Lekcja 24: Praca z typami wartości

Włączmy naszą nową wiedzę na temat wartości i typów związków do naszej struktury danych wydarzeń technicznych. W lekcji 22 ustaliliśmy typ TechEventBase, który obejmuje wszystkie typowe właściwości każdego wydarzenia technicznego:

type TechEventBase = {

title: string,

description: string

date: Date,

capacity: number,

rsvp: number,

kind: string

}

Ostatnia właściwość tego typu nazywa się kind i zawiera informacje o rodzaju zdarzenia technicznego, z którym mamy do czynienia. W tej chwili typ kind to string, ale wiemy, że ten typ może przyjmować tylko trzy różne wartości:

type TechEventBase = {

title: string,

description: string

date: Date,

capacity: number,

rsvp: number,

kind: 'conference' | 'meetup' | 'webinar'

}

To już jest znacznie lepsze niż w poprzedniej wersji. Jesteśmy bardziej zabezpieczeni przed błędnymi wartościami i literówkami. Ma to natychmiastowy wpływ na to, co możemy zrobić z połączonym typem unii TechEvent. Spójrzmy na inną funkcję o nazwie getEventTeaser:

function getEventTeaser(event: TechEvent) {

switch(event.kind) {

case 'conference':

return `${event.title} (Conference)`

case 'meetup':

return `${event.title} (Meetup)`

case 'webinar':

return `${event.title} (Webinar)`

// Again: cool, but not possible

case 'concert':

}

}

TypeScript natychmiast zgłasza błąd, ponieważ typ ‘concert’ nie jest porównywalny z typem ‘conferene’ | ‘meetup’ | ;webinar’. Związki typów wartości są doskonałe do analizy przepływu sterowania. Nie spotykamy się z sytuacjami, które nie mogą się zdarzyć, ponieważ nasze typy nie obsługują takich sytuacji. Dbamy o wszystkie możliwe wartości zestawu.

Dyskryminowane typy związków

Ale możemy zrobić więcej. Zamiast umieszczać sumę trzech typów wartości w TechEventBase, możemy przenieść bardzo różne typy wartości do trzech określonych typów zdarzeń technicznych. Najpierw upuszczamy rodzaj z TechEventBase:

type TechEventBase = {

title: string,

description: string

date: Date,

capacity: number,

rsvp: number,

}

Następnie dodajemy różne typy wartości do każdego konkretnego wydarzenia technicznego.

type Conference = TechEventBase & {

location: string,

price: number,

talks: Talk[],

kind: 'conference'

}

type Meetup = TechEventBase & {

location: string,

price: string,

talks: Talk[],

kind: 'meetup'

}

```
type Webinar = TechEventBase & {

url: string,

price?: number,

talks: Talk,

kind: 'webinar'

}
```

Na pierwszy rzut oka wszystko pozostaje takie samo. Jeśli najedziesz kursorem myszy na właściwość event.kind w instrukcji przełącznika, zobaczysz, że typ rodzaju to nadal „conference” | „meetup” | „webinar”. Ponieważ wszystkie trzy typy zdarzeń technicznych są połączone w jeden typ unii, TypeScript tworzy odpowiedni typ unii dla tej właściwości, tak jak byśmy się tego spodziewali. Ale pod spodem dzieje się coś wspaniałego. Tam, gdzie przed TypeScript po prostu wiedział, że niektóre właściwości dużego typu unii TechEvent istnieją lub nie istnieją, z określonym typem wartości dla właściwości możemy bezpośrednio wskazać otaczający typ obiektu. Zobaczmy, co to oznacza dla funkcji getEventTeaser:

```
function getEventTeaser(event: TechEvent) {

switch(event.kind) {

case 'conference':

// We now know that I'm in type Conference

return `${event.title} (Conference), ` +

// Suddenly I don't have to check for price as

// TypeScript knows it will be there

`priced at ${event.price} USD`

case 'meetup':

// We now know that we're in type Meetup

return `${event.title} (Meetup), ` +

// Suddenly we can say for sure that this

// event will have a location, because the

// type tells us

`hosted at ${event.location}`

case 'webinar':

// We now know that we're in type Webinar

return '${event.title} (Webinar), ' +

// Suddenly we can say for sure that there will

// be a URL
```

```
`available online at ${event.url}`

default:

throw new Error('Not sure what to do with that!')

}

}
```

Używanie typów wartości dla właściwości działa jak haczyk dla TypeScript w celu znalezienia dokładnego kształtu wewnątrz unii. Takie typy nazywane są rozłącznymi typami unii i są bezpiecznym sposobem poruszania się po przestrzeni typów TypeScript.

