04-03 Componenti e stato

Componente funzionale:

- Semplice
- Funzione pura
  - No stato
  - No side effects
- No ciclo di vita
- Può definire handler functions

In React si hanno gli **Hooks** in modo tale da aggirare queste limitazioni in maniera controllata:

- Hook per stato
- Hook per side effect
- Hook per contesto

Bisogna definire un **Hook per ogni funzionalità richiesta**, Hook è una funzione:

Hook	Purpose
`useState`	Define a state variable in the component
`useEffect`	Define a side-effect during the component lifecycle
`useContext`	Act as a context consumer for the current component
`useReducer`	Alternative to useState for Redux-like architectures or complex state logic
`useMemo`	"Memoizes" a value (stores the result of a function and recomputes it only if parameters
`useCallback`	Creates a callback function whose value is memoized
`useRef`	Access to childrens' ref properties
`useLayoutEffect`	Like useEffect, but runs after DOM mutations
`useDebugValue`	Shows a value in the React Developer Tools

useState ci serviva per ricordare lo stato per ogni componente.

Breve riassunto componenti react:

- Componente react fa parte di un element tree quando viene renderizzato
  - **Props**: porzioni di dati immutabili per passaggio di informazioni da genitori a figli
    - Tramite le props passa determinate informazioni ai componenti figli
    - Riceve delle props, può leggere, visualizzarle, usarle ma non cambiarle
      - Read-only
      - ogni attributo in JSX è convertibile in una props
      - Possono essere qualunque oggetto javascript
        - Inclusi elementi react

```
– <Header headerText='Hello'/>
– props.headerText will contain the string "hello"
```

  - **Contesto**: props globale e implicita, passata automaticamente a tutti i componenti
    - per passare la stessa informazione ± contemporaneamente a tanti componenti
      - Es: cambio di lingua, devo passarlo a tutti i componenti che hanno del testo
      - Es: dark mode
  - **Stato**: dove il componente salva le proprie informazioni localmente
    - se cambia lo stato → componente si ri-renderizza
    - Può essere letto e modificato solo dal componente stesso
    - Vive dentro il componente
      - Si può mandare il valore attuale dello stato di un componente tramite props agli altri componenti

Hook useState

Variabile [nomeStato, nomeFunzionePerSetStato]

- setStato → rimpiazza l'intero stato
  - conseguente re-rendering
  - Applica modifiche in maniera asincrona

- `import{ useState } from 'react';`
- `const [hidden, setHidden] = useState(true);`
  - Creates a new state variable
  - `hidden`: name of the variable
  - `setHidden`: update function
  - `true`: default (initial) value
  - Array destructuring assignment to assign 2 values at once

- Creates a state variable of any type
  - **Remembered** across function calls!
- The default value sets the initial *value* (and *type*)
- The variable name can be used inside the function (to affect rendering)
- The `setVariable()` function will replace the current state with the new one
  - And trigger a re-render

```
import React, { useState } from 'react';

function ShortText(props) {
 const [hidden, setHidden] = useState(true);
 return (
    <span>
      {hidden ?
        `${props.text.substr(0, props.
maxLength)}...` : props.text }
      {hidden ? (
        <a onClick={() => setHidden(false)}>more</a>
      ) : (
        <a onClick={() => setHidden(true)}>less</a>
      )}
    </span>
  );
}
```

```
function WelcomeButton(props) {
  let [english, setEnglish] =
    useState(true) ;

  return (<button>
      {english ? 'Hello' : 'Ciao'}
    </button>) ;
}
```

Nota: usare sempre la set, non modificare mai a mano

Aggiornamento dello stato:

- Usando nuovo valore
  - Dipende da props e valori costanti
  - Rimpiazza il corrente
  - Deve avere lo stesso tipo

```
setHidden(false) ;
```

- Con una funzione (callback) → se la logica dipende dalla precedente
  - Esegue callback
  - Nuovo stato dipendente da vecchio
  - Il valore di ritorno della funzione rimpiazzerà il nuovo stato
    - Deve ritornare un nuovo valore di stato
    - Non deve cambiare, deve tornare uno nuovo

```
setSteps(oldSteps => oldSteps + 1);
```

