

Design Patterns

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software **design**

Reusability & Separation of Concerns.

- **The DRY principle – Don't Repeat Yourself.**
- **Techniques to improve DRY(ness) (increase reusability):**
 1. **Inheritance** (is-a relationships, e.g. Car is an automobile)
 2. **Composition** (has-a relationships, e.g. Car has an Engine)
- **React favors composition.**
- **Core React composition Patterns:**
 1. **Containers.**
 2. **Render Props.**
 3. **Higher Order Components.**
- **The Hooks mechanism has diminished the need for these patterns.**

Composition - Children

- **HTML is composable**

```
<div>
  <h2>Some Heading</h2>
  <ul>
    <li> . . . . . </li>
    <li> . . . . . </li>
    <li> . . . . . </li>
  </ul>
</div>
```

```
<div>
  <p>.....</p>
  <img ..... />
  <a href ...../>
</div>
```

<div> has three children.

- **<div> has two children; has three children**


The Container pattern.

All React components have a special children prop so that consumers can pass components directly by nesting them inside the jsx.

```
const Picture = (props) => {  
  return (  
    <div>  
      <img src={props.src}/>  
      {props.children}  
    </div>  
  )  
}
```



```
const OtherComponent = props => {  
  return (  
    <div className='container'>  
      <Picture src={picture.src}>  
        // what is placed here is  
        // passed as props.children  
      </Picture>  
    </div>  
  )  
}
```



- The Picture component renders (a) an image and (b) the JSX the consumer binds to props.children - JSX between the opening and closing tags of Picture.
- This de-couples the Picture component from its content and makes it reusable.

```
const OtherComponent1 = props => {
  return (
    <div className='container'>
      <Picture src={picture.src}>
        <button>.....</button>
      </Picture>
    </div>
  )
}
```

Image

Button

```
const OtherComponent2 = props => {
  return (
    <div className='container'>
      <Picture src={picture.src}>
        <ul>. . . . .</ul>
      </Picture>
    </div>
  )
}
```

Image

List

Picture is
composed
with other
elements /
components

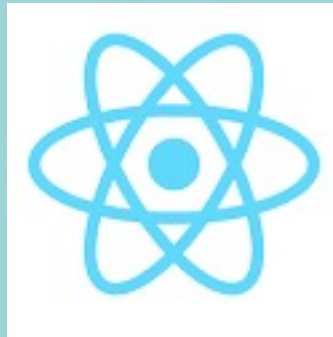
```
const OtherComponent3 = props => {
  return (
    <div className='container'>
      <Picture src={picture.src}>
        <ComplexComponent>
          . . . . .
        </ComplexComponent>
      </Picture>
    </div>
  )
}
```

Image

Complex
Component

Design patterns

- **More later.**



Custom Hooks

Custom Hooks.

- Custom Hooks let you extract component logic into reusable functions.
- Improves code readability and modularity.

Example:

```
const BookPage = props => {  
  const isbn = props.isbn;  
  const [book, setBook] = useState(null);  
  useEffect(() => {  
    fetch(  
      `https://api.for.books?isbn=${isbn}`  
    ).then(res => res.json())  
      .then(book => {  
        setBook(book);  
      });  
  }, [isbn]);  
  . . . rest of component code . . .  
}
```

Objective – Extract the book-related state code into a custom hook.

Custom Hook Example.

Solution:

```
const useBook = isbn => {  
  const [book, setBook] = useState(null);  
  useEffect(() => {  
    fetch(  
      `https://api.for.books?isbn=${isbn}`  
    ).then(res => res.json())  
    .then(book => {  
      setBook(book);  
    });  
  }, [isbn]);  
  return [book, setBook];  
};
```

```
const BookPage = props => {  
  const isbm = props.isbn;  
  const [book, setBook] = useBook(isbn);  
  
  . . . .rest of component code . . . .  
}
```

- Custom Hook is an ordinary function BUT should only be called from a React component function.
- Prefix hook function name with `use` to leverage linting support.
- Function can return any collection type (array, object), with any number of entries.

