

# 웹프론트엔드 SPA 개발자 되기 커리큘럼! ver 1.0

제작: 웹동네(<http://www.webdongne.com>), 뚝동네

## 소개

SPA(Single Page Application)이란?

- \* 쉽게 말해 앱 처럼 동작하는 웹 페이지라고 생각하시면 됩니다.
- \* 주로 서버 데이터를 Ajax를 통해 가져와 웹페이지에 동적으로 출력합니다.(전형적인 앱구조이지요)
- \* 사용자는 이게 앱인지 웹인지 구분할 수 없을 만큼 멋지게 동작합니다.
- \* 즉 앱처럼 페이지(또는 화면) 단위로 처리되는 구조를 SPA라고 합니다.

Vue+TypeScript+Node.js+MongoDB 기반 SPA 개발자 되기 커리큘럼

이번 강의 시리즈는 총 8개의 Level로 구성되어 있습니다.  
자세한 사항은 아래 커리큘럼을 참고해주세요.

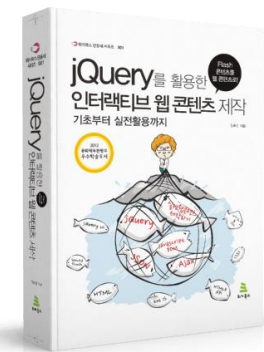
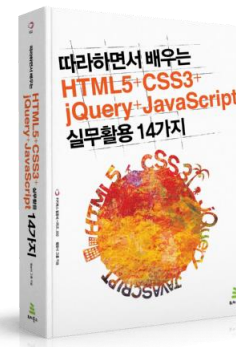
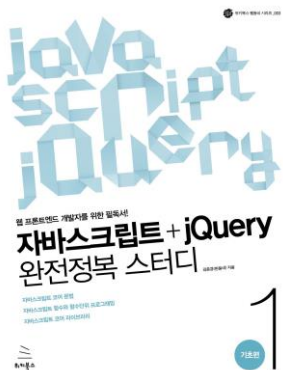
- Level01, 자바스크립트 클래스와 클래스 단위 프로그래밍편
- Level02, ECMAScript 2015 이상 클래스와 클래스 단위 프로그래밍편
- Level03, TypeScript 기초/클래스와 클래스 단위 프로그래밍/모듈편
- Level04, TypeScript 객체지향 프로그래밍(OOP)편
- Level05, Vue.js 기초: Vue CLI + TypeScript + Vue.js 기초편
- Level06, Vue.js 중급: Vue Router + Vuex(Store)편
- Level07, Vue.js 중급: Vue Axios + Ajax편
- Level08, Vue.js 활용: Vue + Node.js +MongoDB 활용한 SPA 제작편

## 01. 강사 소개

- 김춘경(판동네)
- 현) 웹동네 대표 ( <http://www.webdongne.com> )
- 현) 웹프론트엔드 개발자 및 프리랜서 강사
- 네이버 카페 :웹앱을 만드는 사람들의 모임 운영자 ( <http://http://cafe.naver.com/webappdev> )

## 02. 저서

- 자바스크립트+jQuery 완전정복 스터디1-기초편(2015.10, 위키부스)
- 자바스크립트+jQuery 완전정복 스터디2-jQuery 기초와 활용편(2015.10, 위키부스)
- 자바스크립트+jQuery 완전정복 스터디3-중급/고급/활용편(2015.10, 위키부스)
- 따라하면서 배우는 HTML5+CSS3+jQuery+Javascript 실무활용 14가지(2013, 위키북스)
- jQuery를 활용한 인터랙티브 웹 콘텐츠 제작(2012, 위키북스)



# 05부 클래스와 클래스 단위 프로그래밍

## 01장 자바스크립트 클래스 기초

Lesson 01 클래스 소개  
Lesson 02 클래스 관련 기본 개념과 용어 정리  
Lesson 03 오브젝트 리터럴 방식으로 클래스 만들기  
Lesson 04 함수를 활용한 클래스 만들기  
Lesson 05 프로토타입을 활용한 클래스 만들기  
Lesson 06 클래스 정의 방법 3가지 비교  
Lesson 07 미션

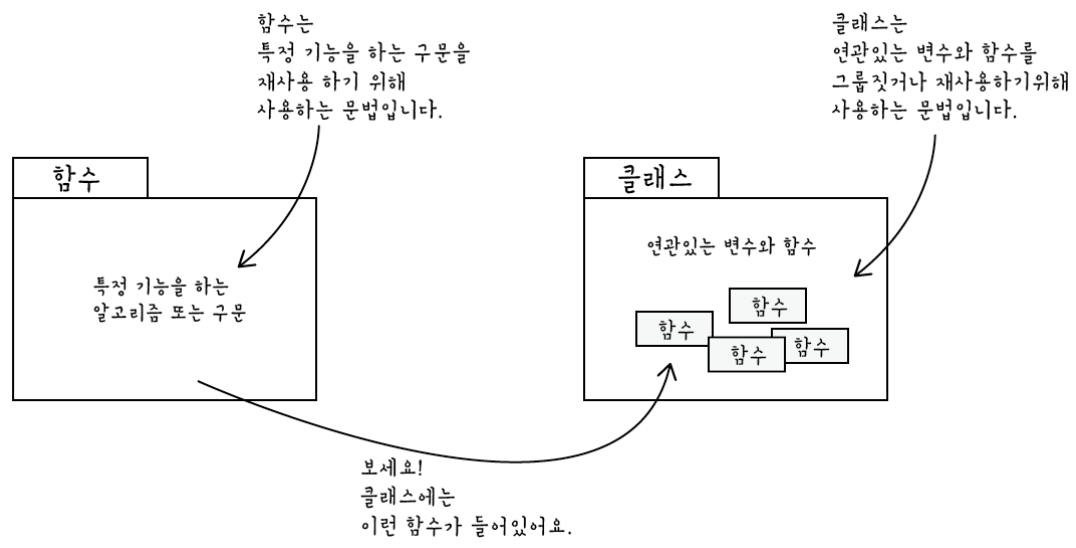
## 01. 클래스를 이해하기 위해 반드시 알고 있어야 하는 내용

1. 함수를 사용하는 이유를 알고 있나요?
2. 지역변수와 전역변수를 이해하고 있나요?
3. 매개변수가 있는 함수를 만들 수 있나요?
4. 리턴 값이 있는 함수를 만들 수 있나요?
5. 중첩 함수를 이해하고 있나요?
6. 콜백 함수를 이해하고 있나요?
7. 클로저를 이해하고 있나요?
8. 함수를 이용해 간단한 탭메뉴를 만들 수 있나요?

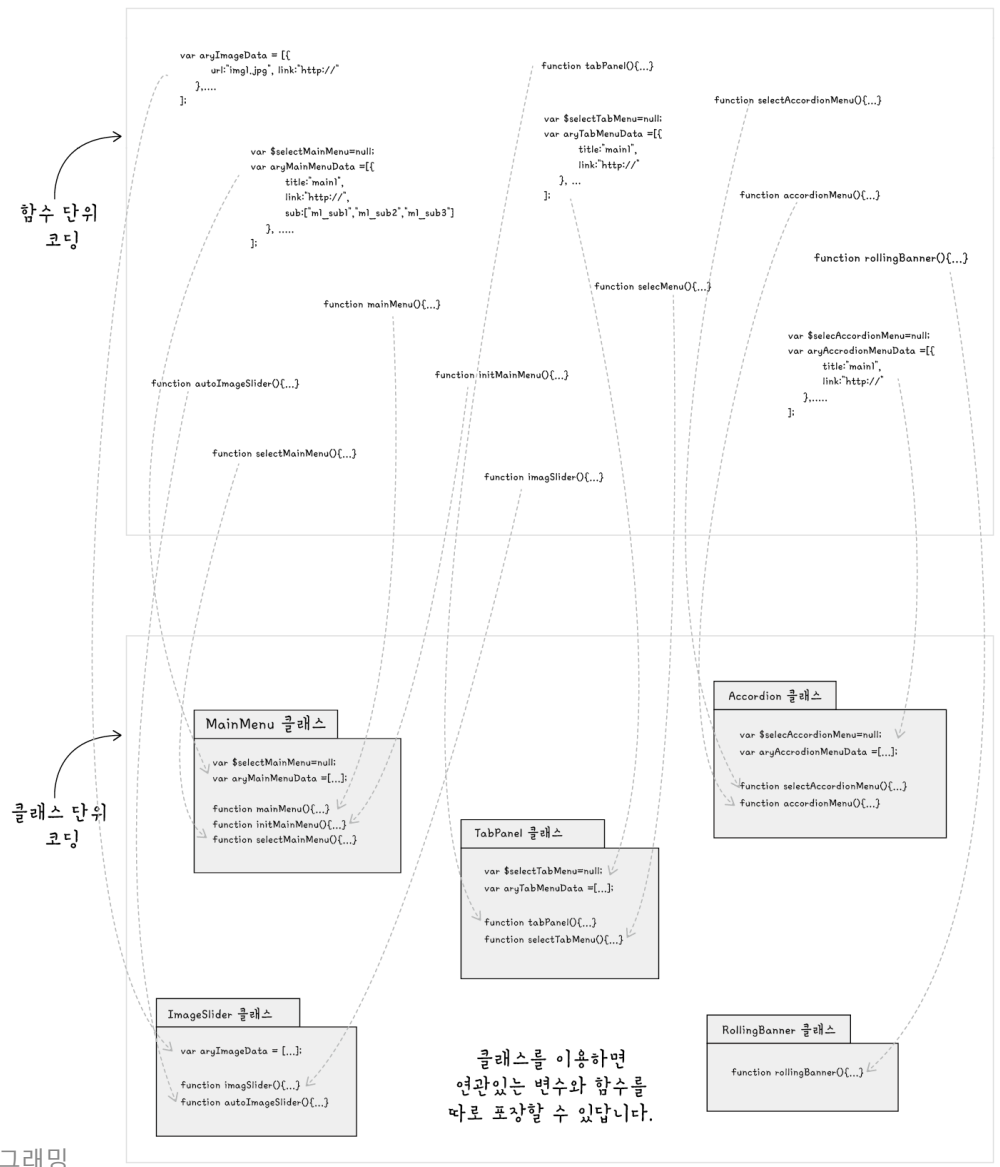
02. 클래스란?

함수가 특정 기능을 하는 구문(알고리즘, 로직)을 묶을 때 사용하는 문법이라면, 클래스는 연관 있는 변수와 함수를 하나로 묶을 때 사용하는 문법입니다.

