4. Reducer (Switch case)

Sunday, 21 December 2025 6:27 PM

Action → data e.g., { type: "CHANGE_NAME", payload: "Alice" }
Dispatch → method call e.g dispatch(action)
Reducer → switch-case logic e.g (state, action) => newState
React → stores state (local)

When the customer clicks the Next button, dispatch is called with an appropriate action. React then calls the reducer, the reducer returns the updated state, and React stores it and re-renders the component.

Example:

```
import { useReducer } from "react";

const initialState = {
  name: "John",
};

function reducer(state, action) {
  switch (action.type) {
    case "CHANGE_NAME":
      return { ...state, name: action.payload };

    default:
      return state;
  }
}

function UserComponent() {
  const [state, dispatch] = useReducer(reducer, initialState);

  function changeName() {
    dispatch({
      type: "CHANGE_NAME",
      payload: "Alice",
    });
  }

  return (
    <div>
      <p>Name: {state.name}</p>
      <button onClick={changeName}>Change Name</button>
    </div>
  );
}

export default UserComponent;
```

👦 Reducer for a 10-year-old

Imagine you have a **magic box** 📦 that keeps your score in a game.

- The box has **one number** inside (your score).
- You can only change the score by giving the box a **note**:
  - “add 1”

- “add 1”
- “subtract 1”
- “reset”

The box **reads the note** and decides what the **new score** should be.
That **magic box is a reducer**.

Example (in simple words)

- Score starts at **0**
- You say: “add 1” → score becomes **1**
- You say: “add 1” → score becomes **2**
- You say: “reset” → score becomes **0**

👉 The reducer:

- remembers the score
- listens to instructions
- updates the score correctly

👦 Reducer for a 25-year-old (React / Programming)

A **reducer** is a **pure function** that:

- takes the **current state**
- takes an **action** (what happened)
- returns the **new state**

Simple definition

Reducer = (state, action) → newState

Real-world analogy 🏦

Think of a **bank clerk**:

- **state** → your current balance
- **action** → “deposit 500”, “withdraw 200”
- **reducer** → rules that decide the new balance

The clerk **does not remember history**, only:

“Given current balance + this action → what’s the new balance?”

Example in JavaScript (Counter)

```
function counterReducer(state, action) {
  if (action.type === "INCREMENT") {
    return state + 1;
  }
if (action.type === "DECREMENT") {
    return state - 1;
  }
if (action.type === "RESET") {
    return 0;
  }
return state;
}
```

Using it

```
let state = 0;
state = counterReducer(state, { type: "INCREMENT" }); // 1
state = counterReducer(state, { type: "INCREMENT" }); // 2
state = counterReducer(state, { type: "RESET" });     // 0
```

Reducer in React (useReducer)

```
const [state, dispatch] = useReducer(counterReducer, 0);
```

- state → current value
- dispatch → sends an action
- counterReducer → decides how state changes

- counterReducer → decides how state changes

Triggering actions

dispatch({ type: "INCREMENT" });
dispatch({ type: "DECREMENT" });

Why use a reducer instead of useState?

useState	useReducer
Simple state	Complex state logic
Few updates	Many related updates
Small components	Large components

👉 Reducers are great when:

- state logic is **complex**
- many actions affect the same state
- you want **predictable updates**

One-line summary 🧠

- **10-year-old**: Reducer is a rule-box that updates things when you give instructions.
- **25-year-old**: Reducer is a pure function that computes new state from old state + action.

If you want, I can also:

- compare useState vs useReducer visually
- explain reducer like **Java switch-case**
- show a **real Next.js example**

Just tell me 👍

Perfect 👍 since you're comfortable with **Java**, this will click fast.

Reducer = Java switch-case

A **reducer** is basically a **switch-case on an action type** that decides **how state changes**.

Java-style thinking 🧠

Java example (bank account)

```
int reducer(int state, String action) {
   switch (action) {
      case "DEPOSIT":
         return state + 100;
      case "WITHDRAW":
         return state - 50;
      case "RESET":
         return 0;
      default:
         return state;
   }
}
```

Usage

```
int balance = 0;
balance = reducer(balance, "DEPOSIT"); // 100
balance = reducer(balance, "WITHDRAW"); // 50
balance = reducer(balance, "RESET"); // 0
```

👉 This is **exactly how reducers work**.

JavaScript Reducer (same idea)

JavaScript Reducer (same idea)

```
function reducer(state, action) {
  switch (action.type) {
    case "DEPOSIT":
      return state + 100;
case "WITHDRAW":
      return state - 50;
case "RESET":
      return 0;
default:
      return state;
  }
}
```

React useReducer version

```
const initialState = 0;
const [state, dispatch] = useReducer(reducer, initialState);
```

Dispatching actions

```
dispatch({ type: "DEPOSIT" });
dispatch({ type: "WITHDRAW" });
dispatch({ type: "RESET" });
```

Key rules (very important)

1️⃣ Reducer must be pure

Like a good Java method:

- ❌ No DB calls
- ❌ No API calls
- ❌ No random values
- ❌ No modifying existing state

```
//❌ wrong
state++;
return state;
//✅ correct
return state + 1;
```

2️⃣ Default case is mandatory

Just like Java switch-case, always return current state:

```
default:
  return state;
```

Reducer vs Java Service class

Java	React
Service.method(state, action)	reducer(state, action)
switch-case	switch(action.type)
Return new object	Return new state
Stateless	Stateless

Complex state (Java POJO style)

Java

```
class UserState {
   String name;
```

```
   int age;
}
```

Reducer

```
function userReducer(state, action) {
  switch (action.type) {
    case "CHANGE_NAME":
      return { ...state, name: action.payload };
case "INCREMENT_AGE":
      return { ...state, age: state.age + 1 };
default:
      return state;
  }
}
```

One-line takeaway 🔑

> **A reducer is a switch-case function that takes current state and action, and returns a new state — nothing more.**

If you want next:

- reducer vs useState in **Java mindset**
- how reducers replace **if-else chaos**
- how Redux uses the same concept

Just say the word 🚀