TypeScript Oddities and Patterns

Charles Habermehl, HALO Link

Enums

```
enum Shape {
  Circle,
  Square,
  Rectangle,
}

```

Alternative: A JavaScript Object?

```
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
}

```

Alternative: A *Heavily* Manipulated JavaScript Object

```
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
} as const

```

With Some Additional Help From TypeScript

```ts
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
} as const;

type Shapes = typeof Shape[keyof typeof Shape];
```

```ts
type Shapes = "Circle" | "Square" | "Rectangle"
```

Discriminating Unions

Like regular unions, but with a twist

Union Type

```
type Shapes = 'Circle' | 'Square' | 'Rectangle';

type DrawShape = {
  shape: Shapes;
  radius?: number;
  width?: number;
  height?: number;
  sides?: number;
  color?: string;
}

```

Union Type

```
type Shapes = 'Circle' | 'Square' | 'Rectangle';

type DrawShape = {
  shape: Shapes;
  radius?: number;
  width?: number;
  height?: number;
  sides?: number;
  color?: string;
}

```

Union Type

```
type Shapes = 'Circle' | 'Square' | 'Rectangle';

type DrawShape = {
  shape: Shapes;
  radius?: number;
  width?: number;
  height?: number;
  sides?: number;
  color?: string;
}

```

```ts
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
} as const;

type Shapes = typeof Shape[keyof typeof Shape];

type ShapeBase = {
  color: string;
}

type DrawShape =
| {
  shape: Shape.Circle;
  radius: number;
} & ShapeBase
| {
  shape: Shape.Square;
  width: number;
} & ShapeBase
| {
  shape: Shape.Rectangle;
  width: number;
  height: number;
} & ShapeBase;

```

```ts
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
} as const;

type Shapes = typeof Shape[keyof typeof Shape];

type ShapeBase = {
  color: string;
}

type DrawShape =
| {
  shape: Shape.Circle;
  radius: number;
} & ShapeBase
| {
  shape: Shape.Square;
  width: number;
} & ShapeBase
| {
  shape: Shape.Rectangle;
  width: number;
  height: number;
} & ShapeBase;

```

```ts
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
} as const;

type Shapes = typeof Shape[keyof typeof Shape];

type ShapeBase = {
  color: string;
}

type DrawShape =
| {
  shape: Shape.Circle;
  radius: number;
} & ShapeBase
| {
  shape: Shape.Square;
  width: number;
} & ShapeBase
| {
  shape: Shape.Rectangle;
  width: number;
  height: number;
} & ShapeBase;

```

```ts
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
} as const;

type Shapes = typeof Shape[keyof typeof Shape];

type ShapeBase = {
  color: string;
}

type DrawShape =
| {
  shape: Shape.Circle;
  radius: number;
} & ShapeBase
| {
  shape: Shape.Square;
  width: number;
} & ShapeBase
| {
  shape: Shape.Rectangle;
  width: number;
  height: number;
} & ShapeBase;

```

```ts
const Shape = {
  Circle: 'Circle',
  Square: 'Square',
  Rectangle: 'Rectangle',
} as const;

type Shapes = typeof Shape[keyof typeof Shape];

type ShapeBase = {
  color: string;
}

type DrawShape =
| {
  shape: Shape.Circle;
  radius: number;
} & ShapeBase
| {
  shape: Shape.Square;
  width: number;
} & ShapeBase
| {
  shape: Shape.Rectangle;
  width: number;
  height: number;
} & ShapeBase;

```

Generic Type Constraining

Where application code stops and library code starts

Generic Type Constraining Basics

```
1  const logString = <TString extends string>(toLog: TString) => {
2      console.log(toLog);
3  }
4
5  logString("this is my string to log");
6  logString(1045);
7
```

Generic Type Constraining Basics

```
const logString = <TString extends string>(toLog: TString) => {
    console.log(toLog);
}

logString("this is my string to log");
logString(1045);

```

Generic Type Constraining Basics

```
const logString = <TString extends string>(toLog: TString) => {
    console.log(toLog);
}

logString("this is my string to log");
logString(1045);

```

Generic Type Constraining In Practice

```tsx
interface IMultiListField<TValue, TComponents> {
  /** `react-select` component overrides */
  components?: TComponents;
  // ... other props omitted for brevity
}

const MultiListField = <
  TValue extends {value: string; label: string},
  TComponents extends SelectComponentsConfig<TValue, true, GroupBase<TValue>>
>(
  props: IMultiListField<TValue, TComponents>
): JSX.Element => {
  return (
    <Select<TValue, true, GroupBase<TValue>>
      components={components}
      // ... other props omitted for brevity
    />
  );
}

```

Generic Type Constraining In Practice

```tsx
interface IMultiListField<TValue, TComponents> {
  /** `react-select` component overrides */
  components?: TComponents;
  // ... other props omitted for brevity
}

const MultiListField = <
  TValue extends {value: string; label: string},
  TComponents extends SelectComponentsConfig<TValue, true, GroupBase<TValue>>
>(
  props: IMultiListField<TValue, TComponents>
): JSX.Element => {
  return (
    <Select<TValue, true, GroupBase<TValue>>
      components={components}
      // ... other props omitted for brevity
    />
  );
}

```

Generic Type Constraining In Practice

```tsx
interface IMultiListField<TValue, TComponents> {
  /** `react-select` component overrides */
  components?: TComponents;
  // ... other props omitted for brevity
}

const MultiListField = <
  TValue extends {value: string; label: string},
  TComponents extends SelectComponentsConfig<TValue, true, GroupBase<TValue>>
>(
  props: IMultiListField<TValue, TComponents>
): JSX.Element => {
  return (
    <Select<TValue, true, GroupBase<TValue>>
      components={components}
      // ... other props omitted for brevity
    />
  );
}

```

Generic Type Constraining In Practice

```tsx
interface IMultiListField<TValue, TComponents> {
  /** `react-select` component overrides */
  components?: TComponents;
  // ... other props omitted for brevity
}

const MultiListField = <
  TValue extends {value: string; label: string},
  TComponents extends SelectComponentsConfig<TValue, true, GroupBase<TValue>>
>(
  props: IMultiListField<TValue, TComponents>
): JSX.Element => {
  return (
    <Select<TValue, true, GroupBase<TValue>>
      components={components}
      // ... other props omitted for brevity
    />
  );
}

```

Generic Type Constraining In Practice

```tsx
interface IMultiListField<TValue, TComponents> {
  /** `react-select` component overrides */
  components?: TComponents;
  // ... other props omitted for brevity
}

const MultiListField = <
  TValue extends {value: string; label: string},
  TComponents extends SelectComponentsConfig<TValue, true, GroupBase<TValue>>
>(
  props: IMultiListField<TValue, TComponents>
): JSX.Element => {
  return (
    <Select<TValue, true, GroupBase<TValue>>
      components={components}
      // ... other props omitted for brevity
    />
  );
}

```

Questions?