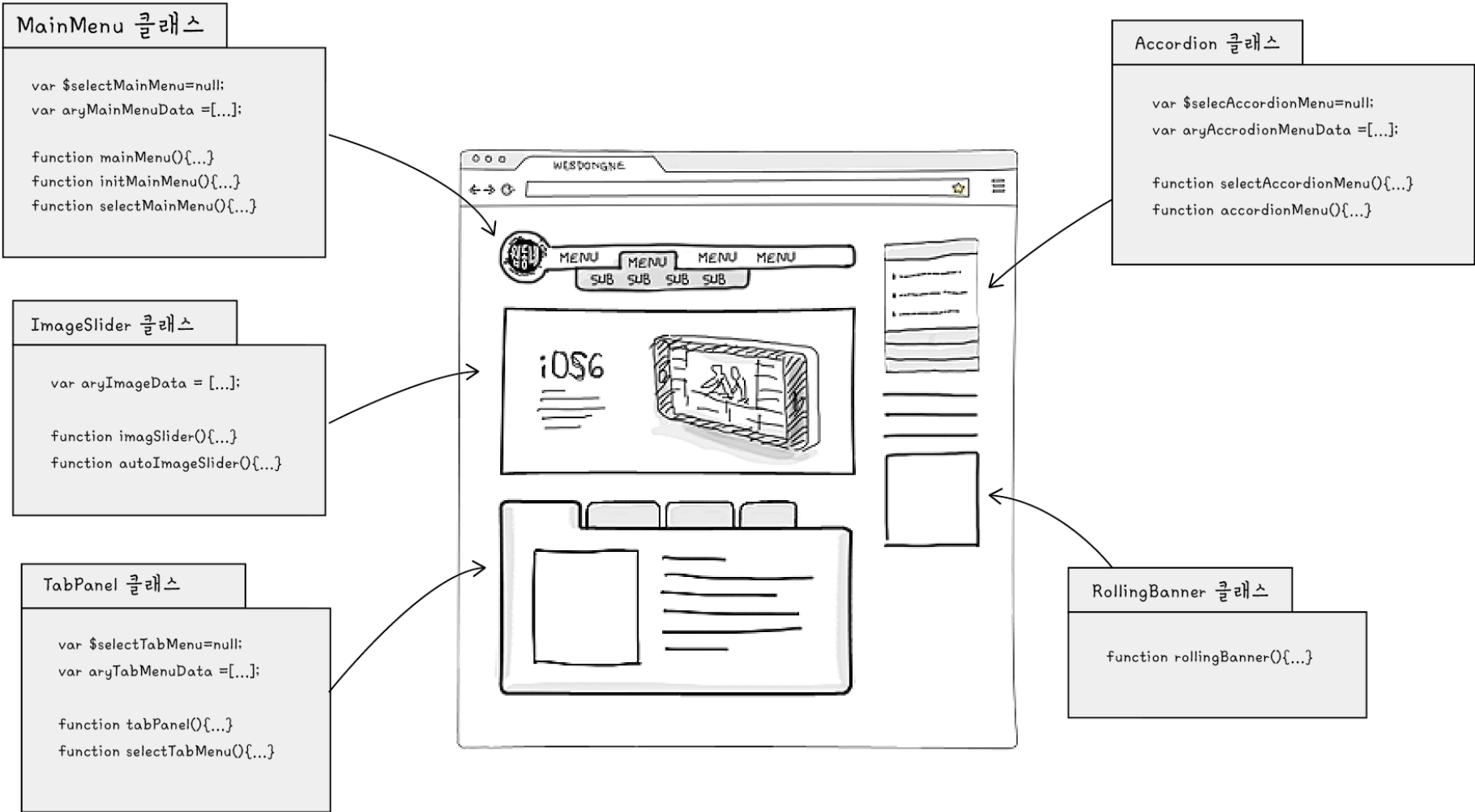
	클래스	함수
포장 내용	연관 있는 변수와 함수	특정 기능을 하는 변수+구문
기능	객체 단위의 코드 그룹화 객체 단위의 중복 코드 제거 및 코드 재사용	기능 단위의 코드 그룹화 기능 단위의 중복 코드 제거 및 코드 재사용



## 1) 객체 단위의 코드 그룹화



# Lesson 01 클래스 소개



03자바스크립트에서 클래스란?

리터럴 방식	함수 방식	프로토타입 방식
<pre>var 인스턴스 = {   프로퍼티1:초기값,   프로퍼티2:초기값,    메서드1:function(){   },   메서드2:function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값;    this.메서드1=function(){   }   this.메서드2=function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값; }  클래스이름.prototype.메서드1=function(){ } 클래스이름.prototype.메서드2=function(){ }</pre>



## 01. 인스턴스

함수를 사용하려면 함수 호출을 해줘야 하듯 클래스를 사용하려면 일반적으로 인스턴스를 생성해줘야 합니다.

붕어빵틀 = 클래스



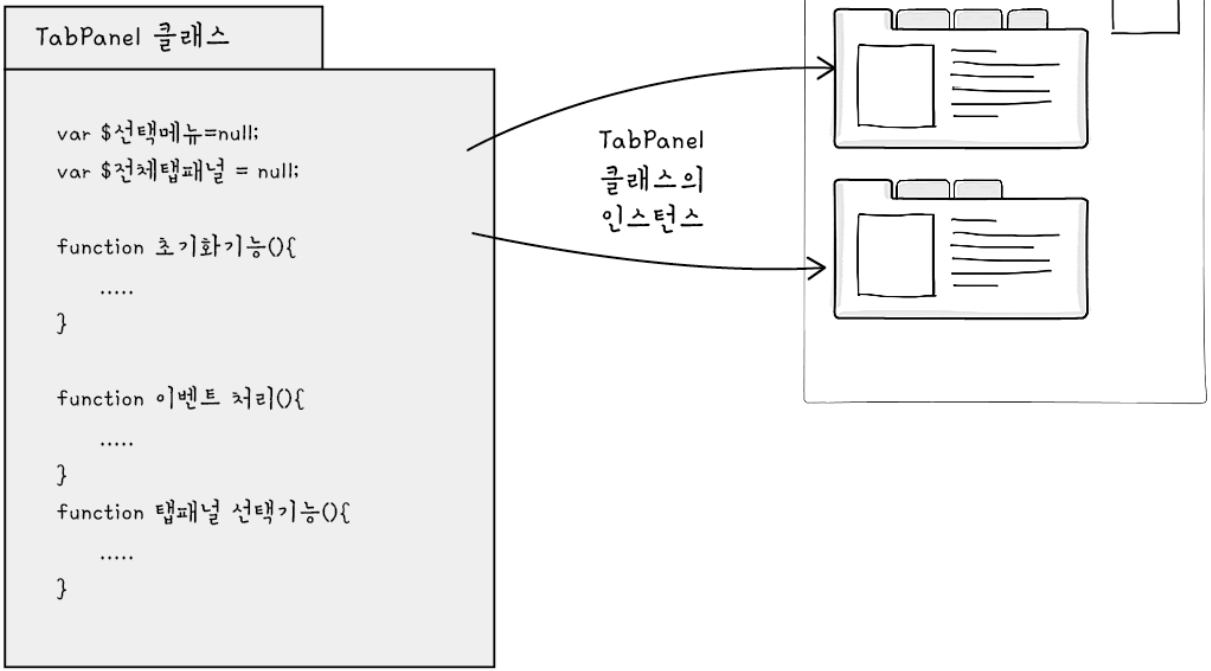
붕어빵 = 인스턴스(또는 객체)



실무예

웹페이지 하나에  
탭패널이 두 개 들어가는 경우

탭패널 클래스 인스턴스를  
두 개 만들어 사용하면 됩니다.



### 02. 객체란?

- 인스턴스의 또 다른 의미이며 클래스의 실체를 나타내는 용어

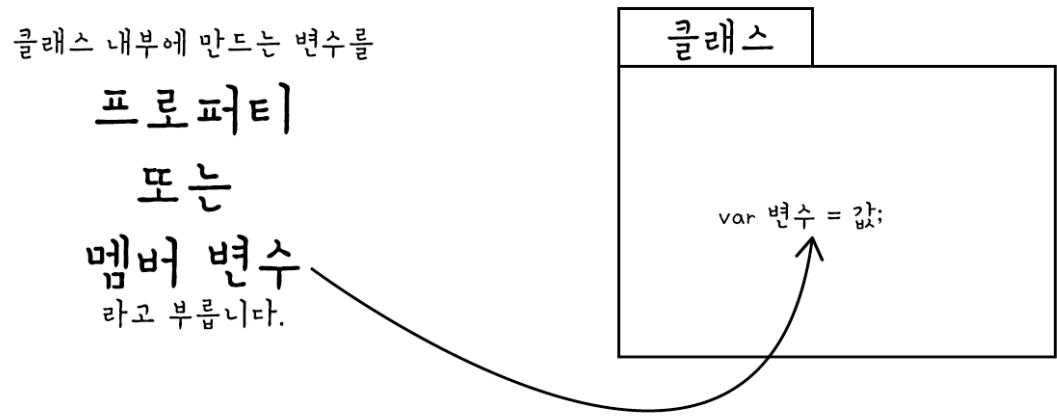
```
var 인스턴스 = new TabPanel();
```

위의 구문을 해석하면 “TabPanel 클래스의 객체 생성”으로 해석함.

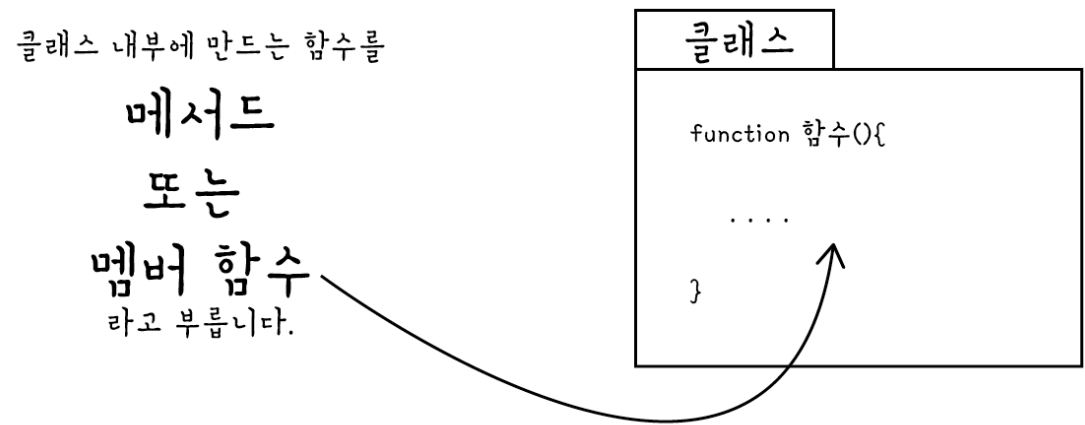
- 주로 인스턴스라는 용어는 new 키워드를 이용해 클래스의 실체를 생성할때 사용

- 객체라는 용어는 인스턴스 생성 후 클래스에서 제공하는 프로퍼티와 메서드를 사용할때 주로 사용.

03. 프로퍼티



04. 메서드



## 01. 사용법

### 1) 문법

```
var 인스턴스 = {  
  프로퍼티: 초깃값,  
  ...  
  메서드:function(){  
    ...  
  },  
  ...  
}
```

### 2) 생성자 정의 방법

생성자는 인스턴스가 만들어지면서 자동으로 호출되는 함수를 말합니다.  
생성자의 주 용도는 프로퍼티 초기화 역할을 담당합니다.  
리터럴 방식에서는 생성자가 존재하지 않습니다.

