Lifecycle hooks

- Component instance has lifecycle hooks which can help you to hook into different events on components.

- Lifecycle ends when component is destroyed

Lifecycle HOOKS - Every hook has a interface for it

* ngOnChanges
* ngOnInit
* ngDoCheck
* ngAfterContentInit
* ngAfterContentChecked
* ngAfterViewInit
* ngAfterViewChecked
* ngOnDestroy

These are called after a compcnent is intialized

- Constructors should not have any blocking code it should be in ngOnInit

* Component communication:

- to facilitate communication b/w components.

Three ways -

1. Using @Input and @Output
2. Using @ViewChild and @ContentChild
3. Using services

1. @Input and @Output

Lets say 2 have 2 components rooms and rooms-list and you want to pass a list of rooms from rooms component to rooms-list and display them in table there.

In rooms-list-component.ts

```
export class RoomsListComponent implements OnInit {

→ @Input() rooms: RoomList[] = [];
make this rooms property as valid html property on app-roomlist selector html element
  constructor() { }

  ngOnInit(): void {
```

}

Now in rooms-list-html.
change name from RoomList to room.

Now in your parent component ie- rooms
Inside rooms.component.html

<app-roomlist [rooms] = "roomlist" > </ app-roomlist>

Rooms ——— Parent / Smart component
- knows where to get data from

Roomslist ——— Child / Dumb component
just needs to know what to render

Since the child component does not know how to perform actions it can ask parent to do by passing data back to parent. Here is where we need @Output

In rooms-list-component.ts

@Input rooms : RoomList[] = [];
@Output() selectedRoom = new EventEmitter <RoomLists>();

selectRoom(room: RoomList) {
this.selectedRoom.emit(room) - emit/give data back to parent who has subscribed to this event
}

In room-list_component.html
Add selectRoom option for rooms in table and on Click call to selectRoom(room)

Inside room.component.ts
selectRoom (room : RoomList) {
this.selectedRoom = room;
console.log (room | json);
}

In room.component-html

<app-rooms-list [rooms] = "roomlist" always $event
(selectedRoom) = "selectRoom($event)" ></app-rooms-list>

event which will be received from child

You can also print the details on your webpage. Above code will log on console.

* Change Detection

- It is the process through which Angular checks to see whether your application state has changed and if any DOM needs to be updated

- Angular runs its change detection periodically so that changes in data model are reflected in an application's view.

- change detection can be triggered either manually or through an asynchronous event.

- By default, angular will run change detection on each and every element.

- You can optimize this by skipping parts of your application and running change detection only when necessary.

In your rooms-list component.ts
Under @ Component ({

changeDetection : ChangeDetectionStrategy.OnPush

})

1) ChangeDetectionStrategy.OnPush → only when asked to detect
2) ——————————— . Default → always look for change on all events

* Before using OnPush you have to satisfy below requisite-
* — OnPush can be applied only in case I am not modifying some data internally in the component. How? Use Input and Output

* — Immutability - we should always return a new instance. In case you are modifying the object return a new instance.

✗ this.roomlist.push(room); → modifying in place

✓ this.roomlist = [... this.roomlist, room];

spread operator.

assigns the previous list + new el to the object

* ngOnchanges:

- can be applied only on compenent/directive which has Input property

```
export class roomlist implements OnChanges {

  ngOnchanges (changes : SimpleChanges): void {

      console.log (changes)
      if (changes['title'])
      {
          this.title =  changes['title'].currentValue.toUpperCase();
      }
  }
```

* ngDoCheck:

- hardly used, very costly
- will be executed everytime you raise an event irrespective of where this component is implemented, it will listen to any changes that have happened inside your entire application.

- DO NOT USE with NgOnChange on same component

- can be used to detect some changes which are not controlled anywhere.

* View child :- Decorator