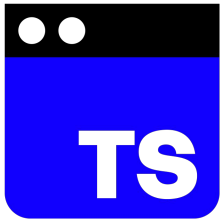
Code Academy



OOP Inkapsuliacija

TypeScript

8 paskaita

Paskaitos eiga

Kas yra inkapsuliacija?

Inkapsuliacijos pasiekiamumo lygiai

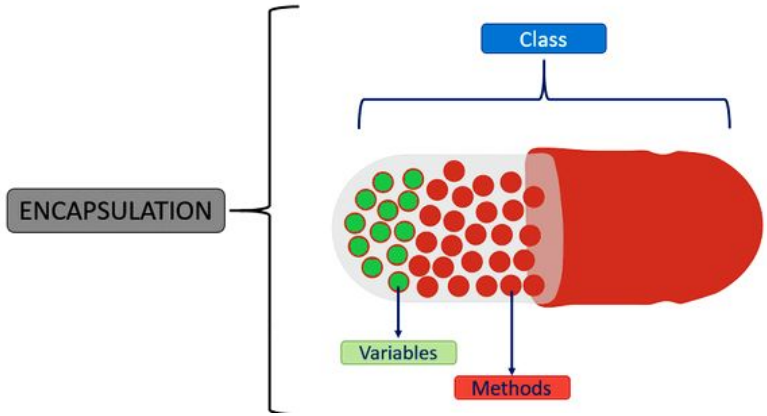
Inkapsuliacijos įgyvendinimas

Kas yra inkapsuliacija?

Kas yra inkapsuliacija?

Inkapsuliacija - tai objekto savybių ir metodų tiesioginio pasiekiamumo ribojimas.

Inkapsuliuotų savybių ar metodų pasiekiamumas įgalinamas naudojant papildomus metodus, kurie atlieka patikrinimus, konvertavimus ir formatavimus.



Inkapsuliacijos tikslai

1. Duomenų korektiškumo užtikrinimas, pvz: Turite klasę Person, kuri turi savybę “name” ir “age”.
    - Ar gali “name” būti: “-155”, arba “******+++”?
    - Ar gali “age” savybė būti -19, 15.4 arba 15666?

Inkapsuliacijos tikslai

2. Objekto savybių tarpusavio suderinamumas, pvz:
   Turite klasę Triangle, kurios objektai sukuriami pagal koordinates.
   Aritmetiniams skaičiavimams reikės naudoti kampus ir kraštinių ilgius.
   - Ar galite pakeisti tik vieną trikampio kampą?
   - Ar galite pakeisti tik vieną trikampio kraštinę?

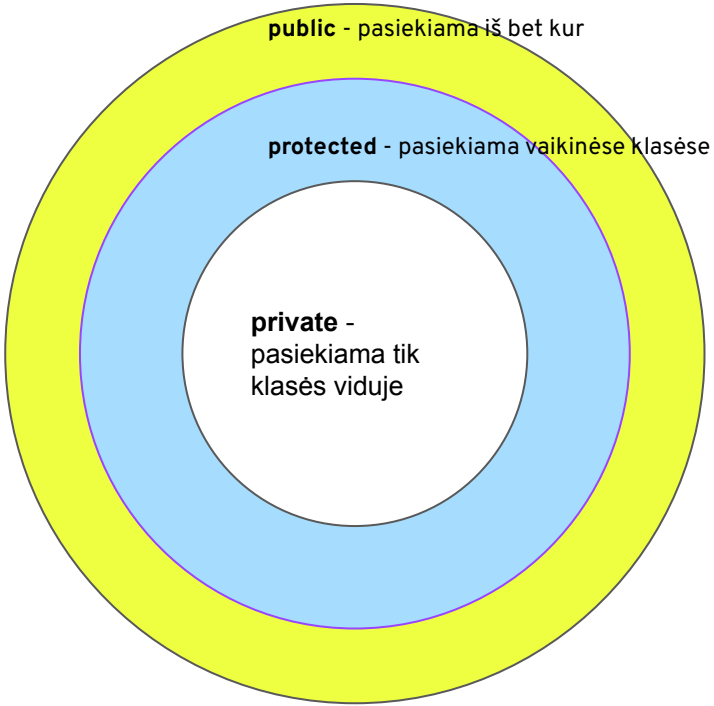
Būtent inkapsuliacija padeda apsisaugoti nuo nekorektiško savybių naudojimo. Praktikoje visos objekto savybės kuriamos inkapsuliuotos, net jei to nereikia šiuo metu.

