ANGULAR 2

COMPONENTS COMMUNICATION VIA @VIEWCHILD AND SERVICE

LUXOFT

LXFT LISTED NYSE

DIRECT PARENT-CHILD COMMUNICATION WITH @VIEWCHILD

PARENT COMPONENT

```
@Component(
   template: "<child-component #child></ child-component >")
class ParentComponent {
  @ViewChild("child")
  childComponent: ChildComponent;

   callChild() {
        childComponent.someMethod();
  }
}
```

CHILD COMPONENT

```
@Component( selector: '[child-component]' )
class ChildComponent {
     someMethod() { … }
}
```

PARENT INTERACTS WITH CHILD VIA LOCAL VARIABLE

```
@Component({
  selector: 'countdown-parent-lv',
  template: `
      <h3>Countdown to Liftoff (via local variable)</h3>
     <button (click)="timer.start()">Start</button>
     <button (click)="timer.stop()">Stop</button>
      <div class="seconds">{{timer.seconds}}</div>
     <countdown-timer #timer></countdown-timer>
 `,
  styleUrls: ['demo.css']
})
export class CountdownLocalVarParentComponent { }
```

We can place a local variable (#timer) on the tag (<countdown-timer>) representing the child component. That gives us a reference to the child component itself and the ability to access any of its properties or methods from within the parent template.

NOTE: The parent component itself has no access to the child.

PARENT INTERACTS WITH CHILD VIA LOCAL VARIABLE

```
@Component({ selector: 'countdown-timer',  template: '<p>{{message}}</p>'})
export class CountdownTimerComponent implements OnInit, OnDestroy {
   intervalId = 0;
   message = '';
   seconds = 11;
   clearTimer() { clearInterval(this.intervalId); }
   ngOnInit()    { this.start(); }
   ngOnDestroy() { this.clearTimer(); }
   start() { this.countDown(); }
   stop()  { this.clearTimer(); this.message = `Holding at T-${this.seconds} seconds`; }
   private countDown() {
      this.clearTimer();
      this.intervalId = window.setInterval(() => {
         this.seconds -= 1;
         if (this.seconds === 0) { this.message = 'Blast off!';
         } else {
            if (this.seconds < 0) { this.seconds = 10; } // reset
            this.message = `T-${this.seconds} seconds and counting`;
         } }, 1000);
}   }
```

PARENT CALLS A VIEWCHILD – INJECTION BY TYPE

We can't use the local variable to access from parent component to child.
In this case we can use @ViewChild

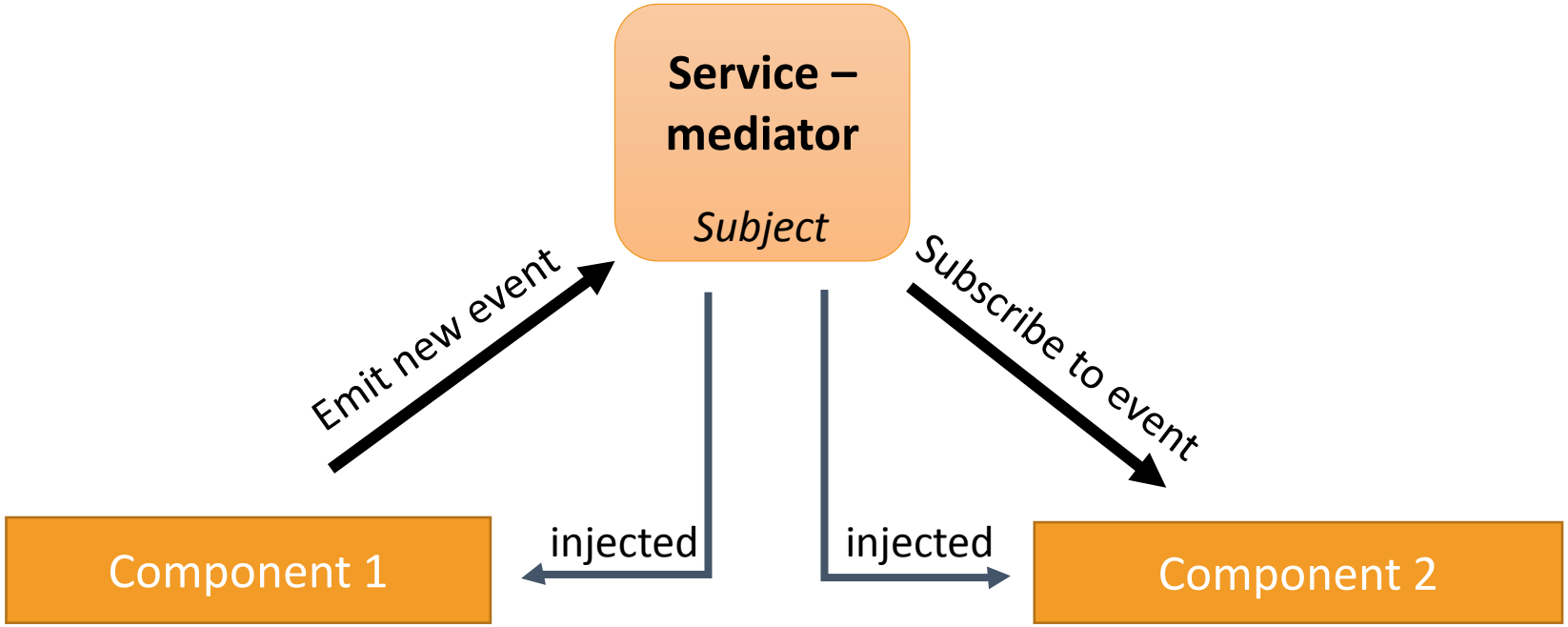
```
@Component({
  selector: 'countdown-parent-vc',
  template: `
    <h3>Countdown to Liftoff (via ViewChild)</h3>
    <button (click)="start()">Start</button>
    <button (click)="stop()">Stop</button>
    <div class="seconds">{{ seconds() }}</div>
     <countdown-timer></countdown-timer>`
})
export class CountdownViewChildParentComponent implements AfterViewInit {
  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;
  seconds() { return 0; }
  ngAfterViewInit() { setTimeout(() =>
                        this.seconds = () => this.timerComponent.seconds, 0); }
  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}
```

