Conditional Rendering

Course: Java

S1

Conditional Rendering

Conditional rendering makes our components flexible. It operates on two levels:

01 On the low level...

Conditional rendering can include an image only if it has a valid URL, makes an input read-only, or chooses from one of two CSS classes.

Because JSX and JavaScript work so well together, we can embed small bits of JavaScript to add behavior that fine-tunes our UI.

02 On the high level...

A parent component can render entirely different component hierarchies based on a condition, or a component can choose not to render at all.

The **React Router** is a library of components that renders views with a declarative syntax.

Conditional Rendering (continued)

JSX is an expression.

It has an almost putty-like quality—it's moldable and flexible.

You can return it from a function, assign it to a variable, pass it to a function, map it, and intermingle it with your JavaScript.

Conditional Rendering (continued)

JSX and JavaScript live together in harmony.

This makes JSX an excellent template language.

There's no special template syntax
to represent loops and conditions—it's
just JavaScript.

Conditional Rendering (continued)

Other template engines require us to learn another syntax for repetition and decisions. They’re often hard to troubleshoot.

JSX is different.

Once we learn component and prop syntax, everything else is JavaScript.

Conditional Rendering (continued)

Even better, we can create our own meta-components (or let the React Router project write them for us) that abstract away the underlying JavaScript and allow us to render components conditionally with a declarative component syntax.

REACT ROUTER

Learning Outcomes

By the end of this lesson, you will be able to:

01	Use conditional statements to render different JSX or nothing at all.
02	Render props and attributes conditionally.
03	Embed a JSX expression inside another JSX expression.
04	Show and hide content with ternary and short-circuit expressions.
05	Set props with spread syntax.
06	Use the React Router in an application.

Conditional Rendering

Remember…you've used several of these techniques before.

Conditional Rendering Types

Today, we'll provide an overview of conditional rendering techniques, including:

- Conditional statements
- Ternary expressions
- Short-circuit expressions
- Boolean attributes
- Spread
- Formal routing

Conditional Statements

- Functional components must return something that represents HTML or the absence of HTML.
- They can return whatever they like at any point.
- In the following code, `<JsxX />` can be any valid JSX expression.

```
if (condition) {
    return <JsxOne />;
} else if (anotherCondition) {
    return <JsxTwo />;
} else {
    return <JsxThree />;
}
```

switch

Essentially, **routing** is inspecting a URL and deciding which view to render. Java calls it **request mapping**, because routing is based on a full HTTP request. React doesn't base its views on requests; it bases them on state. (But note that the React Router can do more than inspect the URL.)

A `switch` statement is useful when there are many scenarios based on one value.

We can use `switch` for basic routing.

```
switch (view) {
    case "home":
        return <Home />;
    case "about":
        return <About />;
    case "contact":
        return <ContactUs />;
    default:
        return <Unknown />;
}
```

Prevent Rendering

On render, `null` means “nothing to render.” However, `undefined` is an error.

That is not the case with embedded JSX. Both `null` and `undefined` embedded in a JSX expression work. The expressions mean “nothing to render.”

Return `null` from a component function to skip rendering altogether.

Returning `undefined` or skipping the return is an error.

```
if (condition) {
    return null;
}

return <Jsx />;
```

Embed a Fragment

JSX is very “plastic,” but it’s also a leaky abstraction.

We never want to do something like the following:

```
let header = <HeaderTwo />;
if (condition) {
    header = <HeaderOne />;
}
```

...because if `condition` is `true`, we’ve now evaluated JSX **twice**.

JSX is a leaky abstraction for one or more calls to `React.createElement`.

Ideally, we want to evaluate the correct JSX once and only once.

Embed a Fragment (continued)

- JSX can be assigned to variables.
- Embed any JSX expression in existing JSX with `{jsxExpression}`.

```
let header;
if (condition) {
    header = <HeaderOne />;
} else {
    header = <HeaderTwo />
}

return (
    <>
        {header}
        <div className="main">
            <p>Content</p>
        </div>
    </>
)
```

Functions

Utility/helper functions are underused. Try to use functions that promote strong software development practices:

- separation of concerns
- “don’t repeat yourself” (DRY) code, and
- single responsibility.