## 01. 사용법

### 3) 프로퍼티 정의 방법

```
var 인스턴스 = {  
  프로퍼티1:초깃값,  
  프로퍼티2:초깃값,  
  ....  
}
```

### 4) 메서드 정의 방법

```
var 인스턴스 = {  
  프로퍼티1:초깃값,  
  프로퍼티2:초깃값,  
  메서드1:function(){  
  },  
  메서드2:function(){  
  }  
}
```

### 5) 인스턴스 생성 방법

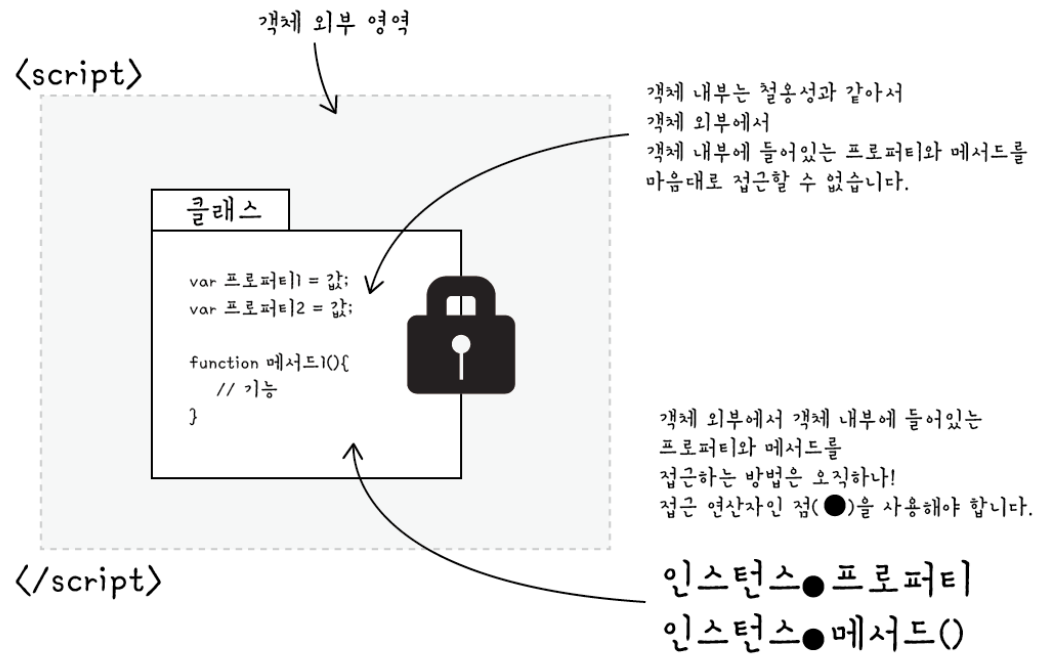
```
var 인스턴스 = {  
  프로퍼티와 메서드 정의  
}
```

01. 사용법

6) 객체 외부에서 프로퍼티와 메서드 접근 방법

```
var 인스턴스 = {
  프로퍼티1:초깃값,
  프로퍼티2:초깃값,
  메서드1:function(){
  },
  메서드2:function(){
  }
}
```

```
인스턴스.프로퍼티;
인스턴스.메서드();
```



## 01. 사용법

### 7) 객체 내부에서 프로퍼티와 메서드 접근 방법

```
var 인스턴스 = {  
  프로퍼티1:초깃값,  
  프로퍼티2:초깃값,  
  메서드1:function(){  
    alert(this.프로퍼티1);  
    this.메서드2();  
  },  
  메서드2:function(){  
  }  
}
```

```
인스턴스.프로퍼티  
인스턴스.메서드()
```

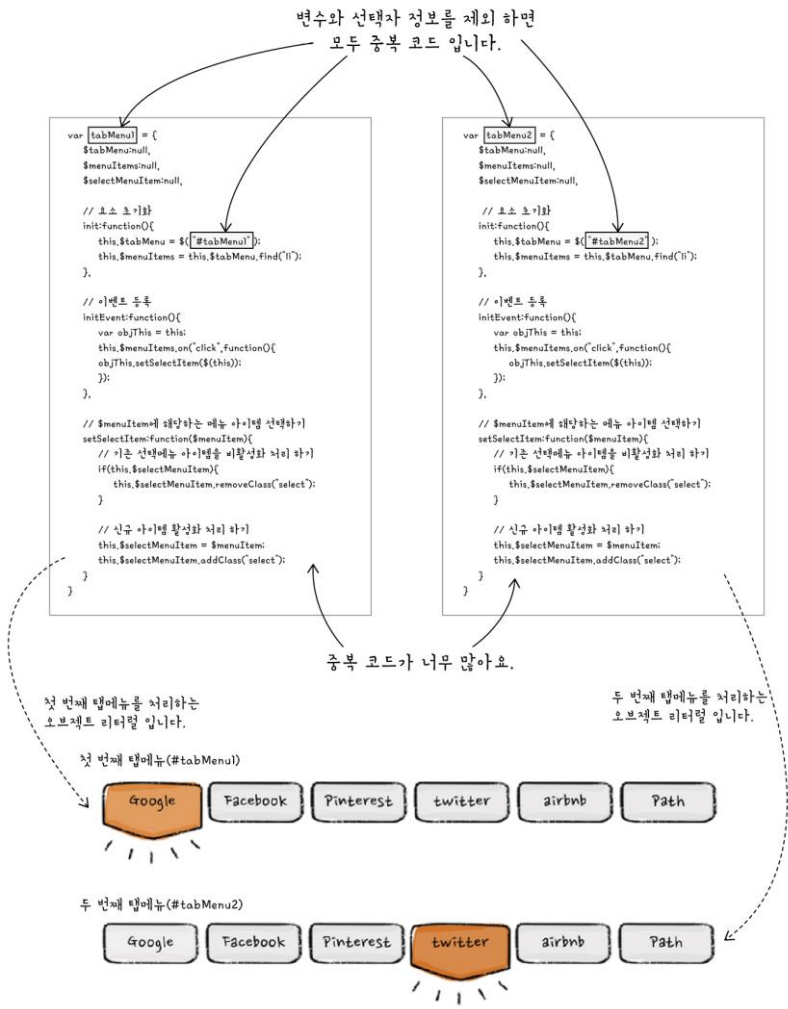
## 02. 예제

함수 단위로 만들어진 탭메뉴를 오브젝트 리터럴 방식 클래스로 만들기



03. 특징

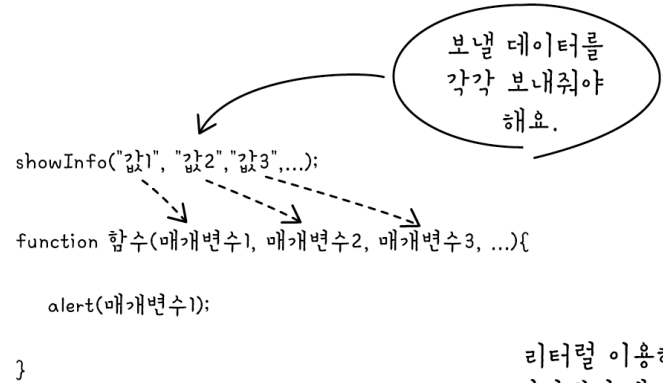
1) 인스턴스를 여러 개 만들 수 없습니다.



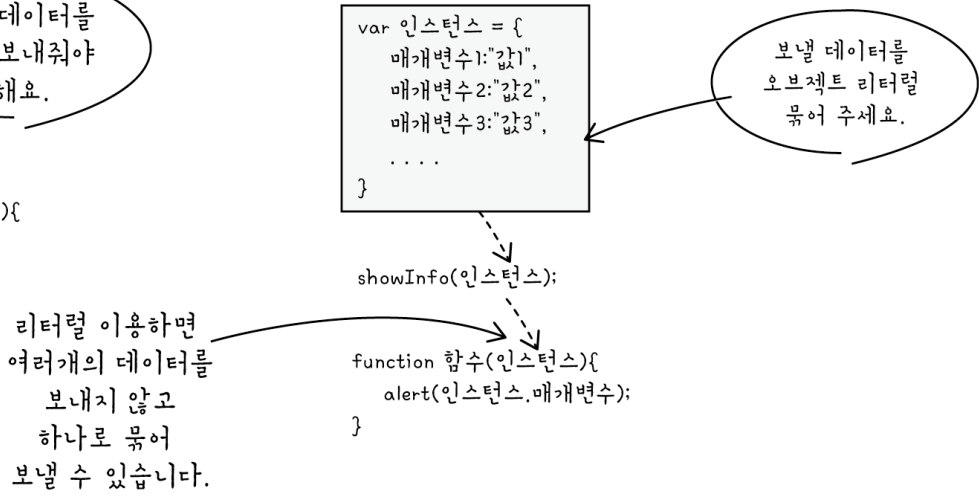
03. 특징

2) 주 용도는 여러 개의 데이터 포장용

오브젝트 리터럴 사용전:



오브젝트 리터럴 사용후:



04. 실무에서 오브젝트 리터럴 사용 예

jQuery에서 스타일 적용시 리터럴 사용하는 예제

## 01. 사용법

### 1) 문법

```
function 클래스이름() {  
    this.프로퍼티1=초깃값;  
    this.프로퍼티2=초깃값;  
    ...  
    this.메서드:function(){  
    }  
    ....  
}  
var 인스턴스= new 클래스이름();
```

### 2) 생성자 정의 방법

```
function 클래스이름(){  
    this.프로퍼티 = 초깃값;  
    this.메서드 = function(){}  
}  
var 인스턴스 = new 클래스이름()
```

## 01. 사용법

### 3) 프로퍼티 정의 방법

```
function 클래스이름(){  
    this.프로퍼티1 = 초깃값;  
    this.프로퍼티2 = 초깃값;  
}
```

### 4) 메서드 정의 방법

```
function 클래스이름(){  
    this.프로퍼티1 = 초깃값;  
    this.프로퍼티2 = 초깃값;  
    this.메서드1=function(){  
    }  
    this.메서드2=function(){  
    }  
}
```

### 5) 인스턴스 생성 방법

```
function 클래스이름(){  
    this.프로퍼티 = 초깃값;  
    this.메서드 = function(){}  
}
```

```
var 인스턴스 = new 클래스이름();
```

## 01. 사용법

### 6) 객체 외부에서 프로퍼티와 메서드 접근 방법

```
function 클래스이름(){  
    this.프로퍼티1 = 초깃값;  
    this.프로퍼티2 = 초깃값;  
    this.메서드1=function(){  
    }  
    this.메서드2=function(){  
    }  
}
```

```
var 인스턴스 = new 클래스이름();  
인스턴스.프로퍼티1;  
인스턴스.메서드1();
```

## 01. 사용법

### 7) 객체 내부에서 프로퍼티와 메서드 접근 방법

```
function 클래스이름(){  
    this.프로퍼티1 = 초깃값;  
    this.프로퍼티2 = 초깃값;  
    this.메서드1=function(){  
        alert(this.프로퍼티1);  
        this.메서드2();  
    }  
    this.메서드2=function(){  
    }  
}  
var 인스턴스 = new 클래스이름()
```

## 02. 예제

함수 단위로 만들어진 탭메뉴를 함수 방식 클래스로 만들기

03. 특징


1) 코드 재사용 가능

```
function TabMenu( selector ){  
  
  this.$tabMenu = null;  
  this.$menuItems = null;  
  this.$selectMenuItem = null;  
  
  // 요소 초기화  
  this.init=function(selector){  
    this.$tabMenu = $(selector);  
    this.$menuItems = this.$tabMenu.find("li");  
  }  
  
  // 이벤트 등록  
  this.initEvent=function(){  
    var objThis = this;  
    this.$menuItems.on("click",function(){  
      objThis.setSelectItem($(this));  
    });  
  }  
  
  // $menuItem에 해당하는 메뉴 아이템 선택하기  
  this.setSelectItem=function($menuItem){  
    // 기존 선택메뉴 아이템을 비활성화 처리하기  
    if(this.$selectMenuItem){  
      this.$selectMenuItem.removeClass("select");  
    }  
  
    // 신규 아이템 활성화 처리하기  
    this.$selectMenuItem = $menuItem;  
    this.$selectMenuItem.addClass("select");  
  }  
  
  // 요소 초기화와 이벤트 등록 호출하기  
  this.init( selector );  
  this.initEvent();  
}
```

TabMenu 클래스  
하나를 이용해  
독립적으로 동작하는 탭메뉴를  
여러 개  
만들 수 있습니다.


```
var tabMenu1 = new TabMenu("#tabMenu1");
```

첫 번째 탭메뉴(#tabMenu1)



```
var tabMenu2 = new TabMenu("#tabMenu2");
```

두 번째 탭메뉴(#tabMenu2)

