Wybrane elementy praktyki projektowania oprogramowania
Zestaw 6

TypeScript

2024-11-12

Liczba punktów do zdobycia: **10/55**
Zestaw ważny do: 2024-11-26

1. (**1p**) Użyć kompilatora TypeScript z linii poleceń oraz w trybie **watch** do kompilowania przykładowego kodu. Pokazać że wytworzony w ten sposób kod można debugować z poziomu Visual Studio Code.

   Użyć **ts-node** lub innego transpilera do uruchamiania kodu TypeScript z niejawną kompilacją. Pokazać że w tym przypadku kod również można debugować z poziomu Visual Studio Code.

   Lista dostępnych transpilerów:

   `https://github.com/privatenumber/ts-runtime-comparison`

2. (**1p**) Przystosować napisaną przy okazji jednego z poprzednich zestawów rekurencyjną implementację funkcji `fib(n)` i funkcję memoizującą `memoize` do TypeScript.

   Uwaga! Zadanie ma wiele rozwiązań, w zależności od tego jakie sygnatury typów przypisze się funkcjom **fib** i **memoize** kontrola typów będzie silniejsza lub słabsza. Znaleźć jakiś kompromis między użytecznością, a skutecznością.

3. (**1p**) Przedstawić po raz kolejny własne implementacje metod `map`, `forEach` i `filter` dla tablic, tym razem dodać poprawne sygnatury. Najwygodniej będzie użyć mechanizmu typów generycznych, na przykład:

```
function filter<T>( a: T[], f: (t: T) => boolean ): T[] {
   // ?
}
```

4. (**1p**) (za `https://typescript-exercises.github.io/`) Dane są typy

```
type User = {
    name: string;
    age: number;
    occupation: string;
}

type Admin = {
    name: string;
    age: number;
    role: string;
}

export type Person = User | Admin;
```

```
export const persons: Person[] = [
    {
        name: 'Jan Kowalski',
        age: 17,
        occupation: 'Student'
    },
    {
        name: 'Tomasz Malinowski',
        age: 20,
        role: 'Administrator'
    }
];
```

oraz funkcja

```
function logPerson(person: Person) {
    let additionalInformation: string;
    if (person.role) {
        additionalInformation = person.role;
    } else {
        additionalInformation = person.occupation;
    }
    console.log(` - ${person.name}, ${person.age}, ${additionalInformation}`);
}
```

która w takiej formie jak wyżej nie skompiluje się - na ścieżkach warunkowych dla **if** kompilator nie jest w stanie zawęzić typu parametru do jednego z dwóch podanych. Jak skorygować powyższy kod tak żeby poprawnie się skompilował i zadziałał?

Uwaga! Jedno z możliwych rozwiązań to zdefiniowanie strażnika typowego (patrz: następne zadanie), więc proszę w tym rozwiązaniu zaproponować inny rodzaj warunku niż strażnik typowy.

5. (**1p**) Do poprzedniego zadania dodano pomocnicze funkcje pełniące rolę strażników typowych (choć do rozwiązania poprzedniego zadania to nie było konieczne!). W zamyśle programisty, funkcje **isAdmin** i **isUser** miały zawęzić typ parametru funkcji **logPerson**, jednak z powodu pewnego błędu w definicji tych funkcji, zamysł nie powiódł się.

   Należy

   - skorygować definicje typów **User** i **Admin** tak żeby zawierały dodatkowe pole, **type**, zawierające informację o rodzaju obiektu (omówiony na wykładzie wzorzec **unia z wariantami**!)
   - dodać dodatkowe pole, z odpowiednią wartością, do przykładowych elementów w tablicy przykładowych danych
   - skorygować definicje pomocniczych funkcji **isAdmin** i **isPerson** tak żeby faktycznie pełniły rolę strażników typowych w funkcji **logPerson**.

```
export function isAdmin(person: Person) {
    return person.type === 'admin';
}

export function isUser(person: Person) {
    return person.type === 'user';
}

export function logPerson(person: Person) {
    let additionalInformation: string = '';
    if (isAdmin(person)) {
        additionalInformation = person.role;
```

```
    }
    if (isUser(person)) {
        additionalInformation = person.occupation;
    }
    console.log(` - ${person.name}, ${person.age}, ${additionalInformation}`);
}
```

6. (**1p**) Pokazać jak używać następujących mechanizmów systemu typów:

   - typy wyższego rzędu - `Extract`, `Exclude`
   - typy wyższego rzędu - `Record`, `Required`, `Readonly`, `Partial`
   - typy wyższego rzędu - `Pick`, `Omit`
   - typy dostępu indeksowanego (Indexed Acess Types)

   Jakie jest zastosowanie dla tych mechanizmów? Jaką wartość wnoszą do systemu typów?

7. (**4p**) Rozwiązać wszystkie zadania z sekcji **Warm up** i **Easy** ze zbioru Type Challenge. Opcjonalnie (dla własnej satysfakcji, bez dodatkowych punktów), rozwiązać wybrane samodzielnie zadania z pozostałych sekcji (zadania z sekcji Hard i Extreme są naprawdę trudne!).

   `https://github.com/type-challenges/type-challenges`

   Uwaga! Każdy z przykładów ma łatwo dostępne rozwiązania, w zadaniu nie chodzi więc o to żeby się wykazać umiejętnością wyszukania gotowego rozwiązania, ale żeby coś z tego wynieść, czyli spróbować mimo wszystko zmierzyć się z tym samodzielnie, a dopiero w ostateczności popatrzeć na prawidłowe rozwiązanie i spróbować je zrozumieć.

Wiktor Zychla