INTERPARK

2017 NOV.

{ .JS }

JavaScript Lab

객체와 프로토타입

JavaScript Lab 4

01. 생성자 와 Prototype

> 자바스크립트는 prototype을 이용, 객체지향 프로그래밍 가능(OOP)

> 자바스크립트 상속은 prototype 기반의 상속

> 자바스크립트에서 클래스는 생성자 함수와 prototype의 조합

> prototype 메소드는 클래스의 인스턴스 간에 공유 (static 함수와 비슷)

예제

```
1
2   //1. 생성자와 Prototype
3   function User(name,passwordHash){
4        this.name = name;
5        this.passwordHash = passwordHash;
6   }
7
8   User.prototype.toString = function(){
9             return "[ User " + this.name + " ]";
10  }
11
12  User.prototype.checkPassword = function(password){
13            return hash(password) === this.passwordHash;
14  }
15
16  var u = new
17  User('eunhak','0ef23ksdflk3lks234l23kjlsd3l');
18
19  u.toString(); // ==> [ User eunhak ]
20  u.checkPassword('N14401'); // ==> false
21
22
23
24
25
26
27
28
29
30
31
32
```

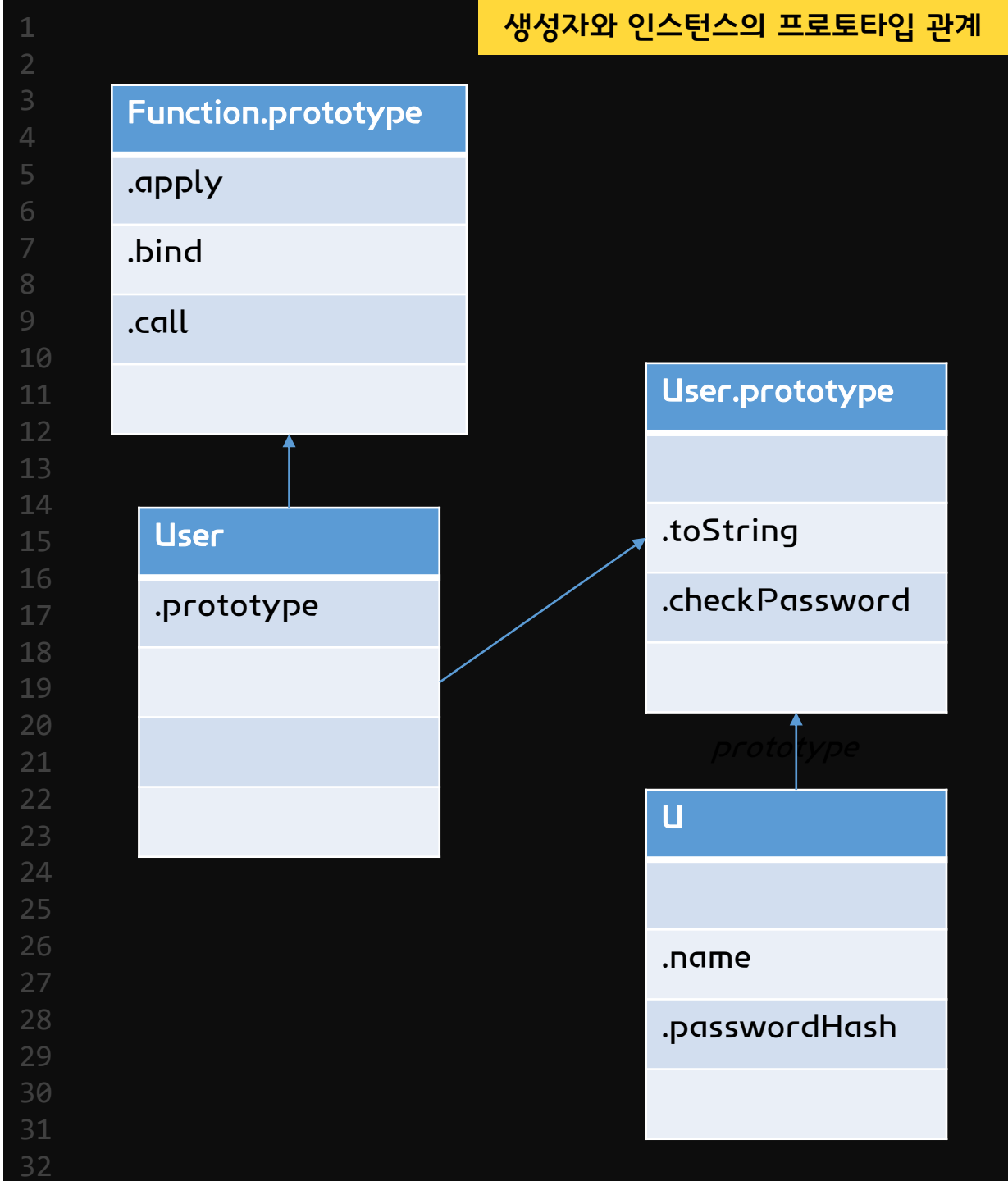
01. 생성자 와 Prototype

> 자바스크립트는 prototype을 이용, 객체지향 프로그래밍 가능(OOP)

> 자바스크립트 상속은 prototype 기반의 상속

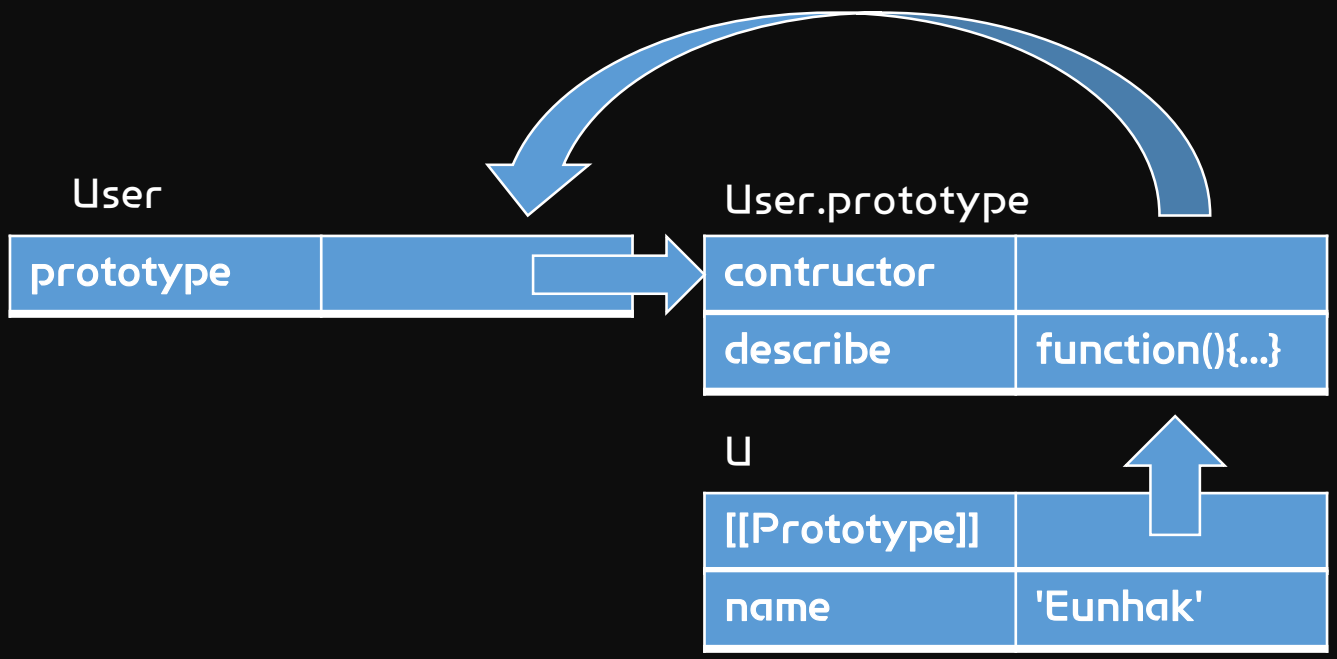
> 자바스크립트에서 클래스는 생성자 함수와 prototype의 조합

> prototype 메소드는 클래스의 인스턴스 간에 공유 (static 함수와 비슷)



