Chap24 클로저

▼ 렉시컬 스코프

함수를 어디서 호출하는지가 아니라 어디에 정의 했는지에 따라 상위 스코프를 결정하는 것

▼ 함수 객체의 내부 슬롯 [[Environment]]

렉시컬 스코프가 가능하려면 함수는 자신의 내부 슬롯 [[Environment]]에 **자신이 정의된 환경, 즉 상위 스코프의 참조를 저장**한다.

[[Environmet]]에 저장된 상위 스코프의 참조는 **현재 실행 중인 실행 컨텍스트의 렉시컬 환경**을 가리킨다.

함수 코드 편가 순서

1. 함수 실행 컨텍스트 생성
2. 함수 렉시컬 환경 생성
    a. 함수 환경 레코드 생성
    b. this 바인딩
    c. 외부 렉시컬 환경에 대한 참조 결정

▼ 클로저와 렉시컬 환경

클로저 : 함수와 렉시컬 환경의 조합

즉, 함수가 생성될 당시의 외부 변수를 기억하고 있어 생성 이후에도 계속 접근이 가능한 것을 말한다.

→따라서 외부 함수보다 중첩 함수가 더 오래 유지되는 경우, 중첩 함수는 이미 생명 주기가 종료한 외부 함수의 변수를 참조할 수 있다. 이러한 **중첩 함수**를 **클로저**(`closure`)라고 부를 수 있다.

```
const x = 1;

// ①
function outer() {
  const x = 10;
  const inner = function () { console.log(x); }; // ②
  return inner;
}

// outer 함수를 호출하면 중첩 함수 inner를 반환한다.
// 그리고 outer 함수의 실행 컨텍스트는 실행 컨텍스트 스택에서 팝되어 제
const innerFunc = outer(); // ③
innerFunc(); // ④ 10
```

- outer함수의 실행 컨텍스트는 inner함수를 반환하며 실행 컨텍스트 스택에서 제거되지만 outer 함수의 렉시컬 환경까지 소멸되는 것이 아니기때문에 중첩함수 inner 함수에서 상위스코프의 식별자 x를 사용할 수 있는 것이다. 이러한 것이 클로저이다.

▼ 클로저에 해당X

- 그러나 상위 스코프의 식별자를 중첩함수에서 사용하지 않으면 일반적으로 클로저로 보지 않는다.
- 중첩함수에서 상위 스코프의 식별자를 참조하지만 외부 함수보다 중첩 함수의 생명주기가 짧을 때

▼ 클로저 활용

클로저는 상태를 안전하게 변경하고 유지하기 위해 사용한다. 다시말해 상태가 의도치 않게 변경되지 않도록 상태를 안전하게 **은닉**하고 **특정 함수에게만 상태 변경을 허용**하기 위해 사용한다.

```
// 카운트 상태 변경 함수
const increase = (function () {
  // 카운트 상태 변수
```

```
  let num = 0;

  // 클로저
  return function () {
    // 카운트 상태를 1만큼 증가 시킨다.
    return ++num;
  };
}());

console.log(increase()); // 1
console.log(increase()); // 2
console.log(increase()); // 3
```

▼ **캡슐화와 정보 은닉**

캡슐화 : 객체의 특정 프로퍼티나 메서드를 감출 목적으로 사용하기도 하는데 이를 정보 은닉이라 한다.

정보 은닉은 외부에 공개할 필요가 없는 구현의 일부를 외부에 공개되지 않도록 감추어 적절치 못한 접근으로부터 객체의 상태가 변경되는 것을 방지해 **정보를 보호**하고, **객체 간의 상호 의존성**, 즉 **결합도**(`coupling`)를 **낮추는 효과**가 있다.

대부분의 객체지향 프로그래밍 언어는 `public`, `private`, `protected`와 같은 접근 제한자(`access modifier`)를 제공하지만 **자바스크립트는 제공하지 않는다. 즉 모든 프로퍼티와 메서드는 `public`이다.**

▼ 자주 발생하는 실수

전역변수로 선언한는 것