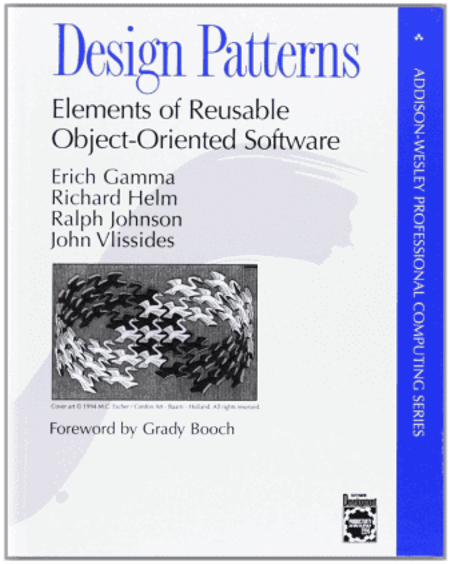

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Design Patterns

(Continues)

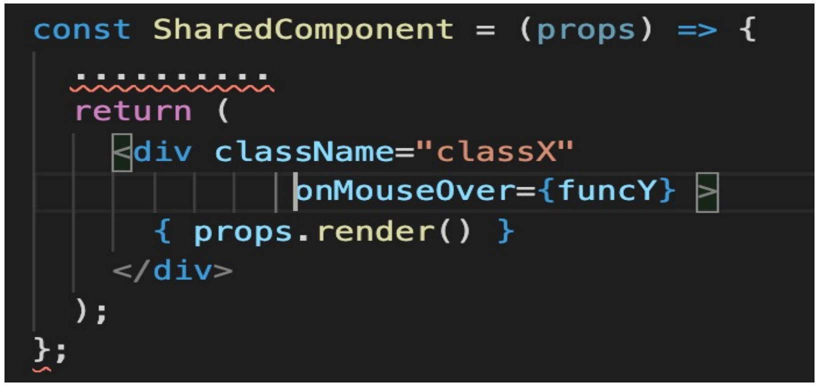
Reusability & Separation of Concerns.

- **The DRY principle – Don't Repeat Yourself.**
- **Techniques to improve DRY(ness) (increase reusability):**
  1. **Inheritance** (is-a **relationships, e.g. Car is an automabile)**
  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.**

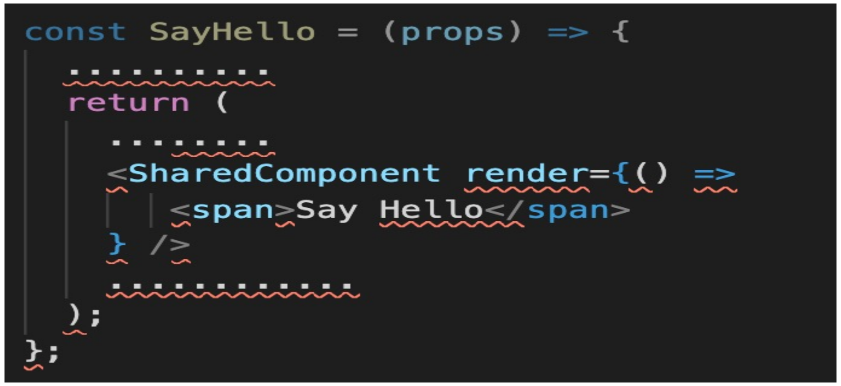
The Render Prop pattern

- **Use the pattern to share logic between components.**
- **Dfn.**: A render prop is a function prop that a component uses to know what to render.

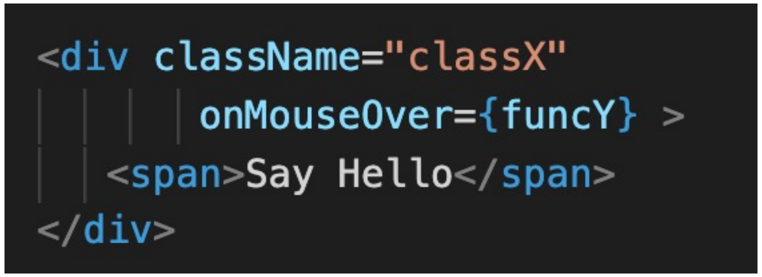


```
const SharedComponent = (props) => {
  ..........
  return (
    <div className="classX"
         onMouseOver={funcY} >
      { props.render() }
    </div>
  );
};
```

