EliceAcademy

JavaScript : 5주차

타입 스크립트와 객체지향

목차

타입스크립트의
기초 문법을 이해한다

클래스와 인터페이스를
이해하고 사용할 수 있다

제네릭을 이해할 수 있
다

EliceAcademy

01

TypeScript

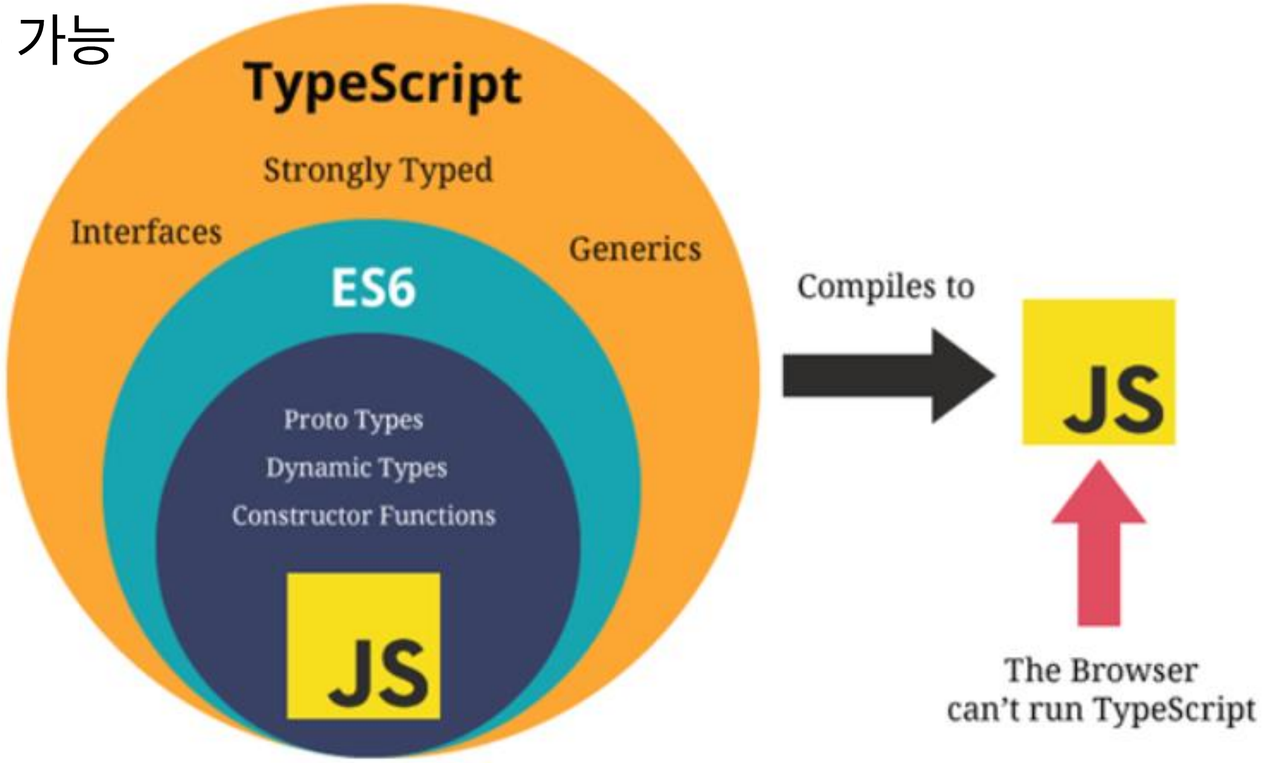
EliceAcademy

타입스크립트

마이크로 소프트에서 만든 프로그래밍언어
정적 타입 언어
자바스크립트가 동작하는 모든 곳 사용 가능
자바스크립트 모든 문법 사용 가능

브라우저는 타입스크립트
인식을 못해서 컴파일 해줘야 함

타입스크립트 안에 내장 컴파일 있다



https://velog.io/@kihyeon8949/TypeScript-Why-TypeScript

타입스크립트 특징

TypeScript = JavaScript + type
Compile time에 type을 결정하기 때문에 속도 상승
타입 에러로 인한 문제점을 초기에 발견할 수 있어 타입의 안정성 증가

동적 타입 – runtime 일 때 타입이 결정
개발 중 예상 못했던 에러를 프로그램 사용자가 예상 못한 값이 들어와서 생긴 에러가 많다.

