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

Adeel Bhutta

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
  - Become familiar with Best practices of UI/UX design

Lesson 6.1 useEffect

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

```
import { clockServer } from './clock.js';

function ClockClient() {

  useEffect(() => {
    const connection = clockServer.createConnection()
    connection.connect();

    return () => {
      connection.disconnect();
    };
  }, []);
  // ...
}
```

Action to take on first render

Action to take when component dismounts

Empty array says: do this on first render only

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

A real example: a display that connects to a self-ticking clock

src/Components/SimpleClockDisplay.tsx

```
export default function ClockDisplay(props: {
    name: string, key: number,
    clock:IClock,
    handleDelete: () => void,
    handleAdd: () => void,
})
    {
    const [localTime, setLocalTime] = useState(0)
    const incrementLocalTime = () => setLocalTime(localTime => localTime + 1)
    const clock = props.clock

    useEffect(() => {
        const listener1 = () => { increme
        clock.addListener(listener1)
        return () => {
            clock.removeListener(listener1)
        }
    }, [])
```

The parent provides the clock

On first render, add this listener to the clock

On dismount, remove the listener.

Display logic will come later…

src/Apps/ThreeClocks/App.tsx

Our app will have three displays of the clock

```
import { Heading, VStack } from '@chakra-ui/react';
import ClockDisplay from './ClockDisplay.tsx';
import SingletonClockFactory from '../../Classes/SingletonClockFactory.ts';

export default function App() {
  const clock = SingletonClockFactory.getInstance(1000);
  clock.start()

  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>
  );
}
```

src/Classes/SingletonClockFactory.ts

The clock itself is simple

```
// the listener does not expect an argument
type Listener = () => void;

class Clock implements IClock {
// the actual clock state.
// it runs all the time, but only notifies
// listeners when this._running is true.

  private _createTimer() {
    setInterval(() => {
      if (this._running) {
        this._notifyAll();
      }
    }, this._interval);
  }

  private _running: boolean = false;
```

```
// the arrow-function idiom guarantees
// that 'this' is bound statically.
public start = () => {
  this._running = true;
};

public stop = () => {
  this._running = false;
};

public constructor(interval: number) {
  this._interval = interval;
  this._createTimer();
  this.start();
}
```

And we'll make the clock a singleton in the usual way

src/Classes/SingletonClockFactory.ts

```
export default class SingletonClockFactory {

    private static theClock: Clock | undefined = undefined

    private constructor () {SingletonClockFactory.theClock = undefined}

    public static instance (interval:number) : Clock {
        if (SingletonClockFactory.theClock === undefined) {
            SingletonClockFactory.theClock  = new Clock(interval)
        }
        return SingletonClockFactory.theClock
    }

}
```

