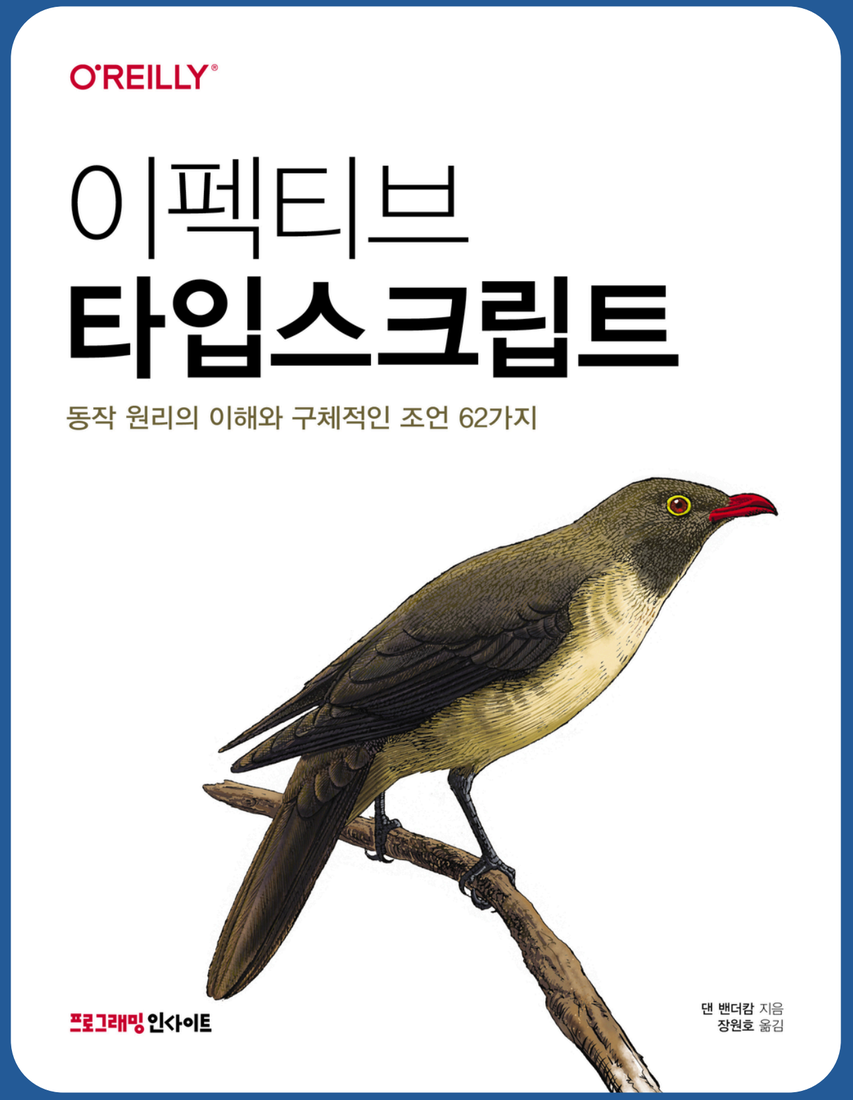
Understanding
Typescript

240511 두선아

Effective Typescript

62 Specific Ways to Improve Your TypeScript

– Dan VanderKam



Understanding Typescript

1장 타입스크립트 알아보기

함께 8주 동안 스터디할 타입스크립트를 알아봅시다

Table of Content

Item 1. TS vs JS

타입스크립트와 자바스크립트의 관계 이해하기

- Superset
- 부분집합
- 자바스크립트와 타입 체커
- 타입 체커를 통과한 타입스크립트 프로그램

Quiz ?

타입스크립트는 자바스크립트의 Superset이다. (O / X)

Superset?

모든 자바스크립트 프로그램은 타입스크립트 프로그램이다. (O / X)

부분 집합?

타입스크립트는 자바스크립트의 런타임 동작을 모델링하는 타입 시스템을 가지고 있다. (O / X)

타입 체커?

Quiz !

타입스크립트는 자바스크립트의 Superset이다! 