01. 생성자 와 Prototype

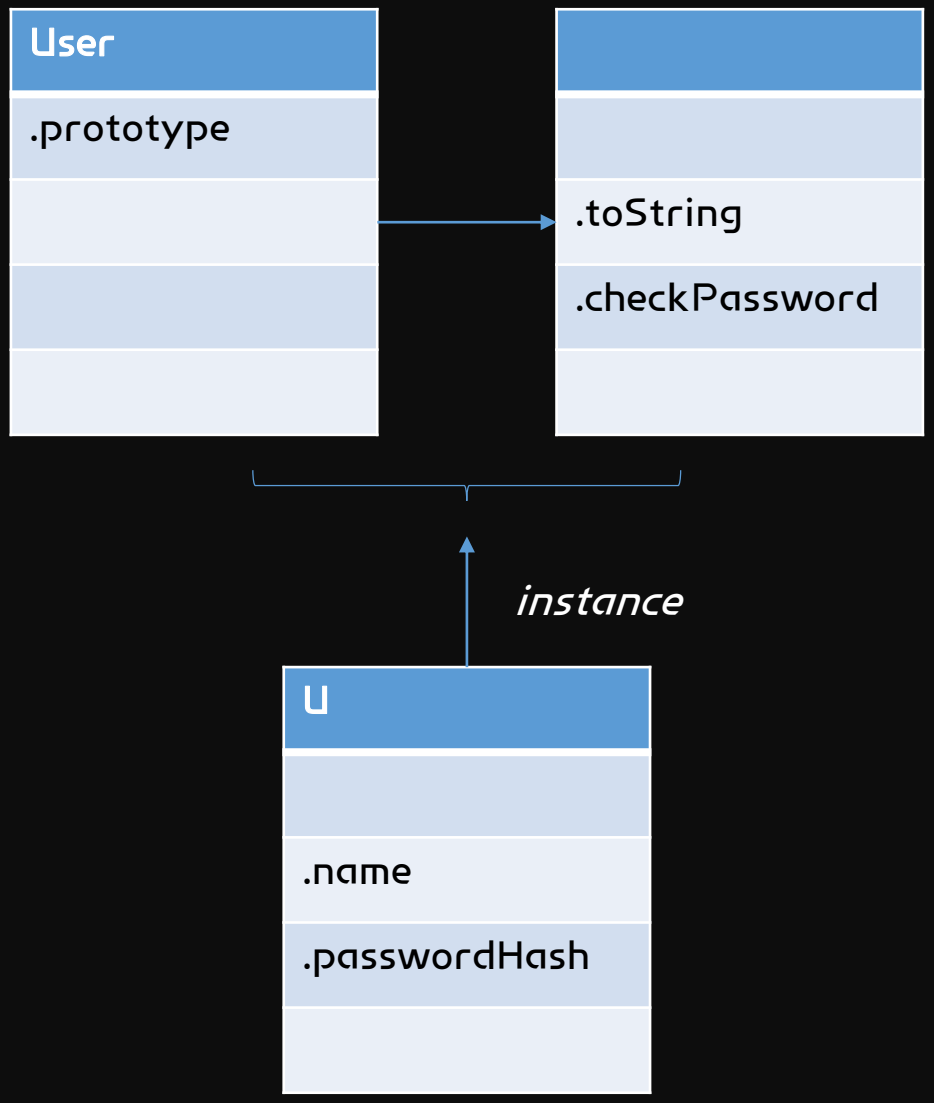
생성자와 인스턴스의 프로토타입 관계



01. 생성자 와 Prototype

> User 클래스의 개념적 관점

User 클래스의 개념적 관점



02. 설계 시 유의점

> new
키워드 누락 시에도
정상적으로 동작할 수 있도록 함

예제

```
function User(name,passwordHash){
    this.name = name;
    this.passwordHash = passwordHash;
}

var u = User('oleh', 'wer2323sD323sdf35344');
u; // undefined
this.name; //oleh
this.passwordHash; //wer2323sD323sdf35344

function UserStrict(name, passwordHash){
    "use strict"
    this.name = name;
