



# 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.**

# 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; <ul> 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>  
  )  
}
```

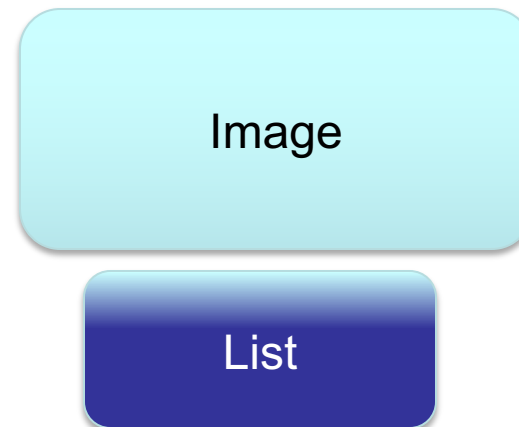
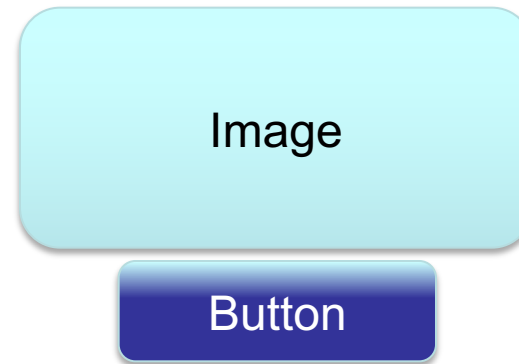


- When the Picture component renders, its `props.children` will display what the consumer places between the opening and closing tags of Picture.
- This de-couples the Picture component from its content and makes it reusable.

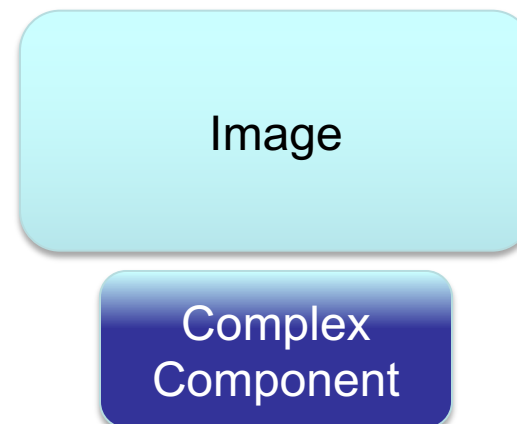
```
const OtherComponent1 = props => {
  return (
    <div className='container'>
      <Picture src={picture.src}>
        <button>.....</button>
      </Picture>
    </div>
  )
}
```

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

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



Picture is **composed** with other elements / components



# The Render Prop pattern

- **Use the pattern to share logic between components.**
- **Dfn.:** A render prop is a function prop that a component uses as part of its rendered output.

```
const SharedComponent = (props) => {  
    
  return (  
    <div className="classX"  
      onMouseOver={funcY}  
      { props.render() }  
    </div>  
  );  
};
```

- SharedComponent **receives its render logic from the consumer, i.e. SayHello.**
- Prop name is arbitrary.

```
const SayHello = (props) => {  
  return (  
    <SharedComponent render={() =>  
      <span>Say Hello</span>  
    } />  
  )  
};
```

```
<div className="classX"  
  | | | | onMouseOver={funcY} >  
  | | <span>Say Hello</span>  
</div>
```

# The Render Prop - Sample App.

## Friends List

- **Jeff Herrera**



- **Michele Denis**



- **Annefleur Hop**



- **Brayden Rice**

## Friends List

- **Önal Kılıççı**

[onal.kilicci@example.com](mailto:onal.kilicci@example.com)

- **Thomas Chen**

[thomas.chen@example.com](mailto:thomas.chen@example.com)

- **Magda Vieira**

[magda.vieira@example.com](mailto:magda.vieira@example.com)

- **Vilma Heikkila**

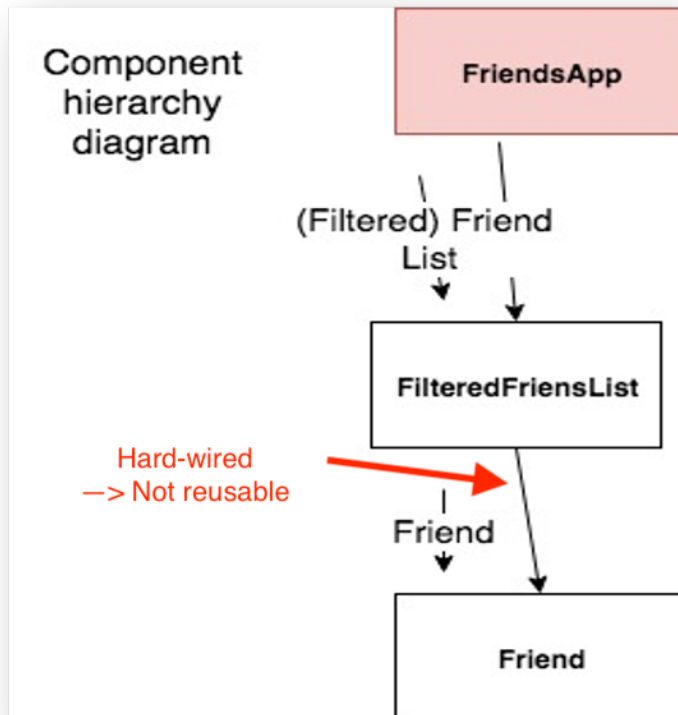
[vilma.heikkila@example.com](mailto:vilma.heikkila@example.com)

- **Herman Hessen**

[herman.hessen@example.com](mailto:herman.hessen@example.com)

- **Sandra Bell**

# The Render Props - Sample App.



- **Updated Solution:** FriendsApp passes a second prop to FilteredFriendList, indicating how Friends should be rendered.
- **New prop is a render prop**
-



```


<FilteredFriendList
  list={filteredList}
  render={(friend) => <FriendImage friend={friend} />}
/>

```

```

1  import React from "react";
2  You, 5 days ago • Initial structure
3  const FilteredFriendList = props => {
4    // console.log('Render of FilteredFriendList')
5    const friends = props.list.map(item => (
6      props.render(item)
7    ));
8    return <ul>{friends}</ul>;
9  };
10
11  export default FilteredFriendList;
12

```



```

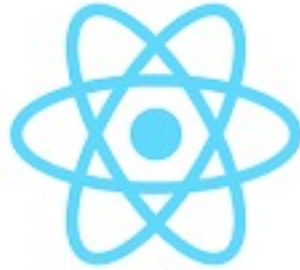
<FilteredFriendList
  list={filteredList}
  render={(friend) => <FriendContact friend={friend} />}
/>

```

- FilteredFriendList is no longer statically importing the component for rendering a friend.
- It now receives the component type via the render prop.
- The prop computes the type of elements in friends array.

- Without this pattern we would need a FilteredFriendList component for each use case, thus violating the DRY principle.

- The prop name is arbitrary; render is a convention.



# 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 can 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.