타입스크립트는 타입이 정의된 자바스크립트의 Superset (상위집합) 입니다
문법적으로도 자바스크립트의 상위집합 입니다

- JS에 문법 오류가 없다면? 유효한 타입스크립트 프로그램!
- JS에 이슈가 있다면? 문법 오류가 아니더라도, 타입 체커에게 지적됩니다

모든 자바스크립트 프로그램은 타입스크립트 프로그램이다! 

JS 에서 TS로 마이그레이션할 때 이점이 됩니다
반대로 모든 타입스크립트 프로그램은 자바스크립트가 아닌 이유는 타입을 명시하는 추가적인 문법을 가지기 때문입니다.

타입스크립트는 자바스크립트의 런타임 동작을 모델링하는 타입 시스템을 가지고 있다! 

런타임 오류를 발생시키는 코드를 찾아내려 합니다
하지만 타입 체커를 통과하면서도 런타임 오류를 발생키는 코드는 충분히 존재합니다
즉, 타입 시스템의 목적이 정적 타입의 정확성을 보장하는 것은 아닙니다

Quiz ?

TS

```ts
const numbers = [1, 2, 3];
console.log(numbers[3].toFixed());
```

이 코드는
자바스크립트일까요? 🤔

코드에서 어떤 일이 발생하나요?

- a) 배열의 인덱스 범위를 초과하여 접근하는 오류
- b) undefined의 속성을 읽을 수 없는 오류
- c) toFixed()에 정확한 타입의 인수를 전달하지 않은 오류
- d) 오류 없이 실행됨

타입 체커는 타입 에러를 발견할 수 있나요? (O / X)

Quiz ?

TS

```ts
const numbers = [1, 2, 3];
console.log(numbers[3].toFixed());
```

타입스크립트이면서
자바스크립트 입니다!

b) undefined의 속성을 읽을 수 없는 오류

- JavaScript 엔진은 배열의 인덱스를 벗어나는 경우 undefined를 반환합니다. (오류가 발생하지 않음)
- undefined 값에 toFixed() 메서드를 호출하면 타입에러가 발생합니다.

```
▶ Uncaught TypeError: Cannot read properties of undefined (reading 'toFixed')
    at <anonymous>:1:23
```

타입 체커는 타입 에러를 발견하지 못했습니다

- 1) 타입 체커는 numbers를 `number[]` 로 추론합니다
- 2) 런타임 이전에 배열의 길이를 알 수 없습니다
  - 만약 고정된 길이의 배열이라면? 튜플을 사용하여 각 요소에 대한 타입과 길이를 고정할 수 있습니다.

TS

```ts
interface Fruits {
  name: string;
  details: {
    info: {
      icon: string;
    };
  };
}

const fruits: Fruits[] = [
  { name: "Apple", details: { info: { icon: "🍎" } } },
  { name: "Orange", details: { info: { iocn: "🍊" } } }, // 1️⃣
  { name: "Grape", details: { info: { icon: "🍇" } } },
  { name: "Cherry", details: null }, // 3️⃣
];

for (const fruit of fruits) {
  console.log(fruit.details.icon); // 2️⃣
}
```

Quiz ?

코드의 1, 2, 3번 위치에
경고가 발생합니다

3가지 경고 메시지와,
발생 위치는 어디일까요?

<보기>

- a) Type 'null' is not assignable to type '{ info: { icon: string; }; }'.
- b) Property 'icon' does not exist on type '{ info: { icon: string; }; }'
- c) Type '{ iocn: string; }' is not assignable to type '{ icon: string; }'.
  Object literal may only specify known properties,
  and 'iocn' does not exist in type '{ icon: string; }'.
- d) 'fruit.details' is possibly 'undefined'.

TS

```ts
interface Fruits {
  name: string;
  details: {
    info: {
      icon: string;
    };
  };
}

const fruits: Fruits[] = [
  { name: "Apple", details: { info: { icon: "🍎" } } },
  { name: "Orange", details: { info: { iocn: "🍊" } } }, // 1️⃣
  { name: "Grape", details: { info: { icon: "🍇" } } },
  { name: "Cherry", details: null }, // 3️⃣
];

for (const fruit of fruits) {
  console.log(fruit.details.icon); // 2️⃣
}
```

Quiz !

