TYPESCRIPT & FP

AN INTRODUCTION

INTRO TO FP

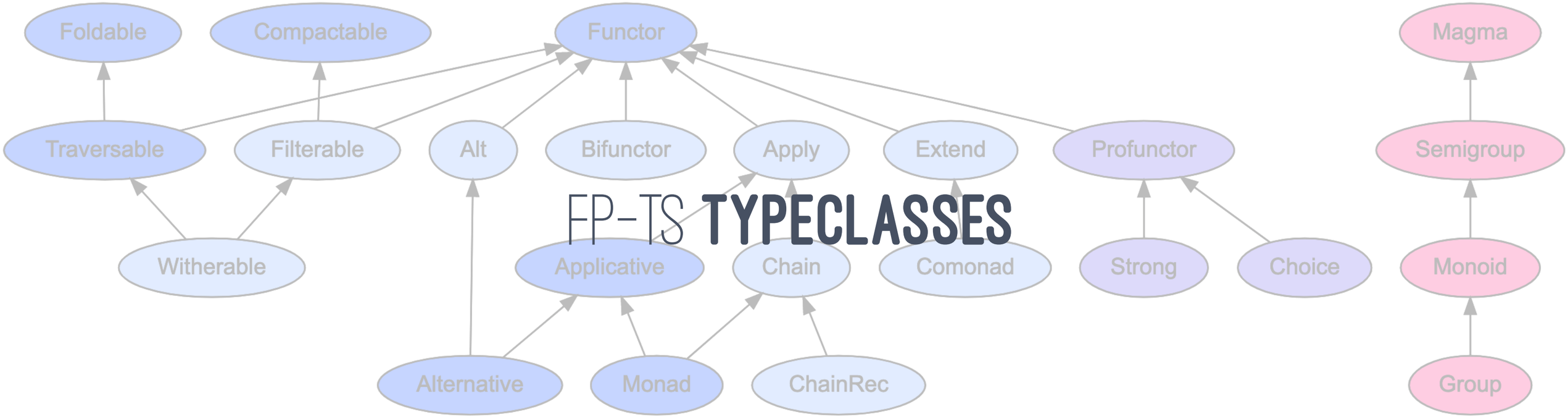
OO	FP
DEPENDENCY INJECTION	FUNCTION
VISITOR PATTERN	FUNCTION
DECORATOR	FUNCTION
MIDDLEWARE	FUNCTION
FACTORY	FUNCTION

HKT

ADT

MONOID

FUNCTOR



FP-TS **TYPECLASSES**

AGENDA

1. RULES OF TYPED FP IN TS
2. MEET FP-TS
3. TYPE-DRIVEN DESIGN
4. PARSING

Typed functional programming in TypeScript

`fp-ts` provides developers with popular patterns and reliable abstractions from typed functional languages in TypeScript.

DATA

- PLAIN VALUES, NO BEHAVIOR ATTACHED
- GENERALLY IMMUTABLE: DIFFERENT VALUE $\iff$ DIFFERENT REFERENCE

FUNCTIONS

- IN THE MATHEMATICAL SENSE: NO "IMPURE" OR "SIDE-EFFECTFUL" FUNCTIONS
- EACH ELEMENT OF THE DOMAIN MAPS TO A SINGLE ELEMENT OF THE CODOMAIN
- "TOTAL": DEFINED FOR EVERY VALUE OF THE DOMAIN

TYPED FUNCTIONAL PROGRAMMING IN TS - RULES

- `"strict": true` IN TSCONFIG.JSON
- ANNOTATE FUNCTION RETURN TYPES
- DEFINE `total` FUNCTIONS
- WHEN POSSIBLE. PREFER `polymorphic` FUNCTIONS
- USE "TYPE DRIVEN DEVELOPMENT": `declare function`

TYPED ~~FUNCTIONAL~~ PROGRAMMING IN TS - RULES

- `"strict": true` IN TSCONFIG.JSON
- ANNOTATE FUNCTION RETURN TYPES
- DEFINE `total` FUNCTIONS
- WHEN POSSIBLE. PREFER `polymorphic` FUNCTIONS
- USE "TYPE DRIVEN DEVELOPMENT": `declare function`

