TypeScript — The Hard Way

Building a Type-Safe, Polymorphic, React Component

Adam Thompson – React Rally 2024

Adam Thompson

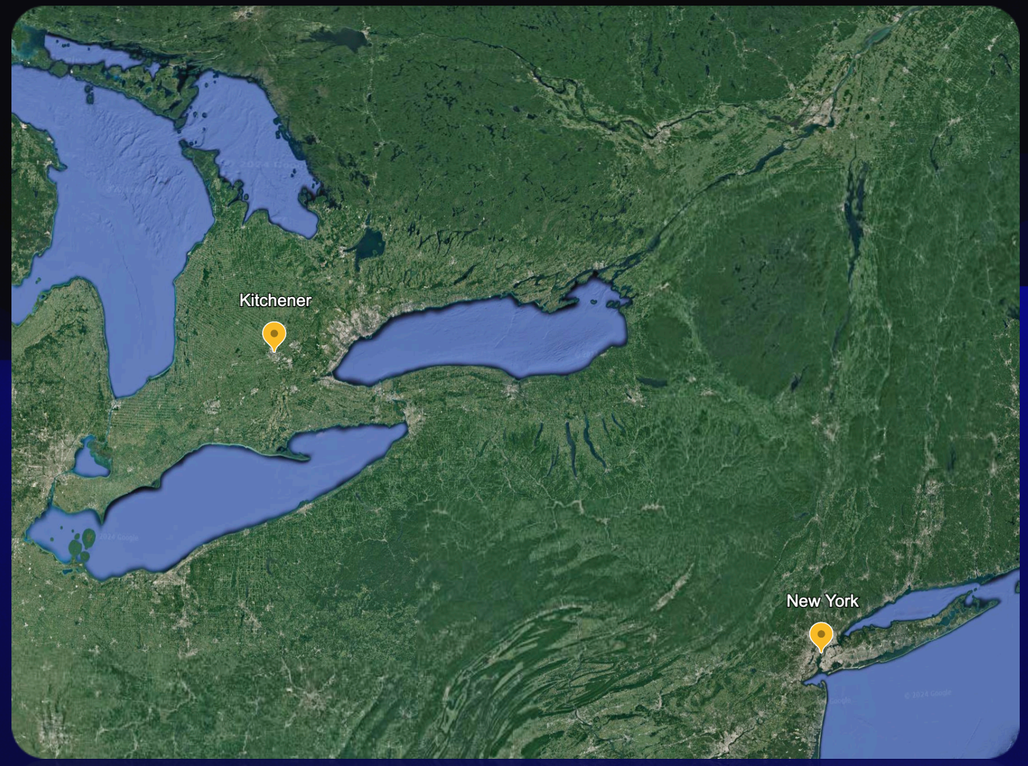
Sr. UI Engineer
MongoDB, LeafyGreen UI

@TheSonOfThomp

Adam Thompson

Canadian in New York.
Musician. Hockey fan. Engineer.

@TheSonOfThomp



TypeScript — The Hard Way

Building a Type-Safe, Polymorphic, React Component

Adam Thompson — Sr. UI Engineer @ MongoDB

Type-Safe, Polymorphic, React Component

Type-Safe, ⁉️
Polymorphic, ⁉️
React Component ✅

Type-Safe, ⁉️

Polymorphic, ⁉️

React Component ✅

TypeScript

TypeScript

JavaScript with syntax for types

typescriptlang.org

Why use TypeScript?

TypeScript helps catch silly bugs 

```
const user = {
  firstName: "Adam",
  lastName: "Thompson",
  role: "speaker",
}

console.log(user.name) // undefined
```

```
const user = {
  firstName: "Adam",
  lastName: "Thompson",
  role: "speaker",
}

console.log(user.name)
// ❌ Property 'name' does not exist on type
'{ firstName: string; lastName: string; role: string; }'
```

```
const user = {
  firstName: "Adam",
  lastName: "Thompson",
  role: "speaker",
}

console.log(user.name)
// ❌ Property 'name' does not exist on type
'{ firstName: string; lastName: string; role: string; }'
```

```
interface User {
  firstName: string,
  lastName: string,
  role: string
}
```

