



# TypeScript Generic

Eko Kurniawan Khannedy

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 13+ years experiences
- [www.programmerzamannow.com](http://www.programmerzamannow.com)
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)





# Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- LinkedIn : <https://www.linkedin.com/company/programmer-zaman-now/>
- Facebook : [fb.com/ProgrammerZamanNow](https://fb.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : [t.me/ProgrammerZamanNow](https://t.me/ProgrammerZamanNow)
- Tiktok : <https://tiktok.com/@programmerzamannow>
- Email : echo.khannedy@gmail.com



# Sebelum Belajar

- Kelas JavaScript dari Programmer Zaman Now
- Kelas NodeJS dari Programmer Zaman Now
- TypeScript OOP

---

# TypeScript Generic



# TypeScript Generic

- Generic adalah fitur dimana kita bisa membuat kode yang sama, dan bisa digunakan berulang kali dengan menggunakan tipe data yang berbeda
- Sebelumnya, saat kita ingin menggunakan tipe data yang bisa berbeda untuk variabel atau parameter, kita menggunakan tipe data any
- Dengan menggunakan Generic, kita bisa melakukan perubahan tipe data ketika digunakan, sehingga lebih aman karena tidak perlu menggunakan tipe data seperti any



# Membuat Project

- Buat folder belajar-typescript-generic
- npm init
- Buka package.json, dan tambah type module



## Menambah Library Jest untuk Unit Test

- `npm install --save-dev jest @types/jest`
- <https://www.npmjs.com/package/jest>





## Menambah Library Babel

- `npm install --save-dev babel-jest @babel/preset-env`
- <https://babeljs.io/setup#installation>



# Menambah TypeScript

- `npm install --save-dev typescript`
- <https://www.npmjs.com/package/typescript>



# Setup TypeScript Project

- `npx tsc --init`
- Semua konfigurasi akan dibuat di file `tsconfig.json`
- Ubah “module” dari “commonjs” menjadi “ES6”



## Setup TypeScript untuk Jest

- `npm install --save-dev @babel/preset-typescript`
- `npm install --save-dev @jest/globals`
- <https://jestjs.io/docs/getting-started#using-typescript>

---

Tanpa Generic




# Tanpa Generic

- Tanpa generic, saat kita ingin membuat Class yang berisi tipe data yang bisa berbeda-beda
- Maka kita harus menggunakan tipe data any



## Kode : Data Class

```
export class Data {  
  value: any;  
  
  constructor(value: any) {  
    this.value = value;  
  }  
}
```





## Kode : Test Data Class

```
import {Data} from "../src/simple";

describe('Tanpa Generic', () => {
  it('should can accept number', () => {
    const data = new Data(123);
    data.value = "eko";
    data.value = true;

    console.info(data);
  });
});
```





---

# Generic Class



# Generic Class

- Generic bisa ditambahkan ketika kita membuat Class, caranya bisa menggunakan tanda <> (diamond) setelah nama class, lalu tentukan tipe data generic nya
- Tipe data generic tersebut bisa digunakan pada class, dan bisa diubah tipenya ketika kita membuat object dari Generic Class tersebut
- Saat kita membuat object dari Generic Class, kita wajib menentukan tipe data yang ingin kita gunakan untuk mengganti tipe data generic nya



## Kode : Generic Class

```
export class GenericData<T> {  
    value: T;  
  
    constructor(value: T) {  
        this.value = value;  
    }  
}
```

## Kode : Membuat Object

```
import {GenericData} from "../src/generic";  
  
describe('Generic Class', () => {  
  it('should can only accept one type', () => {  
    const data = new GenericData<number>(123);  
    data.value = "eko";  
    data.value = true;  
  });  
});
```

## Kode : Menggunakan Data di Generic

```
describe('Generic Class', () => {  
  it('should can only accept one type', () => {  
    const data = new GenericData<number>(123);  
    expect(data.value).toBe(123);  
  
    const dataString = new GenericData<string>("Eko Khannedy");  
    const firstName = dataString.value.substring(0, 3)  
    expect(firstName).toBe("Eko");  
  });  
});
```