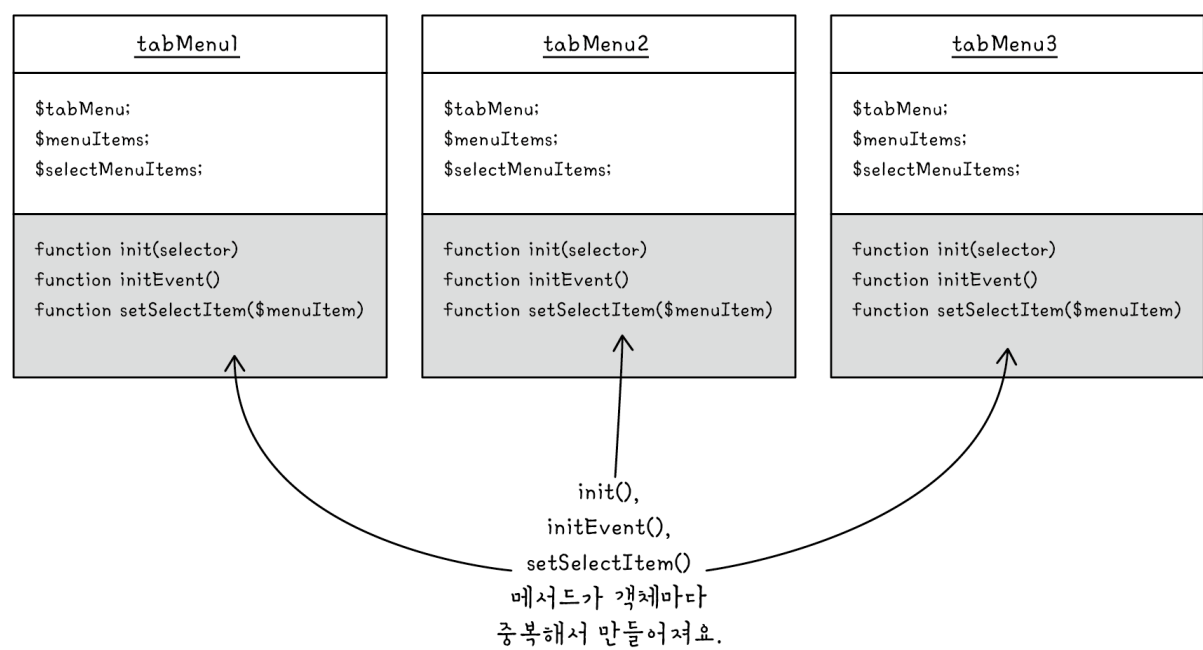


05부 클래스와 클래스 단위 프로그래밍

23

03. 특징

2) 메서드가 중복해서 생성되는 단점



만약 메서드 하나의 크기가 100이고 프로퍼티 크기가 10이라고 했을 때 하나의 탭메뉴 인스턴스 크기는 330이 되고 탭메뉴가 3개이니 총 크기는 990이 됩니다. 탭메뉴의 인스턴스의 개수가 많아질수록 크기는 계속해서 증가합니다. 이런 단점 때문에 실무에서는 함수 방식으로는 클래스를 잘 만들진 않습니다.



## 01. 사용법

### 1) 문법

```
function 클래스이름() {  
  this.프로퍼티1=초깃값;  
  this.프로퍼티2=초깃값;  
  ...  
}
```

```
클래스이름.prototype.메서드=function() {  
  ....  
}
```

### 2) 생성자 정의 방법

```
function 클래스이름(){  
  초기화 작업  
}
```

## 01. 사용법

### 3) 프로퍼티 정의 방법

```
function 클래스이름(){  
  this.프로퍼티1 = 초깃값;  
  this.프로퍼티2 = 초깃값;  
}
```

### 4) 메서드 정의 방법

```
function 클래스이름(){  
  this.프로퍼티1 = 초깃값;  
  this.프로퍼티2 = 초깃값;  
}  
클래스이름.prototype.메서드1=function({});  
클래스이름.prototype.메서드2=function({});
```

### 5) 인스턴스 생성 방법

```
function 클래스이름(){  
  this.프로퍼티 = 초깃값;  
}  
클래스이름.prototype.메서드=function({});  
  
var 인스턴스 = new 클래스이름();
```

## 01. 사용법

### 6) 객체 외부에서 프로퍼티와 메서드 접근 방법

```
function 클래스이름(){  
    this.프로퍼티1 = 초깃값;  
    this.프로퍼티2 = 초깃값;  
}  
클래스이름.prototype.메서드1=function(){};  
클래스이름.prototype.메서드2=function(){};
```

```
var 인스턴스 = new 클래스이름();  
인스턴스.프로퍼티1;  
인스턴스.메서드1();
```

## 01. 사용법

### 7) 객체 내부에서 프로퍼티와 메서드 접근 방법

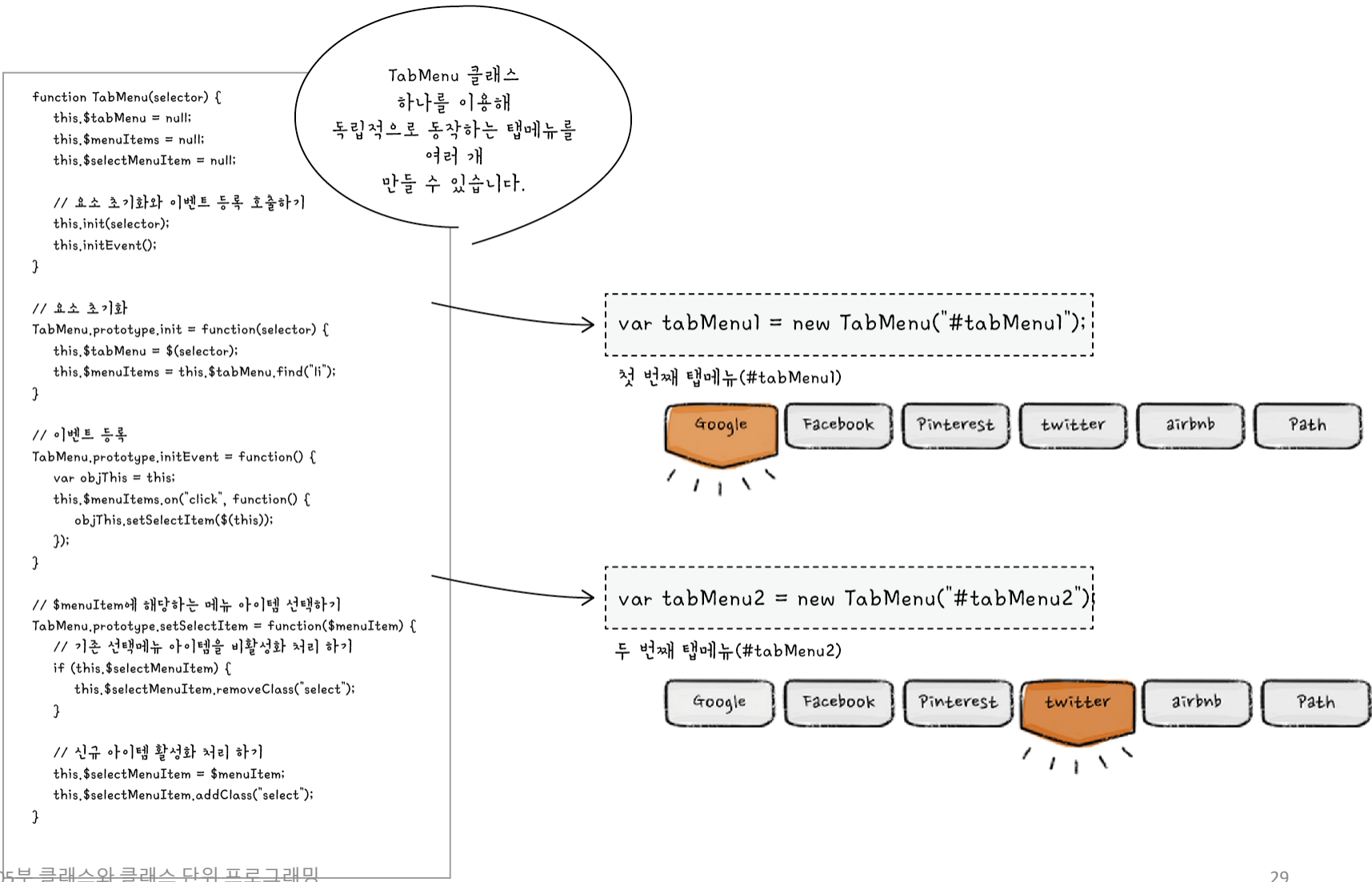
```
function 클래스이름(){  
    this.프로퍼티1 = 초깃값;  
    this.프로퍼티2 = 초깃값;  
}  
클래스이름.prototype.메서드1=function(){  
    alert(this.프로퍼티1);  
    this.메서드2();  
};  
클래스이름.prototype.메서드2=function(){};  
var 인스턴스 = new 클래스이름();
```

## 02. 예제

함수 단위로 만들어진 탭메뉴를 프로토타입 방식 클래스로 만들기

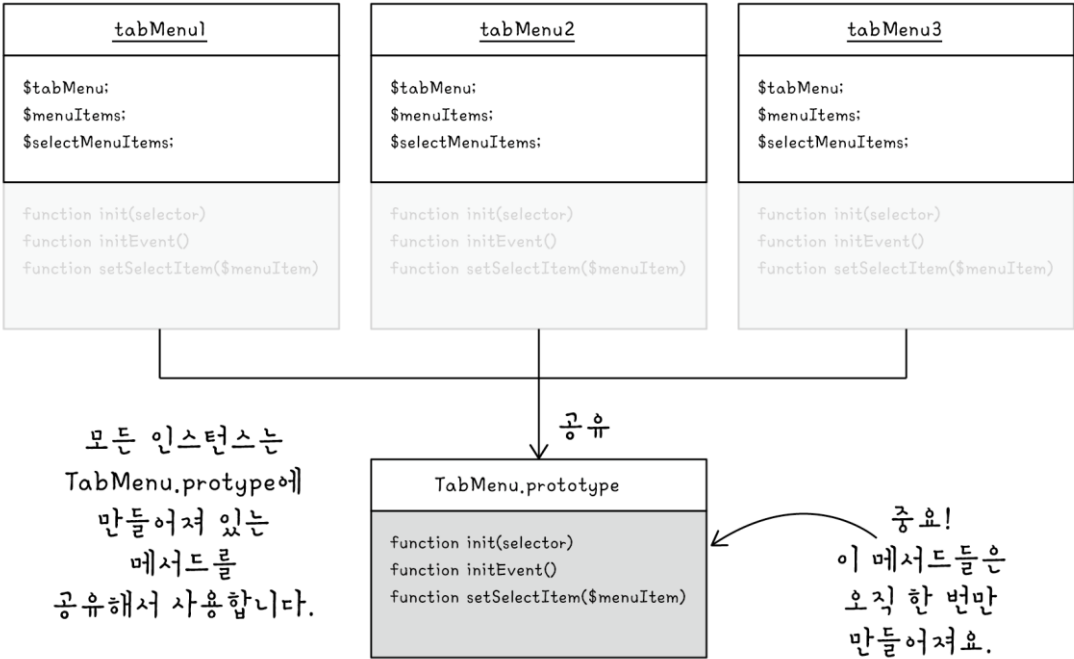
03. 특징

1) 코드 재사용 가능



03. 특징

2) 메서드 공유 기능



만약 메서드 하나의 크기가 100이고 프로퍼티 크기가 10이라고 했을 때 각 인스턴스마다 prototype의 메서드를 공유해서 사용하는 구조이기 때문에, 하나의 탭메뉴 인스턴스 크기는 30이 되고 탭메뉴가 3개이니 총 크기는 390(90+300)이 됩니다.

### 03. 특징

#### 3) 상속 기능

프로토타입 방식의 또 하나의 큰 특징은 자바스크립트에서는 prototype을 이용해 상속을 구현한다는 점입니다. 상속에 대한 내용은 '06부 04장 클래스 상속'편에서 자세히 다룹니다.