    this.passwordHash = passwordHash;
}

var us = UserStrict('oleh', 'wer2323sD323sdf35344');
//Cannot set property 'name' of undefined
////////////////////////////////////////////
function User(name, passwordHash){

  if(!(this instanceof User)){
    return new User(name, passwordHash);
  }

  this.name = name;
  this.passwordHash = passwordHash;
}


var u = User('oleh', 'wer2323sD323sdf35344');

```

02. 설계 시 유의점

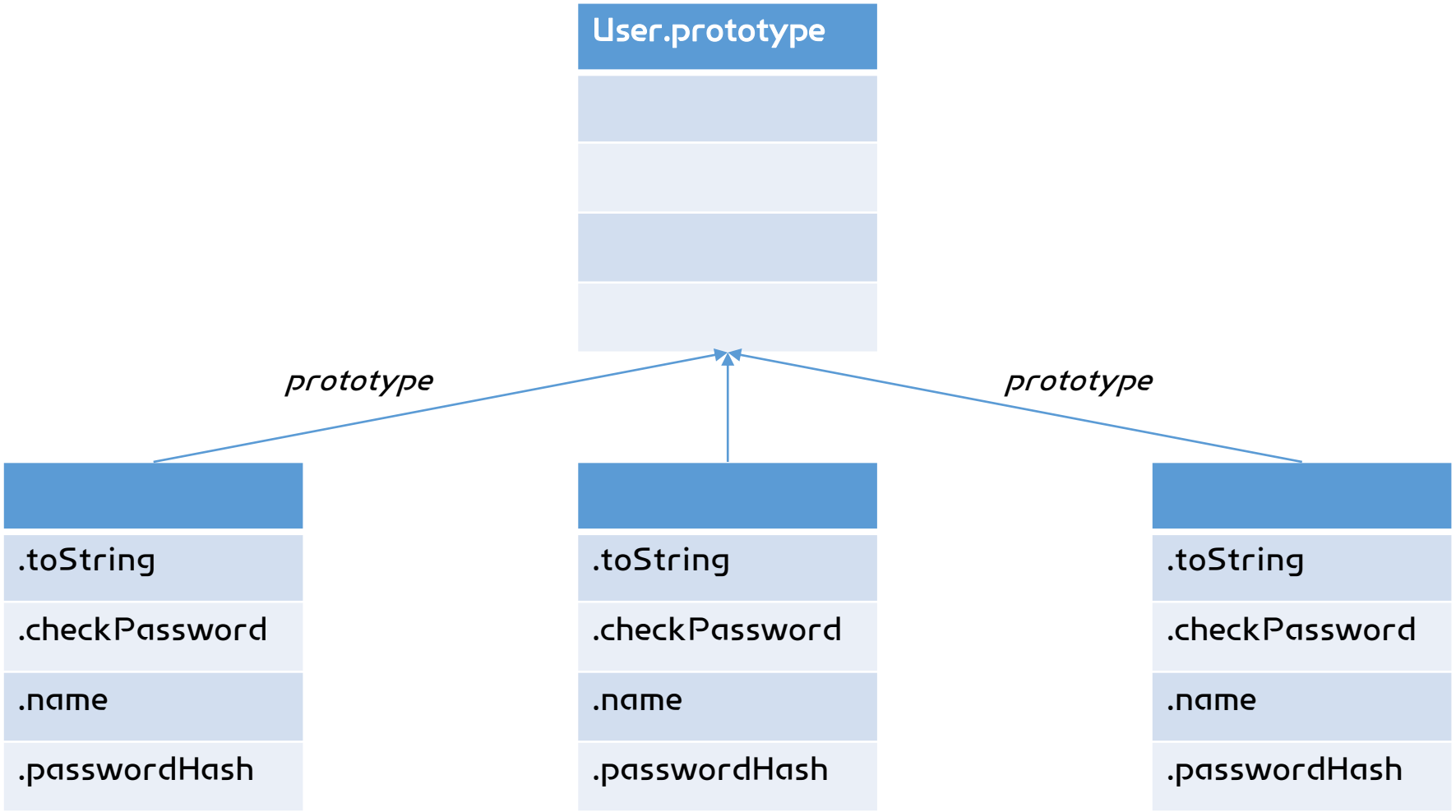
> 메소드는
prototype에 저장(메모리 최적화)