---

# Generic Function



# Generic Function

- Saat kita membuat tipe data generic di Class, tipe data tersebut bisa digunakan diseluruh bagian Class
- Namun, kadang kita tidak membuat class, kita hanya membuat function misalnya
- Generic juga bisa digunakan pada function, kita bisa menggunakan cara yang sama dengan menempatkan tanda < > setelah nama function



## Kode : Generic Function

```
function create<T>(value: T): T {  
    return value;  
}  
  
it('should support', () => {  
    const result = create<string>('Hello');  
    expect(result).toBe('Hello');  
  
    const result2 = create<number>(42);  
    expect(result2).toBe(42);  
});
```



---

# Multiple Generic Type



## Multiple Generic Type

- Tipe data generic bisa kita tambahkan lebih dari satu, baik itu di Class ataupun di Function
- Kita bisa tambahkan pemisah , (koma) di dalam <> jika ingin menambahkan tipe data generic lebih dari satu



## Kode : Multiple Generic Type

```
class Entry<K, V> {  
    constructor(public key: K, public value: V) {  
    }  
}  
  
class Triple<K, V, T> {  
    constructor(public first: K, public second: V, public third: T) {  
    }  
}
```



## Kode : Menggunakan Multiple Generic Type

```
it('should support', () => {  
  const entry = new Entry<number, string>(1, 'Hello');  
  expect(entry.key).toBe(1);  
  expect(entry.value).toBe('Hello');  
  
  const triple = new Triple<number, string, boolean>(1, 'Hello', true);  
  expect(triple.first).toBe(1);  
  expect(triple.second).toBe('Hello');  
  expect(triple.third).toBe(true);  
});
```

---

# Optional Generic Type



## Optional Generic Type

- Saat kita menggunakan generic type di Class, lalu kita menggunakan generic type tersebut di constructor, kita tidak wajib menyebutkan tipe generic nya
- TypeScript bisa secara otomatis mendeteksi tipe yang kita gunakan pada parameter constructor
- Namun jika kita tidak menggunakan tipe tersebut pada constructor, maka typescript tidak bisa menggunakan secara otomatis



## Kode : Optional Generic Type

```
it('should optional type', () => {  
  const entry = new Entry(1, 'Hello');  
  expect(entry.key).toBe(1);  
  expect(entry.value.toUpperCase()).toBe('HELLO');  
});
```



## Kode : Generic Class Tanpa Constructor Parameter

```
class SimpleGeneric<T> {  
  
    private value?: T;  
  
    setValue(value: T) {  
        this.value = value;  
    }  
  
    getValue(): T | undefined {  
        return this.value;  
    }  
}
```





## Kode : Menggunakan Generic Class

```
it('should create simple generic', () => {  
  const simple = new SimpleGeneric();  
  simple.setValue("Eko");  
  expect(simple.getValue().toUpperCase()).toBe("EKO");  
});
```

---

# Generic Parameter Default



## Generic Parameter Default

- Saat kita menggunakan generic data type, kita bisa menentukan tipe data default jika tidak menyebutkan tipe data
- Kita bisa gunakan = tipe data di dalam operator <>



## Kode : Generic Parameter Default

```
class SimpleGeneric<T = string> {  
  
    private value?: T;  
  
    setValue(value: T) {  
        this.value = value;  
    }  
  
    getValue(): T | undefined {  
        return this.value;  
    }  
}
```



## Kode : Menggunakan Generic Parameter Default

```
it('should create simple generic', () => {  
  const simple = new SimpleGeneric();  
  simple.setValue("Eko");  
  expect(simple.getValue().toUpperCase()).toBe("EKO");  
});
```

---

# Generic Constraint



## Generic Constraint

- Secara default, saat menggunakan generic type, kita bisa bebas menggunakan tipe data apapun
- Namun, kadang kita ingin membatasi jenis tipe data yang diperbolehkan
- Kita bisa menggunakan perintah `extends T` pada operator `<>`, yang artinya tipe data yang boleh digunakan hanyalah tipe data `T` dan turunannya



