abc = Q

efg = Point

#JS 33 concept

##Value Type & Reference Type

자바스크립트에는 클래스 개념이 없다.

그래서 클래스를 대체하는 타입 개념이 존재한다.

* 원시타입
* 참조타입

### 값 타입

원시값에 저장하고 수정되는 값은 더 이상 축약할 수 없는 값이며,

원시값을 참조하면 값 자체를 ‘복사, 복제’ 한다는 뜻

var name = “jinju”; // 원시 문자열 객체를 생성(문자 리터럴)

var nameCopy = name; // 값을 새 변수로 복사

var name = null; name 변수에 저장된 값을 새로 할당

console.log(name, nameCopy); // ‘null, jinju’ 출력

.

### 참조 타입

참조 타입은 ES5에서 클래스가 없기 때문에 클래스와 가장 가까운 개념이다.

참조 값은 참조 타입의 인스턴스이며 객체와 같은 말이다.

객체는 순서가 없는 프로퍼티로 이루어져 있으며, 프로퍼티는 이름과 값의 구성이다.

var object1 = new Object(); // 일반 객체를 하나 만들어 object1라는 변수에 저장

var object2 = object1; // object2는 object1과 같은 포인터 참조

>> 참조 타입은 할당된 변수에 값을 직접 저장하지 않는다.

그래서 object에 저장된 값은 객체 인스턴스가 아니라 객체가 있는 메모리상 위치를 가리키는 포인터다.

### 객체 참조 제거

명백히 사용하지 않을 변수에 대해서는 참조 제거를 해주는 것이 좋다.

var object1 = new Object();

object1 = null; // null 값을 넣으면 참조 제거가 됨.

대규모 서비스에선 혹시 모를 상황(?)에 대비해 객체 참조 제거를 필수적으로 하는 것이 좋다.

서로 달라야할 객체가 한 메모리를 참조해 변경 오류가 있을 수 있다.

모듈 내 응집도는 높고 모듈간 결합도는 낮아야 한다는 것!

##명시적 변환, 암시적 변환, Nominal, 구조화, 덕타이밍

* 형변환 : 숫자를 문자로 변환 혹은 문자를 숫자로 변환하는 경우

ex)

### 명시적 변환

개발자가 직접 스크립트를 이용해 어떤 형으로 바꿀지 명시해주는 것

모든 형변환은 전역함수 또는 특정 클래스에서 제공하는 메서드를 이용하여 처리

* 방법  
  1) 문자 -> 숫자

var value = “123.456”;

parseInt(value);

result : 123

var value = “123”;

Number(value);

result : 123

var value = “123.456”;

parseFloat(value);

result : 123.456

var value = “123.456”;

Number(value);

result : 123.456

2) 숫자 -> 문자

-일반 문자형 : String() 사용

var value = 15;

String(value);

result : 15

-16진수 문자형 : Number.toString() 사용

var value = 15;

value.toString(16);

result : f

-실수 문자형 : Number.toFixed() 사용

var value = 123.456;

value.toFixed(2);

result : 123.46 // 반올림 발생

### 암시적 형변환

자바스크립트 엔진이 필요에 의해 암시적으로 형을 자동으로 변환시키는 것

자바스크립트는 규칙을 만들어 암묵적 형변환

* 규칙

1. 숫자형 + 문자형 : 문자형

var a = 10+”10”;

result : a는 문자 “1010”

ex) var a = “30”;

var result =1+a+10;

alert(result);

result : 13010

1. 불린형 + 문자형 : 문자형

var a = true + “10”;

result : a는 문자 “true10”

1. 불린형 + 숫자형 : 숫자형

var a = true +10;

result : a는 숫자 11

(true 1, false 0)

###Nominal 명목

C++, Java, Swift 와 같은 언어는 주로 명목형 시스템 사용

class Foo { method(input : strin) { /”….”/ } }

class Bar { method(input : strin) { /”….”/ } }

let foo: Foo = new Bar(); // Error

>> Bar에서는 Foo 이름이 서로 다르므로 a를 입력하려고 할 때 오류가 생김

###구조화

OCaml, Haskell, Elm과 같은 언어는 기본적으로 구조형 시스템을 가짐.

class Foo { method ( input : string ) : 숫자 }

class Bar { method ( input : string) : 숫자 }

let foo : Foo = new Bar (); // OK  
>> Foo 구조체가 정확히 동일하기 때문에 a가 필요한 곳에 bar를 넣으려고 할 때 전달되는 구조형 시스템의 예시

class Foo { method(input : string ) { /”…”/ } }

class Bar { method(input : number) { /”…”/ } }

let foo: Foo = new Bar(); // Error

>> 모양 변경하니 오류  
??? <https://medium.com/@thejameskyle/type-systems-structural-vs-nominal-typing-explained-56511dd969f4>

###Duck typing

객체의 변수 및 메소드의 집합이 객체 타입을 결정하는 것

즉, 타입 체킹을 runtime에 하는 것

“오리처럼 보이고, 오리처럼 꽥꽥 거릴 수 있으면 오리이다.”

>> 특정 객체가 먼저 무엇인지 “검사”하고 맞으면 해당 액션을 수행하는 것

아래의 예제를 보면, inTheForest() 에서 객체에 quake 속성이 있는지 feathers 속성이 있는지 보고 있으면 해당 함수를 실행한다.  
  
이것이 바로 duck test 또는 duck typing 이라고 한다. 즉, 객체가 무엇인지, 해상 속성이 있는지 검사를 먼저 하고 수행 하는 것을 말한다.

**function** Duck() {  
 *// Dynamically add functions to this object*  
 **this**.quack = **function**() { alert('Quaaaaaack!'); };  
 **this**.feathers = **function**() { alert('The duck has white and gray feathers.'); };  
}  
  
**function** Person() {  
 *// Dynamically add functions to this object*  
 **this**.quack = **function**() { alert('The person imitates a duck.'); };  
 **this**.feathers = **function**() { alert('The person takes a feather from the ground and shows it.'); };  
 **this**.name = **function**() { alert('John Smith'); };  
}  
  
**function** inTheForest(object) {  
**if (object.quack)     *// Check that the .quack() function exists*  
   object.quack();  
 if (object.feathers)  *// Check that the .feathers() function exists*  
   object.feathers();**  
}  
  
**function** game() {  
 **var** donald = **new** Duck();  
 **var** john = **new** Person();  
 inTheForest(donald);  
 inTheForest(john);  
}  
  
*// Execute upon page load*  
game();