TypeScript makes things predictable

```
function fullName(user) {
  return `${user.firstName} ${user.lastName}`
}
```

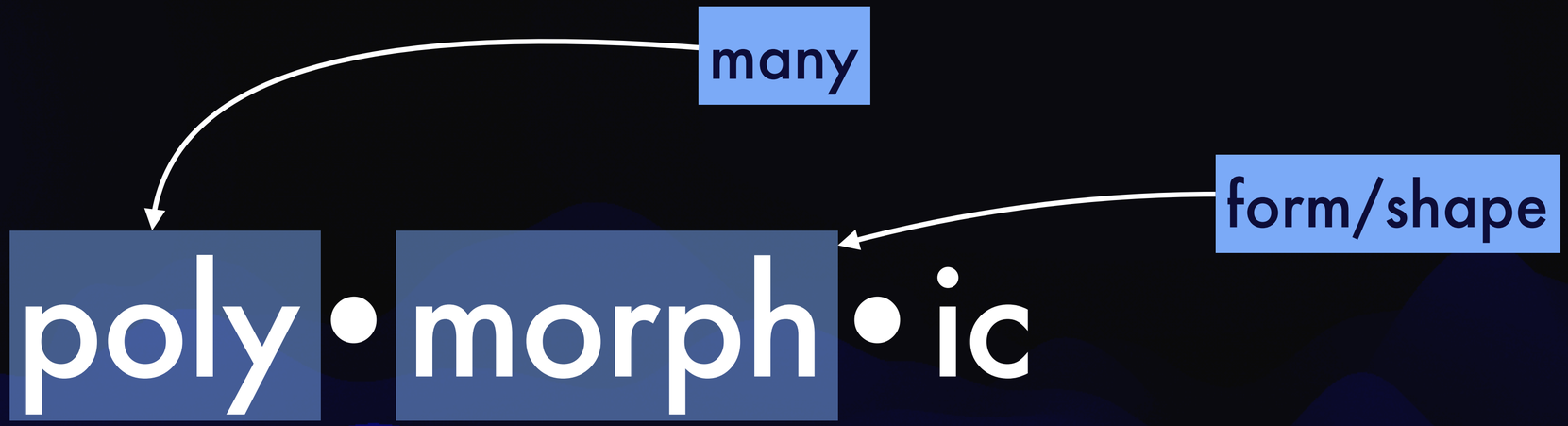
```ts
interface User {
  firstName: string,
  lastName: string,
  role: string
}

function fullName(user: User) {
  return `${user.firstName} ${user.lastName}`
}
```

```ts
interface User {
  firstName: string,
  lastName: string,
  role: string
}

function fullName(user: User) {
  return `${user.firstName} ${user.middleInitial} ${user.lastName}`
}

// ❌ Property 'middleInitial' does not exist on type ‘User'
```

Type-Safe, ✅
Polymorphic, ⁉️
React Component ✅

poly • morph • ic

- occurring in several different forms, in particular with reference to species or genetic variation.
- (of a feature of a programming language) allowing routines to use variables of different types at different times.



- occurring in several different forms, in particular with reference to species or genetic variation.
- (of a feature of a programming language) allowing routines to use variables of different types at different times.

poly • morph • ic

- occurring in several different forms, in particular with reference to species or genetic variation.
- (of a feature of a programming language) allowing routines to use variables of different types at different times.

```
import React from 'react';

const Button = ({ children, ...props}) => {
  return <button {...props}>{children}</button>
}
```

- Semantic HTML
- Remix
- NextJS
- React Router
- Etc...

