Item20. 다른 타입에는 다른 변수 사용하기

Item20. 다른 타입에는 다른 변수 사용하기

- 자바스크립트에서는 한 변수를 다른 목적을 가지는 다른 타입으로 재사용가능
- 타입스크립트에서는 두가지 오류가 발생한다.

```
let id = "12-34-56";
fetchProduct(id);
id = 123456;
fetchProductBySerialNumber(id);
```

변수의 값은 바뀔 수 있지만 타입은 바뀌지 않는다.

➡ 타입의 범위를 좁히기 (타입을 더 작게 제한)

Item20. 다른 타입에는 다른 변수 사용하기

```
let id = "12-34-56";
fetchProduct(id);
id = 123456;
fetchProductBySerialNumber(id);
```

유니온 타입

```
let id : string | number = "12-34-56";
```

```
const id = "12-34-56";
fetchProduct(id);
const serial = 123456;
fetchProductBySerialNumber(serial);
```

별도의 변수 사용
let -> const

Item20. 다른 타입에는 다른 변수 사용하기

재사용되는 변수

```
const id = "12-34-56";
fetchProduct(id);
const serial = 123456;
fetchProductBySerialNumber(serial);
```

가려지는 변수

```
const id = "12-34-56";
fetchProduct(id);
{
    const id = 123456;
    fetchProductBySerialNumber(serial);
}
```

Item21. 타입 넓히기

Item21. 타입 넓히기

```ts
interface Vector3 { x: number; y: number; z: number };
function getComponent(vector: Vector3, axis: 'x' | 'y' | 'z') {
  return vector[axis];
}

let x = 'x';
let vec = { x: 10, y: 20, z: 30};
getComponent(vec, x); // error
```

'"x" | "y" | "z"'

Item21. 타입 넓히기

const 변수 선언

```
const x = 'x'; // 타입이 "x"
let vec = { x: 10, y: 20, z: 30 };
getComponent(vec, x); // 정상
```

'"x" | "y" | "z"'

```
const v = {
    x: 1,
};
v.x = 3;
v.x = '3';
v.y = 4;
v.name = 'Pythagoras';
```

{readonly x: 1}

{x: number}

{{[key: string]: number}}

object

v의 타입은 {x: number}

Item21. 타입 넓히기

타입추론 강도 직접 제어 -> 기본동작 재정의

1. 명시적 타입 구문 제공
2. 타입 체커에 추가적 문맥 제공
3. const 단언문 사용하기

Item21. 타입 넓히기

```ts
const v: {x: 1|3|5 } = {
    x: 1,
};
```

명시적 타입 구문 or const 단언문 추가

```ts
const v1 = {
    x: 1,
    y: 2,
}; // 타입 { x: number; y: number; }

const v2 = {
    x: 1 as const,
    y: 2,
}; // 타입 { x: 1; y: number; }

const v3 = {
    x: 1 ,
    y: 2,
} as const; // 타입 {readonly x: 1; readonly y: 2; }
```

Item22. 타입 좁히기

Item22. 타입 좁히기

```
const el = document.getElementById('foo'); // 타입 HTMLElement | null
if(el) {
  el                                       // 타입 HTMLElement
  el.innerHTML = 'Party Time'.blink();
} else {
  el                                       // 타입 null
  alert('No element #foo');
}
```

```
const el = document.getElementById('foo'); // 타입 HTMLElement | null
if(!el) throw new Error('Unable to find #foo');
el;                                        // 타입 HTMLElement
el.innerHTML = 'Party Time'.blink();
```

Item22. 타입 좁히기

instanceof

```ts
function contains(text: string, search: string|RegExp) {
  if(search instanceof RegExp) {
    search
    return !!search.exec(text);
  }
  search
  return text.includes(search);
}
```

속성체크

```ts
interface A { a: number };
interface B { b: number };
function pickAB(ab: A | B) {
  if('a' in ab) {
    ab // 타입 A
  } else {
    ab // 타입 B
  }
ab // 타입 A | B
}
```

Item22. 타입 좁히기

내장함수

```
function contains(text: string, terms: string|string[]) {
    const termList = Array.isArray(terms) ? terms : [terms];
    termList   //  타입 string[]
}
```

Item22. 타입 좁히기

명시적 '태그'

```ts
interface UploadEvent { type: 'upload'; filename: string; contents: string};
interface DownloadEvent { type: 'download'; filename: string };

type AppEvent = UploadEvent | DownloadEvent;

function handleEvent(e: AppEvent) {
  switch(e.type) {
    case 'download':
    e      // 타입 DownloadEvent
    break;
    case 'upload':
    e;   // 타입 UploadEvent
    break;
  }
}
```

Item22. 타입 좁히기

```
const jackson5 = ['Jackie', 'Tito', 'Jermaine', 'Marlon', 'Michael'];
const members = ['Janet', 'Micahel'].map(
  who => jackson5.find(n => n === who)
  ).filter(who => who !== undefined);  // 타입 (string | undefined)[]
```

사용자 정의 타입 가드

```
function isDefined<T>(x: T | undefined): x is T {
    return x !== undefined;
}
const members = ['Janet', 'Micahel'].map(
    who => jackson5.find(n => n === who)
    ).filter(isDefined); // 타입 string[]
```