예제

```
1
2   //메소드는 prototype에 저장(메모리 최적화)
3
4   function User(name, passwordHash){
5     this.name = name;
6     this.passwordHash = passwordHash;
7     this.toString = function(){
8       return "[ User " + this.name + " ]";
9     };
10    this.checkPassword = function(password){
11      return hash(password) === this.passwordHash;
12    };
13  }
14
15  var oleh = new User('eunhak', 'sdf23d23d23');
16  var gusik = new User('gusik', 'h34ce23');
17  var lnb = new User('nbok', 'dfsf232sd23we');
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

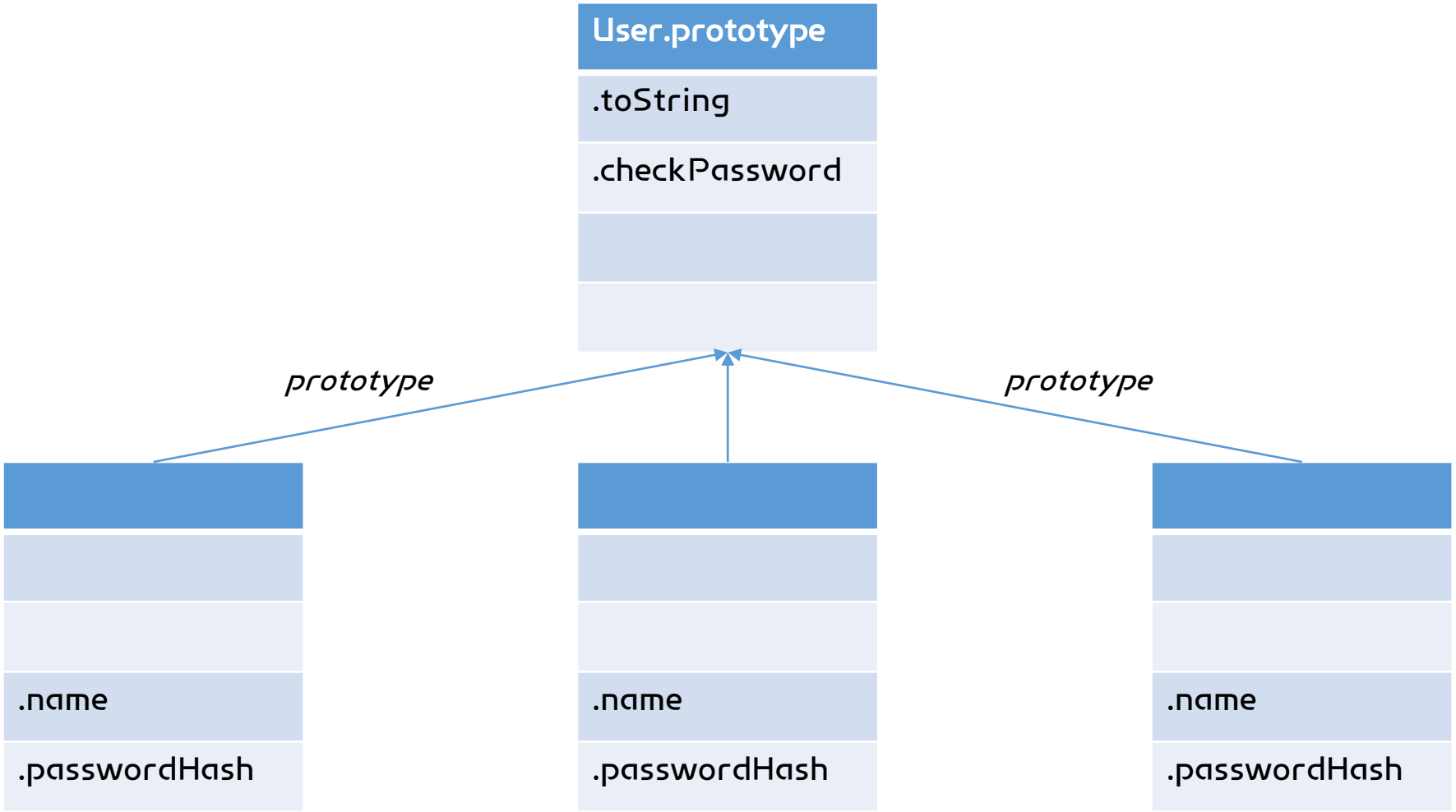
02. 설계 시 유의점

> 생성자에 메소드 생성



02. 설계 시 유의점

> 프로토타입에 메시지 생성



02. 설계 시 유의점

> 정보은닉을 위해 클로저 사용
This의 프로퍼티가 아닌 변수로 참조

> This의 프로퍼티가 없기 때문에 외부 접속 불가

> 메소드는 **인스턴스 객체에 위치 해야함**
(메소드 복사 급증)

예제