PARENT CALLS A VIEWCHILD – INJECTION BY NAME

We can't use the local variable to access from parent component to child.
In this case we can use @ViewChild

```
@Component({
  selector: 'countdown-parent-vc',
  template: `
    <h3>Countdown to Liftoff (via ViewChild)</h3>
    <button (click)="start()">Start</button>
    <button (click)="stop()">Stop</button>
    <div class="seconds">{{ seconds() }}</div>
     <countdown-timer #timer></countdown-timer>`
})
export class CountdownViewChildParentComponent implements AfterViewInit {
  @ViewChild("timer")
  private timerComponent: CountdownTimerComponent;
  seconds() { return 0; }
  ngAfterViewInit() { setTimeout(() =>
                        this.seconds = () => this.timerComponent.seconds, 0); }
  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}
```

COMPONENT COMMUNICATION VIA SERVICE



PARENT AND CHILDREN COMMUNICATE VIA A SERVICE

```
@Injectable()
export class MissionService {
  // Observable string sources
  private missionAnnouncedSource = new Subject<string>();
  private missionConfirmedSource = new Subject<string>();
  // Observable string streams
  missionAnnounced$ = this.missionAnnouncedSource.asObservable();
  missionConfirmed$ = this.missionConfirmedSource.asObservable();
  // Service message commands
  announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }
  confirmMission(astronaut: string) {
    this.missionConfirmedSource.next(astronaut);
  }
}
```

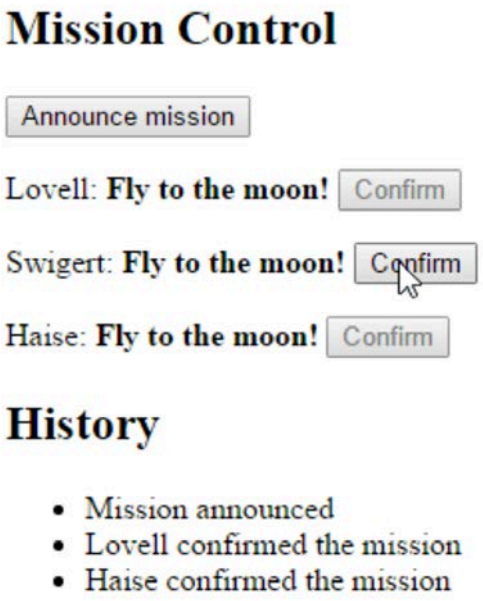
PARENT AND CHILDREN COMMUNICATE VIA A SERVICE

A parent component and its children share a service whose interface enables bi-directional communication within the family.

```
@Injectable()
export class MissionService {
  // Observable string sources
  private missionAnnouncedSource = new Subject<string>();
  private missionConfirmedSource = new Subject<string>();
  // Observable string streams
  missionAnnounced$ = this.missionAnnouncedSource.asObservable();
  missionConfirmed$ = this.missionConfirmedSource.asObservable();
  // Service message commands
  announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }
  confirmMission(astronaut: string) {
    this.missionConfirmedSource.next(astronaut);
  }
}
```

PARENT AND CHILDREN COMMUNICATE VIA A SERVICE

```
@Component({
  selector: 'mission-control',
  template: `<h2>Mission Control</h2>
  <button (click)="announce()">Announce mission</button>
  <my-astronaut *ngFor="let astronaut of astronauts"
   [astronaut]="astronaut”></my-astronaut>
  <h3>History</h3>
  <ul><li *ngFor="let event of history">{{event}}</li></ul>`})
export class MissionControlComponent {
  astronauts = ['Lovell', 'Swigert', 'Haise'];
  history: string[] = [];
  missions = ['Fly to the moon!', 'Fly to mars!', 'Fly to Vegas!'];
  nextMission = 0;
  constructor(private missionService: MissionService) {
    missionService.missionConfirmed$.subscribe( astronaut => {
        this.history.push(`${astronaut} confirmed the mission`); });}
  announce() {
    let mission = this.missions[this.nextMission++];
    this.missionService.announceMission(mission);
    this.history.push(`Mission "${mission}" announced`);
    if (this.nextMission >= this.missions.length) { this.nextMission = 0; }
}  }
```



PARENT AND CHILDREN COMMUNICATE VIA A SERVICE

```
@Component({   selector: 'my-astronaut',
  template: `<p> {{astronaut}}: <strong>{{mission}}</strong>
    <button (click)="confirm()"
    [disabled]="!announced || confirmed">Confirm</button></p>`})
export class AstronautComponent implements OnDestroy {
  @Input() astronaut: string;
  mission = '<no mission announced>';
  confirmed = false;
  announced = false;
  subscription: Subscription;
  constructor(private missionService: MissionService) {
    this.subscription = missionService.missionAnnounced$.subscribe(
      mission => {
        this.mission = mission;
        this.announced = true;
        this.confirmed = false;
      });
  }
  confirm() { this.confirmed = true; this.missionService.confirmMission(this.astronaut); }
  ngOnDestroy() {
     this.subscription.unsubscribe(); // prevent memory leak when component destroyed
}  }
```

Swigert: **Fly to the moon!** Confirm