Webscripting

Hoofdstuk 6

The secret life of objects

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Object literals

- key-value pairs
- {key1:value1, key2:value2, ...}
- string of Symbol als datatype voor key
- value eender welk datatype of function

```
let person = {
    name : 'tim',
    age : 7,
    print : function () {
        console.log(`name: ${this.name} age: ${this.age}`);
    }
};

person.print();
// name: tim age: 7
```

Wanneer een function deel uitmaakt van een object dan verwijst 'this' in deze function naar dit object.

Object literals

- geen encapsulation
- vaak gebruikte conventie: _ voor veld om aan te duiden dat het niet de bedoeling is om het veld te wijzigen buiten het object

```
let person = {
    _name : 'tim',
    print : function () {
        console.log(`name: ${this.name}`);
    },
    setName : function(name) {
        this.name=name;
    },
    getName : function() {
        return this.name;
    },
};
person.setName('sofie');
person.print();  //name: sofie
person._name = 'jan'; // tegen conventie maar lukt toch
person.print(); //name: jan age: 4
```

Object literals

Shorthand property names (ES2015)

```
let name = 'tim';
let age = 7;
const print = function () {
    console.log(`name: ${this.name} age: ${this.age}`);
}
let person = { name , age, print };
// komt op hetzelfde neer als de onderstaande regel
// let person = { name : name , age : age, print : print };

person.print();
```

Object literals

Shorthand method names (ES2015)

```
let person={
    name:'tim',
    age:7,
    print(){
        console.log(`name: ${this.name} age: ${this.age}`);
    }
   // komt op hetzelfde neer als de onderstaande regels
    // print : function(){
   //    console.log(`name: ${this.name} age: ${this.age}`);
    // }
};

person.print();
```

Prototypes

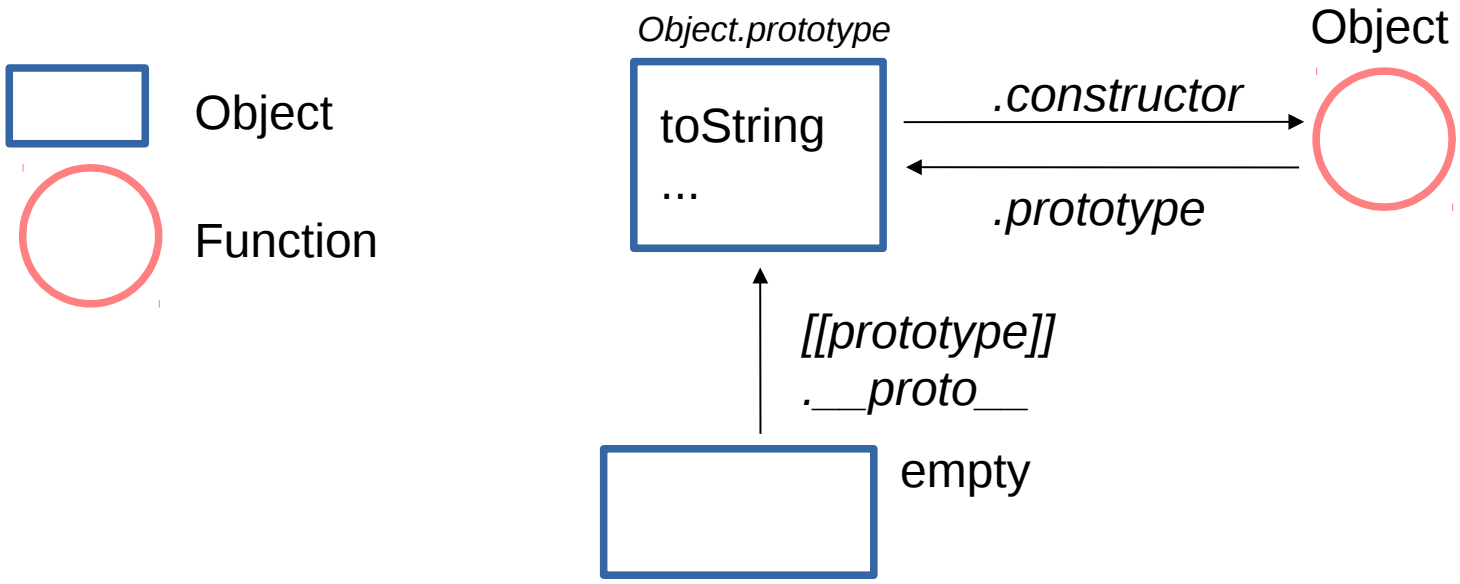
prototype = elk object is verbonden met een prototype
via een associatie-relatie

nut: in het prototype kunnen gemeenschappelijke functies gedefinieerd worden (toString van Object.prototype)

