Ch28 Number

28.1 Number 생성자 함수

표준 빌트인 객체인 **Number** 객체는 **생성자 함수 객체**다. 따라서 **new 연산자**와 함께 호출하여 **Number 인스턴스**를 생성할 수 있다.
Number 생성자 함수에 인수를 전달하지 않고 new 연산자와 함께 호출하면 `[[NumberData]]` 내부 슬롯에 0을 할당한 **Number 래퍼 객체**를 생성한다.

Number 생성자 함수의 인수로 숫자를 전달하면서 new 연산자와 함께 호출하면 `[[NumberData]]` 내부 슬롯에 인수로 전달받은 숫자를 할당한 **Number 래퍼 객체**를 생성한다.

→ 생성자 함수의 인수로 숫자가 아닌 값을 전달하면 인수를 숫자로 강제 변환하고 숫자로 변환할 수 없다면 **NaN**을 `[[NumberData]]` 내부 슬롯에 할당한 Number 래퍼 객체를 생성한다.

new 연산자를 사용하지 않고 Number 생성자 함수를 호출하면 Number 인스턴스가 아닌 숫자를 반환한다. (명시적으로 타입을 변환함)

28.2 Number 프로퍼티

1. `Number.EPSILON`
   - 1과 1보다 큰 숫자 중에서 가장 작은 숫자와의 차이와 같다.
   - 부동소수점으로 인해 발생하는 오차를 해결하기 위해 사용

2. `Number.MAX_VALUE`
   - 자바스크립트에서 표현할 수 있는 가장 큰 양수 값
   - `Number.MAX_VALUE` 보다 큰 숫자는 `Infinity`다.

3. `Number.MIN_VALUE`
   - 자바스크립트에서 표현할 수 있는 가장 작은 양수 값
   - `Number.MIN_VALUE` 보다 작은 숫자는 0이다.

4. `Number.MAX_SAFE_INTEGER`
   - 자바스크립트에서 안전하게 표현할 수 있는 가장 큰 정수값

5. `Number.MIN_SAFE_INTEGER`
   - 자바스크립트에서 안전하게 표현할 수 있는 가장 작은 정수값

6. `Number.POSITIVE_INFINITY`
   - 양의 무한대를 나타내는 숫자값 `Infinity`와 같다.

7. `Number.NEGATIVE_INFINITY`
   - 음의 무한대를 나타내는 숫자값 `-Infinity`와 같다.

8. `Number.NaN`
   - 숫자가 아님을 나타내는 숫자값
   - `window.NaN`과 같다.

28.3 Number 메서드

1. `Number.isFinite`
   - 인수로 전달된 숫자값이 **정상적인 유한수** 인지를 판별하여 불리언 값으로 반환한다.
   - 유한수이면 **true** 무한수이면 **false**
   - 인수가 **NaN**이면 언제나 **false**
   - 빌트인 전역 함수 `isFinite`는 암묵적 타입 변환을 사용해 검사를 수행하지만 `Number.isFinite`는 암묵적 타입 변환하지 않으므로 숫자가 아닌 인수가 주어지면 반환값은 언제나 **false**다.

2. `Number.isInteger`
   - 인수로 전달된 숫자값이 **정수**인지 검사한다.
   - 암묵적 타입 변환하지 않음
   - 정수이면 **true** 정수가 아니면 **false**

3. `Number.isNaN`
   - 인수로 전달된 숫자값이 **NaN**인지 검사한다.
   - NaN이면 **true**
   - 빌트인 전역 함수 `isNaN`은 암묵적 타입 변환을 사용해 검사를 수행하지만 `Number.isNaN`은 암묵적 타입 변환하지 않으므로 숫자가 아닌 인수가 주어지면 반환값은 언제나 **false**다.

```
//Number.isNaN은 인수를 숫자로 암묵적 타입 변환하지 않는다.
Number.isNaN(undefined); // -> false

//isNaN은 인수를 숫자로 암묵적 타입 변환한다. undefined는 NaN으로 암묵적 타입 변환된다.
isNaN(undefined); // -> true
```

4. `Number.isSafeInteger`
   - 인수로 전달된 숫자값이 **안전한 정수**인지 검사한다.
   - 안전한 정수값은 `-(2^53 - 1)과 2^53 - 1` 사이의 정수값
   - 암묵적 타입 변환하지 않음

5. `Number.prototype.toExponential`
   - 숫자를 **지수 표기법으로 변환**하여 **문자열**로 반환한다.

     → **지수 표기법**이란 e 앞에 있는 숫자에 10의 n승을 곱하는 형식으로 수를 나타내는 방식
   - 인수로 소주점 이하로 표현할 자릿수를 전달할 수 있다.
   - 그룹 연산자나 공백을 사용해서 프로퍼티 접근 연산자인지 부동 소수점 숫자의 소수 구분 기호인지 표현할 수 있다.

```
(77.1234).toExponential();  // -> "7.71234e+1"
(77.1234).toExponential(4); // -> "7.7123e+1"
(77.1234).toExponential(2); // -> "7.71e+1"
77.toExponential(); // -> SyntaxError: Invalid or unexpected token
77 .toExponential(); // -> "7.7e+1"
```

6. `Number.prototype.toFixed`
   - 숫자를 **반올림하여 문자열**로 반환한다.

- 반올림하는 소수점 이하 자릿수를 나타내는 0~20 사이의 정수값을 인수로 전달할 수 있다.

```
// 소수점 이하 반올림. 인수를 생략하면 기본값 0이 지정된다.
(12345.6789).toFixed(); // -> "12346"
// 소수점 이하 1자리수 유효, 나머지 반올림
(12345.6789).toFixed(1); // -> "12345.7"
// 소수점 이하 2자리수 유효, 나머지 반올림
(12345.6789).toFixed(2); // -> "12345.68"
```

7. `Number.prototype.toPrecision`
   - 인수로 전달받은 **전체 자릿수까지 유효하도록 나머지 자릿수를 반올림**하여 문자열로 반환한다.
   - 인수로 전달받은 전체 자릿수를 표현할 수 없는 경우 **지수 표기법**으로 결과를 반환한다.
   - 전체 자릿수를 나타내는 0~21 사이의 정수값을 인수로 전달할 수 있다.

```
// 전체 자릿수 유효. 인수를 생략하면 기본값 0이 지정된다.
(12345.6789).toPrecision(); // -> "12345.6789"
// 전체 1자리수 유효, 나머지 반올림
(12345.6789).toPrecision(1); // -> "1e+4"
// 전체 2자리수 유효, 나머지 반올림
(12345.6789).toPrecision(2); // -> "1.2e+4"
// 전체 6자리수 유효, 나머지 반올림
(12345.6789).toPrecision(6); // -> "12345.7"
```

8. `Number.prototype.toString`
   - 숫자를 **문자열로 변환**하여 반환한다.
   - 진법을 나타내는 2~36 사이의 정수값을 인수로 전달할 수 있고 인수를 생략하면 기본값은 **10진법**이다.

```
// 인수를 생략하면 10진수 문자열을 반환한다.
(10).toString(); // -> "10"
// 2진수 문자열을 반환한다.
(16).toString(2); // -> "10000"
// 8진수 문자열을 반환한다.
(16).toString(8); // -> "20"
```