Lekcja 27: Na dole: never

Przy całym tym poszerzaniu i zawężaniu zbiorów, nawet do pojedynczych wartości będących typem, musimy zadać sobie pytanie: czy możemy być jeszcze węższi? Tak możemy. Jest jeden typ, który znajduje się na samym dole hierarchii typów. Jeden typ, który jest jeszcze mniejszym zbiorem niż zestaw z jedną wartością. Typ bez wartości. Pusty zbiór: never.

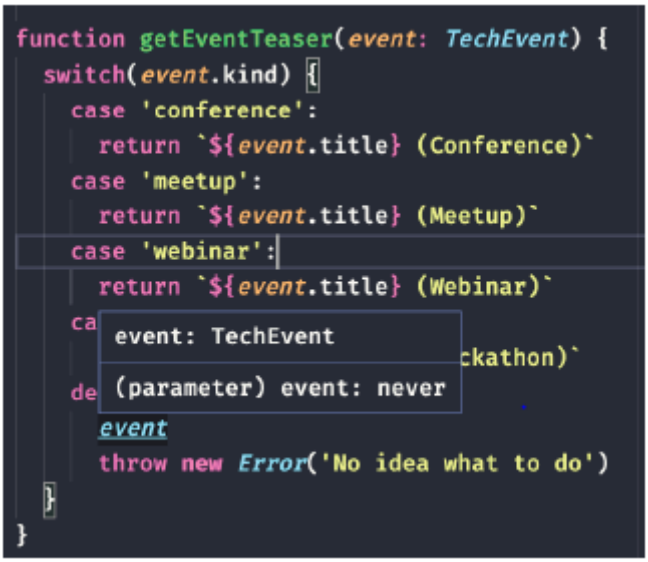
never w analizie przepływu sterowania

never nie zachowuje się prawie jak typ any. Podczas gdy any akceptuje wszystkie wartości i wszystkie operacje na tych wartościach, never w ogóle nie akceptuje pojedynczej wartości. Niemożliwe jest przypisanie wartości i oczywiście nie ma żadnych operacji, które możemy wykonać na typie, którego never nie możemy. Jak więc czuje się typ bez wartości, gdy z nim pracujemy? Poruszyliśmy już to krótko; był ukryty na widoku. Wróćmy do Lekcji 24 i przypomnijmy sobie, co zrobiliśmy, pisząc funkcję getEventTeaser, teraz z uwzględnieniem typu Hackathon:

```
function getEventTeaser(event: TechEvent) {

switch(event.kind) {

case 'conference':

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

`priced at ${event.price} USD'

case 'meetup':

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

`hosted at ${event.location}`

case 'webinar':

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

`available online at ${event.url}`

case 'hackathon':

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

default:

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

}

}
```

Ta instrukcja switch działa przez wszystkie typy wartości w ramach typu unii EventKind: „conferene” | „meetup” | „webinar” | „hackathon”. Z każdą instrukcją case w naszym swith TypeScript wie, że należy wziąć jeden typ wartości z tej listy. Po sprawdzeniu „conference” nie można jej później sprawdzić ponownie. Po wyczerpaniu tej listy nie mamy już żadnych wartości w naszym zbiorze. Lista jest pusta. To jest gałąź default w naszej instrukcji switch. Ale gdybyśmy sprawdzili wszystkie wartości na naszej liście, dlaczego i tak mielibyśmy natknąć się na gałąź default? Czy nie byłoby to błędne zachowanie? Dokładnie! Jest to wysoce błędne, na co wskazujemy, rzucając od razu nowy błąd! Uruchomienie gałęzi

default nigdy się nie zdarzy. Nigdy! To było, słowo never. Więc o to chodzi w typie never. Wskazuje przypadki, które nie powinny się wydarzyć, mówiąc nam, że powinniśmy być bardzo ostrożni, ponieważ nasze zmienne prawdopodobnie nie zawierają oczekiwanych wartości. Jeśli weźmiesz powyższy przykład, wpisz event w pierwszym wierszu domyślnej gałęzi i najedź na niego, TypeScript pokaże ci dokładnie to.