Object.create
Maak een nieuw object vertrekkend van een bestand object.
Het bestaand object wordt het prototype v. het nieuwe object
new
Gebruik een constructor-function om een object aan te maken.
Het prototype van het object wordt het prototype van de constructor-function.

extra

```
let empty = {};
```



[[prototype]] relatie tussen objecten
.__proto__ komt overeen met [[prototype]] (non-standard)

in het prototype worden (meestal) functies van een object vastgelegd (e.g. toString)

.constructor relatie tussen prototype en function (wordt gebruikt via new)

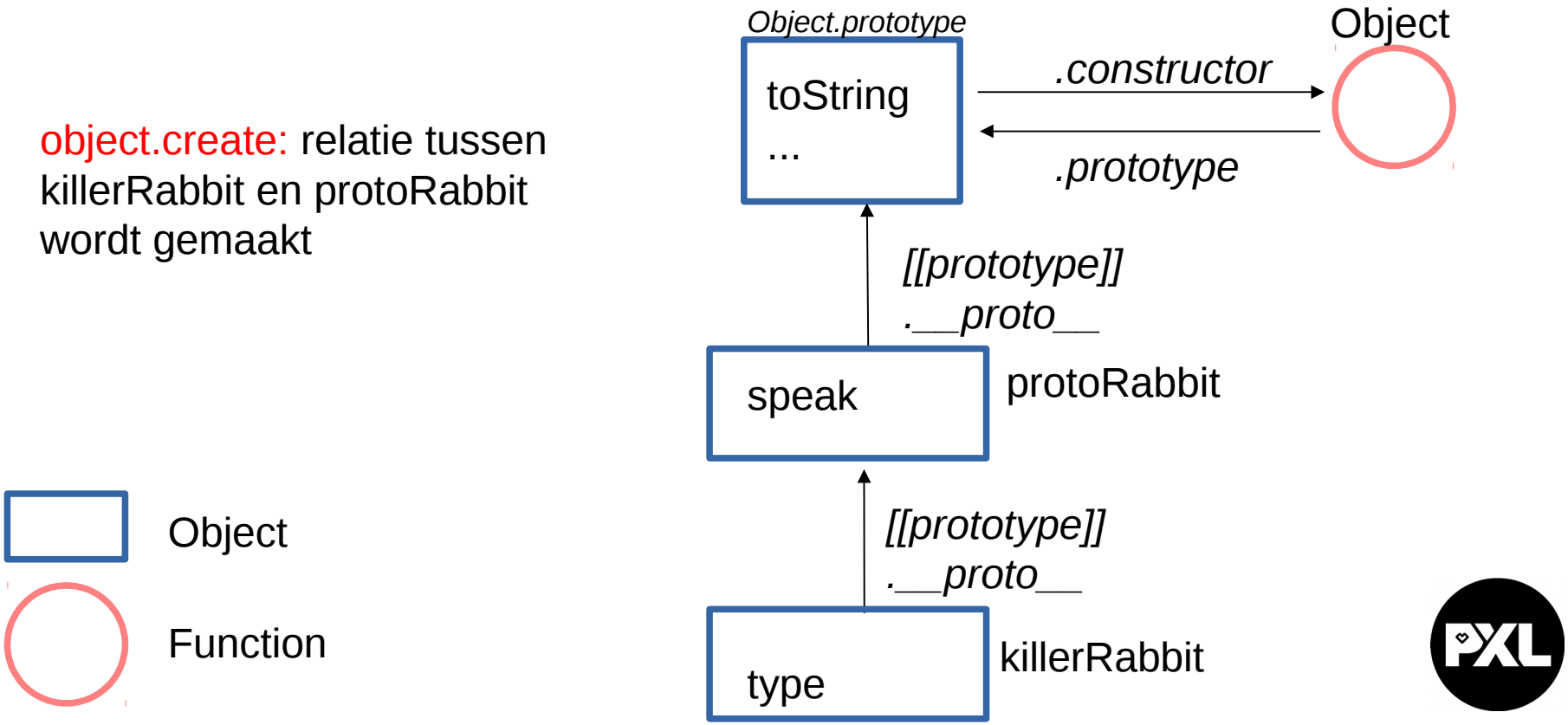
.prototype relatie tussen function en prototype

vb 1
Object.create

extra

```
let protoRabbit = {
    speak(line) {
        console.log(`${this.type} says '${line}'`);
    }
};
let killerRabbit = Object.create(protoRabbit);
killerRabbit.type = "killer";
killerRabbit.speak("SKREEEE!");
// killer says 'SKREEEE!'
```