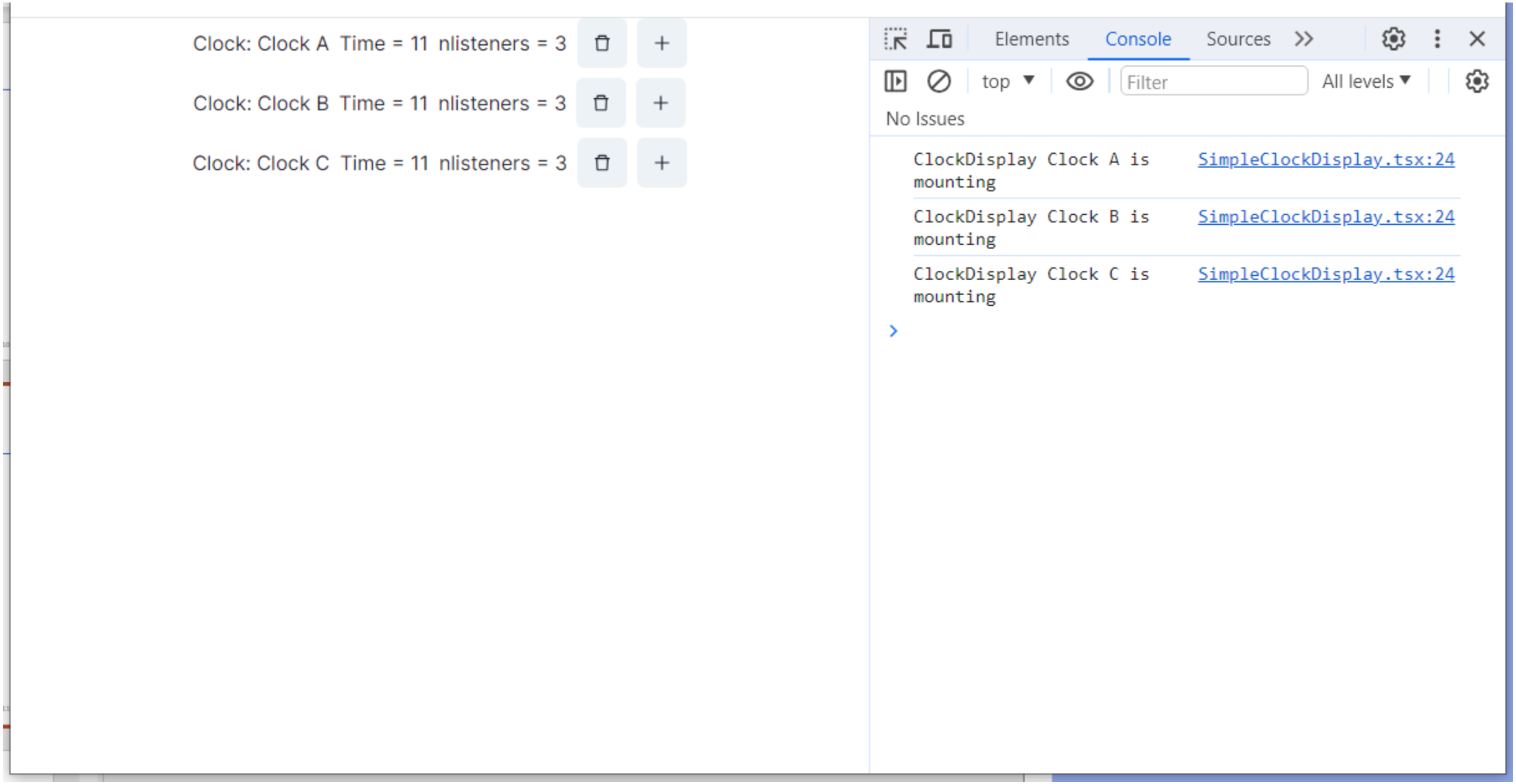
src/Apps/ThreeClocks/ClockDisplay.tsx

Now let's look at <ClockDisplay>

```
export default function ClockDisplay(props: ClockDisplayProps) {
  const [localTime, setLocalTime] = useState(0);
  const incrementLocalTime = () => {
    setLocalTime(localTime => localTime + 1);
  };

  useEffect(() => {
    const listener1 = () => {
      incrementLocalTime();
    };
    props.clock.addListener(listener1);
    console.log(`ClockDisplay ${props.name} is mounting`);
    return () => {
      console.log('ClockDisplay ' + props.name + ' is unmounting');
      props.clock.removeListener(listener1);
    };
  }, [props.clock, props.name]);
```

Clock Display: display logic

```
  return (
    <HStack>
      <Box>Clock: {props.name}</Box>
      <Box>Time = {localTime}</Box>
      <Box>nlisteners = {clock.nListeners}</Box>
      <Button aria-label={'start'} onClick={() => clock.start()}>
        Start
      </Button>
      <Button aria-label={'stop'}  onClick={() => clock.stop()}>
        Stop
      </Button>
    </HStack>
  );
}
```


Clock: Clock A Time = 11 nlisteners = 3
Clock: Clock B Time = 11 nlisteners = 3
Clock: Clock C Time = 11 nlisteners = 3
Elements
Console
Sources
top
Filter
All levels
No Issues
ClockDisplay Clock A is mounting
SimpleClockDisplay.tsx:24
ClockDisplay Clock B is mounting
SimpleClockDisplay.tsx:24
ClockDisplay Clock C is mounting
SimpleClockDisplay.tsx:24

