타입이 값들의 집합이라고 생각하기

Item 7

ryan-dia (강철원)

타입 : 할당 가능한 값들의 집합

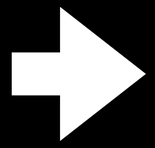
- 공집합 : never

- 한 가지 값만 포함하는 타입 : 리터럴 타입

```ts
type A = 'A';
type B = 'B';
type Tweleve = 12;
```

- 두 개 혹은 세 개이상 : 유니온 타입 (합집합)

```ts
type AB = "A" | "B";
type AB12 = "A" | "B" | 12;
```



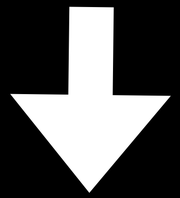
```ts
const a: AB = "A";
const c: AB = "C";
```

교집합 : &(앰퍼샌드)

```ts
interface Person {
  name: string;
}

interface Lifespan {
  birth: Date;
  Death?: Date;
}
type PersonSpan = Person & Lifespan;
```

```ts
  const ps: PersonSpan = {
  name: 'Jun',
  birth: new Date('1912/06/23'),
  Death: new Date('1954/06/07'),
};
```



일반적인 방법

```ts
interface Person {
  name: string;
}
interface PersonSpan extends Person {
  birth: Date;
  death? : Date;
}
```

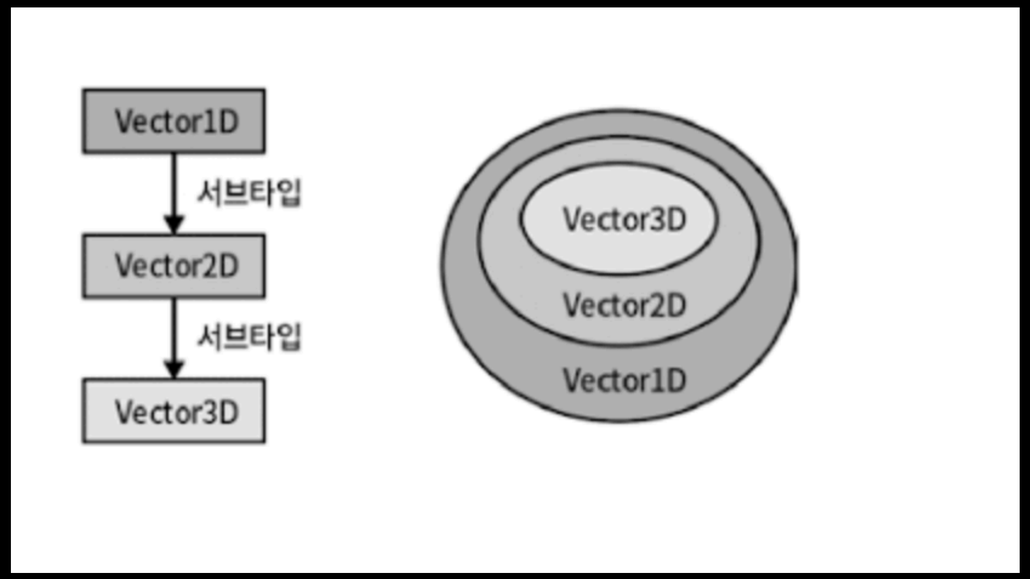
타입이 집합이라는 관점에서 extends의 의미 : "~의 부분 집합"

타입 연산자는 인터페이스의 속성이 아닌, 값의 집합(타입의 범위)에 적용된다.

서브타입

어떤 집합이 다른 집합의 부분집합이다.

```
interface Vector1D {x: number;}
interface Vector2D extends Vector1D { y: number;}
interface Vector3D extends Vector2D { z: number;}
```



Vector3D는 Vector2D의 서브타입이고

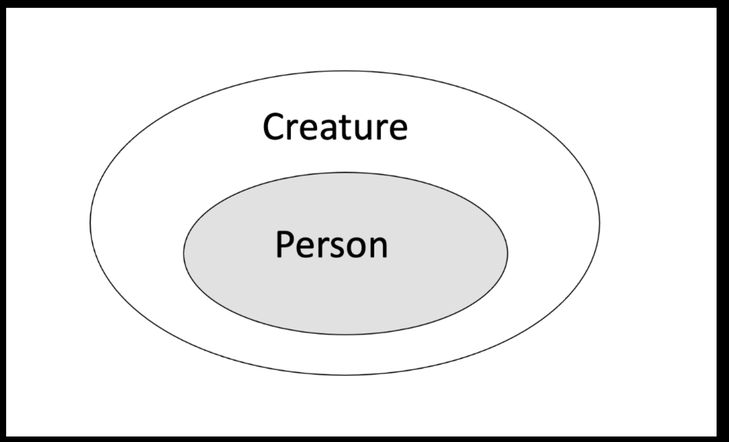
Vector2D는 Vector1D의 서브타입이다.

```ts
interface Creature {
  name: string;
  birth: number;
  gender: "M" | "F";
}

interface Person {
  name: string;
  birth: number;
  gender: "M" | "F";
  nationality: string;
}
```

```ts
interface Creature {
  name: string;
  birth: number;
  gender: "M" | "F";
}

interface Person {
  name: string;
  birth: number;
  gender: "M" | "F";
  nationality: string;
}
```



Creature는 상위 집합이고 Person은 부분 집합이 된다.
Creature와 Person의 교집합은 Person이 된다.

```ts
interface Creature {
  name: string;
  birth: number;
  gender: "M" | "F";
}

interface Person extends Creature {
  nationality: string;
}
```

타입스크립트 타입이 되지 못하는 집합

`Exclude<Type, ExcludedUnion>`

Constructs a type by excluding from `Type` all union members that are assignable to `ExcludedUnion`.

Example

```
type T0 = Exclude<"a" | "b" | "c", "a">;
```

```
type T0 = "b" | "c"
```

```
type T1 = Exclude<"a" | "b" | "c", "a" | "b">;
```

```
type T1 = "c"
```

```
type T2 = Exclude<string | number | (() => void), Function>;
```

```
type T2 = string | number
```

```
type ABC = "A" | "B" | "C"
type ABCWithoutC = Exclude<ABC, "C">
const abcwc:ABCWithoutC = "C"
```

```
type NumberWithoutZero = Exclude<number, 0>
const nwz: NumberWithoutZero = 0;

type StringWithoutZero = Exclude<string, "zero">
const swz: StringWithoutZero = "zero"
```

적절한 타입스립트 타입일 때만 유효

number 타입이나 string 타입에서 특정 유니온 타입을 제거하는 것은
타입스크립트에서 유효한 타입이라고 여기지 않는다.