Event Emitters

Em algum momento, é muito provável que você queira emitir eventos de um componente para o outro. Para isso, nós usamos os *Event Emitters*. Eles servem para criar eventos personalizados para os seus componentes, podendo ser ouvidos por componentes pai. Para criar um evento personalizado utilize, de início, o decorator *Output*. Ele funciona do mesmo do @Input, no entanto, ele serve para criar um objeto para emitir eventos, como o exemplo abaixo:

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

@Component({
  selector: 'meu-componente',
  templateUrl: 'meu-componente.component.html',
  stylesUrls: ['meu-componente.component.css']
})
export class MeuComponenteComponent {

  @Output()
  meuEvento: EventEmitter<any> = new EventEmitter<any>()
  /*
    o nome do evento será o mesmo nome da propriedade,
    nesse caso, o nome do evento será "meuEvento"
  */
}
```

A variável **meuEvento** é do tipo *EventEmitter<T>*. O valor que está entre os sinais “<” e “>” é chamado de **generic**. Generic é uma maneira de criar parâmetros para classes e definir tipos que podem ser substituídos em vários lugares do programa. Os Generics nos permitem fazer com que um objeto não trabalhe somente com um tipo específico de dados. Mas sim, com o tipo de dados que acharmos necessário. Como, por exemplo, uma string ou um número.

No caso do Generic do EventEmitter, ele serve para informar que tipo de dados devem ser enviados para o elemento que está ouvindo o evento. Quando o evento for executado, o tipo de dado que você colocou no generic será enviado juntamente com o evento. Nós podemos recuperar esses dados enviados usando a variável $event, utilizada anteriormente. Nesse caso, utilizamos o tipo *any* do Typescript. Assim, podemos passar o tipo de dados que nós quisermos ou não passar. Se caso você não usar o *any*, o EventEmitter te obrigará a enviar o tipo de dado que você definiu.

Agora, no template desse componente que possui o EventEmitter, devemos ouvir um evento padrão da DOM. Um evento de clique em um botão, por exemplo. Ao botão ser clicado no template, nós executaremos um método da nossa classe que emitirá o nosso evento personalizado para o elemento pai do nosso componente.

```html
<!-- Template do componente com o EventEmitter -->

<button (click)="emitirEventoPersonalizado()">Clique aqui!</button>
```

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

@Component({
  selector: 'meu-componente',
  templateUrl: 'meu-componente.component.html',
  stylesUrls: ['meu-componente.component.css']
})
export class MeuComponenteComponent {

  @Output()
  meuEvento: EventEmitter<any> = new EventEmitter<any>()
  /*
    o nome do evento será o mesmo nome da propriedade,
    nesse caso, o nome do evento será "meuEvento"
  */

  emitirEventoPersonalizado() {
    this.meuEvento.emit()
  }
}
```

O método *emit()* da propriedade **meuEvento** faz a emissão do evento personalizado para o seu elemento pai. Assim, para ouvir esse evento no elemento pai, basta utilizar o conceito de Event Binding visto anteriormente.

```html
<!-- Template de outro componente -->

<meu-component (meuEvento)="f()"></meu-component>
```

Você pode também colocar outro nome para o seu evento que não seja o mesmo nome da

propriedade. Basta que, dentro dos parênteses do Decorator *Output* você coloque uma string com o nome que deseja para o evento.

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

@Component({
  selector: 'meu-componente',
  templateUrl: 'meu-componente.component.html',
  stylesUrls: ['meu-componente.component.css']
})
export class MeuComponenteComponent {

  @Output('eventoPersonalizado')
  meuEvento: EventEmitter<any> = new EventEmitter<any>()
  /*
    o nome do evento será o mesmo nome da propriedade,
    nesse caso, o nome do evento será "meuEvento"
  */

  emitirEventoPersonalizado() {
    this.meuEvento.emit()
  }
}
```

Basta agora utilizar esse novo nome no elemento pai.

```ts
<!-- Template de outro componente -->

<meu-component (eventoPersonalizado)="f()"></meu-component>
```