`Button` needs to be **polymorphic**

Basic Polymorph

```
import React from 'react';

const Button = ({ children, ...props}) => {
  return <button {...props}>{children}</button>
}
```

```tsx
interface ButtonProps {
  type?: 'button' | ‘submit',
  onClick?: React.MouseEventHandler,
  children?: React.ReactNode
}
```

```tsx
interface ButtonProps {
  type?: 'button' | ‘submit',
  onClick?: React.MouseEventHandler,
  children?: React.ReactNode
}

const Button = ({ children, ...props}: ButtonProps) => {
  return <button {...props}>{children}</button>
}
```

```ts
interface PolymorphicButtonProps {
  as?: 'button' | 'a',
  ...
}
```

```tsx
interface PolymorphicButtonProps {
  as?: 'button' | ‘a',

  // <a> props
  href?: string,
  rel?: string,
  target?: string,

  // <button> props
  type?: 'button' | ‘submit’,
  onClick?: React.MouseEventHandler,
  children?: React.ReactNode
}
```

```tsx
interface ButtonProps {
  type?: 'button' | ‘submit',
  onClick?: React.MouseEventHandler,
  children?: React.ReactNode
}
```

```tsx
import { ComponentProps } from ‘react’;

interface ButtonProps extends ComponentProps<'button'> {}
```

```tsx
interface PolymorphicButtonProps {
  as?: 'button' | ‘a',

  // <a> props
  href?: string,
  rel?: string,
  target?: string,

  // <button> props
  type?: 'button' | ‘submit’,
  onClick?: React.MouseEventHandler,
  children?: React.ReactNode
}
```

```tsx
import { ComponentProps } from ‘react’;

interface PolymorphicButtonProps extends ComponentProps<‘button’>,
  ComponentProps<'a'> {
    as?: 'button' | 'a',
}
```

```tsx
interface PolymorphicButtonProps extends ComponentProps<‘button’>,
  ComponentProps<'a'> {
    as?: 'button' | 'a',
}


const Button = ({ as, ...props}: PolymorphicButtonProps) => {
   const Component = as;
   return <Component {…props}/>
}
```

```
<Button as="a" href="reactrally.com">React Rally</Button>
```

```
// ⚠️ No TS error!
<Button as="button" href="reactrally.com">React Rally</Button>

// ⚠️ No TS error!
<Button as=“a" disabled={true}>React Rally</Button>
```

Discriminated Union

Discriminated Union

Like TypeScript's `switch` statement

```ts
interface Shape {
  kind?: "circle" | "square" | "triangle",
  radius?: number,
  length?: number,
  height?: number
}

function calculateArea(shape: Shape) {
   switch(shape.kind) {
     case "circle":
       return Math.PI * shape.radius * shape.radius;
     case "square":
       return shape.length * shape.length;
     case "triangle":
       return (shape.length * shape.height) / 2
   }
}
```

```ts
interface Shape {
  kind?: "circle" | "square" | "triangle",
  radius?: number,
  length?: number,
  height?: number
}

function calculateArea(shape: Shape) {
   switch(shape.kind) {
     case "circle":
       return Math.PI * shape.radius * shape.radius;
     case "square":
       return shape.length * shape.length;
     case "triangle":
       return (shape.length * shape.height) / 2
   }
}
// ❌ ‘shape.radius' is possibly ‘undefined' (and 5 other errors)
```

```ts
type Shape =
  | { kind: "circle", radius: number }
  | { kind: "square", length: number }
  | { kind: "triangle", length: number, height: number }
```

```ts
type Shape =
  | { kind: "circle", radius: number }
  | { kind: "square", length: number }
  | { kind: "triangle", length: number, height: number }

function calculateArea(shape: Shape) {
  switch(shape.kind) {
    case "circle":
      return Math.PI * shape.radius * shape.radius;
    case "square":
      return shape.length * shape.length;
    case "triangle":
      return (shape.length * shape.height) / 2
  }
}
```

Polymorphic Discriminated Union

```tsx
import { ComponentProps } from ‘react’;

interface PolymorphicButtonProps extends ComponentProps<‘button’>,
  ComponentProps<'a'> {
    as?: 'button' | 'a',
}
```

```
import { ComponentProps } from ‘react’;

type PolymorphicButtonProps =
  | ({
      as: ‘button',
  } & ComponentProps<‘button’>)
  | ({
      as: 'a',
  } & ComponentProps<‘a’>)
```

```
<Button as="button" href=“reactrally.com”>React Rally</Button>
// ❌ Property 'href' does not exist on type ...

<Button as=“a" disabled={true}>React Rally</Button>
// ❌ Property 'disabled' does not exist on type …
```