보기 a – 위치 3
보기 c – 위치 1
보기 b – 위치 2

a) Type 'null' is not assignable to type '{ info: { icon: string; }; }'.

- null로 초기화할 수 없습니다. null을 사용하고 싶다면
  details: { info: { icon: string }} | null 과 같이 작성합니다.

c) Type '{ iocn: string; }' is not assignable to type '{ icon: string; }'.
Object literal may only specify known properties,
and 'iocn' does not exist in type '{ icon: string; }'.

- 오타를 작성한 경우입니다!

b) Property 'icon' does not exist on type '{ info: { icon: string; }; }'

- fruit.details.info.icon과 같이 접근해야 합니다.
- 마찬가지로 휴먼에러입니다.

Quiz ?

TS

{ noImplicitAny: false }

```ts
// 경우 A
function 둘_나누기_A(input) {
  return input / 2;
}
console.log(둘_나누기_A("10"));

// 경우 B
function 둘_나누기_B(input: number) {
  return input / 2;
}
console.log(둘_나누기_B("10"));

// 경우 C
function 둘_나누기_C(input: number) {
  return input / 2;
}
console.log(둘_나누기_C("10" as any));
```

경고가 발생할까요?
어떤 위치에서 발생할까요?
이유는 무엇인가요?

둘_나누기_C("10" as any)의 결과는 무엇인가요?

Quiz !

TS

{ noImplicitAny: false }

```ts
// 경우 A
function 둘_나누기_A(input) {
  return input / 2;
}
console.log(둘_나누기_A("10"));

// 경우 B
function 둘_나누기_B(input: number) {
  return input / 2;
}
console.log(둘_나누기_B("10"));

// 경우 C
function 둘_나누기_C(input: number) {
  return input / 2;
}
console.log(둘_나누기_C("10" as any));
```

둘_나누기_C("10" as any)는 5를 반환합니다
자바스크립트는 암시적 타입 변환을 통해 "10"을 10으로
강제 타입변환(type coercion)합니다.

경우A

noImplicitAny가 false 이므로 경고가 발생하지 않습니다

매개변수 input의 타입은 any입니다

경우B

인수의 타입이 일치하지 않아 경고가 발생합니다

```
Argument of type 'string' is not assignable to parameter of type
'number'. ts(2345)
```

경우C

타입 단언(type assertion)된 인수를 전달했기 때문에

경고가 발생하지 않습니다

실제와 다른 타입을 단언했기 때문에

런타임에 에러가 발생할 수 있는 가능성이 있습니다.

Item 2. Which TS

타입스크립트 설정 이해하기

- implicitAny
- strictNullChecks

Quiz ?

타입스크립트 설정을 위해서 tsconfig.json을 사용하는 것이 좋은 이유는?

- a) 타입스크립트 코드의 문제를 더 쉽게 발견할 수 있기 때문
- b) 다른 도구와의 호환성을 보장하기 위해서

noImplicitAny 설정을 true로 설정하는 경우에 대한 설명으로 옳은 것은?

- a) 타입스크립트 컴파일러는 모든 함수의 매개변수에 대한 타입을 명시하지 않으면 경고를 발생시킵니다.
- b) 타입 명시를 하지 않은 경우 코드는 컴파일되지 않습니다.
- c) 개발자가 생산성을 향상시키기 위해 false로 설정하는 것이 일반적입니다
- d) 모든 항목이 옳은 답

strictNullChecks 설정을 true로 설정하는 이유는
'undefined는 객체가 아닙니다'와 같은 런타임 오류를 방지하기 위해서이다. (O / X)

Quiz !