01. 특징

방식	특징
프로토타입 방식	<ul style="list-style-type: none"><li>- 일반적인 클래스 제작 방법</li><li>- 인스턴스마다 공통된 메서드를 공유해서 사용하는 장점이 있음</li><li>- jQuery도 prototype 방식으로 만들어져 있음</li></ul>
함수 방식	<ul style="list-style-type: none"><li>- 간단한 클래스 제작 시 사용</li><li>- 인스턴스마다 메서드가 독립적으로 만들어지는 단점이 있음</li></ul>
리터럴 방식	<ul style="list-style-type: none"><li>- 클래스 만드는 용도는 아니며 주로 여러 개의 매개변수를 그룹으로 묶어 함수의 매개변수로 보낼 때 사용</li><li>- 정의와 함께 인스턴스가 만들어지는 장점이 있음. 단! 인스턴스는 오직 하나만 만들 수 있음</li></ul>

02. 클래스 정의 방법(포장법) 비교

리터럴 방식	함수 방식	프로토타입 방식
<pre>var 인스턴스 = {   프로퍼티1:초기값,   프로퍼티2:초기값,    메서드1:function(){   },   메서드2:function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값;    this.메서드1=function(){   }   this.메서드2=function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값; }  클래스이름.prototype.메서드1=function(){ } 클래스이름.prototype.메서드2=function(){ }</pre>



03. 인스턴스 생성 방법

리터럴 방식	함수 방식	프로토타입 방식
<pre>var 인스턴스 = { }</pre>	<pre>var 인스턴스 = new 클래스 이름()</pre>	

04. 객체 외부에서 프로퍼티와 메서드 접근방식

리터럴 방식	함수 방식	프로토타입 방식
<pre>var 인스턴스 = {   프로퍼티1:초기값,   프로퍼티2:초기값,    메서드1:function(){   },   메서드2:function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값;    this.메서드1=function(){   }   this.메서드2=function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값; }  클래스이름.prototype.메서드1=function (){ } 클래스이름.prototype.메서드2=function (){ }</pre>
<pre>인스턴스.프로퍼티1; 인스턴스.메서드1();</pre>		

05. 객체 내부에서 프로퍼티와 메서드 접근 방법

리터럴 방식	함수 방식	프로토타입 방식
<pre>var 인스턴스 = {   프로퍼티1:초기값,   프로퍼티2:초기값,   메서드1:function(){     alert(this.프로퍼티1);     this.메서드2();   },   메서드2:function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값;   this.메서드1=function(){     alert(this.프로퍼티1);     this.메서드2();   }   this.메서드2=function(){   } }</pre>	<pre>function 클래스이름(){   this.프로퍼티1=초기값;   this.프로퍼티2=초기값; }  클래스이름.prototype.메서드1=function( ){   alert(this.프로퍼티1);   this.메서드2(); }  클래스이름.prototype.메서드2=function( ){ }</pre>

미션01-99단 출력을 클래스로 만들기

미션02 계산기를 클래스로 만들기

미션 03: 심플 갤러리를 클래스로 만들기

미션 04: 탭메뉴 기능 추가하기

# 05부 클래스와 클래스 단위 프로그래밍

## 02장 자바스크립트 클래스 중급

Lesson 01 this의 정체

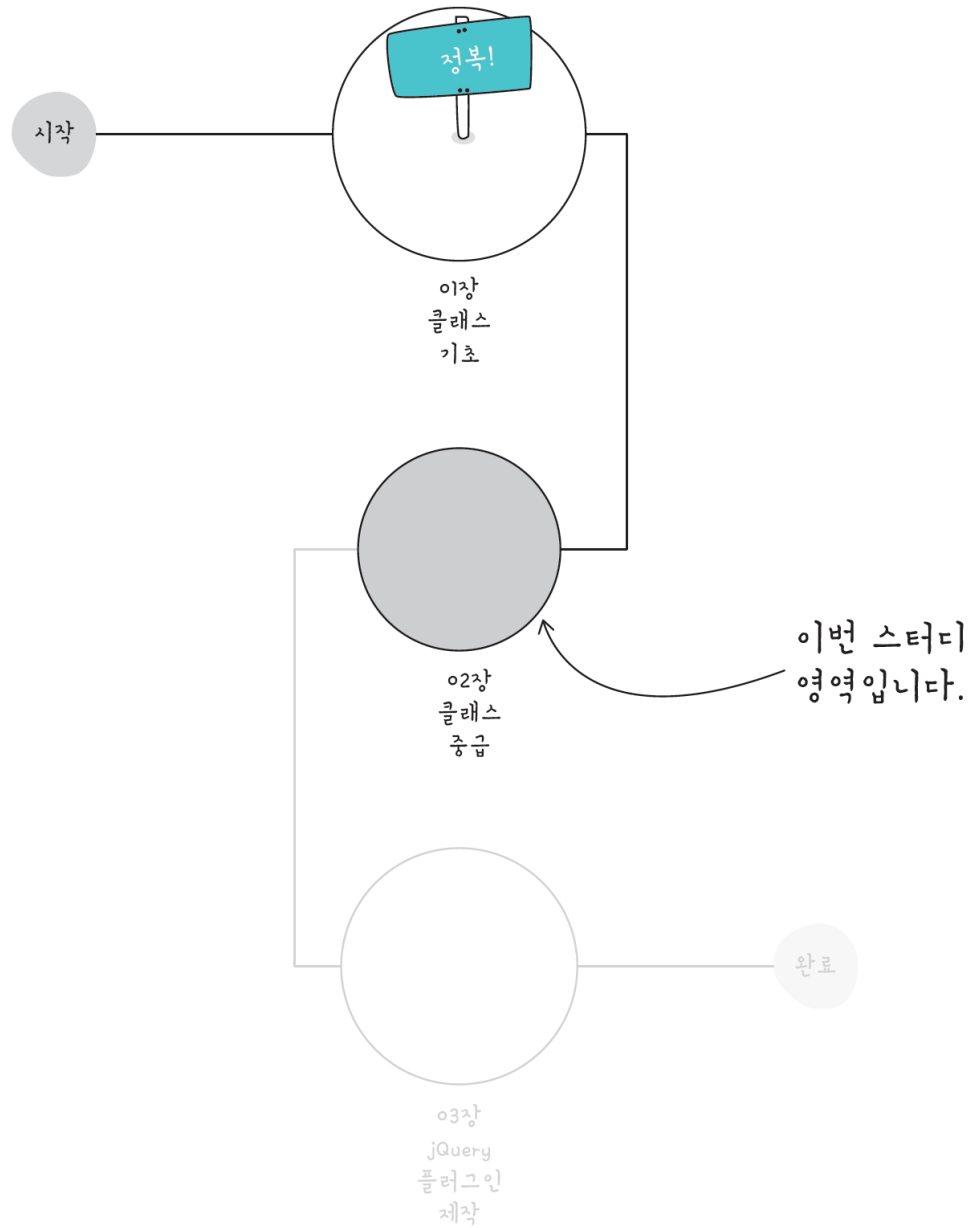
Lesson 02 함수호출() vs. new 함수호출()

Lesson 03 함수 단위 코딩 vs. 클래스 단위 코딩

Lesson 04 인스턴스 프로퍼티와 메서드 vs. 클래스 프로퍼티와 메서드

Lesson 05 패키지

공지:  
원의 크기는 난이도를 나타냅니다.  
앞으로 갈수록 조금씩 어려워지니 차근차근 따라오세요.



## 01. this란?

this는 일반적으로 메서드를 호출한 객체가 저장되어 있는 속성입니다.

```
<script>
function MyClass(){
    this.property1 = "value1";
}
MyClass.prototype.method1=function(){
    alert(this.property1);
}
var my1 = new MyClass();
my1.method1();
</script>
```

상황에 따라 다음과 같이 달라짐

this가 만들어지는 경우	this 값
일반 함수에서 this	window 객체
중첩 함수에서 this	window 객체
이벤트에서 this	이벤트를 발생시킨 객체
메서드에서 this	메서드를 호출한 객체
메서드 내부의 중첩 함수에서 this	window 객체

# Lesson 02 함수호출() vs. new 함수호출()

## 01. 함수호출()

new 없이 함수를 호출하는 경우 일반 함수 호출이 됨, 일반 함수에서의 this는 window

```
var userName = "test";  
  
function User(name){  
    this.userName=name;  
}  
  
// 호출  
  
User("ddandongne");  
  
console.log("uesrName = "+userName);
```

## 02. new 함수호출()

new를 이용해 함수를 호출하는 경우 클래스 인스턴스 생성이 됨, this는 해당 객체가 됨.

```
var userName = "test";  
  
function User(name){  
    this.userName=name;  
}  
  
// 호출  
  
var user = new User("ddandongne");  
  
console.log("1. uesrName = "+userName);
```

## 01. 함수 단위 코딩1

단점1 - tabMenu() 함수를 호출할 때마다 내부에 선언된 중첩 함수가 만들어집니다.

함수 방식 클래스와 동일하게 함수 단위 코딩 역시 함수가 중복해서 만들어지는 단점이 있습니다

단점2 - 외부에서 내부 속성과 함수를 접근할 수 없습니다.

## 02. 함수 단위 코딩2-기능확장법

## 03. 클래스 단위 코딩

단점1 - tabMenu() 함수를 호출할 때마다 내부에 선언된 중첩 함수가 만들어집니다.

해결책: 프로토타입 방식의 경우 여러 개의 인스턴스에서 메서드를 공유해서 사용하기 때문에 단점1을 해결할 수 있습니다.

단점2 - 외부에서 내부 속성과 함수를 접근할 수 없습니다.

해결책 - 함수와 달리 프로퍼티와 메서드는 외부에서 접근할 수 있는 구조로 되어 있기 때문에 언제든지 필요한 메서드를 접근해서 사용할 수 있습니다.



## 01 인스턴스 프로퍼티와 메서드

인스턴스를 생성해야지만 메서드와 프로퍼티를 사용할 수 있는 프로퍼티와 메서드

## 02 클래스 프로퍼티와 메서드

1) 클래스 프로퍼티와 메서드란?  
인스턴스 생성없이 사용할 수 있는 프로퍼티와 메서드

```
2) 만드는 방법
function 클래스이름(){
    ....
}

클래스이름.프로퍼티=값;
클래스이름.메서드이름=function(){}

```

3)사용법

```
클래스이름.프로퍼티;
클래스이름.메서드이름();