```
1
2
3
4   //메소드는 prototype에 저장(메모리 최적화)
5
6   function User(name, passwordHash){
7     this.toString = function(){
8       return "[ User " + this.name + " ]";
9     };
10
11    this.checkPassword = function(password){
12      return hash(password) === this.passwordHash;
13    };
14  }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

02. 설계 시 유의점

> this는
가장 가까이 둘러싼 함수에 의해 바인딩됨

> this 파라메터 누락 시
map 콜백함수 내
this에는 전역객체(window)이 바인딩됨

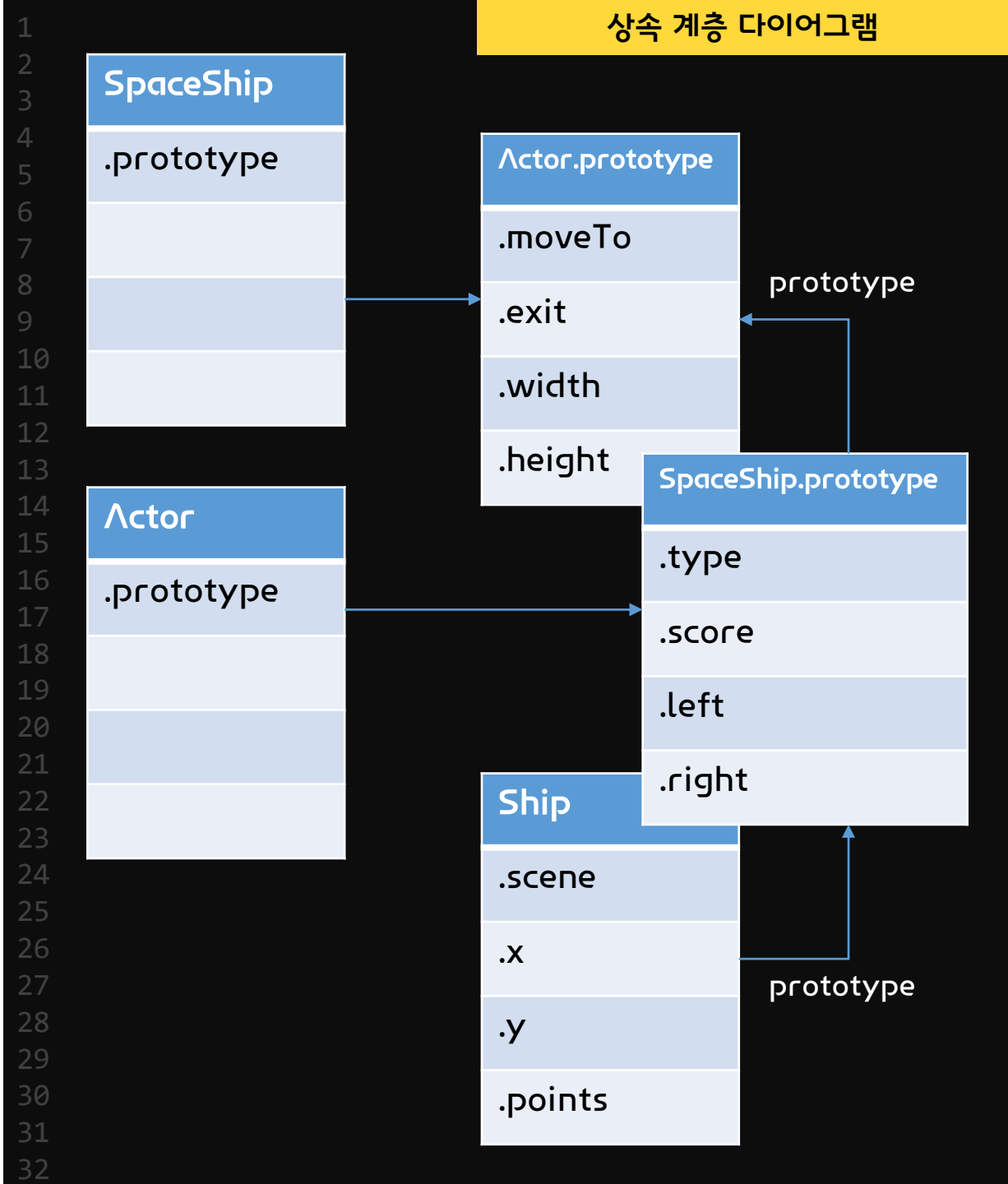
예제

```
function CSVReader(separators){
  "use strict"
  this.separators = separators || [","];
  this.regexp = new
  RegExp(this.separators.map(function(sep){
    return "\\" + seq[0];
  }).join("|"));
};

CSVReader.prototype.read = function(str){
  "use strict"
  var lines = str.trim().split(/\n/);
  return lines.map(function(line){
    return line.split(this.regexp);
  });
};

var reader = new CSVReader();
reader.read("a,b,c\nd.e.f\n");

//nonStict => [["a,b,c"],["d,e,f"]]
//strict => Cannot read property 'regexp' of
undefined
//정상 => [["a,b,c"],["d,e,f"]]