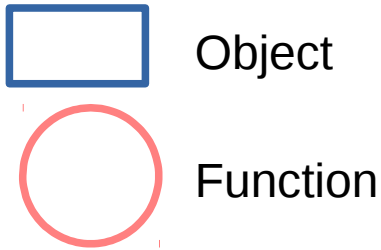
object.create: relatie tussen killerRabbit en protoRabbit wordt gemaakt

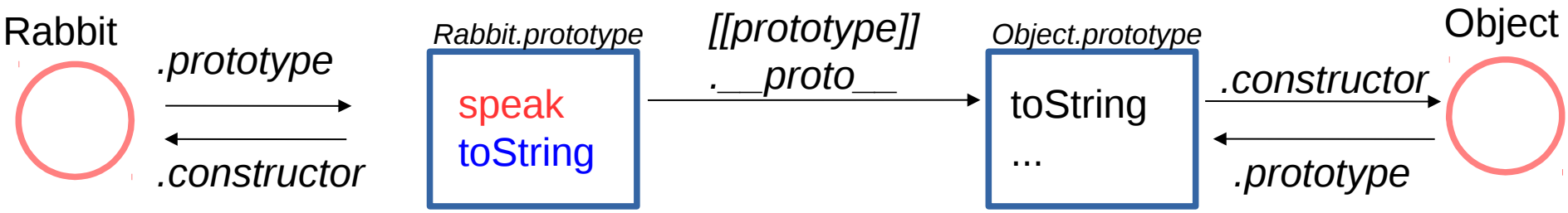


vb 2
new

extra

```
let Rabbit = function( type ){
    this.type = type;
}

Rabbit.prototype.speak = function(line) {
   console.log(`${this.type} says '${line}'`);
};

Rabbit.prototype.toString = function() {
    return `A ${this.type} rabbit`;
};
```





vb 2
new

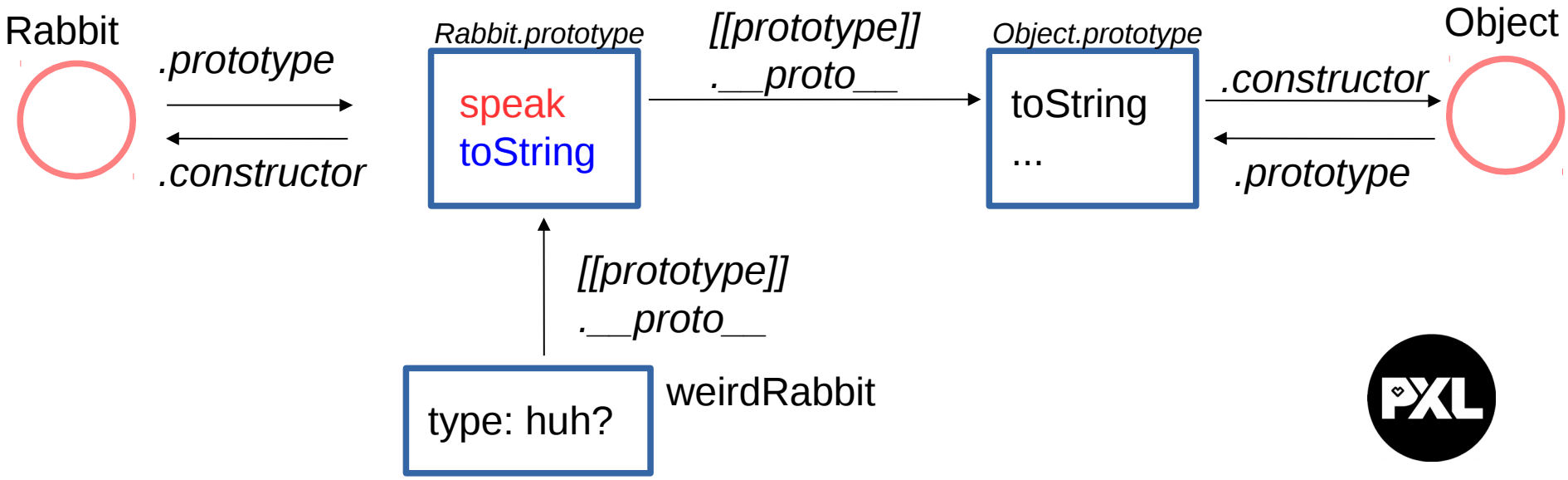
extra

```
let weirdRabbit = new Rabbit("huh?");
weirdRabbit.speak();
// weird says huh?
console.log(String(weirdRabbit));
// A weird rabbit
```