정적타입 – 컴파일할 때 타입 결정 => 개발 할 때 즉각적으로 type-checking errors 를 받아볼 수 있다, 타입이 고정되어 있어 예상 못한 값이 들어올 확률이 줄어 든다

transplie .ts -> .js

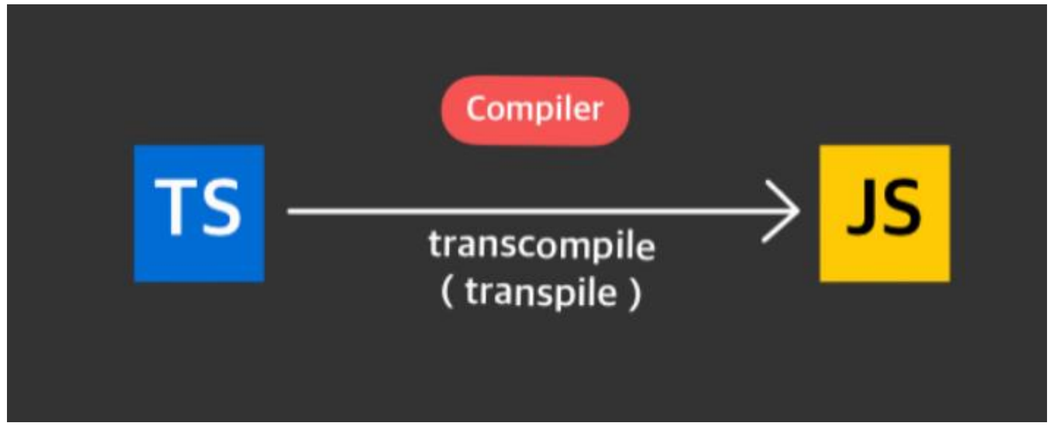
브라우저는 타입스크립트 언어를 이해하지 못해서 자바스크립트로 변경 필요

일반적인 complie은 소스코드를 컴퓨터가 이해하는 기계어로 변경

타입스크립트는 javascript의 소스 코드로 변경해서 transpile 이라고 부른다

tsc(타입스크립트 컴파일러)나 babel을 통해서 JavaScript코드로 변환

TS는 JS Runtime 동작하는 곳에서 다 사용할 수는 있지만, 꼭 TransCompiler를 통해 JS로 변환



정적 vs 동적

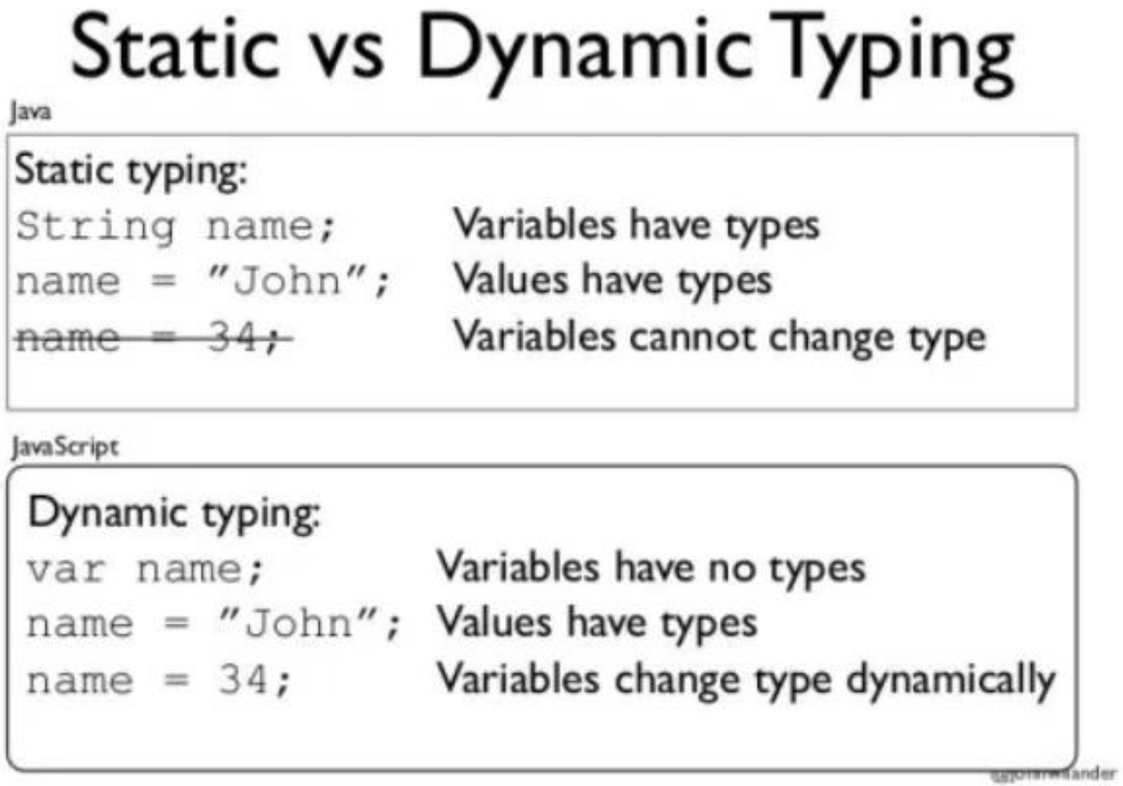
동적 타입 언어

python, ruby, php, javascript ...