////////////////////////////////////////////
//1번째 방법
return lines.map(function(line){
  return line.split(this.regexp);
}, this); //외부 this 바인딩 전달
```

03. 클래스 상속

> 하위클래스 생성자에서
상위클래스 생성자 명시적 호출

> Object.create를 사용하여
하위클래스 프로토타입 객체 생성



03. 클래스 상속

> 하위클래스 생성자에서
상위클래스 생성자 명시적 호출

> Object.create를 사용하여
하위클래스 프로토타입 객체 생성

예제

```
1
2
3   //하위클래스 생성자에서 상위클래스 생성자 명시적 호출
4   //Object.create를 사용하여 하위클래스 프로토타입 객체생성
5
6   function Actor(scene, x, y){
7       this.scene = scene;
8       this.x = x;
9       this.y = y;
10  }
11
12  Actor.prototype.moveTo = function(x, y){
13      this.x = x;
14      this.y = y;
15      return "X:" + this.x + ", Y:" + this.y;
16  }
17
18  function SpaceShip(scene, x, y){
19      Actor.call(this, scene, x, y);
20      this.points = 0;
21  }
```

```
▼ SpaceShip {scene: "spcae", x: 100, y: 200, points: 0}
    points: 1
    scene: "spcae"
    x: 100
    y: 100
  ▼ __proto__: Actor
    ▶ scorePoint: ƒ ()
    ▼ __proto__:
      ▶ moveTo: ƒ (x, y)
      ▶ constructor: ƒ Actor(scene, x, y)
      ▶ __proto__: Object
> |
```

Fin