ARE THESE SIGNATURES OK?

```
declare function sum(a: number, b: number): number;

declare function length(a: string): number;

declare function replicate<A>(a: A, n: number): Array<A>;
```

ARE THESE SIGNATURES OK?

```
declare function parseInt(s: string): number;

declare function head<A>(as: Array<A>): A;
```

MEET COMPOSITION IN FP-TS

PIPE AND **FLOW**

PLAIN JS/TS COMPOSITION

```
result = length(double("foo"));
```

USING THE "PIPE" OPERATOR, E.G. IN F#

```
result = "foo" |> double |> length
```

USING pipe FROM fp-ts

```
import { pipe } from "fp-ts/function";

result = pipe("foo", double, length);
```

USING flow FROM fp-ts

```
import { flow } from "fp-ts/function";

const doubleLength = flow(double, length);

const result = doubleLength("foo");
```

CAR RENTAL FORM EXAMPLE

- **E-MAIL**
- **PAYMENT MODE** AT PICKUP OR ONLINE (NOW)
- **AGE RANGE** 18-25-27+
- EMAIL IS ONLY REQUIRED IF PAYING "ONLINE"
- AT FORM SUBMIT, DISPLAY THE TOTAL PRICE BASED ON USER INPUT

PRODUCT TYPE

```ts
type PaymentMode = "online" | "pickup";

type AgeRange = "18-25" | "25-27" | "27+";

type ConfirmRent = {
  ageRange: AgeRange;
  paymentMode: PaymentMode;
  email?: string;
};
```

PRODUCT TYPE

ALLOWS IMPOSSIBLE STATES

```
export const payload1: ConfirmRent = {
  paymentMode: "pickup",
  ageRange: "18-25",
};

export const payload2: ConfirmRent = {
  paymentMode: "online",
  ageRange: "18-25",
};
```

SUM TYPE

```
export type OnlinePayment = {
  paymentMode: "online";
  ageRange: AgeRange;
  email: string;
};

export type PickupPayment = {
  paymentMode: "pickup";
  ageRange: AgeRange;
};

export type ConfirmRent = OnlinePayment | PickupPayment;
```

SUM TYPE

REPRESENTS POSSIBLE STATES ONLY

```
export const payload3: ConfirmRent = {
  paymentMode: "pickup",
  ageRange: "18-25",
  email: "test@example.com", // correctly errors now
};

export const payload4: ConfirmRent = {
  paymentMode: "online",
  ageRange: "18-25",
};
// Property 'email' is missing in type '{ ... }'
// but required in type 'OnlinePayment'
```

SUM TYPES IN TS RECAP

- ENCODED AS "DISCRIMINATED UNIONS"
- MAKE IMPOSSIBLE STATES NOT REPRESENTABLE
- EXHAUSTIVENESS CHECKING
- USE HELPER LIBS TO REDUCE THE BOILERPLATE

VALIDATION

```
function isFormValid(formState: FormState): boolean;
```

DOES NOT PRESERVE INFORMATION AT TYPE-LEVEL.

FORCING THE CALLER TO "CAST"

```
if (isFormValid(formState)) {
  formState.email; // could still be undefined from TS perspective
}
```

PARSING

```
function parseFormState(formState: FormState): Option<ValidState>;
```

PRESERVES INFORMATION AT TYPE-LEVEL

```
pipe(
  formState,
  parseFormState,
  option.fold(
    () => "Invalid",
    (valid) => valid.email // TS knows this is defined
  )
);
```

PARSING IN TS RECAP

- USING PARSERS, THERE IS NO NEED FOR CASTS
- PARSERS ENCODE IN THE RETURN TYPE THE VALID TYPE
- `io-ts` SIMPLIFIES WRITING PARSERS

LINKS

- SLIDES+CODE
- FP INTRO BY @GCANTI
- FP-TS/IO-TS
- FP-TS ECOSYSTEM/FP-TS LEARNING RESOURCES
- DOMAIN MODELING MADE FUNCTIONAL
- PARSE, DON'T VALIDATE
- FP-CHAT SLACK, #TYPESCRIPT AND #FP-TS CHANNELS

@GIOGONZO

ANY QUESTION?