정적 타입 언어

java , c , c++, go

swift, kotlin, scala, typescript



타입스크립트의 장점

정적으로 타입이 고정되어서 에러의 안정성을 보장
함수의 매개변수 값과 인자 값의 타입을 미리 확인 할 수 있어 가독성이 증가

객체지향 프로그래밍(OOP : Object-Oriented Programing) 가능
=> 생산성, 확장성 가능한 프로그래밍 구현 가능

타입스크립트 기본 셋팅

타입스크립트 컴파일 설치

npm install –g typescript

tsc -v : 버전 확인가능

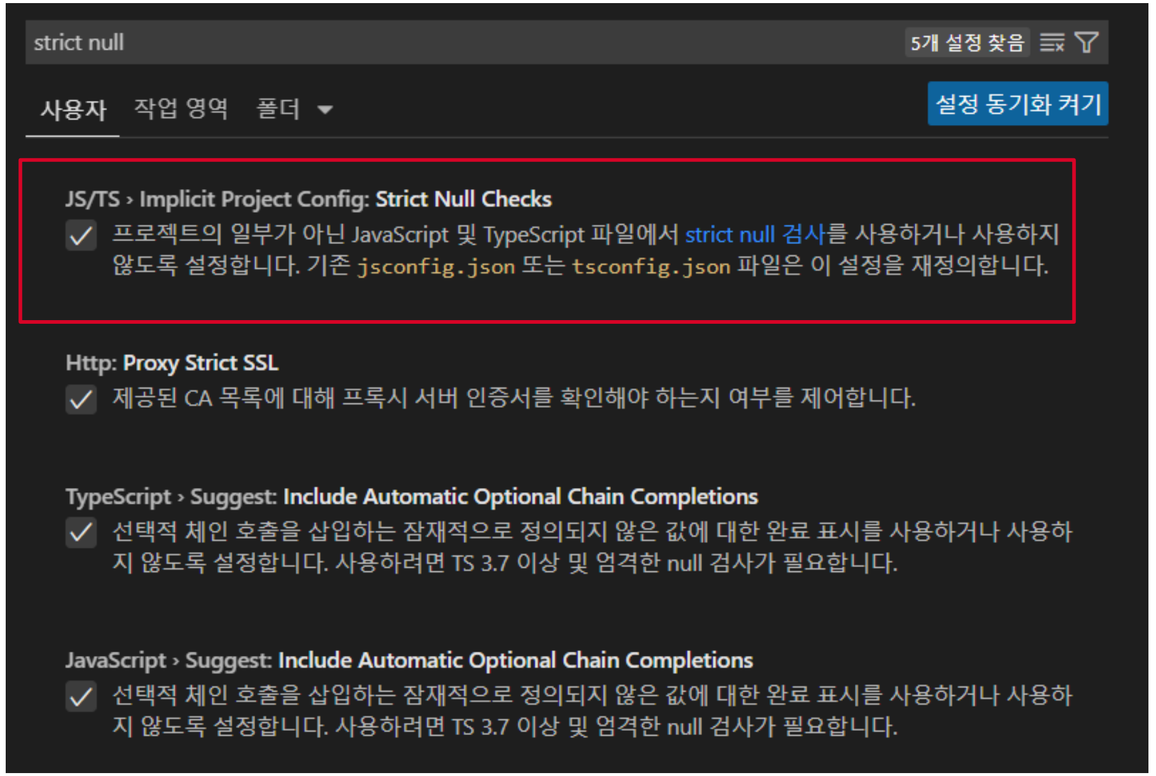
ts -> js 변경

tsc 파일이름.ts –w

타입스크립트 실행

ts-node 파일이름.ts

visual code 가서 설정(ctrl + ,) 검색으로 **‘strict null’**



TypeScript : https://www.typescriptlang.org

타입스크립트 자료형

원시 타입 (primitive type)
string, boolean, number, null , undefined , symbol,
any void never unknown

참조 타입 (reference type)
object (array, function)
tuple enum

타입 지정할 때 사용을 권장하지 않는 타입
null , undefined , any , unknown , void, never , object
기존 자바스트립트의 호환성 때문에 이런 타입들이 생성된 것

함수에서 아무것도 리턴 하지 않을 때 **void**
함수가 리턴을 못 할 때 **never**

tuple : 여러 타입이 올 수 있는 배열 타입
enum : 상수 모음 타입

함수

매개변수의 타입과 리턴 타입을 명시하고 전달 값의 타입이 일치 해야 한다.
매개변수의 개수가 일치 해야 한다.
예외

Optional 선택적 매개변수 (fristName : String , lastName**?** : String)

