CS 4530: Fundamentals of Software Engineering
Module 06: React Hook Patterns

Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

Learning Objectives for this Module

- By the end of this module, you should be able to:
  - Explain the basic use cases for useEffect
  - Explain when a useEffect is executed, and when its return value is executed
  - Construct simple custom hooks and explain why they are useful.
  - Be able to explain the three core steps of a test (assemble, act, assess) can map to UI component testing

Lesson 6.1 useEffect

useEffect is a mechanism for synchronizing a component with an external system

```
export default function ClockDisplay(props: ClockDisplayProps) {
  const clock = props.clock;
  const [localTime, setLocalTime] = useState(0);

  useEffect(() => {
    const listener1 = () => setLocalTime(time => time +1)
    clock.addListener(listener1);


    return () => {clock.removeListener(listener1);
    };
  }, []);
```

Action to take on first render

Action to take when component dismounts

Empty array says: do this on first render only

src/Shared/Components/SimpleClockDisplay.tsx

https://react.dev/reference/react/useEffect

An external system means any piece of code that's not inside your React component

- An event in the lifecycle of a component, like redisplay.
- A timer managed with setInterval and clearInterval
- An event subscription like a chat server (sockets?)
- A call to fetch data from an external API or web site
- An external animation library
- A piece of business logic in an app that is external to your component

Let’s build a clock display

src/Shared/types.ts

- Start by defining the props

```
export interface ClockDisplayProps
{
  name: string;
  key: number;
  clock: ITicker;
  handleDelete?: () => void;
  handleAdd?: () => void;
  noisyDelete?: boolean;
}
```

src/Shared/Components/SimpleClockDisplay.tsx

<ClockDisplay> (1): set up the useEffect

```
import { useState, useEffect } from "react";
import { Box, Button, HStack, VStack } from "@chakra-ui/react";
import type { ClockDisplayProps } from "../types";

// this version simply imports a clock from its parent
export default function ClockDisplay(props: ClockDisplayProps) {
  const clock = props.clock;
  const [localTime, setLocalTime] = useState(0);

  useEffect(() => {
    const listener1 = () => {setLocalTime((localTime) => localTime + 1)};
    clock.addListener(listener1);
    return () => {
      clock.removeListener(listener1);
    };
  }, []);
```

<ClockDisplay> (2): the handlers

```
function handleStart() { clock.start(); }

function handleStop() { clock.stop(); }

function handleReset() { setLocalTime(0); }
```

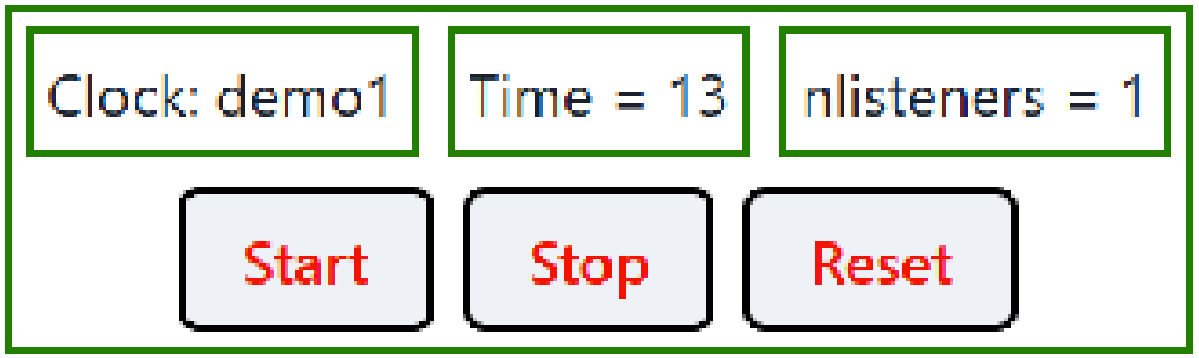
<ClockDisplay> (3): the display logic

```
return (
    <VStack border="2px" borderColor="green" padding="1" width="fit-content" mx="auto">
      <HStack>
        <Box>Clock: {props.name}</Box>
        <Box>Time = {localTime}</Box>
        <Box>nlisteners = {clock.nListeners}</Box>
      </HStack>
      <HStack>
        <Button onClick={handleStart}>Start</Button>
        <Button onClick={handleStop}>Stop</Button>
        <Button onClick={handleReset}>Reset</Button>
      </HStack>
      {props.handleAdd && (
        <HStack>
          <Button onClick={props.handleDelete}>Delete {props.name}</Button>
          <Button onClick={props.handleAdd}>Add a clock</Button>
        </HStack>
      )}
    </VStack>
  );
```