Inkapsuliacijos pasiekiamumo lygiai

Inkapsuliacijos lygiai

TypeScript palaiko tris pasiekiamumo lygius:

- private - savybės ir metodai pasiemiami tik toje klasėje, kurioje jie yra aprašyti.
- protected - tas pats kaip private, tik papildomai savybes ar metodus gali pasiekti vaikinės-paveldinčios klasės viduje aprašyti metodai.
- public - savybės ir metodai yra pasiekiami bet kurioje aplikacijos vietoje, naudojant objektą.



Inkapsuliacijos įgyvendinimas

Kuriant viešas “getter” ir “setter” funkcijas

```
class Person {
  private static readonly ONLY_LETTERS_REGEX = /^[a-zA-ZąčęėįšųūžĄČĘĖĮŠŲŪŽ ]+$/;
  private name: string;

  constructor(name: string) {
    this.setName(name);
  }

  public setName(name: string) {
    if (name === '') throw new Error('Negali būti tuščias');
    if (name.length < 2) throw new Error('Nors 2 raidės');
    if (!Person.ONLY_LETTERS_REGEX.test(name)) throw new Error('Tik raidės ir tarpai');

    this.name = name[0].toUpperCase() + name.slice(1).toLowerCase();
  }

  public getName() {
    return this.name;
  }
```

Kuriant viešas “getter” ir “setter” funkcijas

```
try { const person = new Person(''); }
catch (error) { console.log(error); }  [Error: Negali būti tuščias]

try { const person = new Person('A'); }
catch (error) { console.log(error); }  [Error: Nors 2 raidės]

try { const person = new Person('54546'); }
catch (error) { console.log(error); }  [Error: Tik raidės ir tarpai]

try {
  const person = new Person('peneDIKtas');
  console.log(person);   Person { name: 'Penediktas' }
}
catch (error) { console.log(error); }
```

Kuriant viešas “getter” ir “setter” funkcijas

```
const person = new Person('Furija');

console.log(person);  Person { name: 'Furija' }
```

```
(property) Person.name: string
```

```
Furija
```

```
Property 'name' is private and only accessible within class
'Person'. ts(2341)
```

View Problem No quick fixes available

```
console.log(person.name);  Furija
```

```
getName                    (method) Person.getName(): string
setName
```

Kuriant viešas “getter” ir “setter” funkcijas

```
const person = new Person('Ryčka');

console.log(person);  Person { name: 'Ryčka' }
```

```
(property) Person.name: string

Property 'name' is private and only accessible within class
'Person'. ts(2341)

View Problem    No quick fixes available
```

```
person.name = 'Robertas';
```

Kuriant viešas “getter” ir “setter” funkcijas

```
const person = new Person('Alugirtas');

console.log(person);  Person { name: 'Alugirtas' }
console.log(person.getName());  Alugirtas

person.setName('Činkotkis');

console.log(person);  Person { name: 'Činkotkis' }
console.log(person.getName());  Činkotkis
```

```
try { person.setName('BaValas16') }
catch (error) { console.log(error); }  [Error: Tik raidės ir tarpai]
```