```

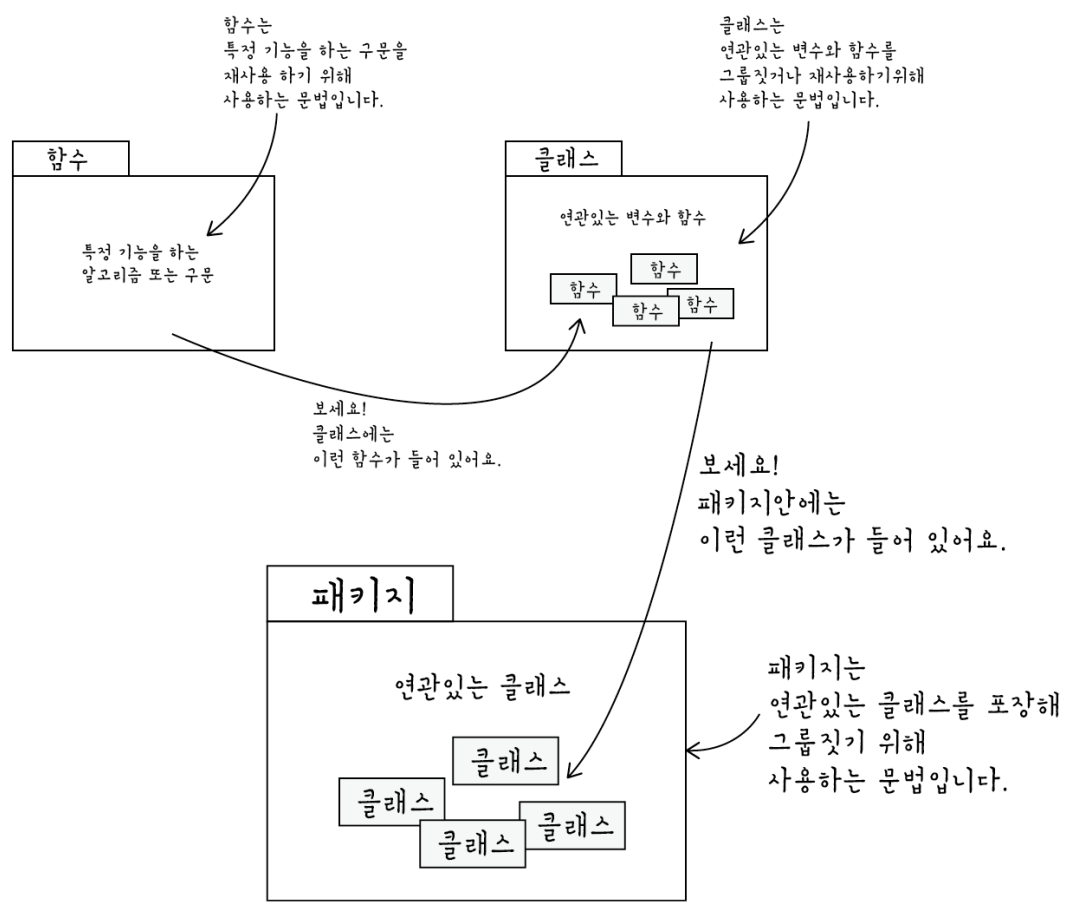
4) 주 용도

주로 도움을 주는 유틸리티성 기능이나 또는 실행하더라도 내부 데이터에 영향을 주지 않고 독립적으로 실행되는 기능이나 정보들을 담을 때 사용

정리

구분	인스턴스 프로퍼티/메서드	클래스 프로퍼티/메서드
작성법	클래스를 만드는 방법 세 가지 중 하나 예) 프로토타입 방식 클래스 <pre>function MyClass(){   this.프로퍼티= 값; } MyClass.prototype.메서드=function(){ }</pre>	<pre>function MyClass(){   this.프로퍼티= 값; } MyClasss.prototype.메서드=function(){ } MyClass.클래스프로퍼티=값; MyClass.클래스메서드=function(){ }</pre>
사용법	<pre>var my1 = new MyClass(); my1.프로퍼티; my1.메서드()</pre>	<pre>MyClass.클래스프로퍼티; MyClass.클래스메서드();</pre>
주용도	객체의 상태 정보를 담거나 다뤄야 하는 경우에 주로 사용	독립적으로 실행 가능한 유틸리티성 정보와 기능을 구현할 때 주로 사용

01. 패키지란?



## 02. 일반 프로그래밍에서 패키지

자바에서는 package라는 명령어를  
이용해 패키지를 구현합니다.

문법

```
package 패키지명;  
public class 클래스이름 {  
    . . . .  
}
```

예)

```
package ddan.utils;  
public class StringUtil {  
    . . . .  
}
```

03. 자바스크립트에서 패키지

문법

```
var packageName = {}  
또는  
var packageName = new Object()  
  
packageName.className = function(){  
}
```

예제

유틸리티 성격의 클래스 그룹

```
var ddan = {}  
ddan.utils = {}  
ddan.utils.String = function()  
ddan.utils.Format = function()  
...  
인스턴스 생성  
var myStr = new ddan.utils.String();
```

UI 컨트롤 클래스 그룹

```
var ddan = {}  
ddan.ui = {}  
ddan.ui.ImageSilder = function()  
ddan.ui.tabMenu = function()  
ddan.ui.tabPanel = function()  
... .
```

# 05부 클래스와 클래스 단위 프로그래밍

## 03장 jQuery 플러그인 제작

Lesson 01 jQuery 확장 소개  
Lesson 02 jQuery 유틸리티 만들기  
Lesson 03 jQuery 플러그인 만들기  
Lesson 04 jQuery 함수 기반 플러그인 만들기  
Lesson 05 jQuery 클래스 기반 플러그인 만들기  
Lesson 06 jQuery 플러그인 그룹 만들기  
Lesson 07 extend() 메서드를 활용한 플러그인 옵션 처리  
Lesson 08 미션

## 01. jQuery 확장이란?

jQuery는 jQuery가 제공하는 기능 이외의 기능이 필요할 경우 다른 개발자가 만들어놓은 기능을 아주 손쉽게 확장해서 사용할 수 있음.

## 02. jQuery 확장 요소 종류

### 1) 유틸리티

```
jQuery.유틸리티()  
또는  
$.유틸리티();
```

### 2) 플러그인

```
$("선택자").플러그인(옵션);  
또는  
var $결과 = $("선택자");  
$결과.플러그인(옵션);
```

## 01. 유틸리티 소개

jQuery 유틸리티는 jQuery 인스턴스를 생성하지 않고 다음과 같이 직접 접근해서 사용합니다.

```
jQuery.유틸리티();  
또는  
$.유틸리티();
```

## 02. 유틸리티 구조

### 1) 문법

```
(function($){  
    $.유틸리티 = function(){  
        // 기능 구현  
    }  
})(jQuery);  
jQuery.유틸리티(); // 사용하기
```

### 2) 사용법

```
$.유틸리티함수();  
또는  
jQuery.유틸리티함수();
```

## 03. 사용자 정의 jQuery 유틸리티 만들기

예제: 3자리수마다 콤마를 추가하는 유틸리티 만들기



01. 플러그인 소개

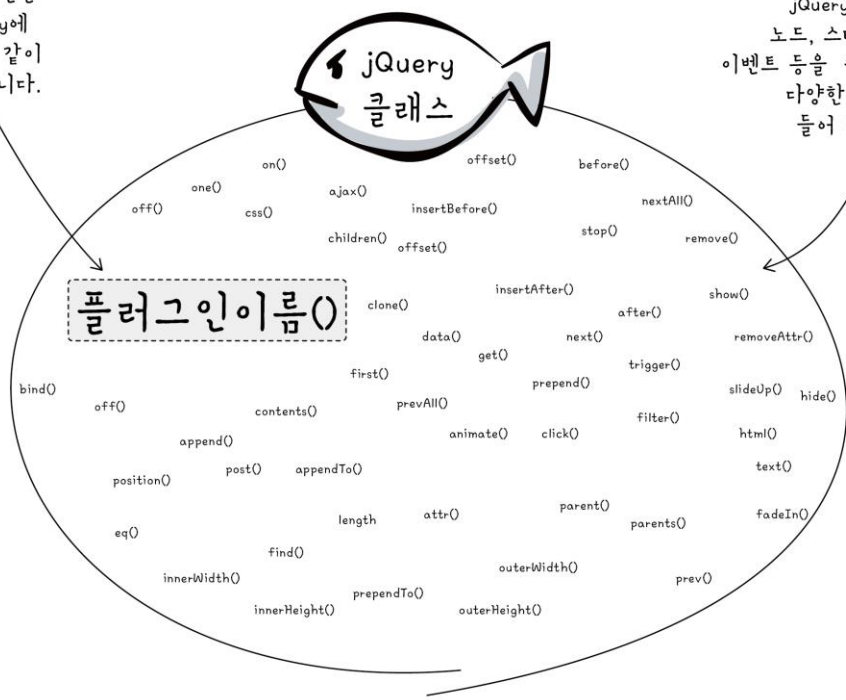
플러그인은 아코디언 메뉴나 탭 메뉴와 같이 특정 기능을 재사용하고자 할 때 사용하는 포장 기능입니다. 쉽게 말해 jQuery 기능 중 jQuery 유틸리티를 제외한 모든 기능은 jQuery 플러그인이라고 생각하면 됩니다.

02. 유틸리티 구조

1) 문법

```
(function($){
  $.fn.플러그인이름 = function(속성값){
    this.each(function(index){
      // 기능 구현
    })
    return this;
  }
})(jQuery)
```

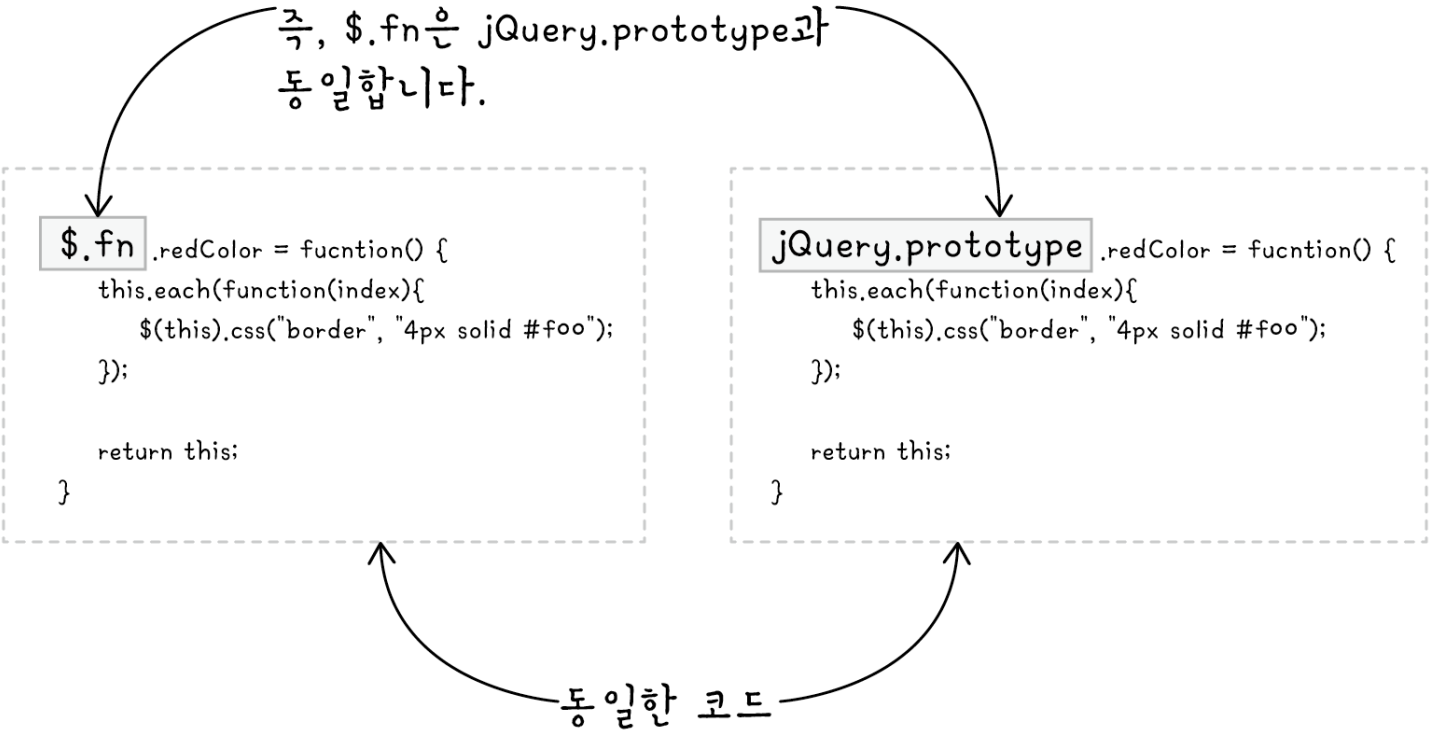
플러그인은  
jQuery에  
다음과 같이  
추가됩니다.



jQuery 객체에는  
노드, 스타일, 속성,  
이벤트 등을 쉽게 다룰 수 있는  
다양한 기능들이  
들어 있습니다.