타입스크립트 설정을 위해서 tsconfig.json을 사용하는 것이 좋은 이유는?

b) 다른 도구와의 호환성을 보장하기 위해서

- tsconfig.json 파일은 TypeScript 프로젝트의 구성 파일로, TypeScript 컴파일러에게 프로젝트에 대한 설정 및 규칙을 알려줍니다.
- 이를 통해 다양한 도구와의 호환성을 보장하고 프로젝트를 일관되게 유지할 수 있습니다.

noImplicitAny 설정을 true로 설정하는 경우에 대한 설명으로 옳은 것은?

a) 타입스크립트 컴파일러는 모든 함수의 매개변수에 대한 타입을 명시하지 않으면 경고를 발생시킵니다.

- noImplicitAny 설정을 true로 설정하면, TypeScript 컴파일러가 모든 함수의 매개변수와 반환값에 대한 타입을 명시하지 않으면 경고를 발생시킵니다. 이를 통해 타입 안전성을 높이고 코드의 가독성을 향상시킬 수 있습니다.

strictNullChecks 설정을 true로 설정하는 이유는 'undefined는 객체가 아닙니다'와 같은 런타임 오류를 방지하기 위해서이다. 👩

- strictNullChecks 설정을 true로 설정하면, null 또는 undefined 값이 있는 경우 이에 대한 명시적인 타입 선언을 요구합니다. 이를 통해 런타임 중 'undefined는 객체가 아닙니다'와 같은 오류를 방지할 수 있습니다.

Quiz ?

TS

{ noImplicitAny: true, strictNullChecks: true }

```ts
function registerCallback(callback) {
  callback();

  if (callback) {
    callback();
  } else {
    console.log("No callback provided.");
  }
}

const button = document.getElementById("myButton");

registerCallback(() => {
  button.addEventListener("click", (e) => {
    e.stopPropagation();
    console.log("Button clicked!");
  });
});
```

코드에서
발생하는 경고의 위치와,
관련된 tsconfig 속성의 이름은?

<보기>

- a) noImplicitThis
- b) noImplicitAny
- c) noImplicitReturns
- d) strictNullChecks

해결 방법은?

Quiz !

TS

{ noImplicitAny: true, strictNullChecks: true }

```ts
function registerCallback(callback) {
  callback();

  if (callback) {
    callback();
  } else {
    console.log("No callback provided.");
  }
}

const button = document.getElementById("myButton");

registerCallback(() => {
  button.addEventListener("click", (e) => {
    e.stopPropagation();
    console.log("Button clicked!");
  });
});
```

b) noImplicitAny

```
Parameter 'callback' implicitly has an 'any' type. ts(7006)
```

noImplicitAny 설정이 true로 설정되어 있기 때문에
callback 매개변수에 대한 타입이 명시되지 않았을 때
TypeScript가 암시적 any 타입을 허용하지 않는 경고가 발생합니다.

d) strictNullChecks

```
'button' is possibly 'null'. ts(18047)
```

strictNullChecks 설정이 true로 설정되어 있기 때문에
null 또는 undefined가 될 수 있는 값에 대한 타입을
명시적으로 지정하거나, null 및 undefined를 체크해야 합니다.

해결 방법은?

b) callback 매개변수에 대한 타입을 명시적으로 지정합니다
d) null 및 undefined를 체크하거나, 옵셔널 체이닝을 사용하거나, 단언합니다

Solutions

경고와 타입 에러를 해결하는 방법은 다양하기 때문에
핵심은 서로 문제 원인을 이해하고 리뷰하고, 소통할 수 있고
더 나은 해결 방법을 적용 & 공유하는 것이라고 생각합니다!

```ts
function registerCallback(callback: Function) {
  callback();

  if (callback) {
    callback();
  } else {
    console.log("No callback provided.");
  }
}

const button = document.getElementById("myButton");

registerCallback(() => {
  button?.addEventListener("click", (e) => {
    e.stopPropagation();
    console.log("Button clicked!");
  });
});
```

callback: Function

callback: () => void

```ts
const button = document.getElementById("myButton") as HTMLButtonElement;
```

```ts
function registerCallback(callback: () => void ) {
  if (callback) {
    callback();
  } else {
    console.log("No callback provided.");
  }
}

const button = document.getElementById("myButton");

if (button) {
  registerCallback(() => {
    button.addEventListener("click", (e) => {
      e.stopPropagation();
      console.log("Button clicked!");
    });
  });
} else {
  console.error("Button not found.");
}
```

as HTMLButtonElement;

button?.