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

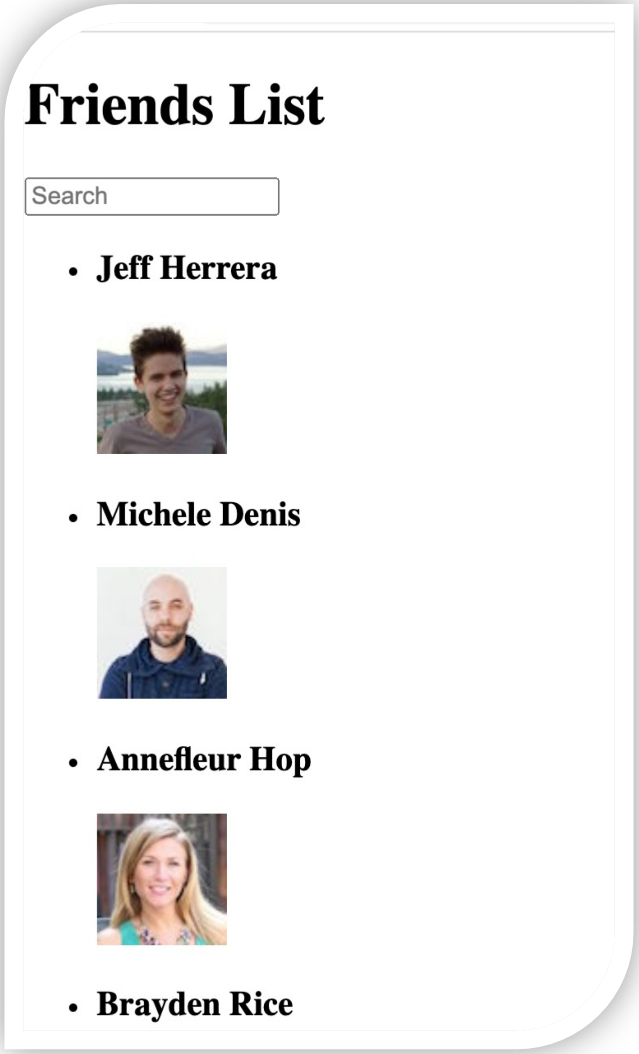


```
const SayHello = (props) => {
  ..........
  return (
    ........
    <SharedComponent render={() =>
      <span>Say Hello</span>
    } />
    .............
  );
};
```



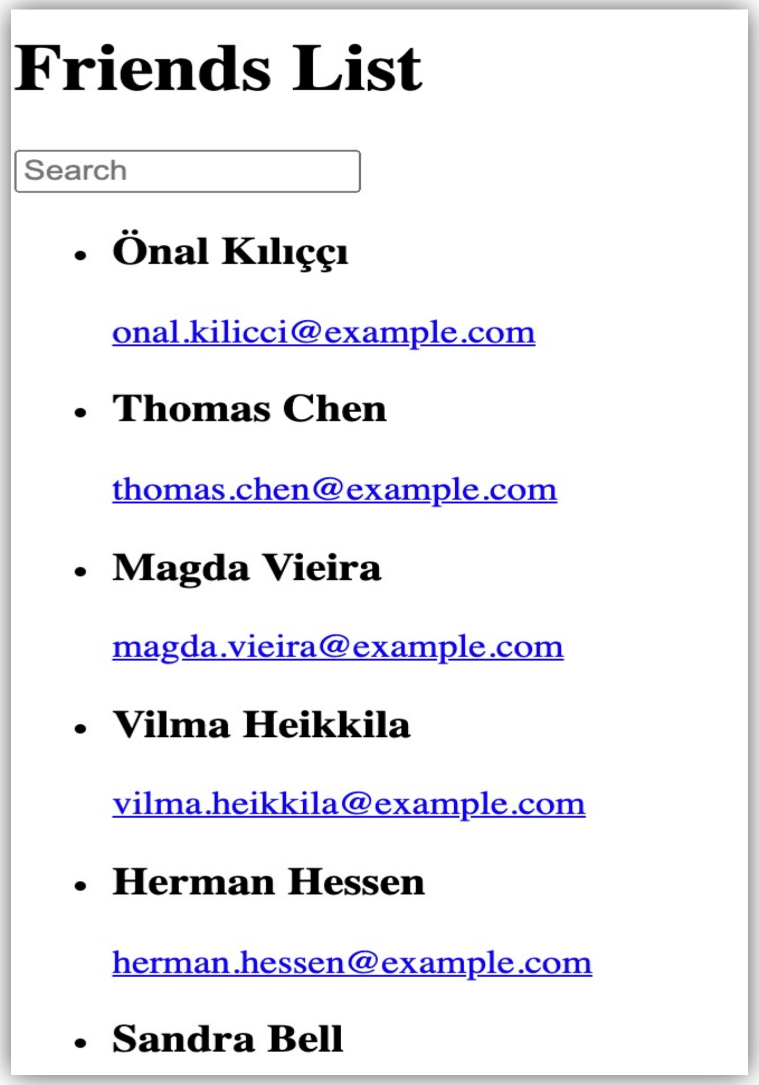
```
<div className="classX"
         onMouseOver={funcY} >
    <span>Say Hello</span>
</div>
```