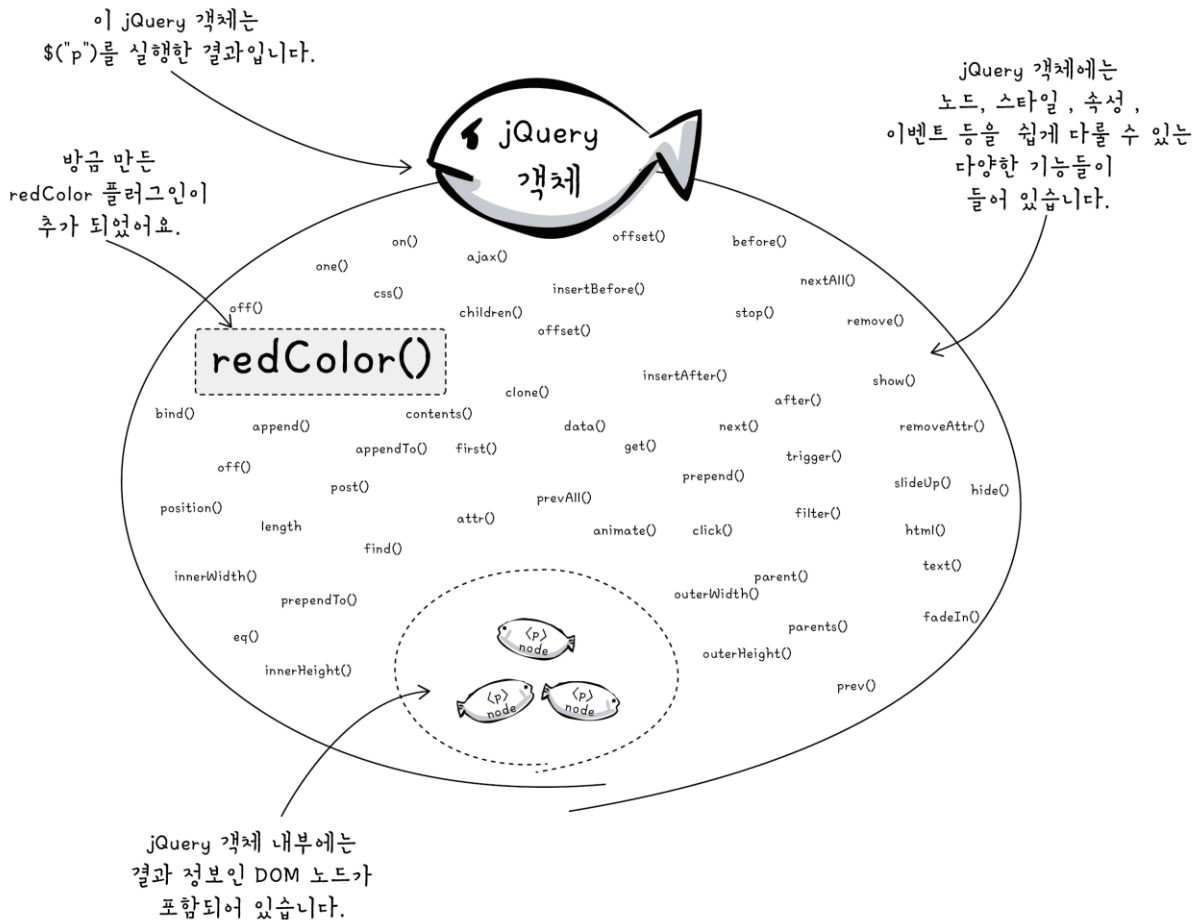
03. jQuery 플러그인 구조 분석

\$는 Query와 동일하며  
fn은 prototype과 동일합니다.  
즉, \$.fn은 jQuery.prototype과  
동일합니다.



03. jQuery 플러그인 구조 분석

```
$(document).ready(function){
  // redColor 플러그인 사용
  $("p").redColor();
}
```



### 04. 사용자 정의 jQuery 플러그인 만들기

예제: 다음 문서에서 li 태그 노드를 찾아 하나씩 차례대로 제거하는 플러그인을 만들어 주세요. 단, 노드 제거 시 노드의 높이를 0으로 서서히 줄이는 애니메이션을 적용한 후 지워주세요.

01. 구문

```
(function($){
  $.fn.플러그인이름 = function(속성값){

    this.each( function(index){

      var 변수1;
      var 변수2;
      ....
      function 함수1(){
      }
      function 함수2(){
      }
      ....

    });

    return this;
  }
})(jQuery)
```

플러그인 구현 코드가 이곳에 위치하게 되요.

02. 예제

01. 구문

```
(function($){
  $.fn.플러그인이름 = function(속성값){

    this.each( function(index){

      var 변수1;
      var 변수2;
      ....
      function 함수1(){
      }
      function 함수2(){
      }
      ....

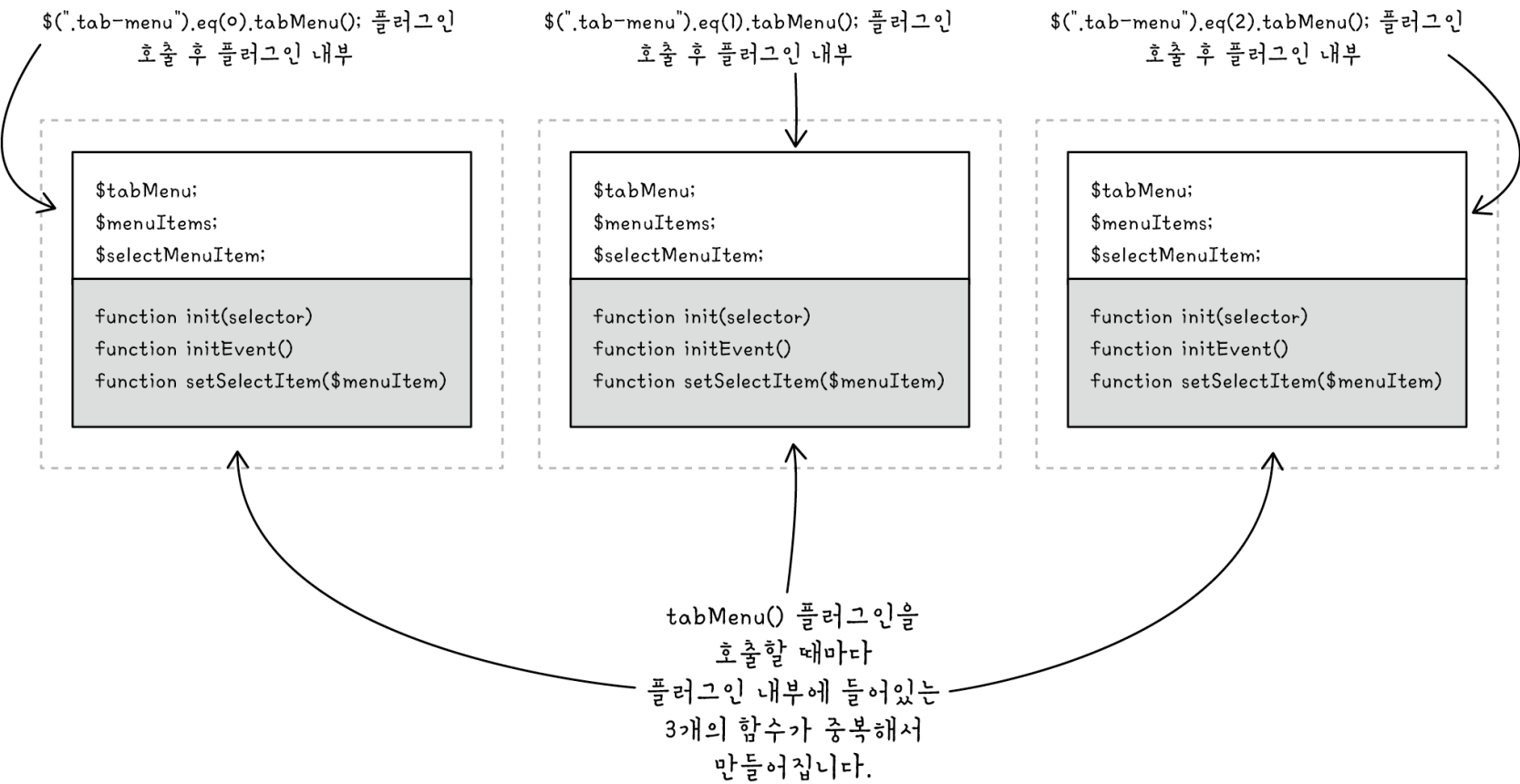
    });

    return this;
  }
})(jQuery)
```

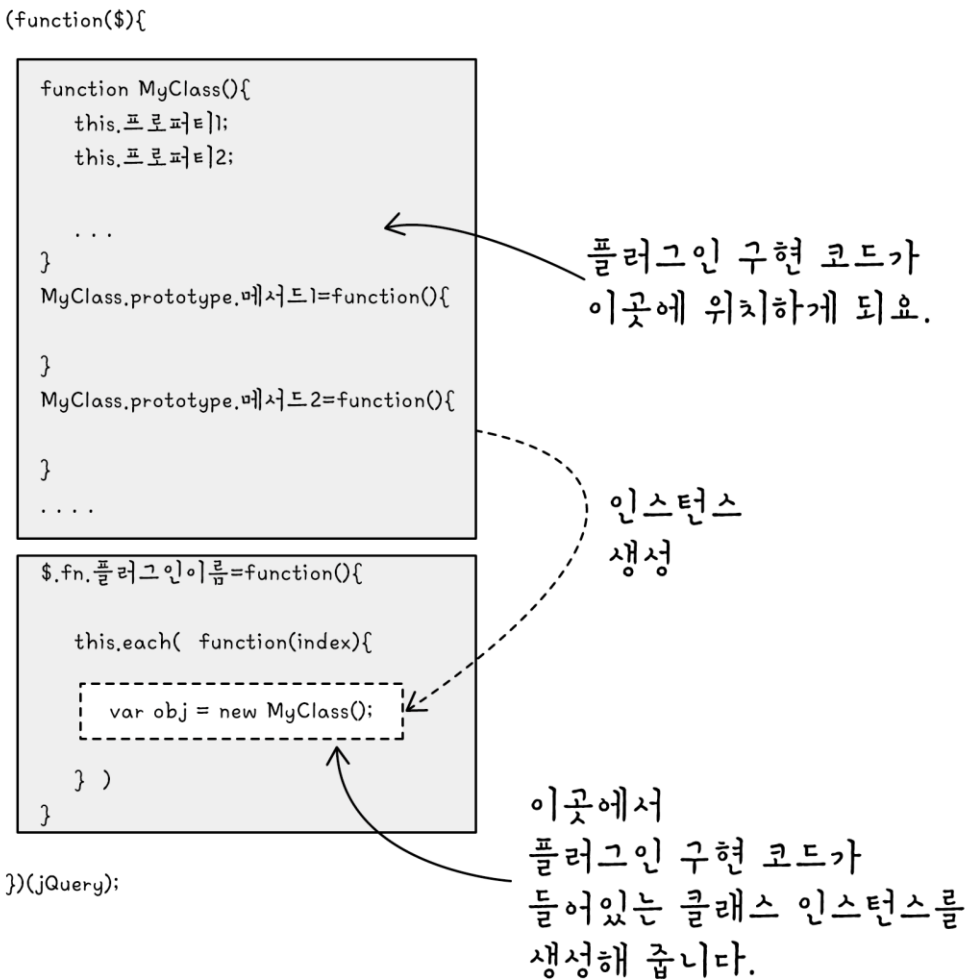
플러그인 구현 코드가  
이곳에 위치하게 되요.