button && button

if(button) { }

Item 3. Independent

코드 생성과 타입이 관계없음을 이해하기

- 런타임에 타입 정보를 유지하기
- 코드 생성은 타입 시스템과 무관하다

Example A, B 런타임에 타입 정보를 유지하기

TS

{ noImplicitAny: true, strictNullChecks: true }

```ts
interface Circle {
  kind: "circle";
  radius: number;
}

interface Square {
  kind: "square";
  width: number;

type Shape = Circle | Square;

// Circle일 때는 Math.PI * shape.radius ** 2을 반환
// Square일 때는 shape.width ** 2를 반환
function calculateArea(shape: Shape) {
  ? ? ?
}

// 테스트
const circle: Circle = { kind: "circle", radius: 5 };
const square: Square = { kind: "square", width: 4 };

console.log(calculateArea(circle)); // 78.54...
console.log(calculateArea(square)); // 16
```

A. 속성을 체크하기

```ts
function calculateArea(shape: Shape): number {
  if ("radius" in shape) {
    return Math.PI * shape.radius ** 2;
  } else if ("width" in shape) {
    return shape.width ** 2;
  } else {
    throw new Error("Unknown shape kind");
  }
}
```

B. tagged union을 사용하기

```ts
function calculateArea(shape: Shape): number {
  if (shape.kind === "circle") {
    return Math.PI * shape.radius ** 2;
  } else if (shape.kind === "square") {
    return shape.width ** 2;
  } else {
    throw new Error("Unknown shape kind");
  }
}
```

Example C

런타임에
타입 정보를 유지하기

TS

{ noImplicitAny: true, strictNullChecks: true }

```ts
class Circle {
  constructor(public radius: number) {}
}

class Square {
  constructor(public width: number) {}
}

type Shape = Circle | Square;

// Circle일 때는 Math.PI * shape.radius ** 2을 반환
// Square일 때는 shape.width ** 2를 반환
function calculateArea(shape: Shape) {
  if (shape instanceof Circle) {
    return Math.PI * shape.radius ** 2;
  } else if (shape instanceof Square) {
    return shape.width ** 2;
  }
}

// 테스트
const circle = new Circle(5);
const square = new Square(4);

console.log(calculateArea(circle)); // 78.54...
console.log(calculateArea(square)); // 16
```

런타임에
타입 정보를 유지하기

C. 클래스로 만들기

interface는 타입으로만 사용합니다
컴파일 시 타입 체크에 사용되고
사라지기 때문에 런타임에 영향을 미치지 않습니다

class로 선언하면 클래스의 인스턴스를 생성하여
instanceof 으로 타입을 참조할 수 있습니다

Example: function overloading

TS

```ts
function format(input: number): string; // 숫자 형식
function format(input: string): string; // 문자열 형식

function format(input: any): string {
  if (typeof input === "number") {
    return `Number: ${input}`;
  } else if (typeof input === "string") {
    return `String: ${input}`;
  } else {
    throw new Error("Unsupported type");
  }
}

const formattedNumber = format(42); // "Number: 42"
const formattedString = format("Hello"); // "String: Hello"
```