The Render Prop - Sample App.


Friends List

Search

- Jeff Herrera
- Michele Denis
- Annefleur Hop
- Brayden Rice


Friends List

Search

- Önal Kılıççı

  onal.kilicci@example.com
- Thomas Chen

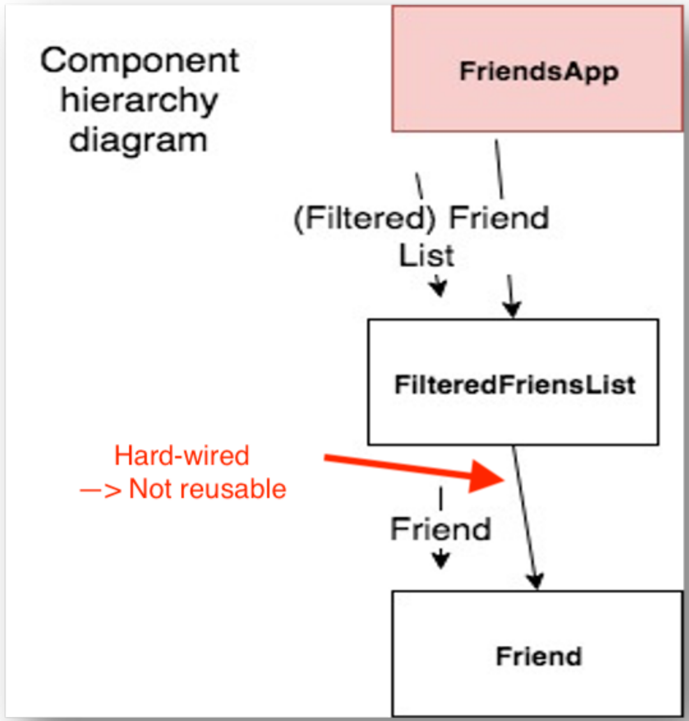
  thomas.chen@example.com
- Magda Vieira

  magda.vieira@example.com
- Vilma Heikkila

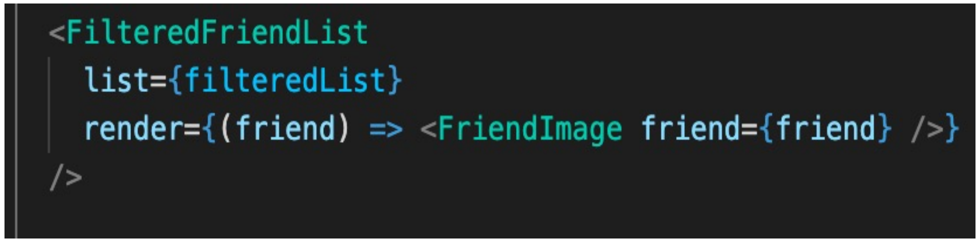
  vilma.heikkila@example.com
- Herman Hessen