Cambi di stato sono spesso determinati **da eventi asincroni:**

- Gestore eventi DOM (onClick)
- Risposta server (api calls)

```
function WelcomeButton(props) {
  let [english, setEnglish] =
    useState(true) ;

  return (<button
    onClick={()=>setEnglish((eng)=>(!eng))}>
      {english ? 'Hello' : 'Ciao'}
    </button>);
}
```

oppure

```
function WelcomeButton(props) {
  let [english, setEnglish] =
    useState(true) ;

    const toggleLanguage = () => {
      setEnglish( e => !e ) ;
    }

  return (<button onClick={toggleLanguage}>
      {english ? 'Hello' : 'Ciao'}
    </button>);
}
```

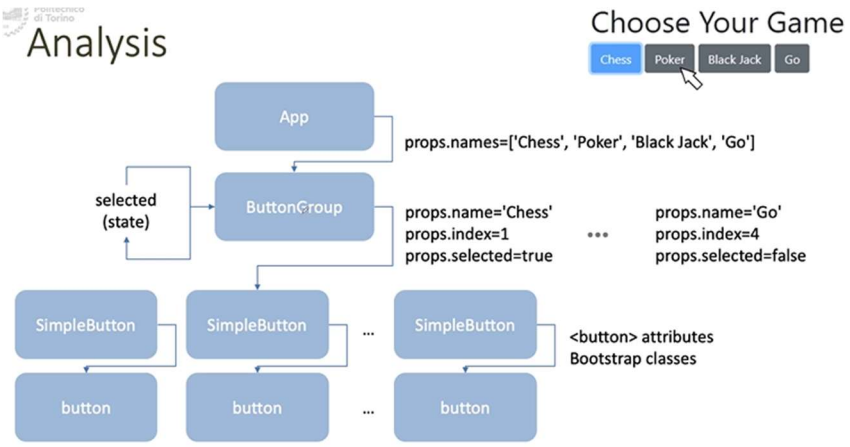
Valore di default (true), viene usato solo al primo rendering del componente (compreso refresh).

Un componente può **avere più stati**, indicando nomi diversi → da fare se ci sono proprietà non collegate tra loro

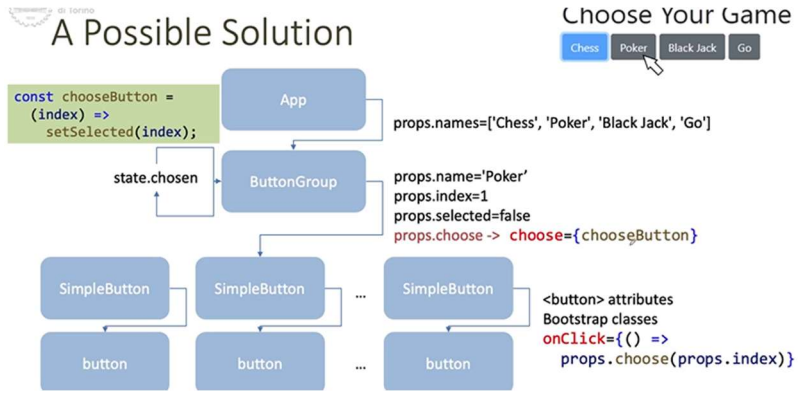
```
function Example(props) {

  [hidden, setHidden] = useState(true) ;
  [count, setCount] = useState(0) ;
  [mode, setMode] = useState('view') ;

  . . .
  setHidden(false) ;
  . . .
  setCount( c => c+1 ) ;
  . . .
  setMode('edit') ;
  . . .
}
```

➔ Componente si renderizza se uno dei suoi stati cambia

Componente figlio vuole cambiare stato padre



Ogni bottone deve avere una reference al metodo per cambiare lo stato del padre.



Nota generale: quando possibile è meglio creare componenti senza stato in quanto riutilizzabili e più veloci. Quello che si fa è portare lo stato più in alto (all'antenato più comune) in modo tale da avere lobstato sul padre e i figli possono leggere con le props, possono aggionrare con le callbacks.