrls: ['./sibling.component.css']
})
export class SiblingComponent implements OnInit {

  message:string;

  constructor(private data: DataService) { }

  ngOnInit() {
    this.data.currentMessage.subscribe(message => this.message = message)
  }

  newMessage() {
    this.data.changeMessage("Hello from Sibling")
  }

}
```

Var , Let and Const:

```
constructor(){
      var a;           Note: I can declare first and later i can initialised
      a = 10;
      let b;
      b = 10;
     const  c = 30;  Note: constant has to be initialised while declaring
}
```

Re-initialising:

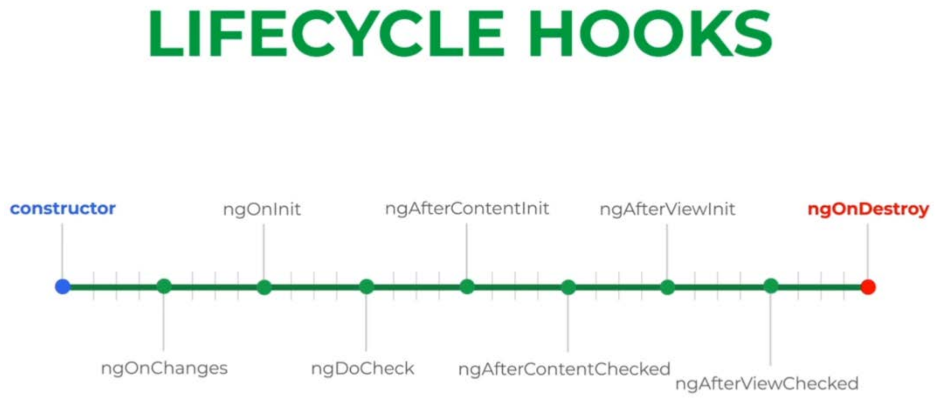
Var	Let	Const

constructor(){
      var a=10;
      console.log(a) ;   // 10
      a = 20;
     console.log(a) ;   // 20
}
	constructor(){
	      let a=10;
	      console.log(a) ; // 10
	      a = 20;
	     console.log(a) ; // 20
	}
		constructor(){
		      const a=10;
		      console.log(a) ; // 10
		      a = 20;
		 //Error ,you cannot re-initialised
		     console.log(a) ;
		}

**showData()
{
     var a=10;
     console.log(a); //10
if(true)
{
    var a=20;
    console.log(a); //20
     a=30;
   console.log(a)   // 30
}
console.log(a);   //30
}**
	**showData()
	{
	     let a=10;
	     console.log(a); //10
	if(true)
	{
	    let a=20;
	    console.log(a); //20
	     a=30;
	console.log(a) // 30
	}
	console.log(a); //10
	}**	

Component lifecycle hooks overview:

- Directive and component instances have a lifecycle as Angular creates, updates, and destroys them
- To see component lifecycle by implementing one or more of the *lifecycle hook* interfaces in the Angular `core` library.
- Each interface has a single hook method whose name is the interface name prefixed with `ng`.
- No directive or component will implement all of the lifecycle hooks.



- Angular only calls a directive/component hook method *if it is defined*.
- The JavaScript language doesn't have interfaces.
- Angular is a platform and framework for building client applications in HTML and TypeScript.
- Angular is written in TypeScript.
- Both components and services are simply classes, with *decorators* that mark their type and provide metadata that tells Angular how to use them.
- Angular instead inspects directive and component classes and calls the hook methods *if they are defined*.
- Angular finds and calls methods like `ngOnInit()`, with or without the interfaces.

.