Kuriant viešas “getter” ir “setter” funkcijas

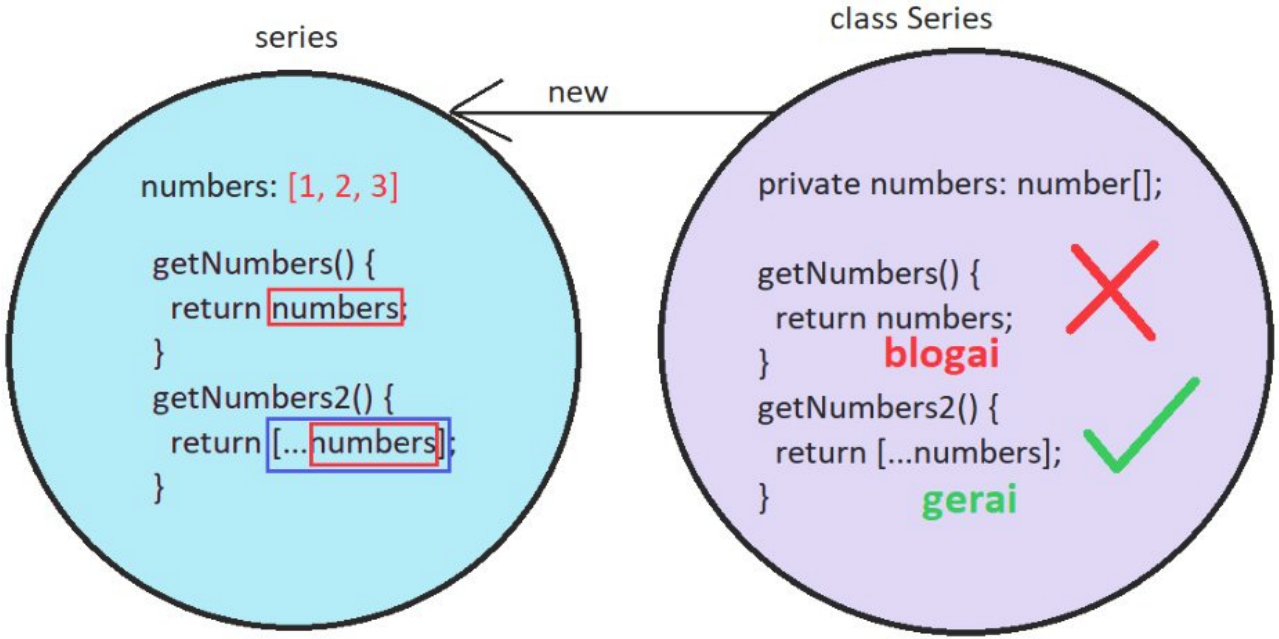
Iš pavyzdžių matome, kad negalime tiesiogiai nei gauti, nei nustatyti “name” savybės. Tai galima tik naudojant papildomas viešas funkcijas - “getter” ir “setter”.

Inkapsuliacijos dėka išvengiame “Person” klasės objektų kūrimo su neteisingais vardais, bei neteisingų vardų priskyrimo jau esantiems “Person” klasės objektams.

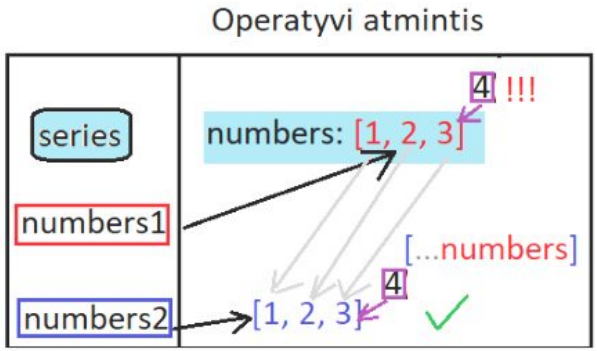
Kuriant viešas “getter” ir “setter” funkcijas

Dažniausiai papildomą funkcionalumą turi tik “setter” funkcija, “getter” funkcija kuriama tik tam, kad gauti duomenis. Visgi kuomet “getter” funkcija yra skirta nuorodos tipo reikšmėms gauti, būtų teisinga grąžinti tos (nuorodos tipo) reikšmės kopiją. Taip nepadarius, naudojant nuorodą, būtų galima netiesiogiai pakeisti vidinius klasės objekto narius.