Running a single clock

```
import { useState, useEffect } from "react";
import { type ITicker } from "../../Shared/types";
import Ticker from '../../Shared/Classes/Ticker'
import ClockDisplay from "../../Shared/Components/SimpleClockDisplay";

export default function App() {
  const [clock, _] = useState<ITicker>(new Ticker(1000));

  return (
   <ClockDisplay name="demo1" key={1} clock={clock} />
  )


}
```

Demo!



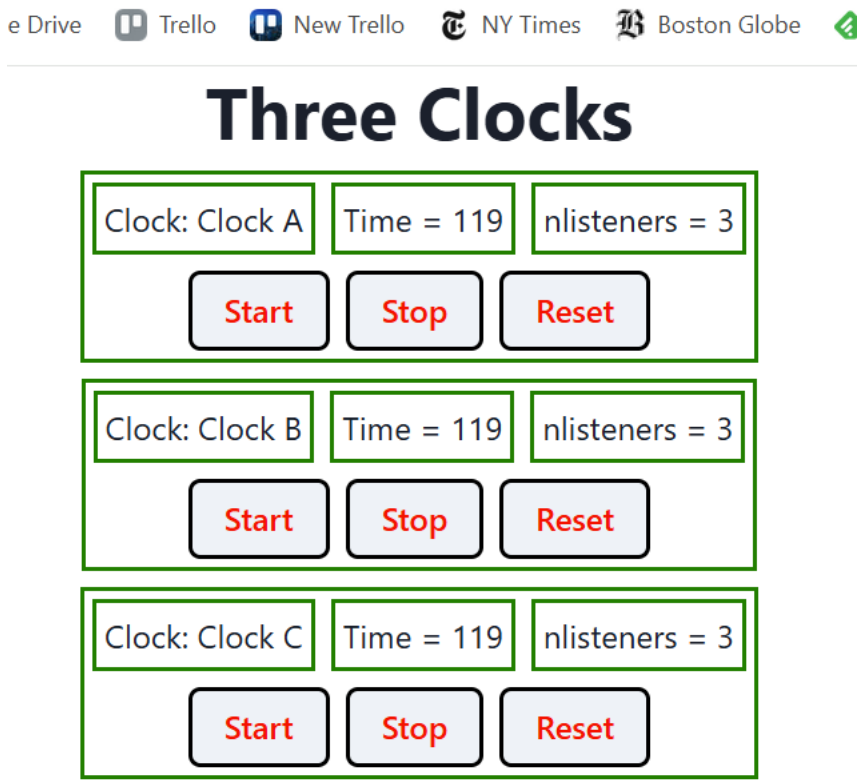
We can use ClockDisplay with a different parent!

src/Examples/ThreeClocks/App.tsx

- Components are modular, so we can use ClockDisplay with a different parent.
- But this required careful design!

```
export default function App() {
  const [clock, _] = useState<ITicker>(new Ticker(1000));

  return (
    <VStack>
      <Heading>Three Clocks</Heading>
      <ClockDisplay key={1} name="Clock A" clock={clock} />
      <ClockDisplay key={2} name="Clock B" clock={clock} />
      <ClockDisplay key={3} name="Clock C" clock={clock} />
    </VStack>
  );
}
```

running ThreeClocks



- Next: back to useEffect in detail….

useEffect's Dependencies Control Its Execution

- useEffect takes an optional array of dependencies
- The effect is only executed if one or more of the values in the dependency change (e.g. by a setter)
- Special Cases:
  - [] means run only on first render
  - No argument means run on every render

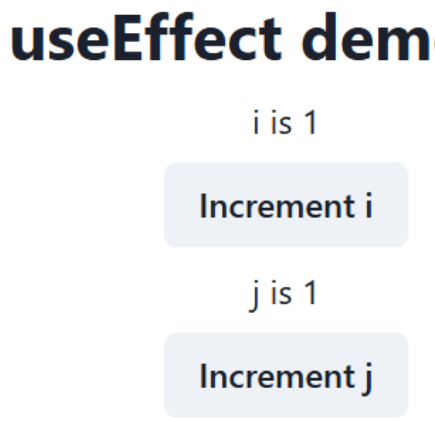
Example (Part 1)