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

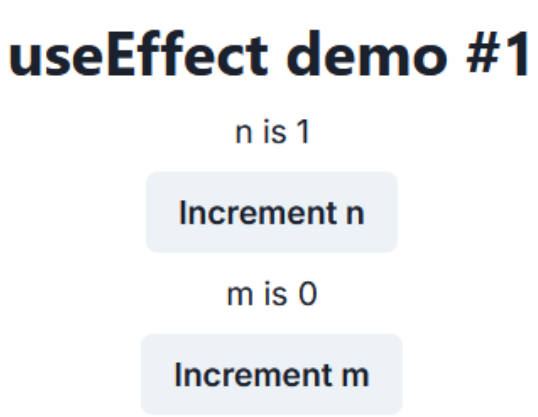
Example (Part 1)

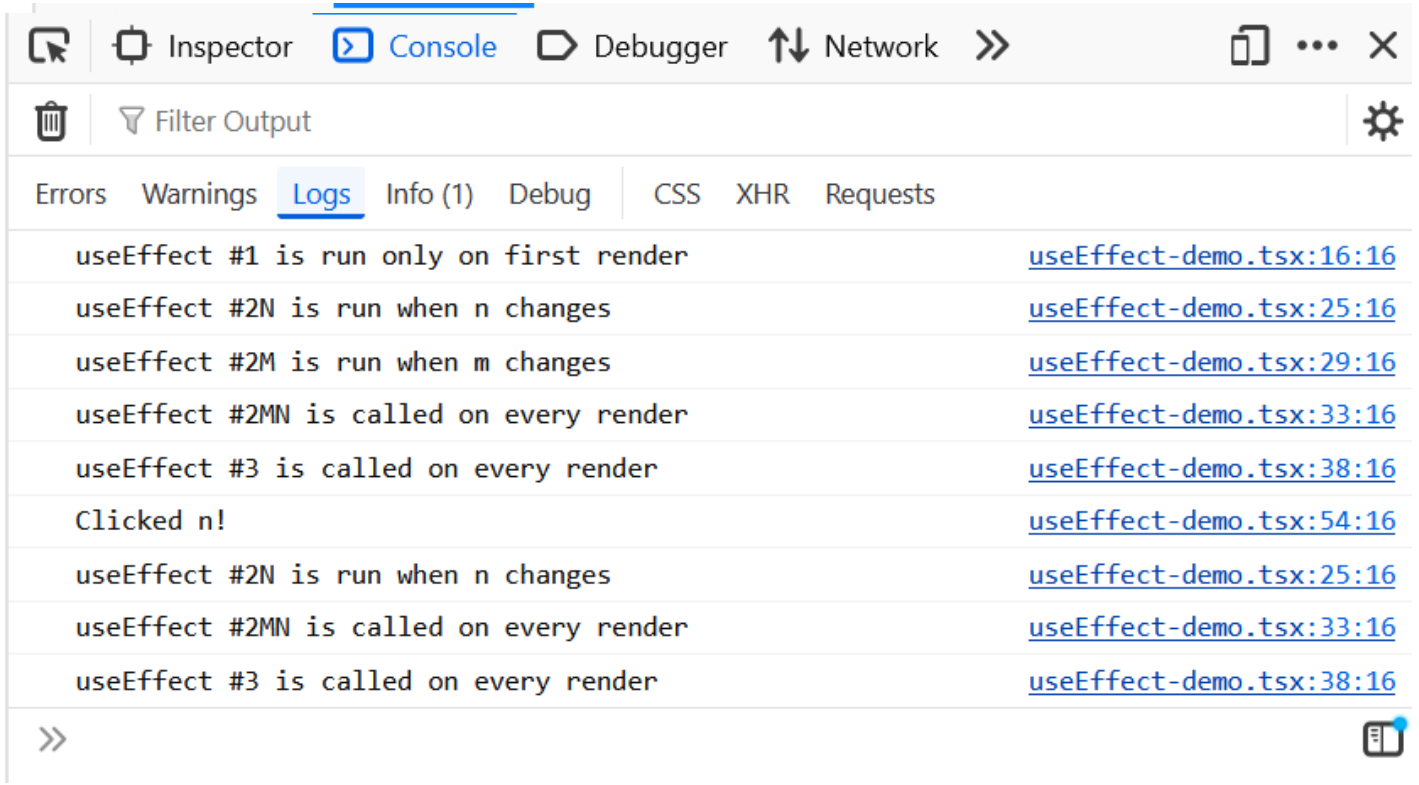
src/app/Apps/useEffect-Demos/useEffect-demo.tsx

```
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


useEffect demo #1
n is 1
Increment n
m is 0
Increment m


Inspector
Console
Debugger
Network
Filter Output
Errors
Warnings
Logs
Info (1)
Debug
CSS
XHR
Requests
useEffect #1 is run only on first render useEffect-demo.tsx:16:16
useEffect #2N is run when n changes useEffect-demo.tsx:25:16
useEffect #2M is run when m changes useEffect-demo.tsx:29:16
useEffect #2MN is called on every render useEffect-demo.tsx:33:16
useEffect #3 is called on every render useEffect-demo.tsx:38:16
Clicked n! useEffect-demo.tsx:54:16
useEffect #2N is run when n changes useEffect-demo.tsx:25:16
useEffect #2MN is called on every render useEffect-demo.tsx:33:16
useEffect #3 is called on every render useEffect-demo.tsx:38:16

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/Apps/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.
```

