Item 37. 공식 명칭에는 상표를 붙이기

'23. 06. 30. (금) 신현호 (@SWARVY)

```ts
interface Vector2D {
  x: number;
  y: number;
}
function calculateNorm(p: Vector2D) {
  return Math.sqrt(p.x * p.x + p.y * p.y);
}

calculateNorm({x: 3, y: 4});  // 정상, 결과는 5
const vec3D = {x: 3, y: 4, z: 1};
calculateNorm(vec3D);         // 정상! 결과는 동일하게 5
```

Vector2D로 2차원 벡터를 표현하려는 의도임을 확인할 수 있어요

하지만 구조적 타이핑의 특성으로, 이와 같이 작성해도 문제는 없어요

구조적 타이핑 관점에서 문제는 없어도 따져보면 좀 이상하잖아요?

이런 일을 방지하려면, 공식 명칭을 사용하면 됩니다

* 공식 명칭이란, 타입이 아니라 값의 관점에서 표현하는거에요

공식 명칭 개념을 TS에서 흉내내려면 상표를 붙이면 됩니다!

더이상 작동하지 않아요!

```ts
interface Vector2D {
  _brand: '2d';
  x: number;
  y: number;
}
function vec2D(x: number, y: number): Vector2D {
  return {x, y, _brand: '2d'};
}
function calculateNorm(p: Vector2D) {
  return Math.sqrt(p.x * p.x + p.y * p.y);  // 기존과 동일합니다.
}

calculateNorm(vec2D(3, 4));  // 정상, 5를 반환합니다.
const vec3D = {x: 3, y: 4, z: 1};
calculateNorm(vec3D);
              // ~~~~ '_brand' 속성이 ... 형식에 없습니다.
```

```ts
interface Vector2D {
  _brand: '2d';
  x: number;
  y: number;
}
function vec2D(x: number, y: number): Vector2D {
  return {x, y, _brand: '2d'};
}
function calculateNorm(p: Vector2D) {
  return Math.sqrt(p.x * p.x + p.y * p.y);  // 기존과 동일합니다.
}

calculateNorm(vec2D(3, 4));  // 정상, 5를 반환합니다.
const vec3D = {x: 3, y: 4, z: 1};
calculateNorm(vec3D);
              // ~~~~ '_brand' 속성이 ... 형식에 없습니다.
```

그런데 이거가지고는 vec3D에 _brand를 '3d'로 적는 악의적인 사용을 막을 수 없어요, 단순 실수방지정도죠

상표 기법은 타입시스템에서 동작하지만
런타임에 상표를 검사하는 것과 동일한 효과를 지녀요

```ts
type AbsolutePath = string & {_brand: 'abs'};
function listAbsolutePath(path: AbsolutePath) {
  // ...
}
function isAbsolutePath(path: string): path is AbsolutePath {
  return path.startsWith('/');
}
```

런타임에는 절대 경로로 시작하는지 체크하기 쉽지만,
타입 시스템에서는 절대경로를 판단하기 어려워서 상표 기법을 사용해요!

string타입이면서 _brand속성을 가지는 객체를 만들수는 없으니
AbsolutePath는 온전히 타임 시스템의 영역이에요

```ts
function f(path: string) {
  if (isAbsolutePath(path)) {
    listAbsolutePath(path);
  }
  listAbsolutePath(path);
                // ~~~~ 'string' 형식의 인수는 'AbsolutePath' 형식의
                //      매개변수에 할당될 수 없습니다.
}
```

상대경로도 가능하다면 타입가드를 사용해서
오류를 방지하면 됩니다!

로직을 분기하는 대신 단언문을 사용하여 오류를 제거할 수도 있지만
단언문은 지양하는게 좋아요

상표 기법은 타입 시스템 내에서 표현할 수 없는 수많은 속성들을 모델링할 수 있습니다.

```
function binarySearch<T>(xs: T[], x: T): boolean {
  let low = 0, high = xs.length - 1;
  while (high >= low) {
    const mid = low + Math.floor((high - low) / 2);
    const v = xs[mid];
    if (v === x) return true;
    [low, high] = x > v ? [mid + 1, high] : [low, mid - 1];
  }
  return false;
}
```

목록이 정렬되어있다는 의도를 표현하고싶다면?

```
type SortedList<T> = T[] & {_brand: 'sorted'};

function isSorted<T>(xs: T[]): xs is SortedList<T> {
  for (let i = 1; i < xs.length; i++) {
    if (xs[i] < xs[i - 1]) {
      return false;
    }
  }
  return true;
}

function binarySearch<T>(xs: SortedList<T>, x: T): boolean {
  // ...
}
```

상표 기법을 사용하여 정렬되었음을 표현할 수 있습니다

```ts
type Meters = number & {_brand: 'meters'};
type Seconds = number & {_brand: 'seconds'};

const meters = (m: number) => m as Meters;
const seconds = (s: number) => s as Seconds;

const oneKm = meters(1000);  // 타입이 Meters
const oneMin = seconds(60);  // 타입이 Seconds
```

number타입에도 상표를 붙일 수 있어요

그런데, 산술 연산 이후에는 상표가 없어져서
실제로 사용하기에는 무리가 있어요

```ts
const tenKm = oneKm * 10;  // 타입이 number
const v = oneKm / oneMin;  // 타입이 number
```

여러 단위가 혼합된 많은 수의 숫자가 들어있으면,
숫자의 단위를 문서화하는건 괜찮은 방법일 수 있어요