Lista jest wyczerpana, wydarzenie nigdy się nie kończy.

Każda operacja na event, inna niż będąca częścią zgłoszonego błędu, spowoduje błędy kompilatora. To jest sytuacja, która w ogóle nie powinna mieć miejsca!

Przygotowanie do aktualizacji dynamicznych

W tej chwili nasza funkcja getEventTeaser obsługuje wszystkie wpisy z EventKind. W przypadku pojawienia się wartości, która nie jest częścią typu unii, zgłaszamy błąd. To jest świetne, ale działa tylko wtedy, gdy zajmiemy się wszystkimi możliwymi przypadkami. A jeśli nie wyczerpaliśmy jeszcze całej naszej listy? Na razie usuńmy „hackathon”:

function getEventTeaser(event: TechEvent) {

switch(event.kind) {

case 'conference':

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

`priced at ${event.price} USD`

case 'meetup':

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

`hosted at ${event.location}`

case 'webinar':

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

`available online at ${event.url}`

```
default:
throw new Error('Not sure what to do with that!')
}
}
```

W domyślnej gałęzi event.kind teraz jest „hackathon”, ale nie mamy z tym do czynienia - po prostu wyrzucamy błąd. Jest to trochę słuszne, ponieważ nie jesteśmy pewni, co z tym zrobić, ale byłoby o wiele lepiej, gdyby TypeScript ostrzegł nas, że o czymś zapomnieliśmy. W końcu chcemy wyczerpać całą naszą listę. W tym celu chcemy się upewnić, że na końcu długiej instrukcji switch-case lub w gałęziach else, które nie powinny wystąpić, typ event zdecydowanie never. Stwórzmy funkcję narzędziową, która zgłasza błąd. Ale zamiast wysyłać tylko wiadomość, chcemy również wysłać winowajcę, który ostatecznie spowodował ten błąd. Wskazówka: typ tego winowajcy jest never.

```
function neverError(
message: string,
token: never // The culprit
) {
return new Error(
`${message}. ${token} should not exist`
)
}
```

Zastępujemy funkcję neverError faktycznym zgłaszaniem błędu w naszej instrukcji switch-case:

```
function getEventTeaser(event: TechEvent) {
switch(event.kind) {
case 'conference':
return `${event.title} (Conference), ` +
`priced at ${event.price} USD`
case 'meetup':
return `${event.title} (Meetup), ` +
`hosted at ${event.location}`
case 'webinar':
return '${event.title} (Webinar), ` +
'available online at ${event.url}`
default:
throw neverError(
```

```
'Not sure what to do with that',
event
)
}
}
```

I natychmiast włącza się funkcja sprawdzania typów w języku TypeScript. W tym momencie event może być hackathon. Po prostu sobie z tym nie radzimy. TypeScript daje nam czerwoną falę i mówi nam, że nie możemy przekazać jakiejś wartości do funkcji, która oczekuje never. Po ponownym dodaniu „hackathonu" do listy, TypeScript ponownie skompiluje się i wszystkie nasze wyczerpujące testy zostaną zakończone.

```
function getEventTeaser(event: TechEvent) {
switch(event.kind) {
case 'conference':
return `${event.title} (Conference), ` +
`priced at ${event.price} USD`
case 'meetup':
return `${event.title} (Meetup), ` +
`hosted at ${event.location}`
case 'webinar':
return `${event.title} (Webinar), ` +
`available online at ${event.url}`
case 'hackathon':
return `even that: ${event.title}`
default:
throw neverError(
'Not sure what to do with that',
event // No complaints
)
}
}
```

Dzięki funkcji never otrzymujemy zabezpieczenie, które może być użyte w sytuacjach, które mogą się zdarzyć, ale nigdy nie powinny mieć miejsca. Zwłaszcza gdy mamy do czynienia z zestawami wartości, które stają się szersze i węższe w miarę kodowania naszych aplikacji. never jest typem dolnym wszystkich innych typów i będzie przydatnym narzędziem w następnych Lekcjach.