클로저 패턴

closer.html

클로저는 자바스크립트 고유의 개념이 아니며 함수와 그 함수가 선언된 어휘적환경(렉시컬)의 조합이다

이미 생명 주기가 종료한 외부 함수의 변수를 중첩 함수에서 참조하는 것을 클로저라고 부른다.

※클로저는 함수가 함수 안에서 선언이 될 때 그 선언될 시점에 외부 환경을 기억하는 겁니다

```
const x = 1;

function foo() {
  const x = 10;
  // 상위 스코프는 함수 정의 환경(위치)에 따라 결정된다.
  // 함수 호출 위치와 상위 스코프는 아무런 관계가 없다.
  bar();
}
// 함수 bar는 자신의 상위 스코프, 즉 전역 렉시컬 환경을 [[Environment]]에 저장하여 기억한다.
function bar() {
  console.log(x);
}
//결과로 둘다 전역변수x에 할당한 값인 1이 출력됨
foo();
bar();

function outer() {
  const x = 20;
  const inner = function () { console.log(x); };
    //inner함수 반환
  return inner;
}
```

```
// outer 함수를 호출하면 중첩 함수 inner를 반환한다.
// 그리고 outer 함수의 실행 컨텍스트는 실행 컨텍스트 스택에서 팝되어 제거된다.
const innerFunc = outer(); // ① outer함수는 중첩함수 inner를 반환하고 실행 컨텍스트 스택에서 제거된다.
innerFunc(); // ② outer 함수의 지역변수 x의 값인 20이다 이미 실행 컨텍스트 스택에서 제거된 outer 함수의 지역 변수 x가 다시 부활 한 듯이 동작하고 있다.

//외부 함수보다 중첩 함수가 더 오래 유지되는 경우 중첩 함수는 이미 생명 주기가 종료한 외부 함수의 변수를 참조할 수 있다. 이러한 중첩 함수를 클로저라고 부른다.
```

▼ 연습 문제(클로저)

콘솔에 출력 되는 결과 값을 맞춰 보세요

```
function foo() {
  const x = 1;
  const y = 2;

    // 중첩 함수 bar는 외부 함수보다 더 오래 유지되며 상위 스코프의 식별자를 참조한다.
  function bar() {
    console.log(x);
  }
  return bar;
}

const bar = foo();
bar(); //출력되는 결과 값을 맞춰보세요
```

▼ 결과

bar를 호출하면 bar에 할당한 foo()함수 의 리턴값이 호출됩니다.

foo함수는 내부의 bar함수를 리턴하므로 bar함수 내부의 console.log(x)가 출력되어 결과값 1을 반환합니다.

따라서 결과 값은 “1”입니다.

클로저의 활용

클로저는 상태를 안전하게 변경하고 유지하기 위해 사용한다.

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

increase.html

```
// 카운트 상태 변수
let num = 0;
// 카운트 상태 변경 함수
const increase = function () {
  // 카운트 상태를 1만큼 증가 시킨다.
  return ++num;
};
console.log(increase()); // 1
console.log(increase()); // 2
console.log(increase()); // 3

//현재 카운트 상태는 전역 변수를 통해 관리되고 있기 때문에 언제든지 누구나 접근할 수 있고 변경할 수 있다
//위 코드는 오류를 발생시킬 가능성을 내포하고 있는 좋지 않은 코드다
```

함수형 프로그래밍에서 클로저를 활용하는 예제

```
// 함수를 반환하는 고차 함수
// 이 함수는 카운트 상태를 유지하기 위한 자유 변수 counter를 기억하는 클로저를 반환한다.
const counter = (function () {
  // 카운트 상태를 유지하기 위한 자유 변수
  let counter = 0;

  // 함수를 인수로 전달받는 클로저를 반환
```

```
  return function (aux) {
    // 인수로 전달 받은 보조 함수에 상태 변경을 위임한다.
    counter = aux(counter);
    return counter;
  };
}());

// 보조 함수
function increase(n) {
  return ++n;
}
// 보조 함수
function decrease(n) {
  return --n;
}
// 보조 함수를 전달하여 호출
console.log(counter(increase)); // 1
console.log(counter(increase)); // 2
// 자유 변수를 공유한다.
console.log(counter(decrease)); // 1
console.log(counter(decrease)); // 0
```

캡슐화

캡슐화는 객체의 상태를 나타내는 프로퍼티와 프로퍼티를 참조하고 조작할 수 있는 동작인 메서드를 하나로 묶는 것을 말한다.

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

capsule.html

```
const Person = (function () {
  let _age = 0; // private

  // 생성자 함수
```

```
  function Person(name, age) {
    this.name = name; // public
    _age = age;
  }

  // 프로토타입 메서드
  Person.prototype.sayHi = function () {
    console.log(`Hi! My name is ${this.name}. I am ${_age}.
`);
  };
    // 생성자 함수를 반환
  return Person;
}());

const me = new Person('Lee', 20);
me.sayHi(); // Hi! My name is Lee. I am 20.
console.log(me.name); // Lee
console.log(me._age); // undefined

const you = new Person('Kim', 30);
you.sayHi(); // Hi! My name is Kim. I am 30.
console.log(you.name); // Kim
console.log(you._age); // undefined

me.sayHi(); // Hi! My name is Lee. I am 30.

//Person 생성자 함수가 여러 개의 인스턴스를 생성할 경우 다음과 같이 _a
ge변수의 상태가 유지되지 않는다는 문제가 있다.

//이는 Person.prototype.sayHi 메서드가 단 한번 생성되는 클로저이기
때문에 발생하는 현상이다.

//자바스크립트는 정보 은닉을 완전하게 지원하지 않는다.

//2021년도에 클래스에 private 필드를 정의할 수 있는 새로운 표준 사양
이 제안되어 있다.
```