Object

Function

new: constructor-function
Rabbit wordt uitgevoerd



'private fields'

extra

Constructor function met IIFE (immediately invoked function expression)

```
let Shape = function (x, y) {
    let _x = x;
    let _y = y;
    this.getX = function () {
        return _x;
    }
    this.getY = function () {
        return _y;
    }
    this.setX = function (x) {
        _x = x;
    }
    this.setY = function (y) {
        _y = y;
    }
}
Shape.prototype.describe = function () {
    console.log(`Shape at (${this.getX()}, ${this.getY()})
`);
}
```

'private fields'

extra

Constructor function met IIFE (immediately invoked function expression)

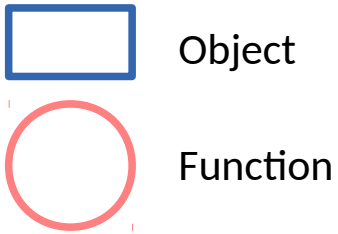
```
let shape1 = new Shape(1, 2);
shape1.describe();
shape1.setX(12);
shape1.describe();
console.log(shape1.y); // undefined
```

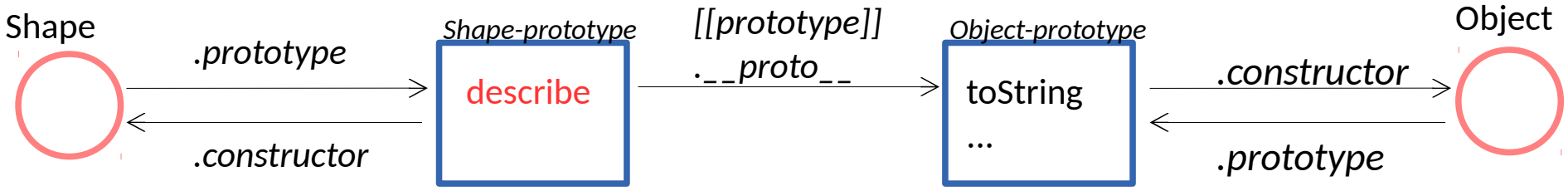
Inheritance ES5

- Eerder associatie (has a) dan inheritance (is a)

extra

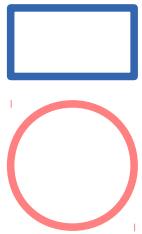
```
let Shape = function(x, y){
    this.x = x;
    this.y = y;
}

Shape.prototype.describe=function(){
   console.log(`Shape at (${this.x}, ${this.y})`);
}
```





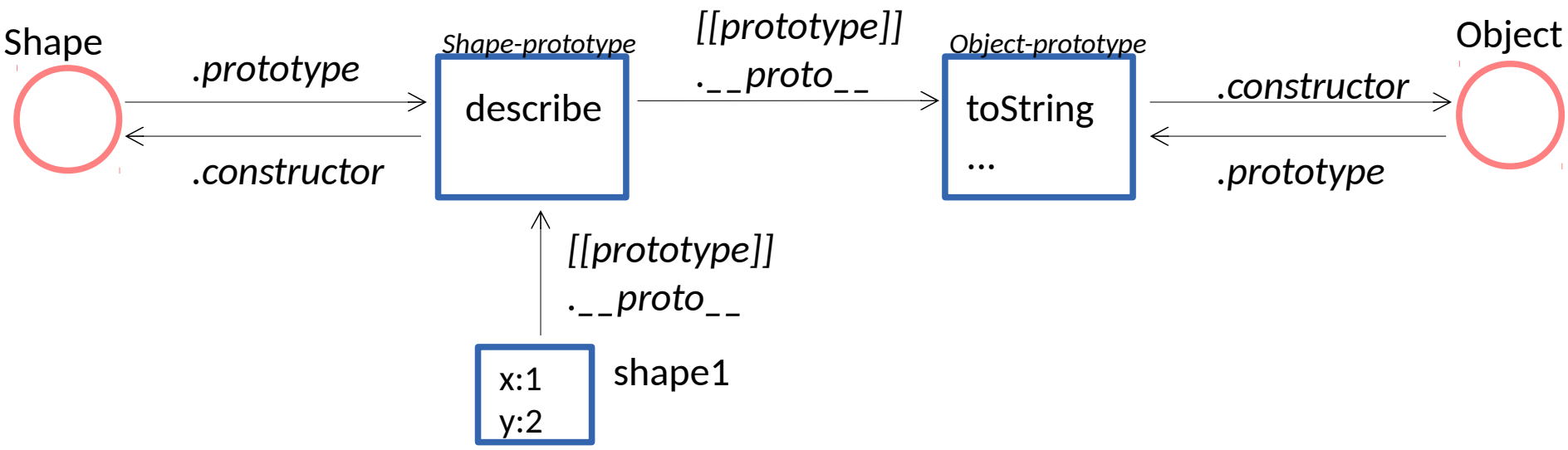
extra

```
let Shape = function(x, y){
    this.x = x;
    this.y = y;
}

Shape.prototype.describe=function(){
   console.log(`Shape at (${this.x}, ${this.y})`);
}

let shape1 = new Shape(1,2);

shape1.describe();
```