Ustalanie typów wartości

Związki dyskryminujące są wspaniałym narzędziem, gdy chcesz skierować swój przepływ kontroli we właściwym kierunku. Ale wiąże się to z pewnymi problemami, gdy w dużym stopniu polegasz na wnioskowaniu o typie (co powinieneś). Zdefiniujmy obiekt konferencji poza tym, co otrzymujemy z zaplecza.

```
const script19 = {

title: 'ScriptConf',

date: new Date('2019-10-25'),

capacity: 300,

rsvp: 289,

description: 'The feel-good JS conference',

kind: 'conference',

price: 129,

location: 'Central Linz',

talks: [{

speaker: 'Vitaly Friedman',

title: 'Designing with Privacy in mind',

abstract: '...'

}]

};
```

Według naszego podpisu typu byłaby to idealnie dobra wartość typu TechEvent (lub Conference). Jednak gdy przekażemy tę wartość do funkcji getEventTeaser, TypeScript uderzy nas czerwoną falistą linią.

```
getEventTeaser(script19)
```

Według TypeScript typy skryptów i TechEvent są niekompatybilne. Problem tkwi w wnioskowaniu o typie. W momencie, gdy przypisujemy tę wartość do zmiennej script19, TypeScript próbuje odgadnąć

poprawny typ każdej wartości właściwości i dąży do zestawu, który z pewnością zadziała. Podobnie jak w przypadku obiektów const, wszystkie właściwości są nadal zmienne, a typy wywnioskowane to głównie łańcuchy i liczby dla prostych właściwości. Oznacza to, że właściwość kind w script19 nie zostanie wywnioskowany jako „onferene", ale jako string. A string to znacznie szerszy zbiór wartości niż „conference". Aby to zadziałało, musimy ponownie powiedzieć TypeScript, że szukamy typu wartości, a nie ich nadzbioru typów. Mamy na to kilka możliwości. Najpierw zróbmy adnotację po lewej stronie.

const script19: TechEvent = {

// Wszystkie właściwości sprzed ...

}

Dzięki temu TypeScript sprawdza typ bezpośrednio przy przypisaniu. W ten sposób wartość „coference" dla kind będzie postrzegana jako typ wartości z adnotacją, a nie znacznie szerszy string. Nie tylko to, ale TypeScript również zrozumie, z jakim podtypem unii typów dyskryminowanych mamy do czynienia. Jeśli najedziesz kursorem na sript19, zobaczysz, że TypeScript poprawnie rozpozna tę wartość jako Conference.

Zadeklarowany jako TechEvent, rozumiany jako Konferencja

Jednak tracimy część udogodnień, które otrzymujemy, gdy polegamy na wnioskowaniu o typie. Przede wszystkim tracimy możliwość wykorzystania strukturalnego pisania i swobodnej pracy z obiektami, które po prostu muszą być zgodne z typami, a nie jawnie mieć określony kształt. W takich scenariuszach możemy naprawić pewne właściwości, wykonując rzutowanie typów. Jednym ze sposobów byłoby rzutowanie typu właściwości konkretnie na typ wartości:

const script19 = {

title: 'ScriptConf',

date: new Date('2019-10-25'),

capacity: 300,

rsvp: 289,

description: 'The feelgood JS conference',

- kind: 'conference',

```
+ kind: 'conference' as 'conference',

price: 129,

location: 'Central Linz',

talks: [{

speaker: 'Vitaly Friedman',

title: 'Designing with Privacy in Mind',

abstract: '...'

}]

};
```

To zadziała, ale tracimy pewne bezpieczeństwo typu, ponieważ możemy również określić „meetup” jako „conference”. Nagle znowu nie wiemy, z jakimi typami mamy do czynienia, a tego chcemy uniknąć. O wiele lepiej jest powiedzieć TypeScript, że chcemy zobaczyć tę wartość w kontekście const:

```
const script19 = {

title: 'ScriptConf',

date: new Date('2019-10-25'),

capacity: 300,

rsvp: 289,

description: 'The feelgood JS conference',

- kind: 'conference',

+ kind: 'conference' as const,

price: 129,

location: 'Central Linz',

talks: [{

speaker: 'Vitaly Friedman',

title: 'Designing with Privacy in mind',

abstract: '...'

}]

};
```

Działa to tak samo, jak przypisywanie wartości pierwotnej do stałej i ustalanie jej typu wartości.

To, co otrzymujemy jako const

Możesz zastosować zdarzenia kontekstu const do obiektów, rzutując wszystkie właściwości na ich typy wartości, skutecznie tworząc typ wartości całego obiektu. Jako efekt uboczny, cały obiekt staje się tylko do odczytu.