  herman.hessen@example.com
- Sandra Bell

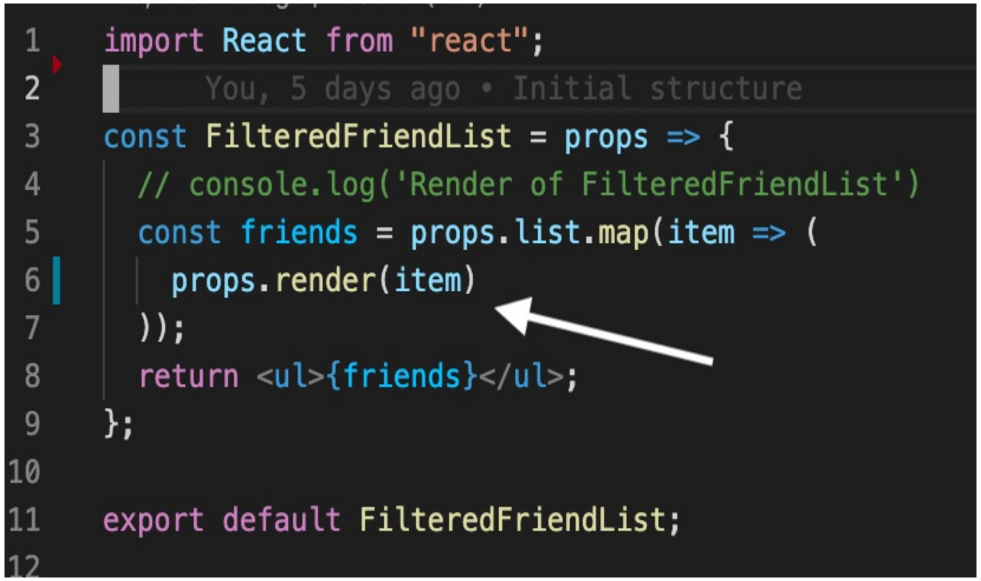
The Render Props - Sample App.



- **Solution: As well as passing the list of matching friends to**
- **, we also tell it <u>how to render a friend</u>**
- **Use a prop to communicate the 'how', i.e. a render prop**

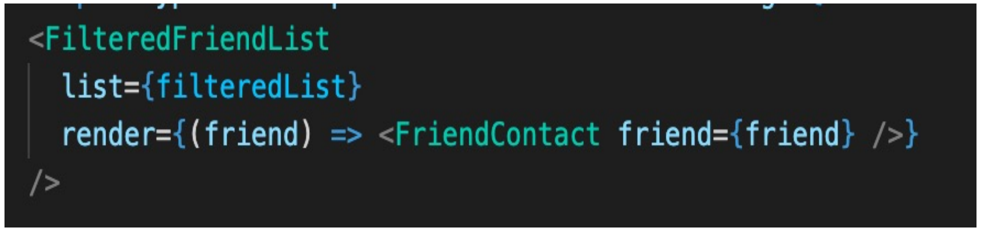


```
<FilteredFriendList
  list={filteredList}
  render={(friend) => <FriendImage friend={friend} />}
/>
```



```
import React from "react";
      You, 5 days ago • Initial structure
const FilteredFriendList = props => {
  // console.log('Render of FilteredFriendList')
  const friends = props.list.map(item => (
    props.render(item)
  ));
  return <ul>{friends}</ul>;
};

export default FilteredFriendList;
```



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

- FilteredFriendList **is no longer statically importing the component for rendering a friend.**
- **It receives this via the render prop.**
- **The friends array elements will be** Friend **components, e.g.** FriendContact, FriendImage

- **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.**

Reusability.

- **Core React composition Patterns:**
  1. **Containers**
  2. **Render Props**
  3. **Higher Order Components.**

- **HOC is a function that takes a component and returns an enhanced version of it.**
  - **Enhancements could include:**
    - **Statefulness**
    - **Props**
    - **UI**
- **Ex –** withRouter **function.**
- **Naming convention:** withXXXXXXXX**()**

Summary.

- **Objectives – Reusability, Separation of Concerns (Single Responsibility), DRY.**
- **Benefits - Maintainability, Understandability, Extendability, Adaptability.**
- **Means – Apply design patterns.**

- **React App.**
  - **Composition.**
  - **Patterns – Container, Render Prop, Higher Order Component.**

- **More patterns later**