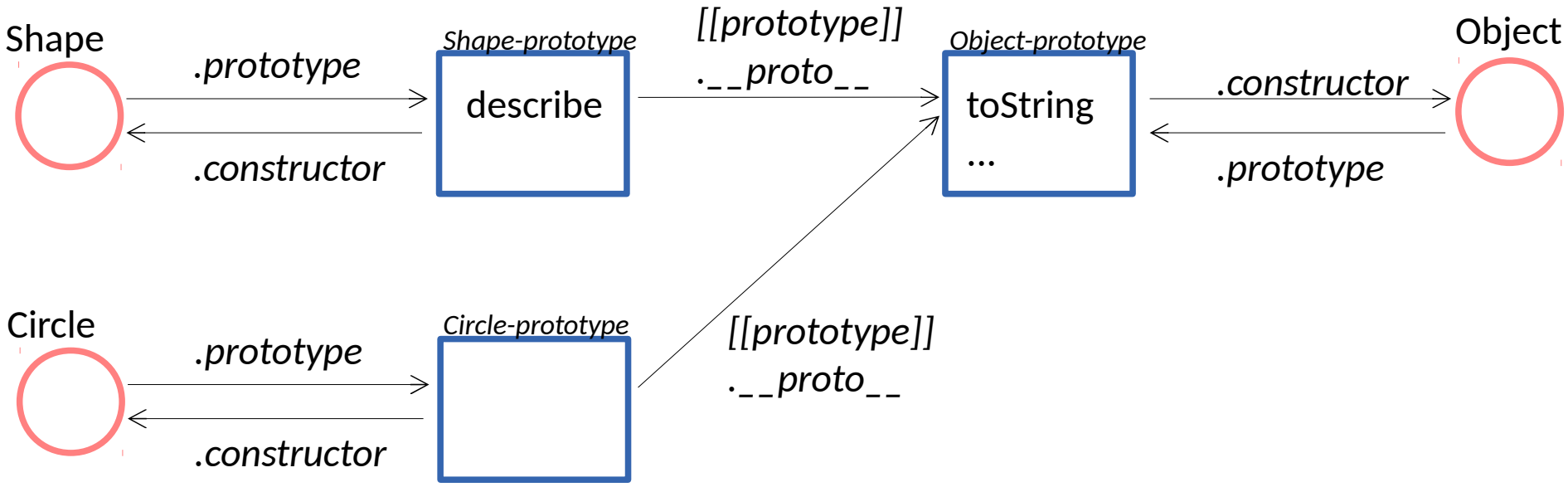
Object

Functio
n



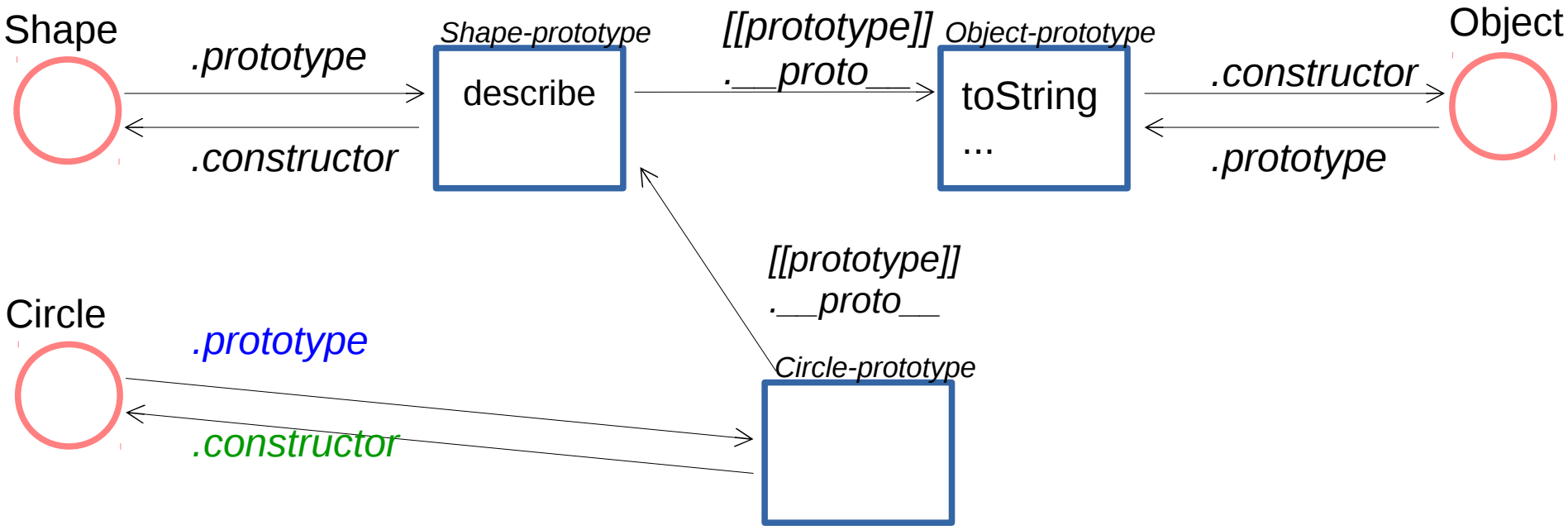
extra

```
let Shape = function(x, y){
    this.x = x;
    this.y = y;
}

Shape.prototype.describe=function(){
    console.log(`Shape at (${this.x}, ${this.y})`);