While components represent single concerns, a UI concern or responsibility is very different from a code unit concern or responsibility.

Use helper functions to clean up complicated JSX expressions.

Since JSX is an expression, it can be returned from a function.

Functions (continued)

```
function makeNavigation({ role }) {
    let adminLink;
    if (role === "ADMIN") {
        adminLink = (<li>
            <button onClick={showAdminView}>Admin</button>
        </li>);
    }
    return (
        <ul>
            <li><button>Profile</button></li>
            <li><button>Favorites</button></li>
            {adminLink}
        </ul>
    );
}

return (
    <>
        <Header />
        {makeNavigation(user)}
        <Body />
    </>
);
```

Break

Conditional Rendering (continued)

Embed Ternary Expressions

Ternary expressions are useful when we want to embed one of two options directly in JSX.

```
return (
    <main>
        {condition ? <StandardHeader /> : <AdminHeader />}
        <Body />
    </main>
);
```

Short-Circuit Expressions

This is a very popular technique in React. It can also be represented with `condition ? <Component /> : null`.

- Short-circuit expressions are useful when a component is optional.
- Take advantage of JavaScript truthiness.
- Hide things like a profile picture, link, or video when they don't have the required data.

```
return (
    <Jsx>
        {condition && <Component />}
        <MoreJsx />
    </Jsx>
);
```

Boolean Attributes

Attribute inclusion or exclusion only works for HTML elements.

For JSX components, the Boolean is passed as a prop.

- In standard HTML elements, any non-value attribute can be included or excluded with a condition.
- If the condition is false, the attribute is not rendered.
- If the condition is true, the attribute is rendered.

```
const data = {
    isDisabled: false,
    isRequired: true
}

return <input disabled={data.isDisabled}
    required={data.isRequired} />;

// result is:
// <input required>
```

Spread

This technique can be a real time-saver when object properties and component props have the same names. When given the choice, try to synchronize names.

Spread syntax can also cause confusion when there's a mix of prop declarations and more than one spread. Don't write overly clever code. The goal is to make code easy to understand.

Any JavaScript object can be spread into a component's attributes/props.

Multiple objects can be spread. If they contain duplicate properties, the last one wins.

```
const attributes = {
    type: "email",
    required: true,
    name: "email"
};

return <input {...attributes} />;

// result is:
// <input type="email" name="email" required />
```

React Router

- Third-party package
- Declarative routing

```
> npm install react-router-dom
```

You can find React Router’s web docs here: https://reactrouter.com/web/guides/quick-start.

01 Basic routing with `Router`, `Link`/`NavLink`, `Switch`, and `Route`.

02 Nested routing.

03 Reading the URL with `useParams` and `useRouteMatch`.

React Router Basics

`Router/BrowserRouter`	All React Router components must live inside it for routing to work.
`Link`	Changes the URL without navigating.
`Switch`	Like JS `switch`. Matches using `Route`.
`Route`	Inspects the URL and renders JSX on a match.

```
import {
    BrowserRouter as Router,
    Switch,
    Route,
    Link
} from "react-router-dom";

function Main() {
    return (
        <Router>
            <Link to="/">Home</Link>
            <Link to="/about">About</Link>
            <Link to="/contact">Contact</Link>
            <Switch>
                <Route path="/about">
                    <h1>About</h1>
                </Route>
                <Route path="/contact">
                    <h1>Contact</h1>
                </Route>
                <Route path="/">
                    <h1>Home</h1>
               </Route>
            </Switch>
        </Router >
    );
}
```

Time to Code

React Router Walkthrough

Suggested Time:

20 Minutes

Questions?

Break

Activity: React Router

Suggested Time:

20 Minutes

RECAP

Recap

- Can you recall all conditional rendering variations?
- Can you name any examples of conditional rendering techniques from past code?
- How might other variations be useful?
- What gets easier when you use other variations?
- What problem does the React Router solve?
- Can we solve it without the React Router?

Learning Outcomes

By the end of this lesson, you will be able to:

01	Use conditional statements to render different JSX or nothing at all.
02	Render props and attributes conditionally.
03	Embed a JSX expression inside another JSX expression.
04	Show and hide content with ternary and short-circuit expressions.
05	Set props with spread syntax.
06	Use the React Router in an application.

The End