# React intermedio

## Abstracción de funcionalidades

### Render props

# ÍNDICE

# Uso de los render props

```
class Library extends React.Component {
    constructor(props) {
      super(props);
      this.state = { books: [] };
    }
    componentDidMount() {
      booksService().then(resp => this.setState({ books: resp }));
    }
    render() {
      return (
        <ul>
          {
              books.map(book =>
                <li>{book.title} - {book.author}</li>
              )
          }
        </ul>
      )
    }
}
```

- El componente representa un proveedor, **pero su renderización es fija**
- La reusabilidad del componente **es mínima**
- Desligar la lógica de la visualización es **un objetivo de las render props**

# Uso de los render props

```
class Book extends React.Component {
    constructor(props) {
      super(props);
      this.goToBook = this.goToBook.bind(this);
    }
    goToBook(e) {
      e.preventDefault();
      const { book } = this.props;
      this.props.history.push(`/books/${book.id}`);
    }
    render() {
      const { book } = this.props;
      return (
        <div className="book">
          <h3>{ book.title }</h3>
          <h4>{ book.author }</h4>
          <button onclick={this.goToBook}>Ir al libro</button>
        </div>
      )
    }
}
```

# Uso de los render props

```
class Library extends React.Component {
    constructor(props) {
      super(props);
      this.state = { books: [] };
    }
    componentDidMount() {
      booksService().then(
          resp => this.setState({ books: resp }));
    }
    render() {
      return (
        <>
          {
            books.map(
                book => this.props.render(book)
            )
          }
        </>
      )
    }
}
```

```
const App = () => {
    return (
        <div className="book-list">
            <Library render={ book => (
                <Book book={book}/>
            )}/>
        </div>
    )
}
```

- Esto renderizará los datos proveídos utilizando el **componente Book** que hemos definido para ello

# Render props para preocupaciones transversales

- Como habremos podido fijarnos, los render props **también pueden ocuparse de las preocupaciones transversales** de la aplicación
- Sin embargo utilizar esta solución nos ofrece una alternativa que va de **arriba hacia abajo** en vez de **abajo hacia arriba**
- Los HOCs y las render props se complementan, de hecho la mayoría de HOCs se pueden implementar usando render props

# Render props para preocupaciones transversales

```
const withBooks = component => {
  return class extends React.Component {
    render() {
      return (
        <Library render={book => (
          <Component {...this.props} book={book} />
        )}/>
      );
    }
  }
}
```

- Este HOC nos devolverá un fragmento **compuesto por la renderización usando Component de los libros en Library**

# Ejemplo de render props

```
class Mouse extends React.Component {
  constructor(props) {
    super(props);
    this.state = {mouseX: 0, mouseY: 0}
    this.handleMouse = this.handleMouse.bind(this);
  }
  handleMouse(e) {
    this.setState({
      mouseX: e.clientX,
      mouseY: e.clientY
    });
  }
  render() {
    return (
      <div style={{ height: '100vh' }} onMouseMove={this.handleMouseMove}>
        {this.props.render(this.state)}
      </div>
    );
  }
}
```

# Ejemplo de render props

```
const withMouse = Component => {
    class WithMouse extends React.Component{
        constructor(props) {
            super(props);
            this.state = {mouseX: 0, mouseY: 0}
            this.handleMouse = this.handleMouse.bind(this);
        }
        handleMouse(e) {
            this.setState({
                mouseX: e.clientX,
                mouseY: e.clientY
            });
        }
        render() {
            return (
                <div style={{ height: '100vh' }} onMouseMove={this.handleMouseMove}>
                    <Component mouse={this.state} {...this.props}/>
                </div>
            )
        }
    }
    WithMouse.displayName = `
        WithMouse(${getDisplayName(InputComponent.displayName || InputComponent.name ||
'Component')})
    `;
    return WithMouse;
}
```

# PARA RESUMIR

- Las **render props** son componentes que entran por propiedades de otros componentes y **realizan su renderización**, recibiendo parámetros del primero
- Tanto las **HOC** como las **render props** son usadas para solucionar preocupaciones transversales, **son complementarias y combinables**
- Las render props **no tienen que ser únicas por componente** y el nombre de la propiedad **puede ser arbitrario**