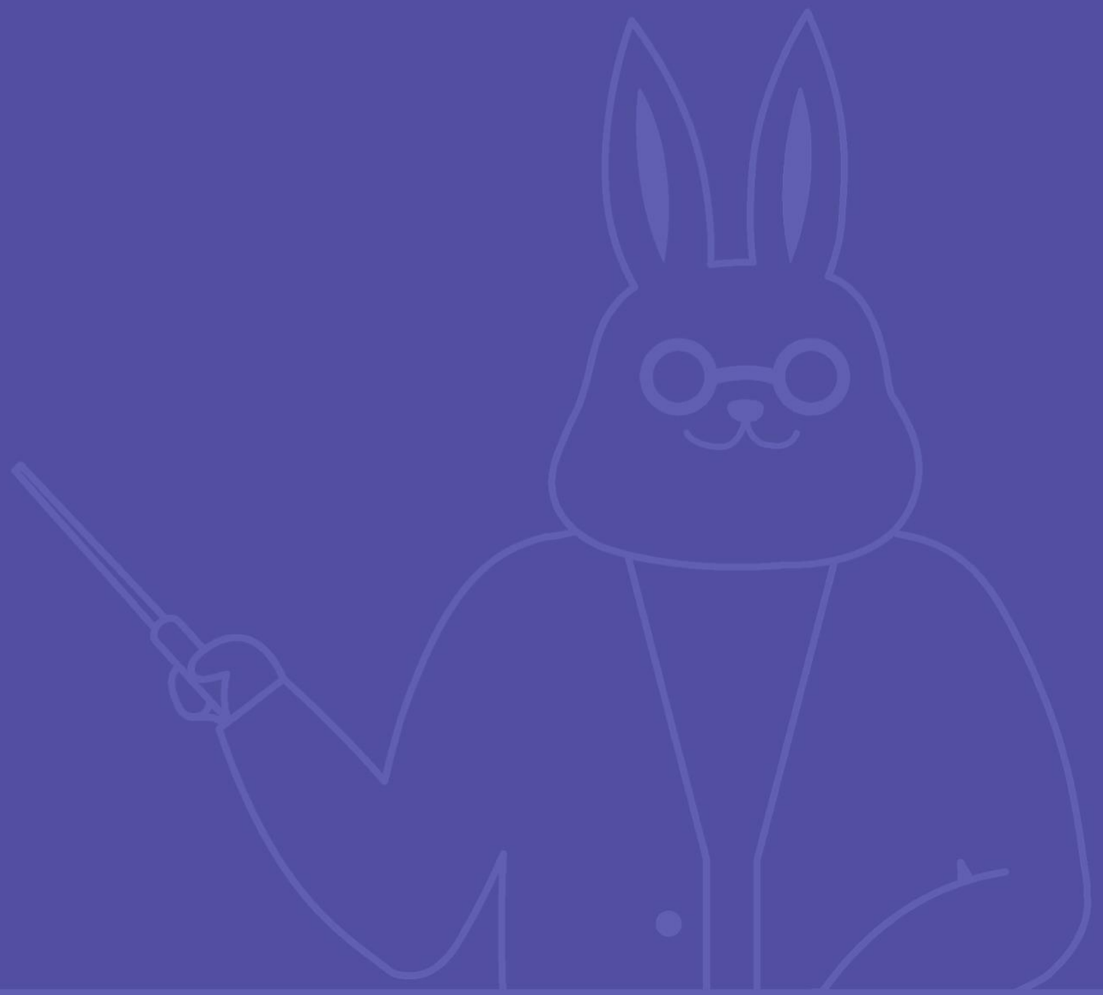
타입스크립트 I

04 Generic

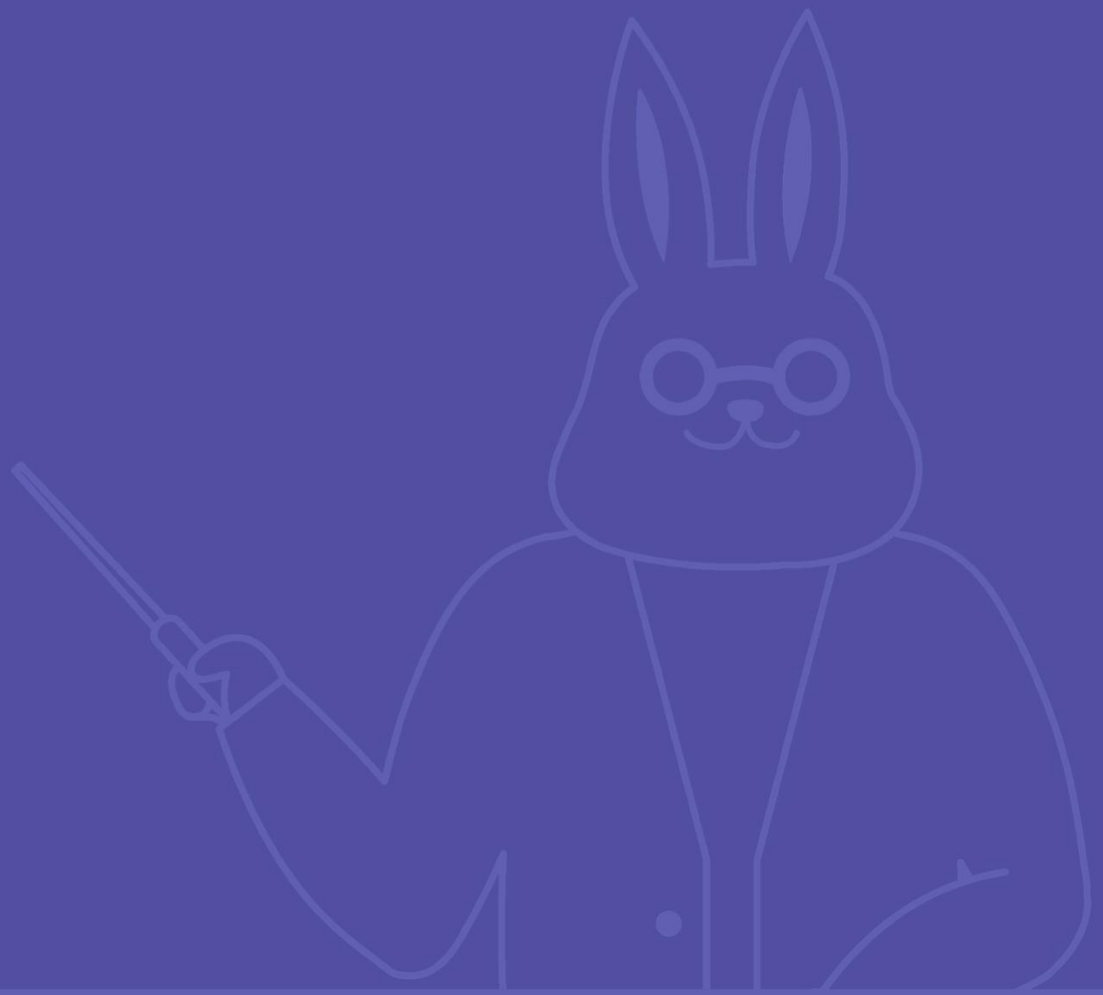


/* elice */

목차

01

Generic이란?



Generic

- 정적 type 언어는 클래스나 함수를 정의할 때 type을 선언해야 한다.

  ex) C 언어는 int type 변수를 선언하면 정수형 값만 할당할 수 있다.

- Generic은 코드를 작성할 때가 아니라 코드가 수행될 때 타입을 명시한다.
- 코드를 작성할 때 식별자를 써서 아직 정해지지 않은 타입을 표시한다.

  일반적으로 식별자는 T, U, V, ...를 사용한다.

  필드 이름의 첫 글자를 사용하기도 한다.

Generic을 사용하는 이유

- 재사용성이 높은 함수와 클래스를 생성할 수 있다.

  여러 타입에서 동작이 가능하다. (한 번의 선언으로 다양한 타입에 재사용할 수 있다.)

  코드의 가독성이 향상된다.