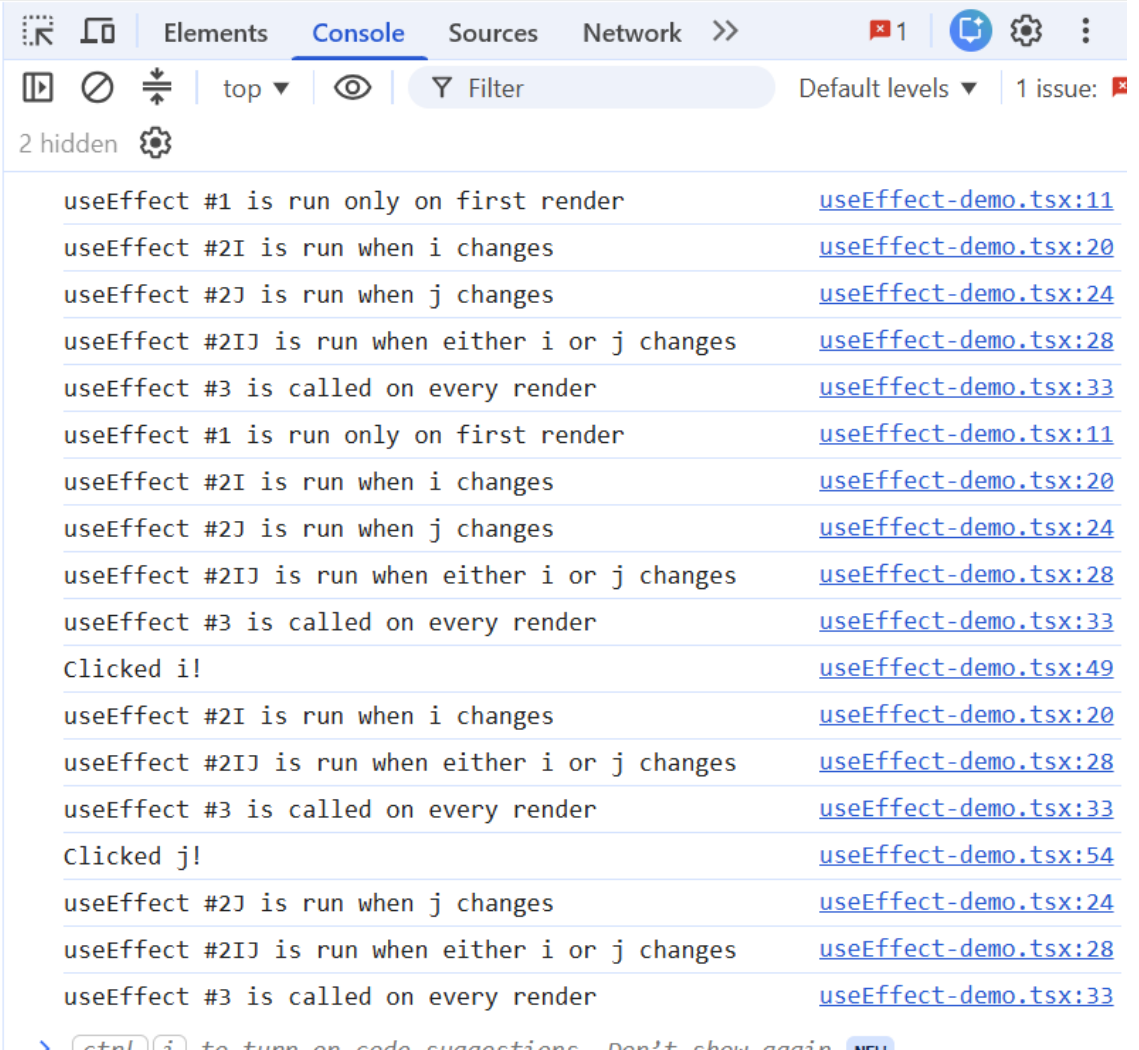
src/Examples/useEffect-demo.tsx

```
export default function App() {
  const [i, setI] = useState(0);
  const [j, setJ] = useState(0);

  // runs only on first render.
  useEffect(() => {
    console.log('useEffect #1 is run only on first render');
  }, []);

  useEffect(() => {
    console.log('useEffect #2I is run when i changes');
  }, [i]);

  useEffect(() => {
    console.log('useEffect #2J is run when j changes');
  }, [j]);

  useEffect(() => {
    console.log('useEffect #2IJ is run when either i or j
changes');
  }, [i,j]);

  // runs on every render
  useEffect(() => {
    console.log('useEffect #3 is called on every render');
  });
```

```
  function onClickI() {
      console.log('Clicked i!');
      setI(lastI => lastI+1);
      // bound variable name doesn't matter, of course
      // setI(i+1) is a bug, because value of i may not
      // be current.
    }

    function onClickJ() {
      console.log('Clicked j!');
      setJ(j => j+1);
    }

    return (
      <VStack>
        <Heading>useEffect demo #1IJ</Heading>
        <Text> i is {i} </Text>
        <Button onClick={onClickI}>Increment i</Button>
        <Text> j is {j} </Text>
        <Button onClick={onClickJ}>Increment j</Button>
      </VStack>
    );
```