02. 예제

03. 정리



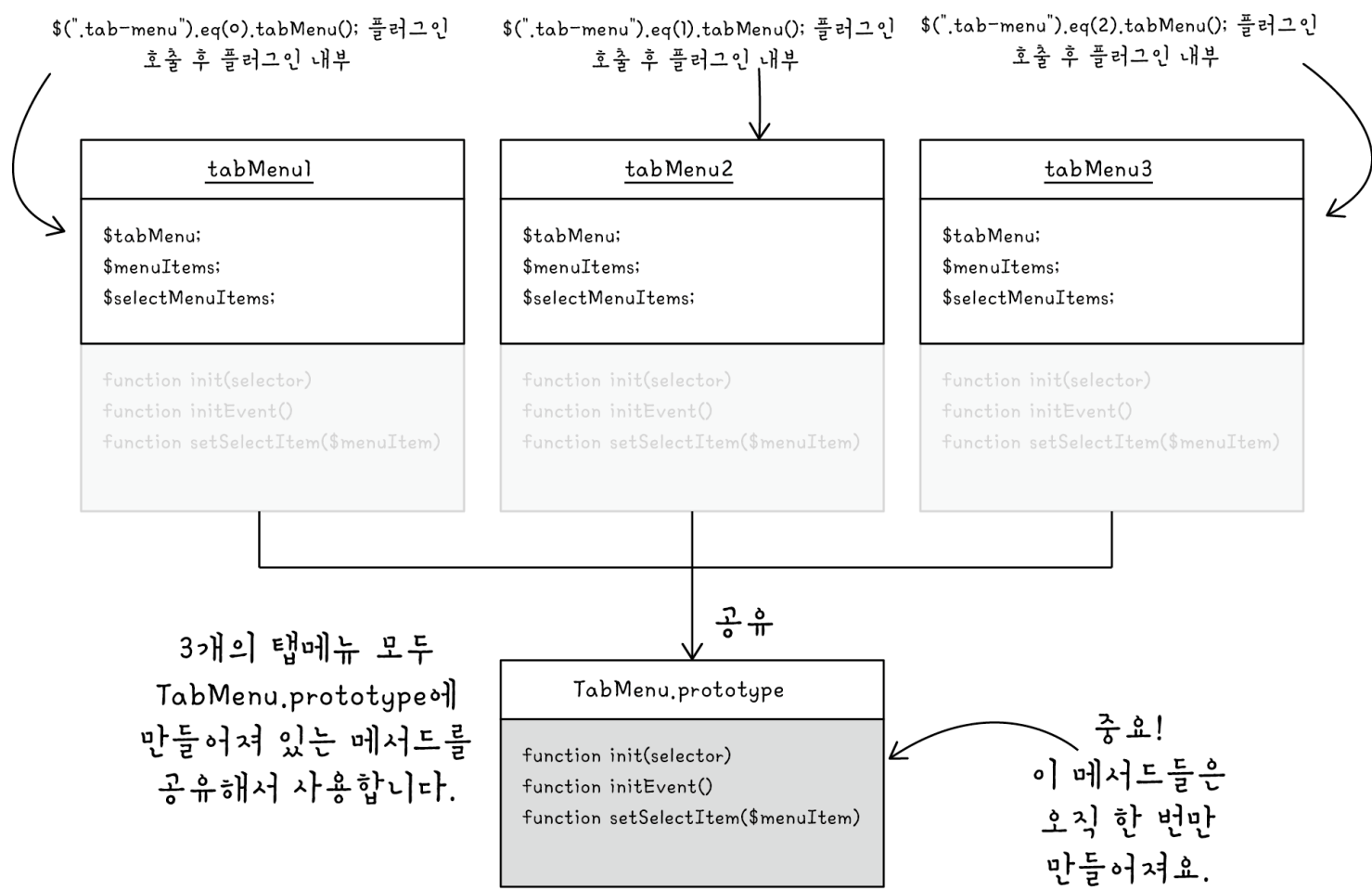
01. 구문



02. 예제

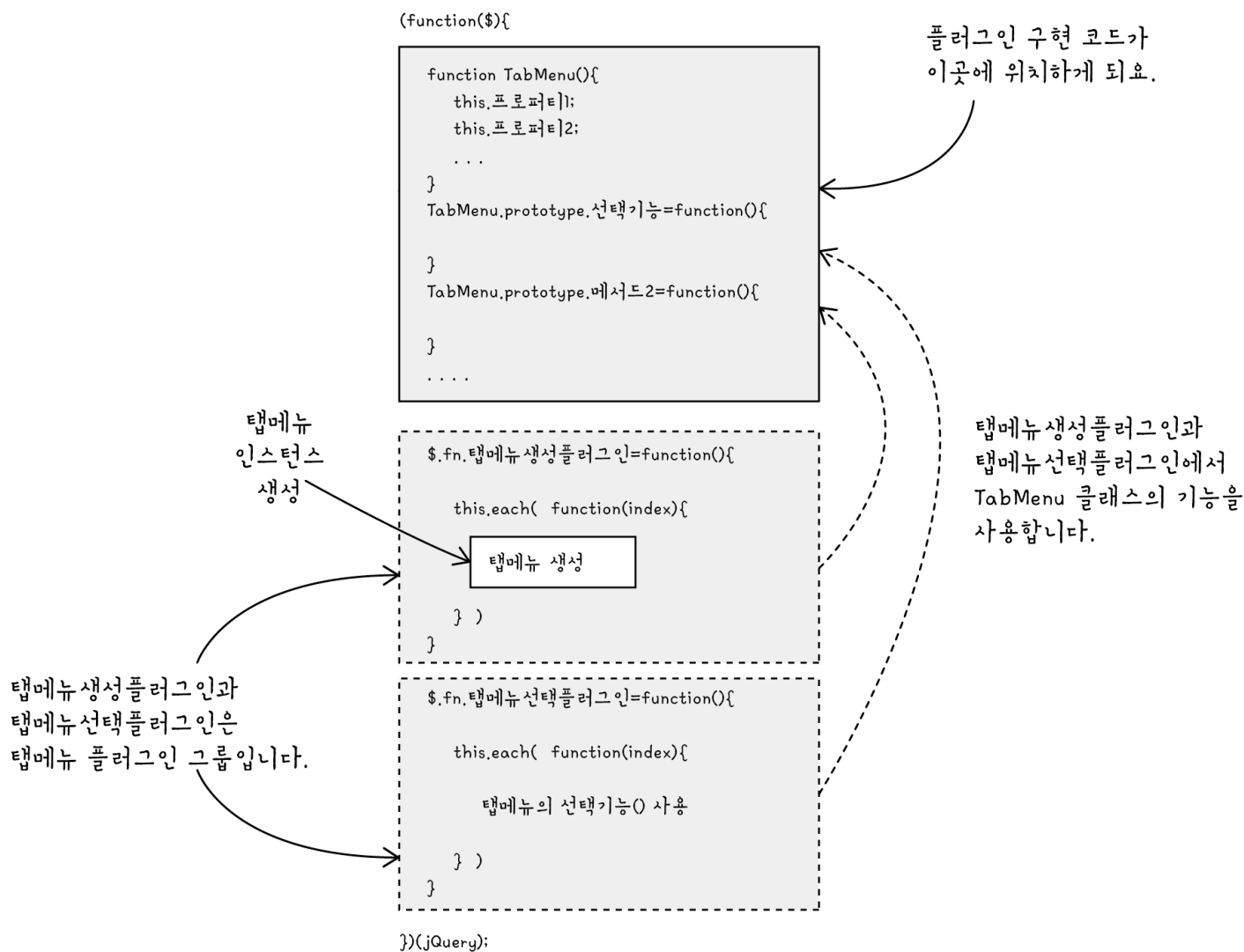


03. 정리



01. 소개

여기서 말하는 '플러그인 그룹'이란 다음과 같이 연관된 클래스 기반으로 jQuery 플러그인을 만들 때 클래스 인스턴스를 연관 있는 플러그인에서 공유해서 사용하는 구조를 말합니다.



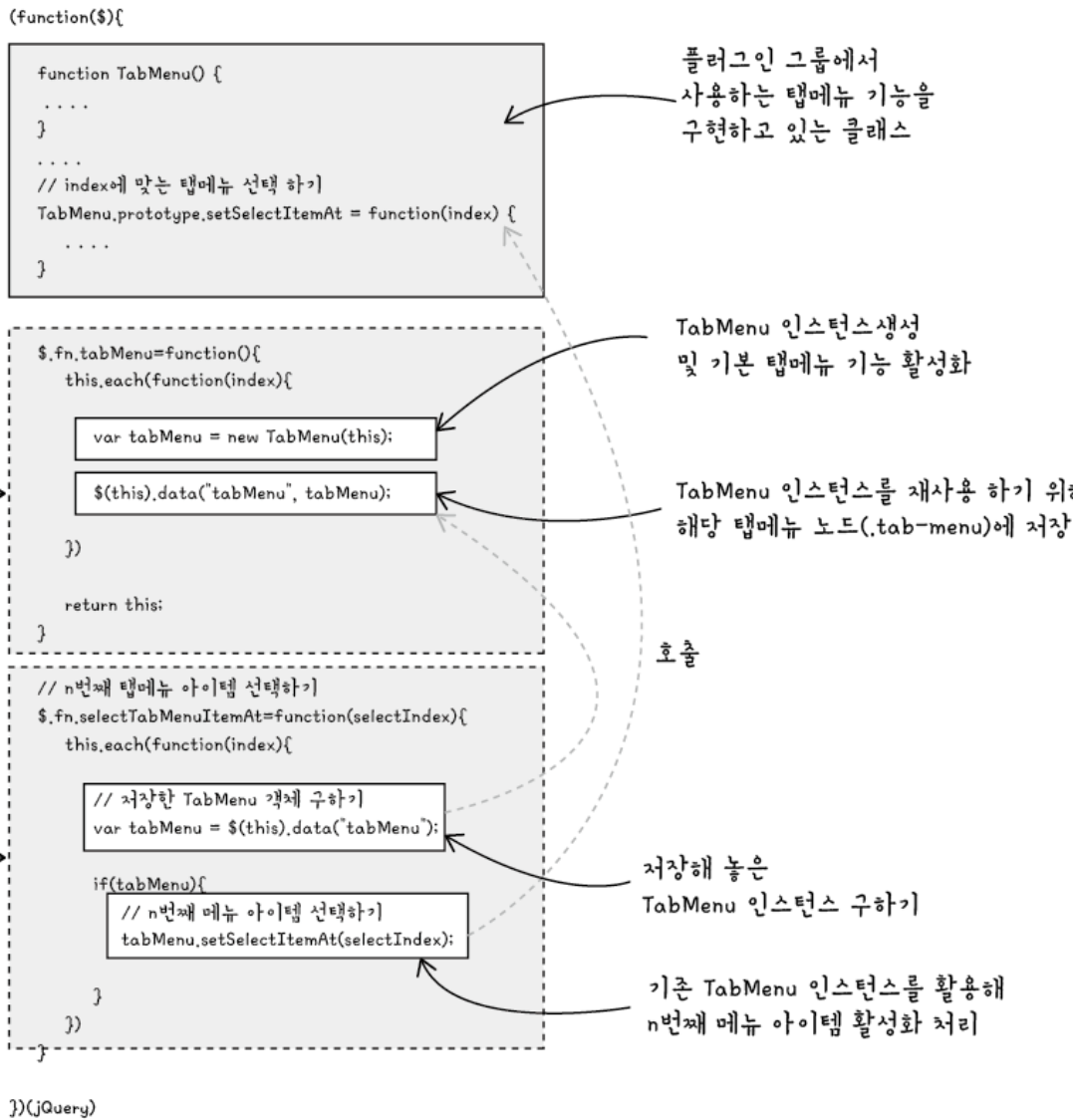
02. 예제

03. 정리

1) 플러그인 그룹이란?  
예제를 가지고 설명하면 tabMenu 플러그인과 selectTabMenuItemAt 플러그인이 탭메뉴 플러그인 그룹이 됩니다.

2) 플러그인 그룹 구조  
특정 플러그인에서 생성한 클래스의 인스턴스를 다른 플러그인에서 재사용해야 하는 경우 jQuery의 data() 메서드를 활용해 생성한 인스턴스를 저장해 재사용하면 됩니다.

탭 메뉴 관련 플러그인 그룹 (플러그인 그룹에서는 TabMenu 클래스의 인스턴스가 오직 한번만 만들어져요.)



## 01. 기본 옵션 값

플러그인 호출시 기본으로 설정되는 값

## 02. jQuery의 extend() 메서드 소개

### 1) 소개

사용법:

```
var result = jQuery.extend(target[,object1][,objectN]);
```

매개변수:

- target: 합쳐진 기능을 최종적으로 저장할 객체
- object1, objectN: 합쳐질 기능을 가진 객체

리턴값:

리턴 값은 target에 저장되는 값과 같습니다.

### 2) 기능 소개

예제

### 3) extend() 메서드 사용 시 주의사항 및 해결책

예제

## 03) extend() 메서드를 활용한 플러그인 옵션 처리

예제

## 미션 01 심플 갤러리를 플러그인으로 만들기