# 1. any를 구체적으로 변형해서 사용하기
# 2. 함수 안으로 타입 단언문 감추기

이펙티브 타입스크립트

@Leemainsw

# 함수의 매개변수에 ANY 사용하기 - 배열

## Any를 구체적으로 변형해서 사용하기

- 함수 내의 array.length 타입이 체크된다.
- 함수의 반환 타입이 any 대신 number로 추론된다.
- 함수가 호출될 때 매개변수가 배열인지 체크한다.

```
function getLengthBad(array: any) {  // 이렇게 하지 맙시다!
  return array.length;
}

function getLength(array: any[]) {
  return array.length;
}
```

# 함수의 매개변수에 ANY 사용하기 - 배열

## Any를 구체적으로 변형해서 사용하기

- 매개변수에 배열이 아닌 값을 넣었을 경우 아래와 같은 결과를 반환

```
getLengthBad(/123/);  // 오류 없음, undefined를 반환합니다.
getLength(/123/);
          // ~~~~~ 'RegExp' 형식의 인수는
          //       'any[]' 형식의 매개변수에 할당될 수 없습니다.
```

# 함수의 매개변수에 ANY 사용하기 - 객체

## Any를 구체적으로 변형해서 사용하기

- 함수의 매개변수가 객체이지만 값을 알 수 없을 경우 {[key: string]: any}로 선언

```
function hasTwelveLetterKey(o: {[key: string]: any}) {
  for (const key in o) {
    if (key.length === 12) {
      return true;
    }
  }
  return false;
}
```

# 함수의 매개변수에 ANY 사용하기 - 객체

## Any를 구체적으로 변형해서 사용하기

- 모든 비기본형 타입을 포함하는 object로 사용해도 됨
- Object은 객체의 키를 열거할 수 있지만 속성에는 접근할 수 없음

```
function hasTwelveLetterKey(o: object) {
  for (const key in o) {
    if (key.length === 12) {
      console.log(key, o[key]);
                  // ~~~~~~ '{}' 형식에 인덱스 시그니처가 없으므로
                  //        요소에 암시적으로 'any' 형식이 있습니다.
      return true;
    }
  }
  return false;
}
```

# 함수 타입에 any 사용하기

## Any를 구체적으로 변형해서 사용하기

- 함수 타입에서 최소한으로 구체화할 수 있는 세가지 방법

```
type Fn0 = () => any;                 // 매개변수 없이 호출 가능한 모든 함수
type Fn1 = (arg: any) => any;         // 매개변수 1개
type FnN = (...args: any[]) => any;   // 모든 개수의 매개변수
                                      // "Function" 타입과 동일합니다.
```

# 결론

## Any를 구체적으로 변형해서 사용하기

- any를 사용할 때는 정말 모든 값이 허용되어야만 하는지 면밀히 검토해야 한다.
- Any 보다 더 정확하게 모델링할 수 있도록 아래 예시처럼 구체적인 형태를 사용해야 한다.
  - Any []
  - {[id: string]: any}
  - () => any

# 함수 안으로 타입 단언문 감추기

- 함수의 모든 부분을 안전한 타입으로 구현하는 것이 이상적이지만, 불필요한 예외 상황까지 고려해 가며 타입 정보를 힘들게 구성할 필요는 없다.
- 프로젝트 전반에 위험한 타입 단언문이 드러나 있는 것보다, 제대로 타입이 정의된 함수 안으로 타입 단언문을 감추는 것이 더 좋은 설계이다.

# 함수 안으로 타입 단언문 감추기

- 마지막 호출을 캐싱하는 함수를 작성했지만 오류 발생

```
declare function shallowEqual(a: any, b: any): boolean;
function cacheLast<T extends Function>(fn: T): T {
  let lastArgs: any[] | null = null;
  let lastResult: any;
  return function (...args: any[]) {
    // '(...args: any[]) => any' 형식은 'T' 형식에 할당할 수 없습니다.
    if (!lastArgs || !shallowEqual(lastArgs, args)) {
      lastResult = fn(...args);
      lastArgs = args;
    }
    return lastArgs;
  };
}
```

# 함수 안으로 타입 단언문 감추기

- As unknown as T 타입 단언 추가
- 반환 함수와 타입 T와의 관계성을 모르기 때문에 오류가 발생했던 것

```
declare function shallowEqual(a: any, b: any): boolean;
function cacheLast<T extends Function>(fn: T): T {
  let lastArgs: any[] | null = null;
  let lastResult: any;
  return function (...args: any[]) {
    if (!lastArgs || !shallowEqual(lastArgs, args)) {
      lastResult = fn(...args);
      lastArgs = args;
    }
    return lastArgs;
  } as unknown as T;
}
```

# 함수 안으로 타입 단언문 감추기

- 두 가지의 문제점
  - 함수를 연속으로 호출하는 경우 this의 값이 동일한지 체크하지 않는다.
  - 원본 함수가 객체처럼 속성 값을 가지고 있었다면 래퍼 함수에는 속성 값이 없기 때문에 타입이 달라진다.

# 함수 안으로 타입 단언문 감추기

- If 구분의 k in b 체크로 b 객체에 k 속성이 있다는 것을 확인했지만 b[k] 부분에서 오류가 발생

```
declare function shallowEqual(a: any, b: any): boolean;
function shallowObjectEqual<T extends object>(a: T, b: T): boolean {
  for (const [k, aVal] of Object.entries(a)) {
    if (!(k in b) || aVal !== b[k]) {
                          // ~~~~ '{}' 형식에 인덱스 시그니처가 없으므로
                          //      요소에 암시적으로 'any' 형식이 있습니다.
      return false;
    }
  }
  return Object.keys(a).length === Object.keys(b).length;
}
```

# 함수 안으로 타입 단언문 감추기

- 실제 오류가 아니라는 것을 인지하고 있기 때문에 any로 단언
- b as any 타입 단언문은 안전

```
function shallowObjectEqual<T extends object>(a: T, b: T): boolean {
  for (const [k, aVal] of Object.entries(a)) {
    if (!(k in b) || aVal !== (b as any)[k]) {
      return false;
    }
  }
  return Object.keys(a).length === Object.keys(b).length;
}
```

# 결론

## 함수 안으로 타입 단언문 감추기

- 타입 선언문은 일반적으로 타입을 위험하게 만들지만 상황에 따라 필요할 수 있기 때문에 불가피하게 사용해야 한다면 정확한 정의를 가지는 함수 안으로 숨겨야 한다.