Demo





Open the console by pressing F12 (on Windows) or Cmd-

When is the cleanup function executed?

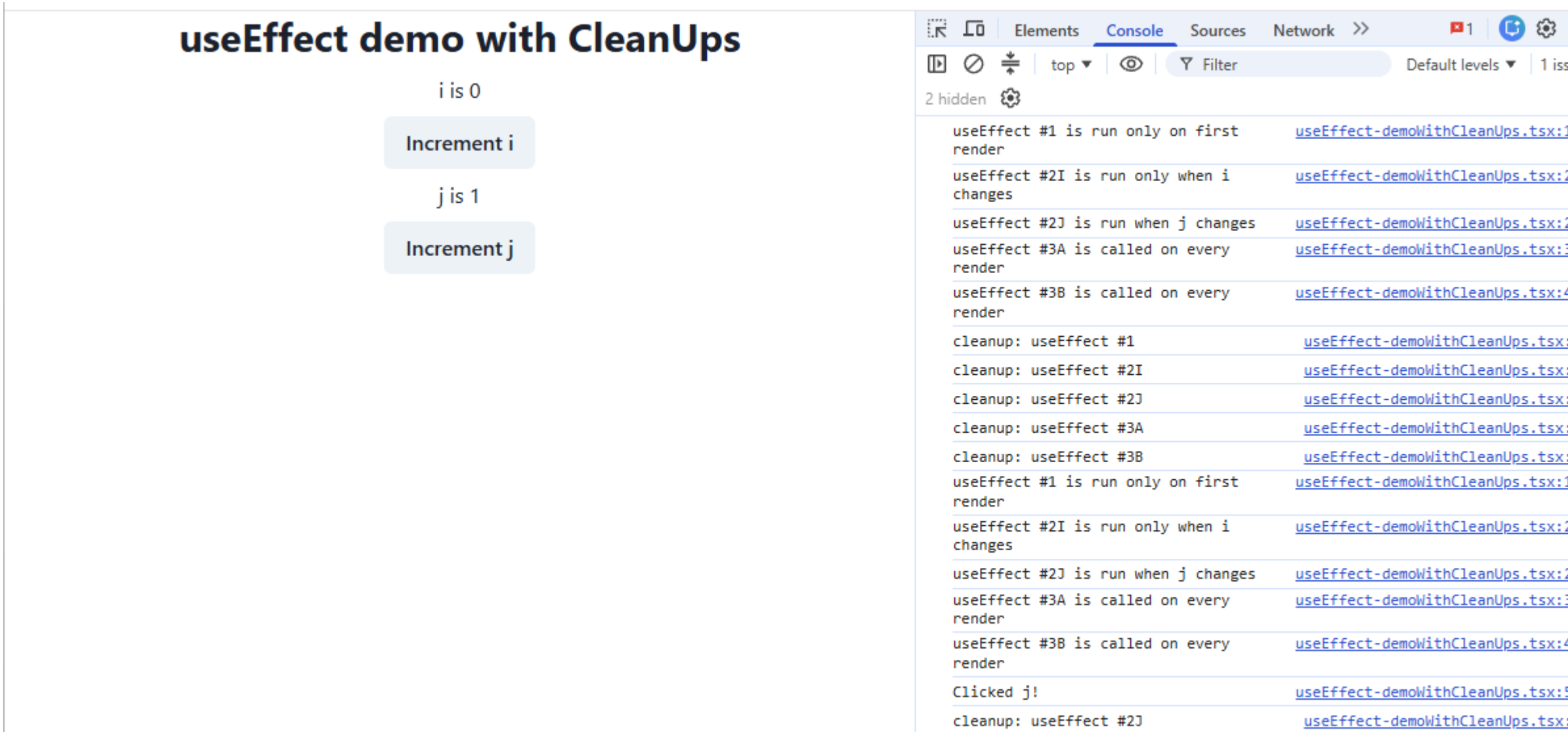
- In general, the cleanup function is executed sometime before the next time the hook is run.
- For the first-time-only case, this means when the component is dismounted.
- Let's look at useEffect demo again, this time with noisy cleanups.