Problemos iliustracija kitoje skaidrėje →




```
const series = new Series(1, 2, 3);
const numbers1 = series.getNumbers();
const numbers2 = series.getNumbers2();
numbers1.push(4);
numbers2.push(4)
```



Naudojant ES6 “get” ir “set”

Naudotis “getter” ir “setter” funkcijomis nepatogu. Norint gauti ar nustatyti reikšmes reikia kviesti funkcijas. Todėl su Ecma - 6 standartu buvo sukurtas būdas paprasčiau rašyti “setter” ir “getter” funkcijas.

Naudojant ES6 “get” ir “set”

Pritaikius šią sintaksę, galime aprašyti apsaugas specialiose “get” ir “set” funkcijose, tačiau naudoti galime kaip savybes.

Tokiu būdu, galime turėti inkapsuliuotas reikšmes ir nesirūpinti funkcijų kvietimu. Funkcijų kvietimo sintaksė yra nuo mūsų paslepiama, ir atrodo tarsi naudotumėte savybę tiesiogiai.

Naudojant ES6 “get” ir “set”

```
class Person {
    private static readonly ONLY_LETTERS_REGEX = /^[a-zA-ZąčęėįšųūžĄČĘĖĮŠŲŪŽ ]+$/;
    private _name: string;

    constructor(name: string) {
      this.name = name;
    }

    public set name(val: string) {
      if (val === '') throw new Error('Negali būti tuščias');
      if (val.length < 2) throw new Error('Nors 2 raidės');
      if (!Person.ONLY_LETTERS_REGEX.test(val)) throw new Error('Tik raidės ir tarpai');

      this._name = val[0].toUpperCase() + val.slice(1).toLowerCase();
    }

    public get name() {
      return this._name;
    }
```

Naudojant ES6 “get” ir “set”

```
try { const person = new Person(''); }
  catch (error) { console.log(error); }  [Error: Negali būti tuščias]

  try { const person = new Person('A'); }
  catch (error) { console.log(error); }  [Error: Nors 2 raidės]

  try { const person = new Person('54546'); }
  catch (error) { console.log(error); }  [Error: Tik raidės ir tarpai]

  try {
    const person = new Person('peneDIKtas');
    console.log(person);   Person { name: 'Penediktas' }
  }
  catch (error) { console.log(error); }
```

Naudojant ES6 “get” ir “set”

```
const person = new Person('Ryčka');

console.log(person);  Person { _name: 'Ryčka' }
console.log(person.name);  Ryčka

person.name = 'Robertas';

console.log(person);  Person { _name: 'Robertas' }
console.log(person.name);  Robertas

person.  Person { _name: 'Robertas' }
         name                    (property) Person.name: string
```

```
try { person.name = 'BaValas16' }
catch (error) { console.log(error); }  [Error: Tik raidės ir tarpai]
```

Klausimai?

Paskaitos darbas

Paskaitos darbas

Paskaitoje atliksime užduotis, tokia eiga:

1. Sprendžiame užduotis savarankiškai
2. Po savarankiško sprendimo laiko (10-30 min.) dėstytojas išsprendžia 1 užduotį argumentuodamas sprendimą
3. Studentai užduoda klausimus apie sprendimą
4. Sprendimų palyginimas
5. Atliekama sekanti užduotis

Jeigu išsprendėte užduotį anksčiau nei kiti, spręskite sekančias užduotis.

Užduoties aptarimo metu, nesidrovėkite klausti kuo daugiau klausimų. Nebūtinai jūsų sprendimas yra prastesnis. Galbūt net geresnis?

Iki kito karto!