Child to parent communication

First example

childComp

```
<template>
    <lightning-card title="Child component shows value">
        <h1 style="background-color:aquamarine;">
        Show value of child parent:{myName}


        </h1>



    </lightning-card>

</template>
import { LightningElement,api } from 'lwc';

export default class ChildComp extends LightningElement {

    @api myName=" child Value from child comp";
}
```

.....

parentComp

```
<template>
    <lightning-card title="Parent component">
<h1 style="color:blue">
    <c-child-comp></c-child-comp>
</h1>
```

```
    </lightning-card>
</template>
```

Nothing in Js

.....

Second example Passing a value to child from parent

By using parameter and that always be public property.

No change in childComp

```
<template>
    <lightning-card title="Parent component">
        <h2 style="color:brown">
        This my private property value :{parentValue} <br/>
        </h2>
        <h1 style="color:blue">

    <c-child-comp my-name={parentValue}></c-child-comp>
</h1>

<div class="slds-box">
    <lightning-button label="change a value" onclick={handle}></lightning-
button>
</div>
    </lightning-card>
</template>
import { LightningElement,track} from 'lwc';

export default class ParentComp extends LightningElement {
@track parentValue="Value is in parent";

handle(){
this.parentValue="This is parent value showung in child component";
```

```
}
}
```

.....

Third example

How to call Public method created in a child using parent component

Changes in childCompjs file

```
import { LightningElement,api } from 'lwc';


export default class ChildComp extends LightningElement {


    @api myName=" child Value from child comp";
    //for using public property user have to import api plus decorator
@api
    //public property and component when calling convert into kabab case


    @api testChildMethod(parentParam)
{
alert('calling child method '+parentParam.firstName);
}
}
```

Changes in parentComp.html

```
<template>
    <lightning-card title="Parent component">
        <h2 style="color:brown">
        This my private property value :{parentValue} <br/>
        </h2>
        <h1 style="color:blue">
```

```
    <c-child-comp my-name={parentValue}></c-child-comp>
</h1>


<div class="slds-box">
    <lightning-button label="change a value" onclick={handle}></lightning-button>
    <lightning-button label="call a method" onclick={handlecall}></lightning-button>
</div>
    </lightning-card>
</template>
```

Changes in parentCompjs

```
import { LightningElement,track} from 'lwc';

export default class ParentComp extends LightningElement {
@track parentValue="Value is in parent";

handle(){
this.parentValue="This is parent value showung in child component";

}

handlecall(){
var childCompvar=this.template.querySelector('c-child-comp');
var sendParam={'firstName':'Tantul'};
childCompvar.testChildMethod(sendParam);
```

```
}


}
```

.....

Implement how to change multiple HTML files and using of render method

Lifecylehook.html

```
<template>

    <div>Inside First Template</div>

    <lightning-button label="gO TO SECOND TEMPLATE"
onclick={onchange}></lightning-button>

</template>


<template>

    <div>inside second temp</div>

    <lightning-button label="gotofirsttemplate"
onclick={onchange}></lightning-button>

</template>

import { LightningElement,api } from 'lwc';

import firsttemplate from './lifehook.html';

import secondtemplate from './lifehook2.html';


export default class Lifehook extends LightningElement {


    templatenumber = 'temp1';


    constructor()

    {

        super();

        console.log('inside constructor life cysle hook');

    }
```

```
connectedCallback()
{
    console.log('inside connected call back');

}
disconnectedCallback()
{
    console.log('inside  disconnected call back');
}

onchange()
{
    console.log('Inside change template');

    if(this.templatenumber==='temp1')
    {
    this.templatenumber='temp2';

    }
    else
    {
    this.templatenumber='temp1';
    }
    }

    render()
    {
    console.log('Inside render');
    if(this.templatenumber==='temp1')
    return firsttemplate;
    else return secondtemplate;
```

```
        }

 }
```

.....

Example

```
<template>
    Parent Component Message:- { Message}
    </template>
import { LightningElement, track, api } from 'lwc';
export default class ChildComponent extends LightningElement {
@track Message;
@api
changeMessage(strString) {
this.Message = strString.toUpperCase();
}
}


<template>
    <lightning-card title="Parent lightning card">
        <lightning-layout-item flexibility="auto" padding="around-small">
            <lightning-input  label="Enter the message"
onchange={handleChangeEvent}></lightning-input>




      <c-child-lwc></c-child-lwc>
    </lightning-layout-item>
    </lightning-card>
```

```
</template>
```

```
import { LightningElement } from 'lwc';
```

```
export default class ParentComponent extends LightningElement {
```

```
handleChangeEvent(event){
```

```
this.template.querySelector('c-child-
lwc').changeMessage(event.target.value);
```

```
}
```

```
}
```

Task1: Create contact list and add in a parent component

Task2:Multiple rendering change the background color and color of text.

Temp1<green:blue> sldx-box

Hello how are you doing

Temp2<orange:yellow> sldx-box

Hello how are you doing

Change Colour

Task3: lifecycle hook phases

childComp1

childComp2

ParentComp →<c-child-comp1> <c-child-comp2>

Write lifecycle hook [don’t use render]

Task2

<template>

<h1 style="font-size:50px">How are you doing</h1>

```
    <lightning-button label="gO TO SECOND TEMPLATE"
onclick={onchange}></lightning-button>

</template>

<template>

    <div class="slds-box"  style="border: 4px solid black; background-
color:rgb(30, 4, 122) ;color:rgb(219, 29, 146);"><h1 style="font-
size:50px">Hello how are you doing

    </h1> </div>

    <lightning-button label="gotofirsttemplate"
onclick={onchange}></lightning-button>

</template>

import { LightningElement,api } from 'lwc';

import firsttemplate from './lifehook.html';

import secondtemplate from './lifehook2.html';









export default class Lifehook extends LightningElement {


    templatenumber = 'temp1';


    constructor()

    {

        super();
```

```
    console.log('inside constructor life cysle hook');
}
connectedCallback()
{
    console.log('inside connected call back');

}
disconnectedCallback()
{
    console.log('inside  discon
```