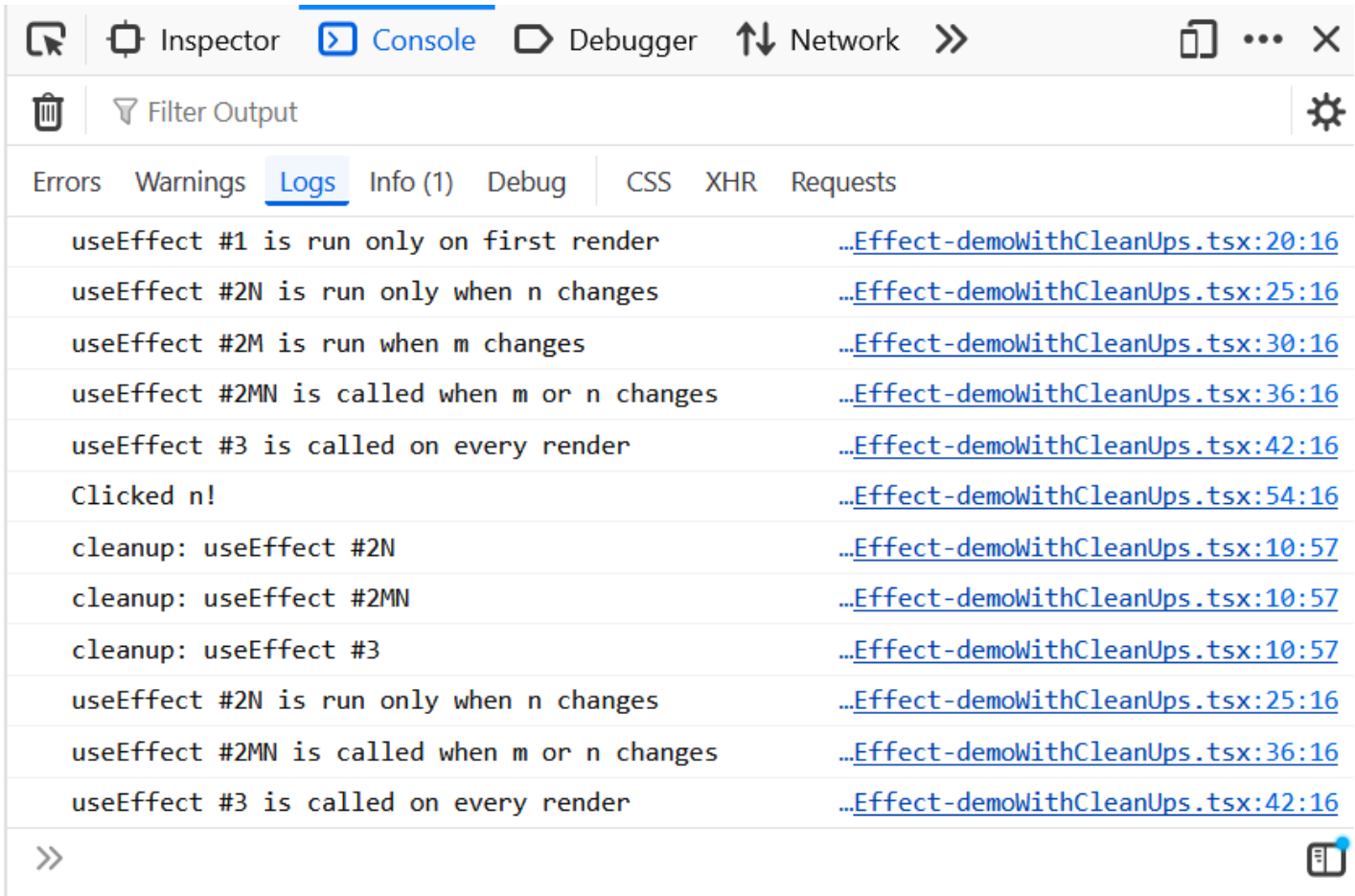
useEffect demo with CleanUps

n is 1

Increment n

m is 0

Increment m



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/Hooks/useClock.ts

```
import { useEffect } from 'react';
import SingletonClockFactory from '../Classes/SingletonClockFactory.ts';
import { type IClock } from '../Interfaces/IClock.ts';


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

src/Apps/ThreeClocksWithUseClock/ClockDisplay.tsx

Using useClock

```
import { useClock } from '../Hooks/useClock';

export function ClockDisplay(props: {
    name: string, key: number,
    handleDelete: () => void, handleAdd: () => void,
    noisyDelete?: boolean

}) {
    const [localTime, setLocalTime] = useState(0)
    const incrementLocalTime = () => setLocalTime(localTime => localTime + 1)
    const clock:IClock = 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

A somewhat larger 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 (1): 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...`
  );
}
```

This function runs *after* the fetch returns

And this function runs after *that*

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

Example (2, 3): Setting Up and Cleaning Up a Chat Subscription or 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();
  };
}, []);

This allows you to subscribe to live chat messages

This function runs at cleanup to avoid memory leaks

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

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

client/tests/end-to-end/login.e2e.spec.ts

```
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

run with: npx playwright test

Cypress commands work on a "virtual DOM"

`.visit()`	Visit a remote URL. Many tests begin with this command.