}

let Circle = function( x, y, radius ){
    Shape.call( this, x, y );  // constructor Shape
    this.radius=radius;
}
```



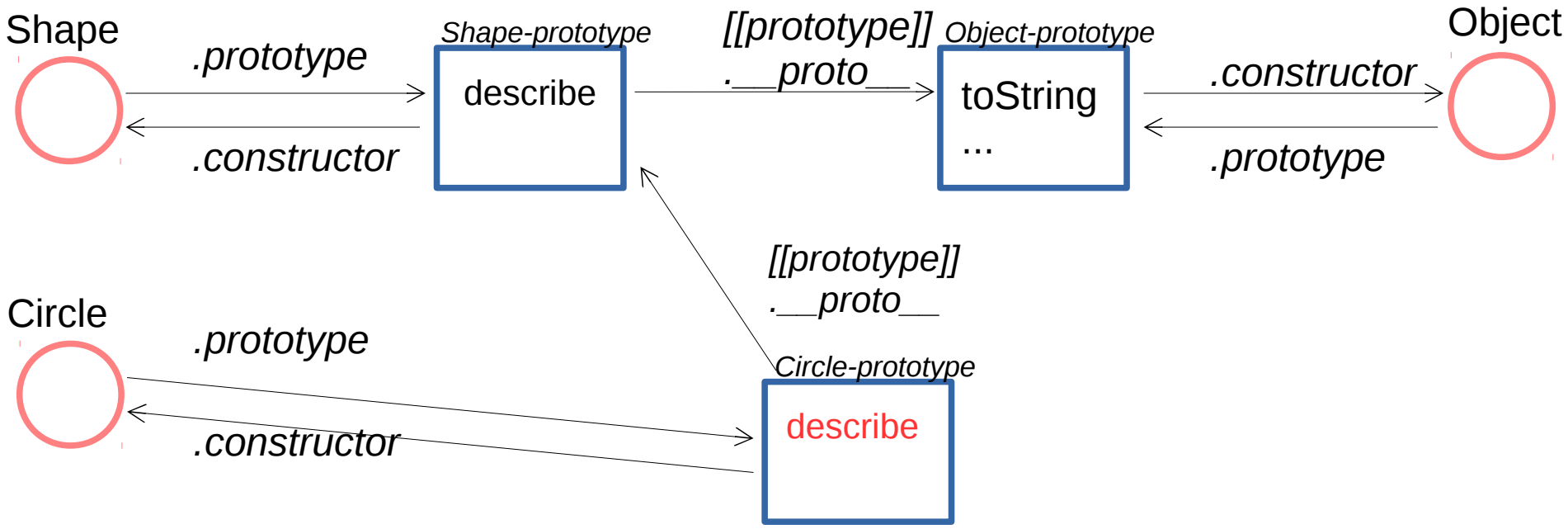
extra

```
...

Circle.prototype=Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;
```