## Kode : Employee Inheritance

```
interface Employee {  
    id: string;  
    name: string;  
}  
  
interface Manager extends Employee {  
    totalEmployee: number;  
}  
  
interface VP extends Manager {  
    totalManager: number;  
}
```





## Kode : Generic Constraint

```
class EmployeeData<T extends Employee> {  
    constructor(public employee: T) {  
    }  
}
```



## Kode : Menggunakan Generic Constraint

```
it('should support constraint', async () => {  
    const data1 = new EmployeeData<Employee>({  
        id: "100",  
        name: "Eko"  
    });  
    const data2 = new EmployeeData<Manager>({  
        id: "100",  
        name: "Eko",  
        totalEmployee: 10,  
    });  
  
    const data3 = new EmployeeData<string>("Eko");  
    const data4 = new EmployeeData<number>(1234);  
}).
```

---

# Generic Collection



# Generic Collection

- Sebelumnya kita pernah menggunakan tipe data Array, dimana Array sebenarnya adalah tipe data Generic, oleh karena itu kita bisa menggunakan `Array<tipe>` ketika membuat Array
- Selain Array, ada tipe data Collection (kumpulan data) yang berupa tipe data Generic, yaitu :
- `Set<T>`, yaitu tipe data collection yang berisi data unique dan tidak ada jaminan urutan data
- `Map<K, V>` yaitu tipe data collection yang berisi data key-value



## Array<T>

- Generic type Array<T> sebenarnya merupakan representasi dari tipe data array di JavaScript
- Oleh karena itu, cara penggunaannya sama seperti array di JavaScript
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)



## Kode : Array<T>

```
it('should support array', () => {  
  const array = new Array<string>();  
  array.push('Eko');  
  array.push('Kurniawan');  
  
  expect(array[0].toUpperCase()).toBe('EKO');  
  expect(array[1].toUpperCase()).toBe('KURNIAWAN');  
});
```



## Set<T>

- Generic type Set<T> sebenarnya merupakan representasi dari tipe data Set di JavaScript
- Oleh karena itu, cara penggunaannya sama seperti Set di JavaScript
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set)



## Kode : Set<T>

```
it('should support set', () => {  
  const set = new Set<string>();  
  set.add('Eko');  
  set.add('Kurniawan');  
  set.add('Eko');  
  
  expect(set.size).toBe(2);  
  expect(set.has('Eko')).toBe(true);  
  expect(set.has('Kurniawan')).toBe(true);  
});
```





## Map<K, V>

- Generic type Map<K, V> sebenarnya merupakan representasi dari tipe data Map di JavaScript
- Oleh karena itu, cara penggunaannya sama seperti Map di JavaScript
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map)



## Kode : Map<K, V>

```
it('should support map', () => {  
  const map = new Map<string, number>();  
  map.set('Eko', 1);  
  map.set('Kurniawan', 2);  
  
  expect(map.get('Eko')).toBe(1);  
  expect(map.get('Kurniawan')).toBe(2);  
});
```

---

# Generic Promise



## Generic Promise

- Saat kita menggunakan JavaScript Async, kita akan sering bertemu dengan Promise
- TypeScript menggunakan Generic sebagai representasi untuk tipe data `Promise<T>`
- Oleh karena itu, ketika membuat function yang mengembalikan Promise, kita bisa menentukan tipe data apa yang akan dikembalikan oleh Promise tersebut



## Kode : Promise Function

```
async function fetchData(value: string): Promise<string> {  
  return new Promise<string>((resolve, reject) => {  
    setTimeout(() => {  
      if (value === 'Eko') {  
        resolve('Hello ' + value);  
      } else {  
        reject('Not Found');  
      }  
    }, 1000);  
  });  
}
```



## Kode : Menggunakan Promise

```
it('should support promise', async () => {  
  const result = await fetchData('Eko');  
  expect(result.toUpperCase()).toBe('HELLO EKO');  
  
  try {  
    await fetchData('Budi');  
  } catch (e) {  
    expect(e).toBe('Not Found');  
  }  
});
```



Penutup