TypeScript는 JavaScript와 달리
함수 오버로딩을 지원합니다

하나의 함수에 대해
여러 개의 시그니처(선언문)를 작성하여
다양한 형태의 매개변수나 반환값을 지정할 수 있습니다

그러나 실제로 JavaScript로 컴파일될 때는
하나의 구현체만 존재합니다

TypeScript의 함수 오버로딩은
타입 수준에서 동작하며
주로 타입체크를 위해 사용합니다

Item 4. Structural

구조적 타이핑 익숙해지기

- 구조적 타이핑으로 발생한 문제 코드

구조적 타이핑

Example: structural typing

타입 체커의 타입에 대한 이해도는 사람과 다르다

```ts
type 맛_단계 = 1 | 2 | 3 | 4 | 5;

// 마라탕과 탕후루를 정의하는 인터페이스
interface 탕후루 {
  재료: string[];
  단맛: 맛_단계;
}

function 마라탕만들기(마라탕: 탕후루) {
  console.log("재료로 마라탕 만들기:", 마라탕.재료);
  // 마라탕 만드는 로직
}

function 탕후루만들기(탕후루: 탕후루) {
  console.log("재료로 탕후루 만들기:", 탕후루.재료);
  // 탕후루 만드는 로직
}

// 마라탕 재료와 매운 정도
const 마라탕재료 = ["소고기", "두부", "버섯"];
const 마라탕요리 = {
  재료: 마라탕재료,
  매운맛: 5,
  얼얼맛: 5,
  단맛: 1 as 맛_단계,
};

// 탕후루 재료와 매운 정도
const 탕후루재료 = ["딸기", "샤인머스캣"];
const 탕후루요리: 탕후루 = {
  재료: 탕후루재료,
  단맛: 5,
};

// 마라탕과 탕후루 만들기
마라탕만들기(마라탕요리);
탕후루만들기(탕후루요리);
탕후루만들기(마라탕요리);
```

function 마라탕만들기는
매개변수로 받을 값을 interface 탕후루 타입으로 지정했습니다.

하지만 함수에 실제로 전달된 인수는
interface 탕후루와 다르게
매운맛, 얼얼맛 속성을 추가로 가지고 있습니다

함수의 매개변수 값이 모두 주어진다면 (구조적으로 호환된다면)
값이 어떻게 만들어졌는지 신경쓰지 않는
덕 타이핑 Duck Typing 기반의 자바스크립트를 모델링하기 위해서
타입스크립트 또한 구조적 타이핑을 사용합니다

걸어다니고, 날아다니고, 꽥꽥대는 것은 오리인 것처럼
재료들을 가지고 있고, 단맛을 조절 가능한 것은? 탕후루 재료입니다.

객체가 필요한 속성을 가지고 있다면
해당 객체를 그 타입으로 사용할 수 있습니다

Structural

Item 5. any

any 타입 지양하기

- 타입 체커와 타입스크립트 언어를 무력화시키는 any

any 쓰지 않기

- any 타입에는 **타입 안정성이 없다**
- any는 **함수 contract (시그니처)를 무시**한다
- any 타입에는 **언어 서비스가 적용되지 않는다**
- any 타입은 **코드 리팩터링 때 버그를 감춘다**
- any는 **타입 설계를 감춘다**
- any는 **타입 시스템의 신뢰도를 떨어뜨린다**

any 쓰지 않기 운동

Example:
any를 멀리하지 않은 결과
(feat. 함수가 any)

```ts
interface ExchangeRate {
  currency: string;
  rate: number;
}

const DOLLAR_TO_WON_RATE: ExchangeRate = { currency: "KRW", rate: 1150 }; // 1달러 = 1150원
const WON_TO_DOLLAR_RATE: ExchangeRate = { currency: "USD", rate: 1 / 1150 }; // 1만원 = 1/1150 달러

const exchange = (
  amount: number,
  exchangeRate: ExchangeRate,
  bonus: number
) => {
  return bonus + amount * exchangeRate.rate;
};

// 원화를 달러로 환전하는 함수
const exchangeWonToDollar = (amount: number, bonus: number = 0) => {
  const rate = WON_TO_DOLLAR_RATE;
  const money = exchange(amount, WON_TO_DOLLAR_RATE, bonus); // 1달러를 이벤트 보너스로 지급
  return `${money} ${rate.currency}`;
};

// 달러를 원화로 환전하는 함수
const exchangeDollarToWon: any = (amount: number, bonus: number = 0) => {
  const rate = DOLLAR_TO_WON_RATE;
  const money = exchange(amount, DOLLAR_TO_WON_RATE, bonus); // 1000원을 이벤트 보너스로 지급?
  return `${money} ${rate.currency}`;
};

const event_bonus = {
  wonToDollaer: 1,
  dollarToWon: "1000",
};
```

any는 함수 시그니처를
무시합니다

범인은 여기 있는 any

```ts
// 10만원을 달러로 환전
console.log(exchangeWonToDollar(100000, event_bonus.wonToDollaer));

// 100달러를 원으로 환전
console.log(exchangeDollarToWon(100, event_bonus.dollarToWon));
```