Only Buttons & Links…

```
type PolymorphicButtonProps =
  | ({
      as: ‘button',
    } & ComponentProps<‘button’>)
  | ({
      as: 'a',
    } & ComponentProps<‘a’>)
```

```
type PolymorphicButtonProps =
  | ({
      as: ‘button',
    } & ComponentProps<‘button’>)
  | ({
      as: 'a',
    } & ComponentProps<‘a’>)
  | ({
      as: 'li',
    } & ComponentProps<‘li’>)
  | ({
      as: 'span',
    } & ComponentProps<‘span’>)
  | ({
      as: 'div',
    } & ComponentProps<‘div’>)
  ...
```

```tsx
import NextLink from ‘next/link’;
import { Link as RemixLink } from ‘@remix-run/react’

type PolymorphicButtonProps =
  ...
  | ({
      as: ‘div',
  } & ComponentProps<‘div’>)
  | ({
      as: NextLink,
  } & ComponentProps<NextLink>)
  | ({
      as: RemixLink,
  } & ComponentProps<RemixLink>)
  ...
```

```tsx
import NextLink from ‘next/link’;
import { Link as RemixLink } from ‘@remix-run/react’

type PolymorphicButtonProps =
  ...
  | ({
      as: ‘div',
    } & ComponentProps<‘div’>)
  | ({
      as: NextLink,
    } & ComponentProps<NextLink>)
  | ({
      as: RemixLink,
    } & ComponentProps<RemixLink>)
  ...
```

Don’t Repeat Yourself

Generics

Generics

Like TypeScript's function parameters

```
const myArray: Array<string> = ["Adam", "Brooke", "Chris"]
```

```
const myArray: Array<string> = ["Adam", "Brooke", "Chris"]
```