extra

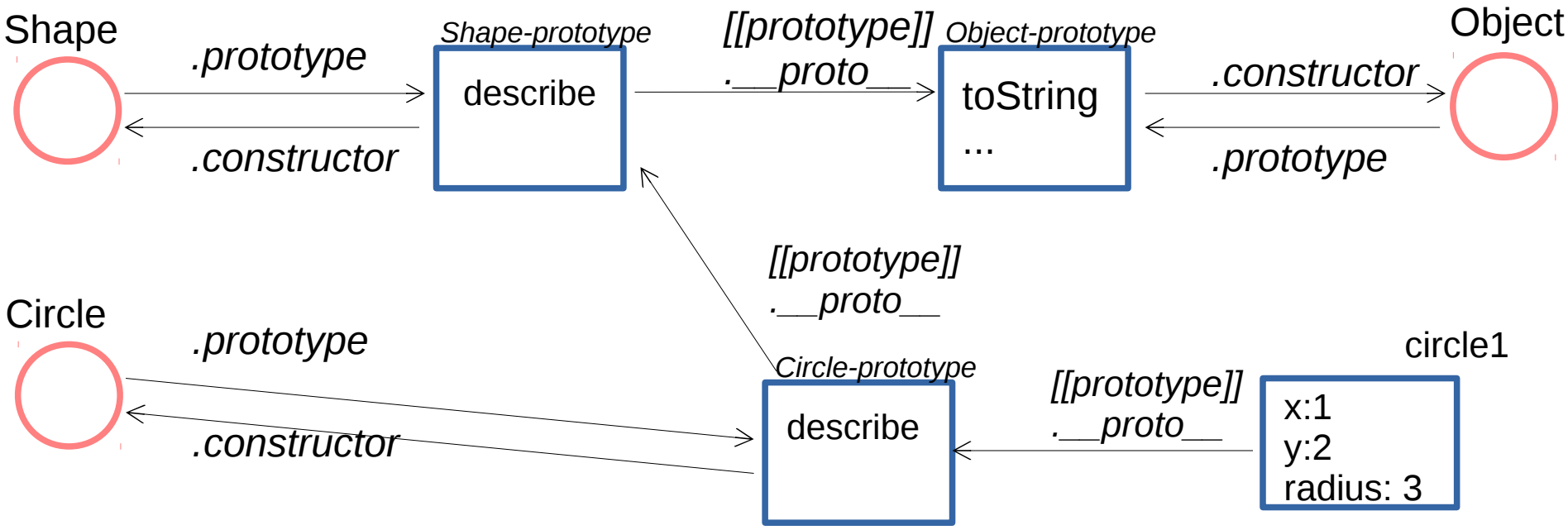
```
...

Circle.prototype=Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;

Circle.prototype.describe=function(){
    Shape.prototype.describe.call(this);
    console.log('I am also a Circle');
    console.log(`My radius is ${this.radius} `);
}
```