변수 이름 뒤에 물음표가 오면 매개변수가 있어도 되고 없어도 된다 (firstName 값은 필수 값)

Default 초기 매개변수 (fristName : String , lastName **= “Kim”**)

변수 초기값을 미리 넣어 줄 수 있다. lastName은 문자열과 undefined 값(자동 Kim)만 가능

Rest 나머지 매개변수 (num : number, ...nums : number[ ])
첫번째 인자 값은 필수이고 두번째 인자 값은 안 넣어줄 수 있고 혹은 해당 타입 여러 개를 받아올 수 있는데 배열로 받아온다

타입 별칭 (type aliases) 과 타입 추론 (type inference)

타입 별칭
나만의 타입을 정의할 수 있다
type 변수 이름 = 타입 종류
변수이름은 대문자로 시작한다 : 클래스 이름 정의랑 동일
타입 종류는 특정한 객체로도 올 수 있다

타입 추론 : 타입스크립트가 코드를 해석해 나가는 동작으로 자동으로 타입을 추론 한다
const 변수 에 값을 넣으면 고정이라서 변하지 않고 그 값을 우리가 알기 때문에 생략
함수 void 는 리턴 타입이 없어서 return; 할 경우 보통 생략
보통 함수의 인자 값을 변수의 초기값으로 할 때 그 변수 타입은 생략

타입추론
https://joshua1988.github.io/ts/guide/type-inference.html#%ED%83%80%EC%9E%85-%EC%B6%94%EB%A1%A0-type-inference

union type 과 intersection type

union : or

“ | ” (파이프) 사용해 두개 이상 타입을 선택적으로 사용할 수 있다.

intersection : and

“ & ” (앤퍼센트) 사용해 타입을 합쳐서 사용 할 수 있다.

enum : 열거타입

서로 연관 있는 독립적인 상수를 묶어서 정의

하지만 union 타입으로 대체 가능하다

보통 상수는 모두 대문자랑 스네이크테일 표기법을 사용한다
const MAX_NUM = 10;