- 오류를 쉽게 포착할 수 있다.
  - any 타입을 사용하면 컴파일 시 타입을 체크하지 않는다.

    타입을 체크하지 않아 관련 메소드의 힌트를 사용할 수 없다.

    컴파일 시에 컴파일러가 오류를 찾지 못한다.

  - Generic도 any처럼 미리 타입을 지정하지는 않지만, 타입을 체크해 컴파일러가 오류를 찾을 수 있다.

02

Generic으로 함수와 클래스 만들기

function

코드

```
function sort<T>(items: T[]): T[] {
 return items.sort();
}

const nums: number[] = [1, 2, 3, 4];
const chars: string[] = ["a", "b", "c", "d"];

sort<number>(nums);
sort<string>(chars);
```

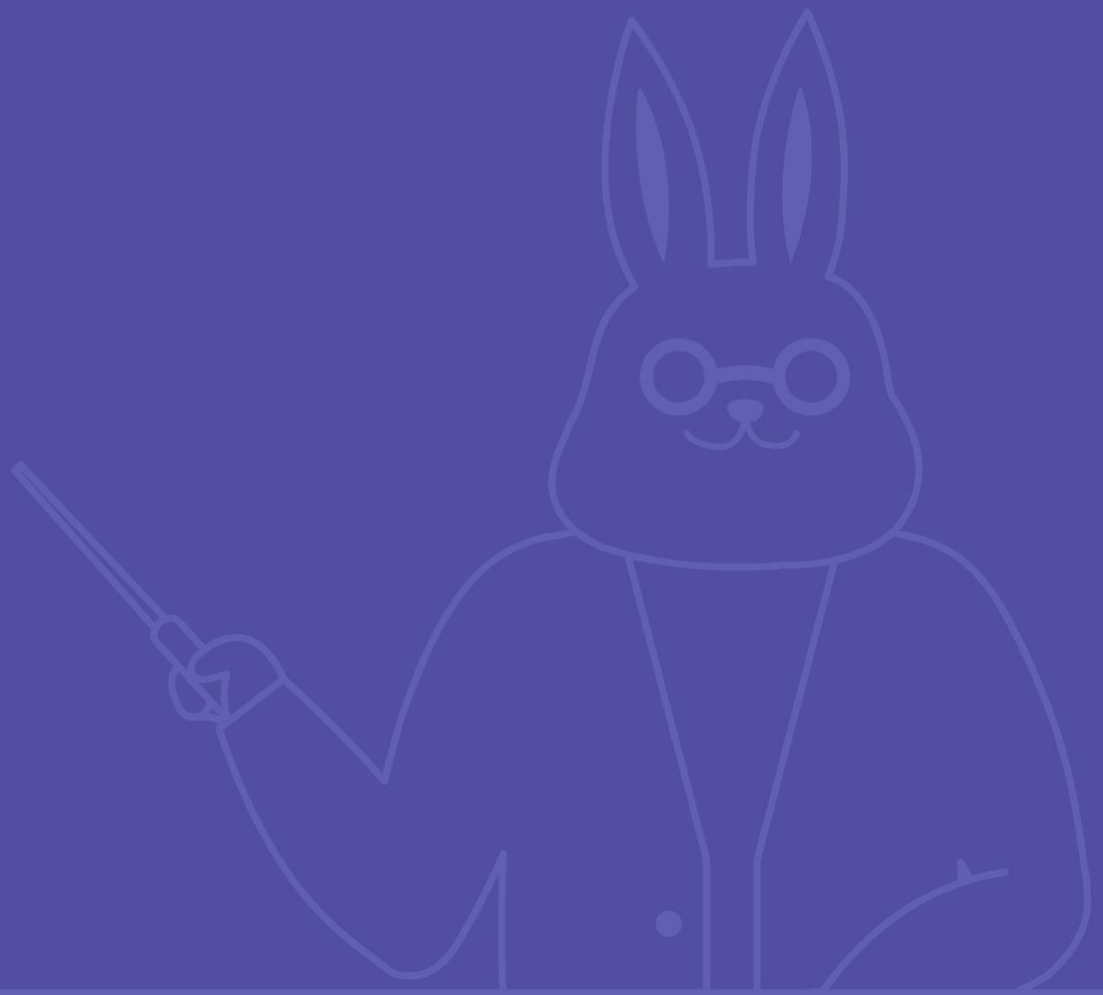
class

코드

```
class Queue<T> {
 protected data: Array<T> = [];
 push(item: T) {
  this.data.push(item);
 }
 pop(): T | undefined {
  return this.data.shift();
 }
}

const numberQueue = new Queue<number>();

numberQueue.push(0);
numberQueue.push("1"); // 의도하지 않은 실수를 사전 검출 가능