extra

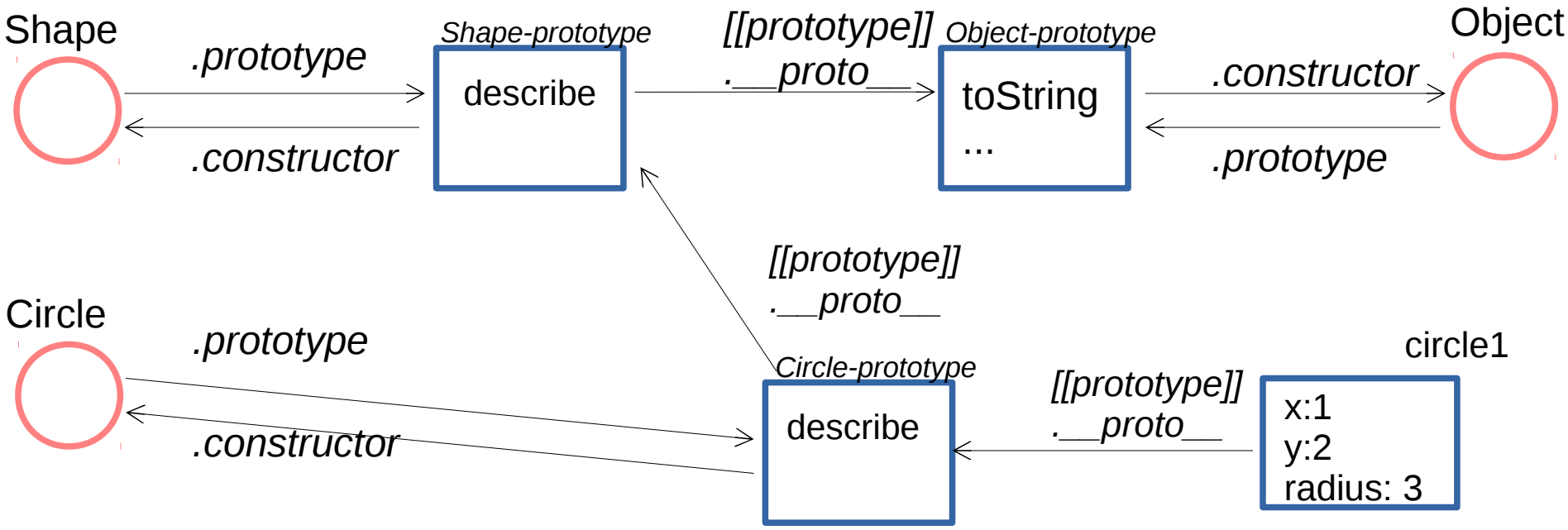
```
...

Circle.prototype=Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;

Circle.prototype.describe=function(){
    Shape.prototype.describe.call(this);
    console.log('I am also a Circle');
    console.log(`My radius is ${this.radius} `);
}

let circle1=new Circle(1,2,3);
circle1.describe();
```

```
jan@laptop-jan ~/Desktop/code/test $ node shape.js
Shape at (1, 2)
I am also a Circle
My radius is 3
```



extra

```
console.log(circle1 instanceof Circle);
//true
console.log(circle1 instanceof Shape);
//true
console.log(circle1 instanceof Object);
//true
```



Classes (ES6)

Eenvoudigere syntax
Intern worden nog altijd prototypes gebruikt

```
class Rabbit {
    constructor(type) {
        this.type = type;
    }
    speak(line) {
        console.log(`${this.type} says '${line}'`);
    }
   toString() {
       return `A ${this.type} rabbit`;
   }
}
let killerRabbit = new Rabbit("killer");
let blackRabbit = new Rabbit("black");
```

Classes: setters & getters

Indien setter voorzien voor key:

overal waar object.key gewijzigd wordt, wordt de setter aangeroepen

Indien getter voorzien voor key:

overal waar object.key uitgelezen wordt, wordt de getter aangeroepen

Classes: setters & getters

```
class Person  {
    constructor(name) {
       this._name=name;
    }
    set name (name) {
        this._name=name;
    }
    get name () {
        return this._name;
    }
};


person = new Person('tim');
person.name = 'sofie';       // setter wordt aangeroepen
console.log(person.name);    // getter wordt aangeroepen
```

Classes: setters & getters

Geen underscores in fields: onderstaande code werkt niet!

```
class Person  {
    constructor(name) {
       this.name = name;
    }
    set name (name) {
        this.name = name;   // setter voor name w. aangeroepen
    }                       // oneindige lus -> stack overflow
    get name () {
        return this.name;
    }
};


person = new Person('tim');
person.name = 'sofie';
```

Classes: inheritance

```
class Shape{
    constructor(x,y){
        this.x=x;
        this.y=y;
    }
    describe(){
        console.log(`Shape at (${this.x}, ${this.y}) `);
    }
    static describe(){
        console.log('Shape');
    }
}
let shape1 = new Shape(1,2);
shape1.describe();  // Shape at (1, 2)
Shape.describe();   // Shape
```

```
class Circle extends Shape {
    constructor(x, y, radius) {
        super(x, y);
        this.radius = radius;
    }
    describe(){
        super.describe();
        console.log('Circle');
        console.log(`My radius is ${this.radius} `);
    }
    static describe(){
        super.describe();
        console.log('Circle');
    }
}

let circle1=new Circle(4,5,6);
circle1.describe();
// Shape at (4, 5)
// Circle
// My radius is 6
Circle.describe();
// Shape
// Circle
```

Classes: instanceof

```
console.log(shape1 instanceof Shape);
// true
console.log(shape1 instanceof Circle);
// false
console.log(circle1 instanceof Shape);
// true
console.log(circle1 instanceof Circle);
// true
```

Besluit

Elk object is verbonden met een prototype

new: gebruik een function om een object te maken

Object.create: maak een object met als prototype een bestaand object

Eenvoudigere syntax adhv classes