Generic Values

JavaScript values that can be typed

- Functions
- Classes
- Components

Generic Types

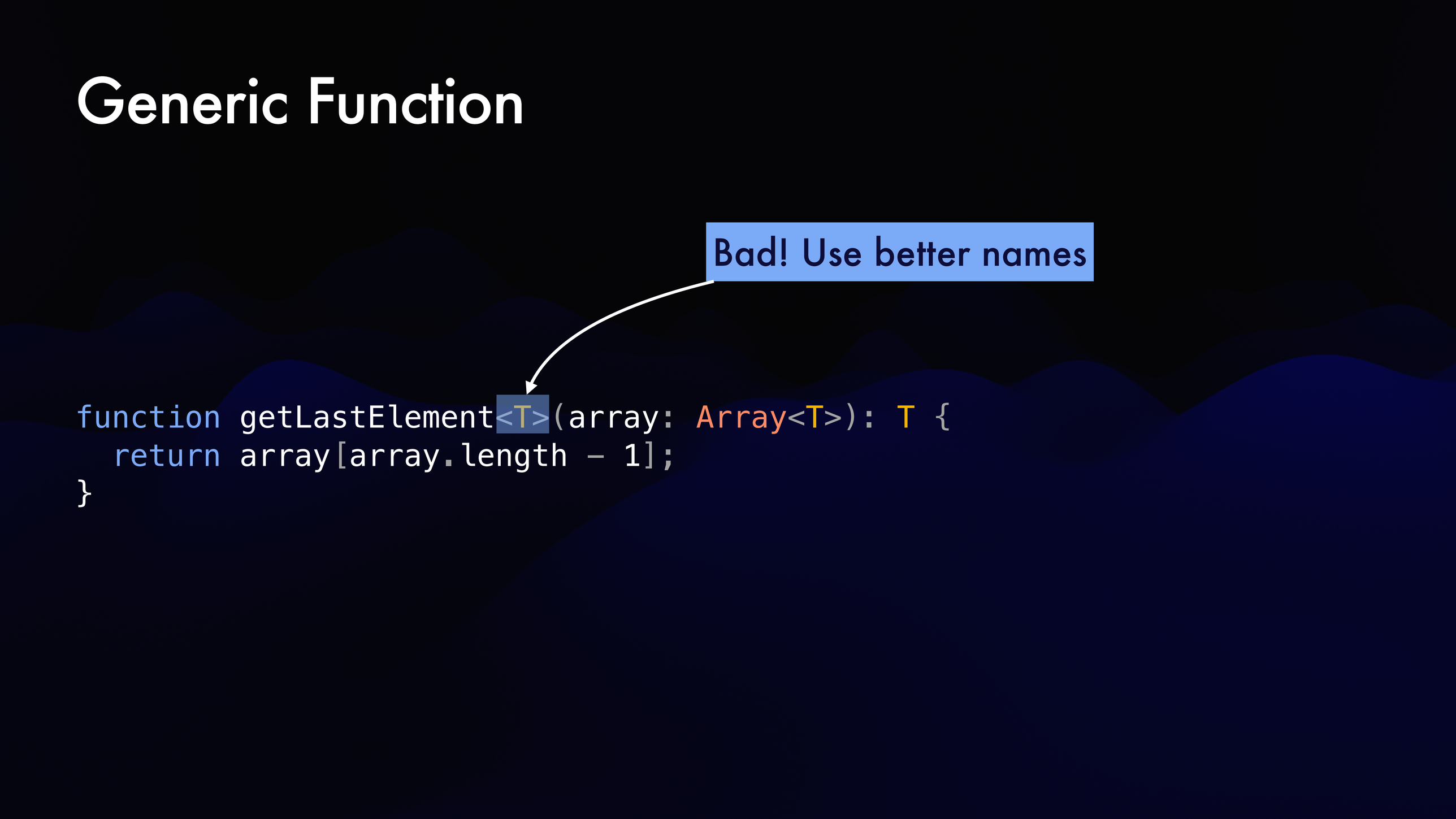
TypeScript entities that are variable

- Interfaces
- Types

Generic Function

```
function getLastElement<T>(array: Array<T>): T {
  return array[array.length - 1];
}
```

Generic Function




```
function getLastElement<T>(array: Array<T>): T {
  return array[array.length - 1];
}
```

Generic Function

```
const lastString = getLastElement<string>(["Adam", "Brooke", "Dave"]);
//    ^ const lastString: string

const lastNumber = getLastElement<number>([1, 14, 93]);
//    ^ const lastNumber: number
```

Generic Function

```
function getLastElement<T>(array: Array<T>): T {
  return array[array.length - 1];
}

const lastString = getLastElement(["Adam", "Brooke", "Dave"]);
//    ^ const lastString: string

const lastNumber = getLastElement([1, 14, 93]);
//    ^ const lastNumber: number

const lastThing = getLastElement(["Adam", 93, false]);
//    ^ const lastThing: string | number | boolean
```

Generic Values

JavaScript values that can be typed

- Functions
- Classes
- Components

Generic Types

TypeScript entities that are variable

- Interfaces
- Types

Generic Types

```
interface Selection<T>{
  current: T,
  options: Array<T>
}
```

Generic Types

```
interface Selection<T>{
  current: T,
  options: Array<T>
}

const speaker: Selection<string> = {
  current: "Adam",
  options: [“Dev", “Adam", “Corbin", "Kent"]
}
```

Generic Types

```
interface Selection<T>{
  current: T,
  options: Array<T>
}

const element: Selection<HTMLElement> = {
  current: document.activeElement(),
  options: Array.from(document.querySelectorAll('*'))
}
```

Generic Types

```
interface SelectedElement<T extends HTMLElement>{
  current: T | null,
  options: Array<T>
}

const focusedElement: SelectedElement<HTMLButtonElement> = {
  current: null,
  options: Array.from(document.querySelectorAll('button'))
}
```

Generic Types


```
interface SelectedElement<T extends HTMLElement>{
  current: T | null,
  options: Array<T>
}

const focusedElement: SelectedElement<HTMLButtonElement> = {
  current: null,
  options: Array.from(document.querySelectorAll('button'))
}
```

Generic React Components

Generic Components

```
interface SelectProps<M extends boolean> {
  multiselect: M;
  value: M extends true ? Array<string> : string;
}
```