Demo

src/Examples/useEffectsDemos/useEffect-demoWithCleanUps.tsx

```
function cleanup(message: string) {
  return () => {
    console.log('cleanup: ' + message);
  };
}


export default function App() {
  const [i, setI] = useState(0);
  const [j, setJ] = useState(0);


  // runs only on first render.
  useEffect(() => {
    console.log('useEffect #1 is run only on first render');
    return cleanup('useEffect #1');
  }, []);


  useEffect(() => {
    console.log('useEffect #2I is run when i changes');
    return cleanup('useEffect #2I');
  }, [i]);

  // etc.
```

Demo


useEffect demo with CleanUps
i is 0
Increment i
j is 1
Increment j
Elements
Console
Sources
Network
top
Filter
Default levels
2 hidden
useEffect #1 is run only on first render
useEffect #2I is run only when i changes
useEffect #2J is run when j changes
useEffect #3A is called on every render
useEffect #3B is called on every render
cleanup: useEffect #1
cleanup: useEffect #2I
cleanup: useEffect #2J
cleanup: useEffect #3A
cleanup: useEffect #3B
useEffect #1 is run only on first render
useEffect #2I is run only when i changes
useEffect #2J is run when j changes
useEffect #3A is called on every render
useEffect #3B is called on every render
Clicked j!
cleanup: useEffect #2J

Communication with an (Actual) External System

- In the previous example, everything was present in the one codebase, including the clock.
- More common situation is that the user interface components need to communicate with a remote system, not necessarily in the same codebase.

Example: Interacting With a REST-based Server

```
export default function RandomNASA() {
  const [image, setImage] = useState<null | string>(null);
  const [error, setError] = useState<unknown>(null);

  useEffect(() => {
    fetch("https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY")
      .then((response) => response.json())
      .then((json) => {
        const data = z.object({ url: z.string() }).parse(json);
        setImage(data.url);
      })
      .catch((err) => setError(err));
  }, []);

  return error !== null ? (
    `Could not retrieve image: ${error}`
  ) : image !== null ? (
    <img src={image} />
  ) : (
    `Loading...`
  );
}
```

Doing this in the real world is somewhat more complicated than this (see Resources for details)

This function runs *after* the fetch returns

And this function runs after *that*

This function runs if there's an error in the previous

Lesson 6.2 Custom Hooks

Custom Hooks

- REACT lets us combine useState and useEffect to build custom hooks.
- Custom Hooks let us separate business logic from display logic

Example: useClock

Src/Shared/Hooks/useClock.ts

```
import { useEffect } from "react";
import Ticker from "../Classes/Ticker";
import { type ITicker } from "../types";


// takes a listener function as an argument
// returns the clock object
export function useClock(listener1: () => void): ITicker {
  const clock = new Ticker(1000);
  useEffect(() => {
    clock.addListener(listener1);
    return () => {
      clock.removeListener(listener1);
    };
  }, []);
  return clock;
}
```