enum 타입은 클래스와 동일하게 첫글자 만 대문자로 해준다

02

객체지향 OOP

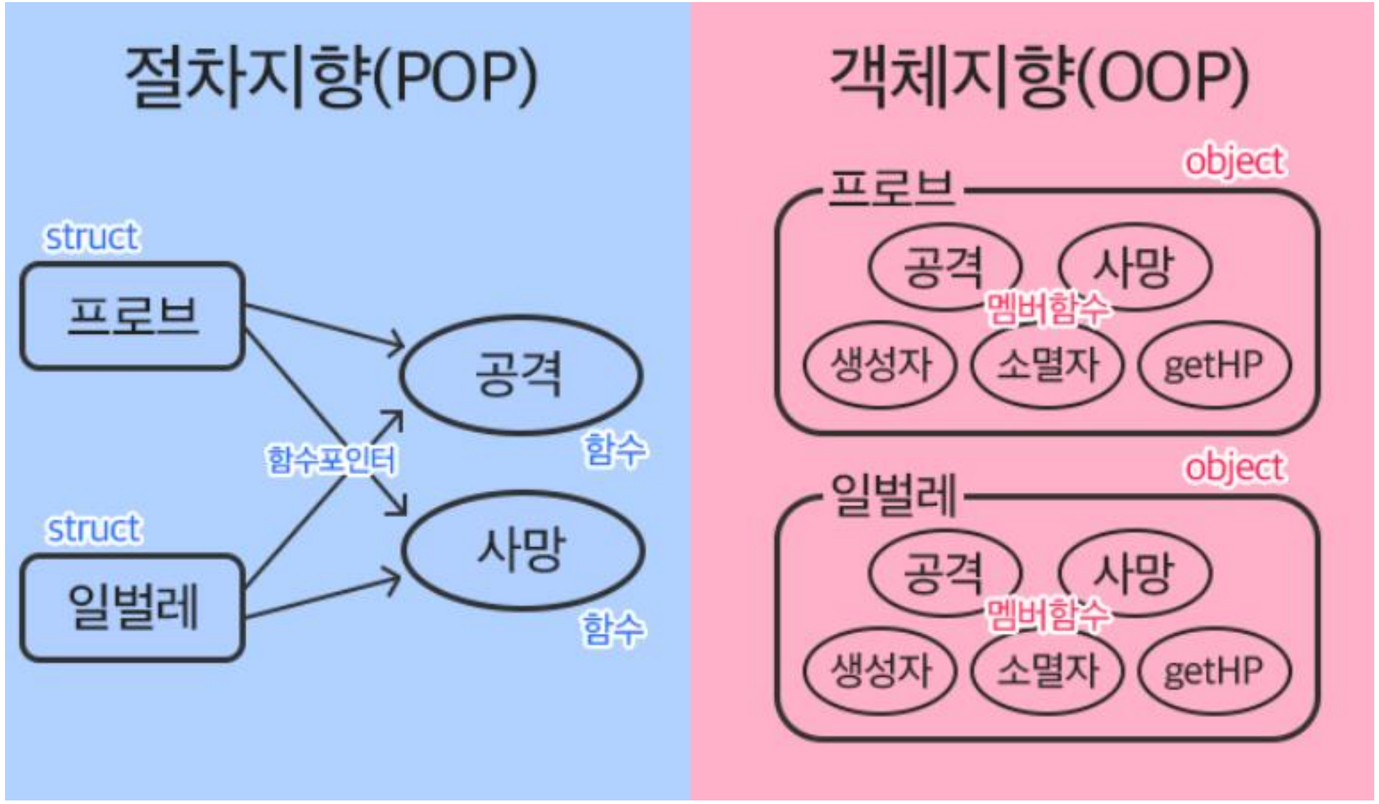
EliceAcademy

절차지향(POP) vs 객체지향(OOP)

Procedure Oriented Programming
절차지향 언어
실행 순서, 즉 절차가 중심 : 함수 호출 순서

Object Oriented Programming

필요한 객체들의 종류와 속성이 중심
프로그램은 즉 " 객체 " 들의 의사소통
객체들은 서로 메시지를 주고 받으면서
데이터를 처리



객체지향의 장점

모듈화 : 객체를 위주로 모듈성 있는 코드를 작성 가능
재사용성 : 모듈별로 원하는 것을 재사용이 편리
확장성 : 객체 단위로 확장 가능 (상속이나 인터페이스 구현을 통해)
유지보수성 : 위의 특징으로 기존 코드의 이슈해결 / 새로운 기능 추가 쉬움
생산성 ↑
코드 퀄리티 ↑
제품 개발 속도 ↑

Javascript도 프로토타입 기반 베이스로 OOP 이지만 완벽한 객체지향 프로그래밍은 아니다
TypeScript는 "class", "interface", "generic"등 을 활용
다른 Class를 베이스로하는 프로그래밍 언어처럼 강력한 OOP가 가능

