

일급객체 1

2023.07.23

일급객체란?

- 어떤 객체가 다음 조건을 만족하면 일급 객체로 간주한다.
1. 변수나 데이터 구조안에 담을 수 있다.
 2. 파라미터로 전달 할 수 있다.
 3. 반환값으로 사용할 수 있다.
 4. 배열이나 객체에 담을 수 있다.

일급함수 & 고차함수

- 일급함수 : 일급객체로 취급되는 함수
- 고차 함수 : 함수를 인자로 전달받거나 함수를 결과로 반환하는 함수를 말한다

Javascript의 경우

- JS에서 함수는 일급으로 취급된다.
- 다음의 것들은 일급이 아니다
 1. 수식 연산자
 2. 반복문
 3. 조건문
 4. Try/catch 블록

개래새 으쭈르그



함수형 프로그래밍에 사용

- 일급이 아닌 것을 함수로 감싸서 일급으로 바꿀 수 있다
- 함수는 함수를 리턴 할 수 있다

-> 위 사실을 사용해서 코드 악취를 줄일 수 있다

함수 이름에 있는 암묵적 인자

암묵적 인자 들어내기

```
1 function setPriceByName(cart, name, price) {
2   var item = cart[name];
3   var newItem = objectSet(item, "price", price);
4   var newCart = objectSet(cart, name, newItem);
5
6   return newCart;
7 }
8
9 function setQuantityByName(cart, name, quantity) {
10  var item = cart[name];
11  var newItem = objectSet(item, "quantity", quantity);
12  var newCart = objectSet(cart, name, newItem);
13
14  return newCart;
15 }
16
17 function setShippingByName(cart, name, shipping) {
18  var item = cart[name];
19  var newItem = objectSet(item, "shipping", shipping);
20  var newCart = objectSet(cart, name, newItem);
21
22  return newCart;
23 }
```

함수 이름에 있는 암묵적 인자

- 함수의 이름이 구현의 차이를 결정한다
- 함수의 동작에 중복되는 부분이 많다

함수 이름에 있는 암묵적 인자

암묵적 인자 들어내기

```
1 function setFieldByName(cart, name, field, value) {  
2   var item = cart[name];  
3   var newItem = objectSet(item, field, value);  
4   var newCart = objectSet(cart, name, newItem);  
5  
6   return newCart;  
7 }  
8  
9 cart = setFieldByName(cart, "apple", "price", 0.99);  
10 cart = setFieldByName(cart, "apple", "quantity", 3);  
11 cart = setFieldByName(cart, "apple", "shipping", 0.2);  
12
```

- 변경할 필드명을 인자로 받게 수정
- 필드명을 일급으로 취급

함수 이름에 있는 암묵적 인자

암묵적 인자 들어내기

```
1  var validItemFields = ["price", "quantity", "shipping"];
2
3  function setFieldByName(cart, name, field, value) {
4      if (validItemFields.indexOf(field) === -1) {
5          throw new Error("Invalid field: " + field);
6      }
7
8      var item = cart[name];
9      var newItem = objectSet(item, field, value);
10     var newCart = objectSet(cart, name, newItem);
11
12     return newCart;
13 }
14
15 cart = setFieldByName(cart, "apple", "price", 0.99);
16 cart = setFieldByName(cart, "apple", "quantity", 3);
17 cart = setFieldByName(cart, "apple", "shipping", 0.2);
```

- 엉뚱한 필드명 인자가 입력될 수 도 있으니 필드명 확인 추가
- 언어에 따라 Enum, 합 타입, Literal Type 등으로 처리 가능

함수 이름에 있는 암묵적 인자

본문을 콜백으로 바꾸기

```
1  function cookAndEatArray(array) {
2      for (let i = 0; i < array.length; i++) {
3          var item = array[i];
4          cook(item);
5          eat(item);
6      }
7  }
8
9  function cleanArray(array) {
10     for (let i = 0; i < array.length; i++) {
11         var item = array[i];
12         wash(item);
13         dry(item);
14         putAway(item);
15     }
16 }
```

- 반복되는 코드 존재
- 함수 이름에 암묵적 인자 존재

함수 이름에 있는 암묵적 인자

본문을 콜백으로 바꾸기

```
1 function forEach(array, f) {
2   for (let i = 0; i < array.length; i++) {
3     var item = array[i];
4     // f is handler function aka callback
5     f(item);
6   }
7 }
8
9 function cookAndEat(food) {
10   cook(food);
11   eat(food);
12 }
13
14 function clean(dish) {
15   wash(dish);
16   dry(dish);
17   putAway(dish);
18 }
19
20 forEach(foods, cookAndEat);
21 forEach(dishes, clean);
```

- 함수 이름에 내포된 정보가 동작(함수)
- 함수를 인자로 받는
고차함수로 리팩토링가능

foreach, map, reduce, filter

함수형 패턴으로 배열 순회

```
1 // ES5
2 const cart = [
3   {
4     name: "apple",
5     type: "fruit",
6     price: 1000,
7     amount: 1,
8   },
9   {
10    name: "banana",
11    type: "fruit",
12    price: 2000,
13    amount: 2,
14  },
15  {
16    name: "pencil",
17    type: "stationery",
18    price: 500,
19    amount: 10,
20  },
21 ];
22
23 const fruitPriceSum = cart
24   .filter((item) => item.type === "fruit")
25   .map((item) => item.price)
26   .reduce((prev, curr) => prev + curr, 0);
```

- Javascript 등 언어에 존재하는 Array method
- Array prototype(class)에 foreach 같은 고차함수들이 이미 구현 되어있음
- 직접 구현해보는 것도 좋은공부?

함수를 리턴하는 함수

함수 감싸기

```
1  try {
2    saveUserData(user);
3  } catch (error) {
4    logToErrorService(error);
5  }
6  // try catch is not a first class citizen
7
8  function wrapLogging(fn) {
9    return function (args) {
10      try {
11        fn(args);
12      } catch (error) {
13        logToErrorService(error);
14      }
15    };
16  }
17
18  const saveUserDataWithLogging = wrapLogging(saveUserData);
```

- try/catch로 error logging
- 고차함수로 코드 반복 줄일 수 있음

React, function as prop

```
1  import React, { ReactElement, ReactNode } from "react";
2  import { PlusIcon } from "lucide-react";
3
4  type IconProps = {};
5
6  type ButtonProps = {
7    children: ReactNode;
8    renderIcon: () => ReactElement<IconProps>;
9  };
10
11  const ButtonWithIcon = ({ children, renderIcon }: ButtonProps) => {
12    const icon = renderIcon();
13
14    return (
15      <button>
16        {icon}
17        {children}
18      </button>
19    );
20  };
21
22  export const Main = () => {
23    return (
24      <ButtonWithIcon renderIcon={() => <PlusIcon />}>button here</ButtonWithIcon>
25    );
26  };
```

React, higher order component

```
22  function withLogging<T>(Component: React.ComponentType<T>) {
23    return function (props: T & React.JSX.IntrinsicAttributes) {
24      useEffect(() => {
25        console.log(`Rendering ${Component.name} with props`, props);
26      }, []);
27      return <Component {...props} />;
28    };
29  }
30
31  const ButtonWithIconAndLogging = withLogging(ButtonWithIcon);
32
33  export const Main = () => {
34    return (
35      <>
36        <ButtonWithIcon renderIcon={() => <PlusIcon />}>
37          button here
38        </ButtonWithIcon>
39        <ButtonWithIconAndLogging renderIcon={() => <PlusIcon />}>
40          button with logging
41        </ButtonWithIconAndLogging>
42      </>
43    );
44  };
```