Lekcja 10: Kontrola przepływu

any jest przydatny, jeśli nie wiesz, jakich typów się spodziewać. Następująca funkcja wybiera adres dostawy, który został przesłany jako parametr (typu string) lub jeden z tablicy ciągów deliveryAddress.

function selectDeliveryAddress (addressOrIndex: any) {

if (typeof addressOrIndex === 'number') {

return deliveryAddresses [addressOrIndex]

}

adres zwrotnyOrIndex

}

Dzieje się tutaj kilka rzeczy.

Wpisz zwężenie

Z if (typeof addressOrIndex === 'number'), robimy coś, co łączy świat typów z naszym kodem JavaScript. Ponieważ JavaScript ma liczby, możemy wykonać kilka testów typu w czasie wykonywania, aby upewnić się, że określony typ jest podany. Nie ma to nic wspólnego z TypeScript, ale z JavaScriptem.

Jednak TypeScript może to wykorzystać. Od tego momentu TypeScript wie, że addressOrIndex jest typu number. Od tego momentu mamy dostęp do wszystkich cech number. Możemy sformatować liczbę do reprezentacji stałoprzecinkowej:

if (typeof addressOrIndex === 'number') {

// OK, ponieważ addressOrIndex to liczba

console.log (addressOrIndex.toFixed (2))

}

Wszystkie operacje numeryczne możemy wykonać na adresOrIndex:

if (typeof addressOrIndex === 'number') {

// OK, ponieważ addressOrIndex to liczba

console.log (addressOrIndex * 2 + 3)

}

Lub, w naszym przypadku, możemy użyć go jako indeksu liczbowego dla naszej tablicy. Powinniśmy jednak sprawdzić, czy mieści się w zakresie tablicy:

// Porównanie, aby sprawdzić, czy addressOrIndex to

// mniejsze niż liczba

// z pozycji w DeliveryAdress jest również tylko

// możliwe, ponieważ wiemy

// adresOrIndex to liczba

```
if (typeof addressOrIndex === 'number' &&

addressOrIndex <deliveryAddresses.length) {

return deliveryAddresses [addressOrIndex]

}
```

W tych kilku wierszach kodu widzimy kilka pojęć TypeScript:

1. Wpisz ochronę. Ochrona typów sprawdza typy w czasie wykonywania, podobnie jak operator typeof upewnia się, że w tym momencie mamy do czynienia z liczbą.

2. Analiza przepływu kontrolnego. Zabezpieczenia typu służą do wyzwalania analizy przepływu sterowania w języku TypeScript. TypeScript może analizować przepływ programu, aby zapewnić odpowiednie typy do następnych kroków.

3. Zawężenie. Od wszechogarniającego dowolnego typu zawężamy do typu number.

Wszystkie trzy koncepcje są ze sobą powiązane i mają kluczowe znaczenie dla wszystkiego, co zamierzasz zrobić z TypeScript od tego momentu.

Nadal any

Jednak równie ważne, jak rzeczy, które widzimy, są rzeczy, których nie robimy. Po zawężeniu typu addressOrIndex do number z naszą ochroną typów, wyjaśniliśmy, że w tym momencie oczekujemy, że będzie to liczba. Ale co dzieje się poza instrukcją if? Jaki jest typ adressOrIndex? Co jest bez number? To any! any to wszechogarniający typ topowy. To może być wszystko i wszystko jest dozwolone. Więc nawet po upewnieniu się, że mamy do czynienia z number w innym punkcie, reszta to wciąż całkiem sporo! A any w tej pozycji jest bardzo delikatny. Oczekujemy, że addressOrIndex będzie to string lub number , ale any pozwala nam przekazać cokolwiek i zwrócić wszystko, nawet jeśli jawnie wpiszemy wartość zwracaną:

```
function selectDeliveryAddress (addressOrIndex: any):

string{

if (typeof addressOrIndex === 'number' &&

addressOrIndex <deliveryAddresses.length) {

return deliveryAddresses [addressOrIndex]

}

// Całkowicie OK z dowolnym plikiem

return adressOrIndex

}

// O nie! To jest całkowicie OK w TypeScript, ale

// myFavouriteAddress jest teraz ciągiem znaków, mimo że po prostu

// powrót prawda? To wybuchnie w czasie wykonywania!

const myFavouriteAddress = selectDeliveryAddress (true)
```

Elastyczność any wiąże się z kruchością i ogromnym potencjałem niedopasowania typów. Dlatego powinniśmy za wszelką cenę unikać any.

Podtypy i supertypy

Wspominamy o podtypach i supertypach. Wszystkie typy w TypeScript zajmują swoje miejsce w hierarchii. Na przykład any jest supertypem wszystkich typów, a string jest podtypem any. Każda wartość string może być przypisana do jego nadtypu any, ale nie każda wartość any może być przypisana do jego podtypu string. To samo dotyczy klas i obiektów. HTML Element jest supertypem wszystkich elementów HTML w DOM. HTMLAnchorElement jest podtypem HTMLElement. Każdy element HTMLAnchorElement można przypisać do typu HTML Elementu , ale nie każdą wartość HTML Element można przypisać do elementu HTMLAnchorElement. Przy zawężaniu typów schodzimy w dół hierarchii typów od supertypów do podtypów. Później zobaczymy pełny potencjał podtypów Typescript.

Wpisz unknown

Na szczęście TypeScript ma partnera dla any : unknown. unknown jest również kompatybilne z każdym typem w TypeScript, więc jest to również najpopularniejszy typ.

Ale to też jest bardzo hamujące. Tam, gdzie wolno nam robić wszystko z any, nie wolno nam robić niczego z unknown.

unknown powinno sprawić, że będziesz ostrożny: musimy zapewnić odpowiedni przepływ sterowania, aby zapewnić bezpieczeństwo typu. Zobaczmy, co się stanie, gdy zmienimy any na unknown:

```
function selectDeliveryAddress (addressOrIndex:

unknown): string {

if (typeof addressOrIndex === 'number' &&

addressOrIndex <deliveryAddresses.length) {

return deliveryAddresses [addressOrIndex]

}

return adressOrIndex

}
```

Bum! Dokładnie tego chcemy: „Nie można przypisać numeru typu do ciągu znaków”. Musimy sprawdzić typ i wyzwolić analizę przepływu sterowania; w przeciwnym razie TypeScript zgłosi błąd!

```
function selectDeliveryAddresses (addressOrIndex: unknown): string {

if (typeof addressOrIndex === 'number' &&

addressOrIndex <deliveryAddresses.length) {

return deliveryAddresses [addressOrIndex]

} else if (typeof addressOrIndex === 'string') {

adres zwrotnyOrIndex

}
```

```
return ''

}
```

Przepływ sterowania jest zakończony. Jeśli otrzymamy wartość jednego z dwóch możliwych typów, number lub string, wiemy, co zrobić: albo zwrócić wpis z listy adresów dostawy, albo zwrócić właśnie wprowadzony adres dostawy. Jeśli przekażemy cokolwiek innego, zwracamy pusty łańcuch!