객체지향 4가지 속성

캡슐화 Encapsulation
추상화 Abstraction
상속 Inheritance
다형성 Polymorphism

클래스 요소 : 멤버(==필드) , 생성자, 메서드
클래스로 만든 객체 = 인스턴스
this => 클래스로 만든 인스턴스 객체의 주소

접근제어자

public : 같은 폴더안에 있는 곳까지 접근 가능
protected : 부모 자식 클래스에서만 접근 가능
private : 자기 자신 클래스에서만 접근 가능

public > protected > private

캡슐화

외부에 노출되고 싶은 부분만 보여준다

private – getter – setter

readonly: 속성을 읽기 전용으로 하기

static : 클래스 멤버 변수 => new 하지 않고도 접근할 수 있다.

싱글톤 패턴

객체를 단 한 개만 만들고 싶을 때 사용

private 접근 제어자를 사용해 constructor() 앞에 붙이면
new 키워드를 통해 인스턴스를 생성하지 못하도록 제한
=> 클래스 안에서만 객체를 생성할 수 있다.

클래스 외부에서는 이미 만들어진 클래스 객체를 스테틱 메서드 getInstance()를 통해
불러올 수 있다. 새로운 인스턴스 생성 불가

상속과 추상화

상속을 통해 클래스를 세부화 시킬 수 있다
자식 클래스 뒤에 **extends**를 붙인다 **자식클래스는 단 한 개의 부모 클래스만 가질 수 있다**
자식 클래스는 항상 부모가 먼저 생성되야 사용 가능하다 constructor (**super()**);

보통 부모클래스는 보편화된 클래스로 추상화를 사용한다
추상클래스는 클래스 앞에 **abstract** 키워드를 붙인다

추상클래스 특징
직접 인스턴스화 할 수 없다. => new 통해 객체 생성 못함
추상 메서드 ➔ 바디{}가 없는 메서드 가 단 한 개라도 있으면 추상 클래스
 자식클래스를 통해서 추상 메서드를 반드시 구현 해야 한다 => 강제

interface

상속의 문제점을 개선하기 위해서 등장!
상속의 깊이가 깊어질수록, 서로의 관계가 복잡해짐
부모클래스가 수정되면, 자식 클래스들에 다 영향을 줌
TypeScript 에서는 두 가지 이상의 부모 클래스를 상속할 수 없음 (extends)

추상클래스보다 더 추상 클래스 개념 => 설계도 , 규약 , 계약서

모든 메서드가 추상 메서드라서 abstract 키워드 붙지 않는다
자식클래스 뒤에 implements 통해서 클래스처럼 상속 할 수 있다

타입케스팅

자식 타입을 부모타입으로 자료형을 바꾸는 것을 업케스팅
부모 타입을 다시 자식타입으로 바꾸는 것을 다운케스팅

같은 부모로 여러 자식을 만들면 배열로 묶어줄 수 있다

interface 와 type alias 차이점

interface 는 규격사항 -> 올바르게 구현 하는 것이 목적
type alias 는 데이터의 묘사 -> 데이터를 담을 목적

interface 는 또 다른 interface 상속이 가능하다
interface 는 병합이 가능하다

type은 인덱스 속성 값을 사용할 수 있다
type은 유니온 타입 사용할 수 있다

EliceAcademy

03

제네릭 generic

generic

제네릭이란 타입을 마치 함수의 파라미터처럼 사용하는 것을 의미한다.
정적 type 언어는 클래스나 함수를 정의할 때 type을 선언해야 한다.
Generic은 코드를 작성할 때가 아니라 코드를 수행될 때(런타임) 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.

https://lakelouise.tistory.com/188

generic 사용 이유

한 가지 타입보다 여러 가지 타입에서 동작하는 컴포넌트를 생성하는 데 사용된다.
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.
any타입을 사용하면 컴파일 시 타입을 체크하지 않는다.
타입을 체크하지 않아 관련 메서드의 힌트를 사용할 수 없다.
컴파일 시에 컴파일러가 오류를 찾지 못한다.
generic도 any처럼 타입을 지정하지 않지만, 타입을 체크해 컴파일러가 오류를 찾을 수 있다.