src/Examples/SimpleClockDisplayWithUseClock.tsx

Using useClock

```
import { useState } from "react";
import { type ClockDisplayProps } from "../Shared/types";
import { type ITicker } from "../Shared/types";
import { useClock } from "../Shared/Hooks/useClock";
import { Box,  HStack, IconButton } from "@chakra-ui/react";
import { AiOutlineDelete, AiOutlinePlus } from "react-icons/ai";

export function ClockDisplay(props: ClockDisplayProps) {
  const [localTime, setLocalTime] = useState(0);
  const incrementLocalTime = () => setLocalTime((localTime) => localTime + 1);
  const clock: ITicker = useClock(incrementLocalTime);

  return (
    <HStack>
      <Box>Clock: {props.name}</Box>
      <Box>Time = {localTime}</Box>
      <Box>nlisteners = {clock.nListeners}</Box>
      <IconButton aria-label={"delete"} onClick={props.handleDelete} icon={<AiOutlineDelete />}
/>
      <IconButton aria-label={"add"} onClick={props.handleAdd} icon={<AiOutlinePlus />} />
    </HStack>
  );
}
```

src/Apps/ToDoApp/App.tsx

Example: ToDoList

```
export default function ToDoApp () {
  const [todoList,setTodolist] = useState<ToDoItem[]>([])
  const [itemKey,setItemKey] = useState<number>(0)   // first unused key

  function handleAdd (title:string, priority:string) {
    if (title === '') {return}   // ignore blank button presses
    setTodolist(todoList.concat({title: title, priority: priority, key: itemKey}))
    setItemKey(itemKey + 1)
  }

  function handleDelete(targetKey:number) {
    const newList = todoList.filter(item => item.key !== targetKey)
    setTodolist(newList)
  }
```

business logic

```
  return (
  <VStack>
    <Heading>TODO List</Heading>
    <ToDoItemEntryForm onAdd={handleAdd}/>
    <ToDoListDisplay items={todoList} onDelete={handleDelete}/>
  </VStack>
  )
}
```

display logic

src/Apps/ToDoAppWithCustomHooks/App.tsx

Refactoring ToDoList

```
export default function ToDoApp () {

  const {todoList, handleAdd, handleDelete} = useToDoItemList()


  return (
  <VStack>
    <Heading>TODO List</Heading>
    <ToDoItemEntryForm onAdd={handleAdd}/>
    <ToDoListDisplay items={todoList} onDelete={handleDelete}/>
  </VStack>
  )
}
```

business logic is encapsulated

src/Apps/ToDoAppWithCustomHooks/useToDoItemList.tsx

The hook encapsulates the business logic

```
export default function useToDoItemList () {
  const [todoList,setTodolist] = useState<ToDoItem[]>([])
  const [itemKey,setItemKey] = useState<number>(0)   // first unused key

  function handleAdd (title:string, priority:string) {
    if (title === '') {return}   // ignore blank button presses
    setTodolist(todoList.concat({title: title, priority: priority, key: itemKey}))
    setItemKey(itemKey + 1)
  }

  function handleDelete(targetKey:number) {
    const newList = todoList.filter(item => item.key !== targetKey)
    setTodolist(newList)
  }

  return {todoList: todoList, handleAdd: handleAdd, handleDelete: handleDelete}
}
```

The hook is like a class managing a piece of state

```
export default function useToDoItemList () {
  const [todoList,setTodolist] = useState<ToDoItem[]>([])
  const [itemKey,setItemKey] = useState<number>(0)   // first unused key

  function handleAdd (title:string, priority:string) {
    if (title === '') {return}   // ignore blank button presses
    setTodolist(todoList.concat({title: title, priority: priority, key: itemKey}))
    setItemKey(itemKey + 1)
  }

  function handleDelete(targetKey:number) {
    const newList = todoList.filter(item => item.key !== targetKey)
    setTodolist(newList)
  }

  return {todoList: todoList, handleAdd: handleAdd, handleDelete: handleDelete}
}
```

handleAdd and handleDelete are the only methods for manipulating the state

The hook's state becomes part of its user's state.

```
export default function useToDoItemList () {
  const [todoList,setTodolist] = useState<ToDoItem[]>([])
  const [itemKey,setItemKey] = useState<number>(0)   // first unused key

  function handleAdd (title:string, priority:string) {
    if (title === '') {return}   // ignore blank button presses
    setTodolist(todoList.concat({title: title, priority: priority, key: itemKey}))
    setItemKey(itemKey + 1)
  }

  function handleDelete(targetKey:number) {
    const newList = todoList.filter(item => item.key !== targetKey)
    setTodolist(newList)
  }

  return {todoList: todoList, handleAdd: handleAdd, handleDelete: handleDelete}
}
```

calling these setters redisplays the whole component

The Rules of Hooks

1. Only call hooks at the top level
   - Not within loops, inside conditions, or nested functions
   - Rationale: The order of hooks called must always be the same each time a component renders
2. Only call hooks from React Components or Custom Hooks
   - Not from any other helper methods or classes
   - Rationale: React must know the component that the call to the hook is associated with

```
export function LikeButton(){
  const [isLiked, setIsLiked] = useState(false);
  const [count, setCount] = useState(0);
  ...
}
```