numberQueue.push(+"1"); // 실수를 사전 인지하고 수정할 수 있다
```

03

Union type



Union type

- Union type

  “|”를 사용해 두 개 이상의 타입을 선언하는 방식

- Union과 Generic 모두 여러 타입을 다룰 수 있다.

  Union은 선언한 공통된 메소드만 사용할 수 있다.

  리턴 값이 하나의 타입이 아닌 선언된 Union 타입으로 지정된다.

Union

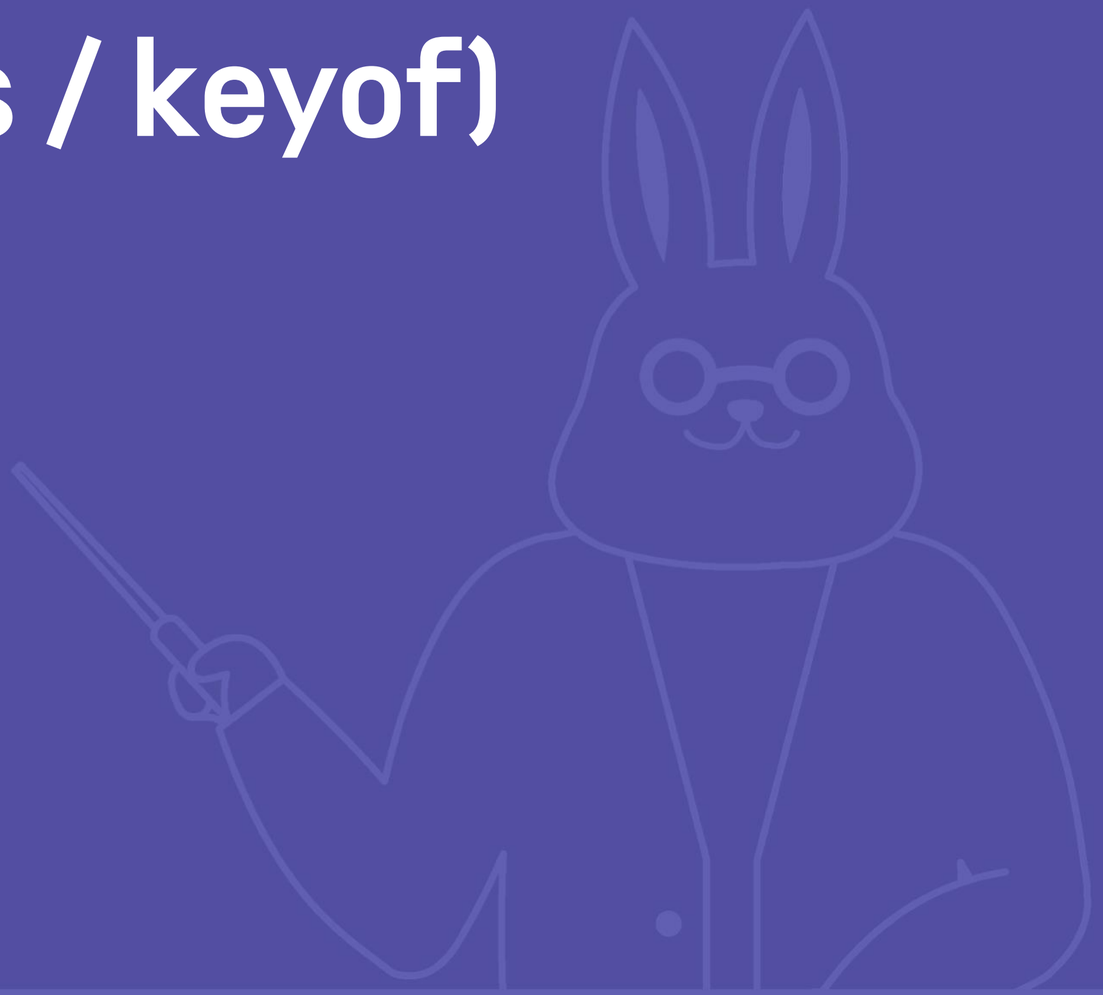
코드

```
// 1. Union type
const printMessage = (message: string | number) => {
 return message;
}

const message1 = printMessage(1234);
const message2 = printMessage("hello world !");

// string과 number type의 공통된 메소드만 사용 가능하다
// error: length does not exist on type string | number
message1.length;
```

```
// 2. generic
const printMessage2 = <T>(message: T) => {
 return message;
}

const message1 = printMessage2<string>("hello world !");
message1.length;
```