Generic Components

```
const Select = <M extends boolean>({
  multiselect,
  value
}: SelectProps<M>) => {
  const label = `Select ${multiselect ? 'some' : 'a'} fruit`;
  return (
    <>
      <label htmlFor="select-element">{label}</label>
      <select id="select-element" value={value}>
        <option>Apple</option>
        // ...
      </select>
    </>
  );
};
```

Generic Components

```
const Select = <M extends boolean>({
  multiselect,
  value
}: SelectProps<M>) => {
  const label = `Select ${multiselect ? 'some' : 'a'} fruit`;
  return (
    <>
      <label htmlFor="select-element">{label}</label>
      <select id="select-element" value={value}>
        <option>Apple</option>
        // ...
      </select>
    </>
  );
};
```

Must use `extends` syntax!

Generic Components

```
<Select multiselect value={['Apple', 'Banana']} />;

<Select multiselect={false} value={'Apple'} />;

<Select multiselect value={'Apple'} />;
// ❌ Type 'string' is not assignable to type 'string[]'
```

Generic Components

```
interface SelectProps<M extends boolean> {
  multiselect: M;
  value: M extends true ? Array<string> : string;
}
```

Generic Components

```
interface SelectProps<M extends boolean> {
  multiselect: M;
  value: M extends true ? Array<string> : string;
}
```

Conditional Types

Like TypeScript's ternary operator

Conditional Types

```
type MyType = (SomeType extends OtherType) ? TypeIfTrue : TypeIfFalse
```

Conditional Types

```
type NameOrId<T extends number | string> = T extends number
  ? { id: number }
  : { name: string };

const x: NameOrId<string> = {
  name: "Adam"
}
```

Conditional Types

```tsx
interface SelectProps<M extends boolean> {
  multiselect: M;
  value: M extends true ? Array<string> : string;
}

<Select multiselect={true} value={['Apple', 'Banana']} />;

<Select multiselect={false} value={'Apple'} />;
```

Conditional Types

```
interface SelectProps<M extends boolean> {
  multiselect: M;
  value: M extends true ? Array<string> : string;
}

const myProps: SelectProps<true> = {
  multiselect: true,
  value: ["Apple", “Banana"]
}

React.createElement(Select, myProps);
```

Conditional Types

```
interface SelectProps<M extends boolean> {
  multiselect: M;
  value: M extends true ? Array<string> : string;
}

const myProps: SelectProps<true> = {
  multiselect: true,
  value: ["Apple", “Banana"]
}

React.createElement(Select, myProps);
```

Type Literal

JavaScript value

Generic Polymorphic Component

```
type PolymorphicButtonProps =
  | ({
      as: ‘button',
    } & ComponentProps<‘button’>)
  | ({
      as: 'a',
    } & ComponentProps<‘a’>)
  | ({
      as: 'li',
    } & ComponentProps<‘li’>)
  | ({
      as: 'span',
    } & ComponentProps<‘span’>)
  | ({
      as: 'div',
    } & ComponentProps<‘div’>)
  ...
```

```tsx
type PolymorphicButtonProps <T> = {
  as?: T,
} & ComponentProps<T>
```

```tsx
type PolymorphicButtonProps <T extends React.ElementType> = {
  as?: T,
} & ComponentProps<T>
```

```tsx
const Button = <T extends React.ElementType>(
  {
    as,
    children,
    ...props
  }: PolymorphicButtonProps<T>) => {
   const Component = as || "span";
   return <Component {...props}>{children}</Component>
}
```

```
import { Link as RemixLink } from "@remix-run/react";

<Button as={RemixLink} to="reactrally.com">React Rally</Button>

<Button as={"button"} onClick={handleClick}>Like & Subscribe</Button>
```

TypeScript the Hard Way

- Polymorphism
- Interfaces & Types
- Built-in React types
- Interface extension
- Type literals
- Discriminated Unions
- Conditional Types
- Generics

Adam Thompson

@TheSonOfThomp

References

github.com/mongodb/leafygreen-ui/blob/main/packages/polymorphic/README.md

Director of Design Systems

MongoDB