10만원은 이제
약 87.9달러 입니다

100달러는 이제
1,000,115,000원 입니다 😱

Example: any를 멀리하지 않은 결과 (feat. api response)

any 쓰지 않기 운동

```ts
interface UserData {
  id: string;
  name: string;
  height: string;
}

interface DataProvider {
  fetchData: (userId: string) => any;
}

class UserApi implements DataProvider {
  private data: any;

  constructor() {
    this.data = {};
  }

  setData(data: UserData) {
    this.data = data;
  }

  fetchData(userId: string): UserData {
    // 네트워크 요청~
    // 실제 시나리오에서는 사용자 ID를 기반으로 API에서 비동기로 사용자 데이터를 가져올 것
    return {
      id: userId,
      name: "김유저",
      height: "180.5",
    };
  }
}
```

서버에서 받는 값은
일단 any로 작성했구요
고칠거예요🤫

any를 멀리하지 않은 자

여기서 리턴 타입도
명시했으니까
문제 없을 것 같은데요😛

```ts
const userApi: DataProvider = new UserApi();
const fetchedUserData = userApi.fetchData("유저아이디");
console.log(`User Height: ${fetchedUserData.height.toFixed(2)}`);
// 🚫 런타임 오류: toFixed는 함수가 아님
```

Runtime Error

any 때문에 언어 서비스가 안되나요?
잘못 작성했다구요?

그럼 respone 데이터도
타입 지정할게요😒

```ts
class UserApi implements DataProvider {
  private data: UserData | undefined;

  constructor() {
    this.data = undefined;
  }
```

```ts
const userApi: DataProvider = new UserApi();
const fetchedUserData = userApi.fetchData("유저아이디");
console.log(`User Height: ${fetchedUserData.height.toFixed(2)}`);
// 🚫 런타임 오류: toFixed는 함수가 아님
```

경고가 안뜬다구요?
어딜 또 수정해야 하나요?😡

범인은....

```ts
interface DataProvider {
  fetchData: (userId: string) => any;
}
```

any....

타입을
소중히
합시다

타입을 명시하도록 하고, 유형을 모르는 타입은
타입 안정성을 유지하는 unknown을 사용하자

```ts
interface UserData {
  id: string;
  name: string;
  height: string;
}

interface DataProvider<T> {
  fetchData: (userId: string) => T;
}

class UserApi implements DataProvider<UserData> {
  private data: UserData | undefined;

  constructor() {
    this.data = undefined;
  }

  setData(data: UserData) {
    this.data = data;
  }

  fetchData(userId: string) {
    // 네트워크 요청~
    // 실제 시나리오에서는 사용자 ID를 기반으로 API에서 비동기로 사용자 데이터를 가져올 것
    return {
      id: userId,
      name: "김유저",
      height: "180.5",
    };
  }
}

const userApi = new UserApi();
const fetchedUserData = userApi.fetchData("유저아이디");
console.log(`User Height: ${fetchedUserData.height.toFixed(2)}`);
// 🚫 런타임 오류: toFixed는 함수가 아님
```

Example: 제네릭을 사용하자

제네릭 쓰기 운동!

장점

- 타입 변환과 타입 확인을 최소화
- 코드가 더 간결해지고 가독성이 향상됨
- 데이터 유형에 관계없이 여러 유형의 데이터에 대해 작동하는 재사용이 가능한 코드를 작성할 수 있음

코드 체커 일하는 중

```
Property 'toFixed' does not exist on type 'string'. Did you mean 'fixed'?
 ts(2551)

lib.es2015.core.d.ts(483, 5): 'fixed' is declared here.
```

+a

감사합니다!

reference

이펙티브 타입스크립트, Dan Vanderkam – 1장 타입스크립트 알아보기

Typescript Handbook – https://www.typescriptlang.org/docs/handbook/intro.html

예제 – chatGPT랑 같이 만듦