04

제약조건 (Constraints / keyof)



제약조건 (Constraints / keyof)

- 원하지 않는 속성에 접근하는 것을 막기 위해 Generic에 제약조건을 사용한다.
  1. Constraints : 특정 타입들로만 동작하는 Generic 함수를 만들 때 사용한다.
  2. Keyof : 두 객체를 비교할 때 사용한다.

Constraints

- Generic T에 제약 조건을 설정한다. (문자열 or 숫자)
- 제약 조건을 벗어나는 타입을 선언하면 에러가 발생한다.

코드

```
const printMessage = <T extends string | number>(message: T): T => {
 return message;
}



printMessage<string>("1");
printMessage<number>(1);
printMessage<boolean>(false) // Error: Type  'boolean' does not satisfy the
                                    constraint 'string | number'.
```

keyof

코드

```
const getProperty = <T extends object, U extends keyof T>(obj: T, key: U) => {
 return obj[key]
}


getProperty({ a: 1, b: 2, c: 3 }, "a");

// error: Argument of type '"z"' is not assignable to parameter of type '"a" | "b" | “    c"'
getProperty({ a: 1, b: 2, c: 3 }, "z");
```

- 두번째 함수에서 오류가 발생한다.
- Generic T는 키 값이 a, b, c만 존재하는 object이다.
- U의 값인 ‘z’ 가 Generic T의 키 값 중 존재하지 않기 때문에 오류가 발생한다.

05

디자인 패턴 (Factory Pattern with Generics)

Factory Pattern with Generics

- 객체를 생성하는 인터페이스만 미리 정의하고,
  인스턴스를 만들 클래스의 결정은 서브 클래스가 내리는 패턴

- 여러 개의 서브 클래스를 가진 슈퍼 클래스가 있을 때,
  입력에 따라 하나의 서브 클래스의 인스턴스를 반환한다.

Factory Pattern

코드

```
interface Car {
 drive(): void
 park(): void
}

class Bus implements Car {
 drive(): void {}
 park(): void {}
}
class Taxi implements Car {
 drive(): void {}
 park(): void {}
}
```

```
class CarFactory {
 static getInstance(type: string): Car {
// car의 type이 추가 될 때마다, case 문을 추가 해야
하는 단점
  switch (type) {
   case "bus":
    return new Bus();
   default:
    return new Taxi();
  }
 }
}

const bus = CarFactory.getInstance("bus");
const taxi = CarFactory.getInstance("taxi");
```

Factory Pattern with Generics

코드

```
interface Car {
 drive(): void
 park(): void
}

class Bus implements Car {
 drive(): void {}
 park(): void {}
}
class Taxi implements Car {
 drive(): void {}
 park(): void {}
}
class Suv implements Car {
 drive(): void {}
 park(): void {}
}
```

```
export class CarFactory {
 static getInstance<T extends Car>(type: { new (): T }): T {
  return new type();
 }
}

const bus = CarFactory.getInstance(Bus);
const taxi = CarFactory.getInstance(Taxi);
```

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
김준영

강사
김준영

감수자
이재성

디자이너
김루미

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io