React knows which useState is which by tracking calls to them from components in the render tree

We Use Two ESLint Rules for React Hooks

- You should not violate the rules of hooks. These linter plugins help detect violations
- React-hooks/rules-of-hooks
  - Enforces that hooks are only called from React functional components or custom hooks
- React-hooks/exhaustive-deps
  - Enforces that all variables used in useEffects are included as dependencies

Putting it All Together

- In the previous examples, we learned
  - how to introduce side effects
  - how to create our own hooks
- We can use these concepts to make React components interact with a remote system (e.g., REST services).

Example: Setting Up and Cleaning Up a Chat Subscription; Showing Unread Messages

```
useEffect(() => {
  const socket = new WebSocket('wss://api.randomsite.com/chat');

socket.onmessage = (event) => {
    const message = JSON.parse(event.data);
    setMessages((prev) => [...prev, message]);
  };
  return () => {
    socket.close();
  };
}, []);


useEffect(() => {
  document.title = `You have ${unreadMsgs} unread mess
}, [unreadMsgs]);
```

This allows you to subscribe to live chat messages

This function runs at cleanup to avoid memory leaks

This updates the browser tab title to show the number of unread messages or notifications

There are several other React Hooks that might be useful.

- React Context allows you to share state between deeply nested components more easily

```
import { useContext } from ‘react';

function myButtonComponent() {
  const theme = useContext(ThemeContext);
      // ...
}
```

- useRef Hook lets you reference a value that’s not needed for rendering
- useEffectEvent Hook lets you extract non-reactive logic from your Effects into a reusable function.

https://react.dev/reference/react/useContext

Lesson 6.3 Testing your React components

Testing React components

- The AAA pattern ("Assemble/Act/Assess") still applies
- Need a test double for the React system
  - render components into a "virtual dom" or into a captive web browser
- We'll be using a package called "Playwright" to do this.

Recall: Most tests are in AAA form: Assemble/Act/Assess

```
test('addStudent should add a student to the data
    // const db = new DataBase ()
    expect(db.nameToIDs('blair')).toEqual([])

    const id1 = db.addStudent('blair’);

    expect(db.nameToIDs('blair')).toEqual([id1])
});
```

Assemble (and check that you’ve assembled it

Act (do the action that you are trying to test)

Assess: check to see that the response is correct

Playwright commands work on a ”page object”

`.goto()`	Navigate to a URL. Many tests begin with this command.
`.getByLabel()`	Find form elements by their associated label text.
`.getByRole()`	Find elements by their ARIA role(button, textbox, etc) .
`.getByText()`	Find elements by their text content.
`.waitForURL()`	Wait for navigation to complete to a specific URL.
`.fill()`	Type text into an input field.

These will fail if the specified element does not exist

https://playwright.dev/docs/locators

A typical playwright test

```
test('should allow an existing user to log in', async ({ page }) => {
  await page.goto('/login');

 await page.getByLabel('Username’)
  .fill("Frau Drei");

 await page.getByLabel('Password’,
 { exact: true }).fill(password);

 await page.getByRole('button’,
 { name: 'Log In' }).click();

  await page.waitForURL('/');
 await expect(page.getByText
 ('signed in as Frau Drei')).toBeVisible();
 });
```

Assemble (navigate to the page)

Act (fill form and click login)

Assess: check to see that the response is correct

Learning Objectives for this Lesson

- By the end of this lesson, you should be able to:
  - Explain the basic use cases for useEffect
  - Explain when a useEffect is executed, and when its return value is executed
  - Construct simple custom hooks and explain why they are useful.
  - Be able to explain the three core steps of a test (assemble, act, assess) can map to UI component testing