`.contains()`	Select a DOM element by text content.
`.get()`	Find DOM elements by selector
`.click()`	Click a DOM element.
`.type()`	Type into a DOM element.

These will fail if the specified element does not exist

https://docs.cypress.io/api/table-of-contents#Cypress-API

A typical cypress test

testing/cypress/e2e/addAnswer.cy.ts
(from IP2 starter)

```
it("5.1 | Created new answer should be displayed at the top of the answers page",
() => {
    const answers = [
      "Test Answer 1",
      A1_TXT,
      A2_TXT,
    ];
    cy.visit("http://localhost:3000");
    cy.contains(Q1_DESC).click();
    cy.contains("Answer Question").click();
    cy.get("#answerUsernameInput").type("joym");
    cy.get("#answerTextInput").type(answers[0]);
    cy.contains("Post Answer").click();
    cy.get(".answerText").each(($el, index) => {
      cy.contains(answers[index]);
    });
    cy.contains("joym");
    cy.contains("0 seconds ago");
  });
```

Assemble (and check that you've assembled it correctly)

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

Assess: check to see that the response is correct

run with: npx cypress run

Lesson 6.4 Best Practices for UI Design

UI/UX Design principles

- The most commonly repeated UI/UX design principles across companies like Apple, Google Material Design, Microsoft Design, IBM Carbon Design System, and the Nielsen Norman Group are surprisingly consistent:
  - Consistency
  - Clarity
  - Accessibility
  - Visual hierarchy
  - Simplicity
  - User control
  - Efficiency
  - Predictability
  - Responsive/adaptive design
  - Feedback/system status

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
  - Become familiar with Best practices of UI/UX design