OOPS in Nodejs

Index

- BackGround
  - Data-Types
  - Objects
- OOPS Concept
- OOPS in Node
- Exercise
- Key Take-away

Data Types

- NULL
- Undefined
- Boolean
- Numbers
- String
- Symbol (ES-6)
- Objects

```
var id1 = Symbol("id");
var id2 = Symbol("id");

console.log(id1 === id2);
```

```
var myObject = {
    a: 1,
    b: "hi",
    c: function random(){},
    d: {}
};
```

Paytm

Property Flags

- writable
- enumerable
- configurable

Special Objects

- Functions

```
> function baseObject(){}
```

Name	baseObject
Length	0
Prototype	…

writable
enumerable
configurable

- Arrays

```
> var array = new Array(3)
> array[0] = “Paytm”
```

0	Paytm
Length	1

Paytm

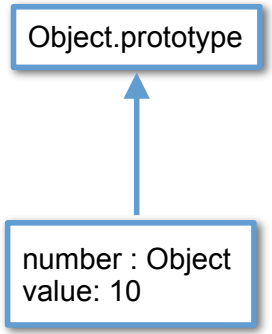
Object

- Creation
  - new Object()
  - Object.create(<other object>)
  - Object.assign(User, <other1>, <other2>) [ES-6]
- Property
  - Object.getOwnPropertyDescriptors(<obj>)
  - Object.getPrototypeOf(<obj>)

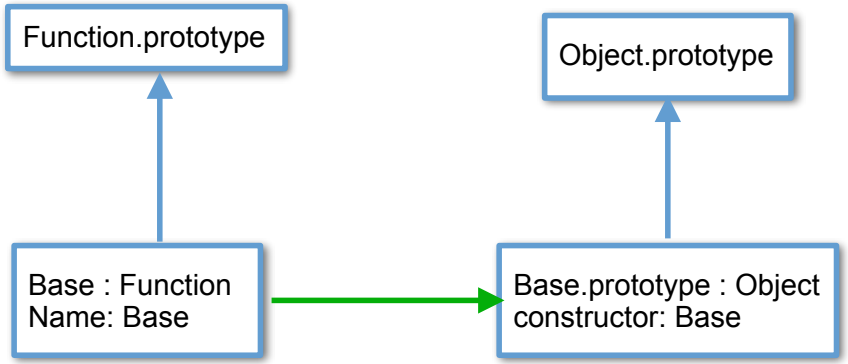
Object Prototype

- [[prototype]]
  - __proto__
- prototype

```
> var number = 10;
> number.toString()
'10'
```




```
> function Base(){}
```



OOPS concept

- Class
- Objects
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Exception Handling

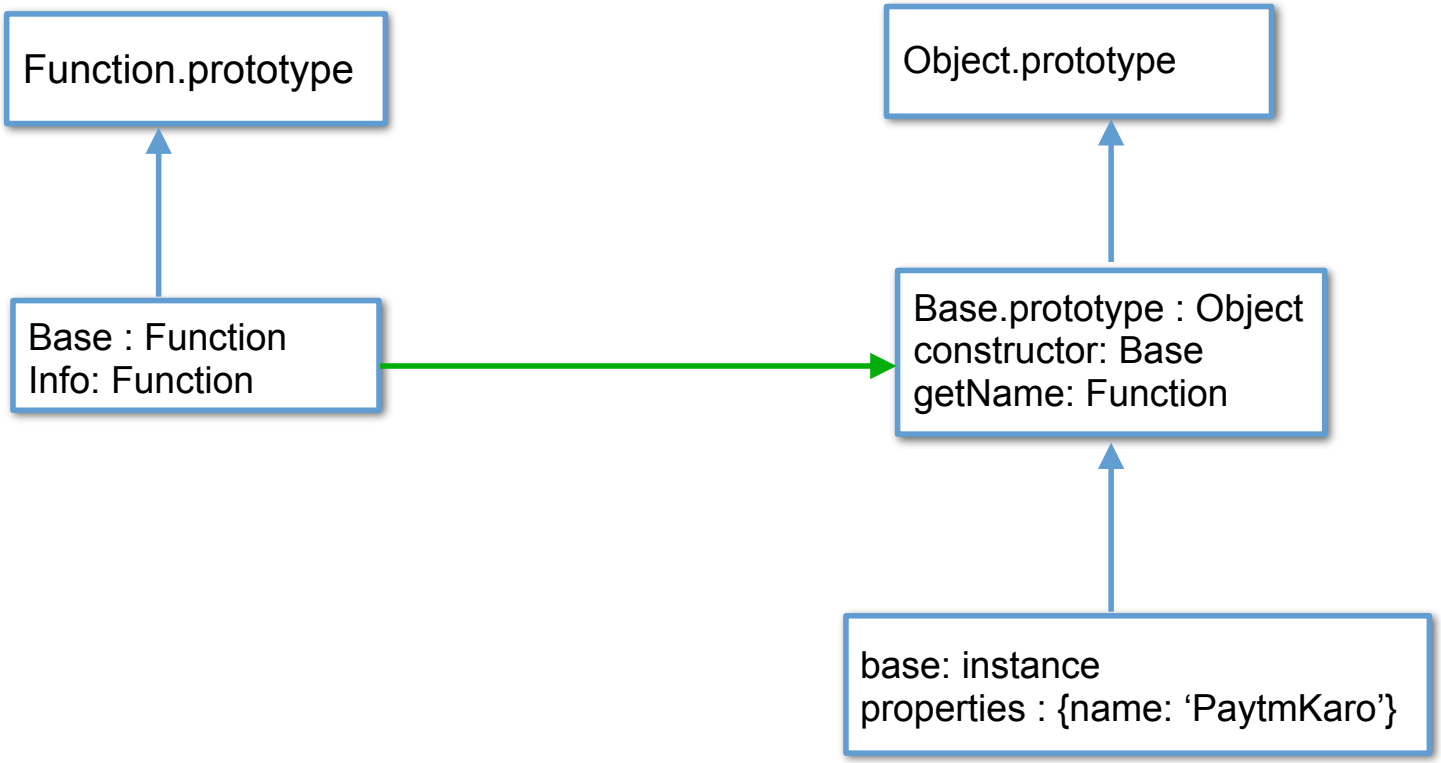
Class Definition

```
// Classical way
function Base(properties){
    this.properties = properties;
}

Base.prototype = {
    constructor: Base,
    getName: function getName(){
        return this.properties.name;
    }
};

Base.Info = function(){
    return "This is base Object";
};
```

```
// ES6
class Base {
    constructor(properties) {
        this.properties = properties;
    }
    getName(){
        return this.properties.name;
    }

    static Info(){
        return "This is ES-6 Base Object";
    }
}
```

```
// Usage :-
var base = new Base({name: "Paytm Karo!"});
console.log('Movie Name ' + base.getName());
```

Prototype Chain



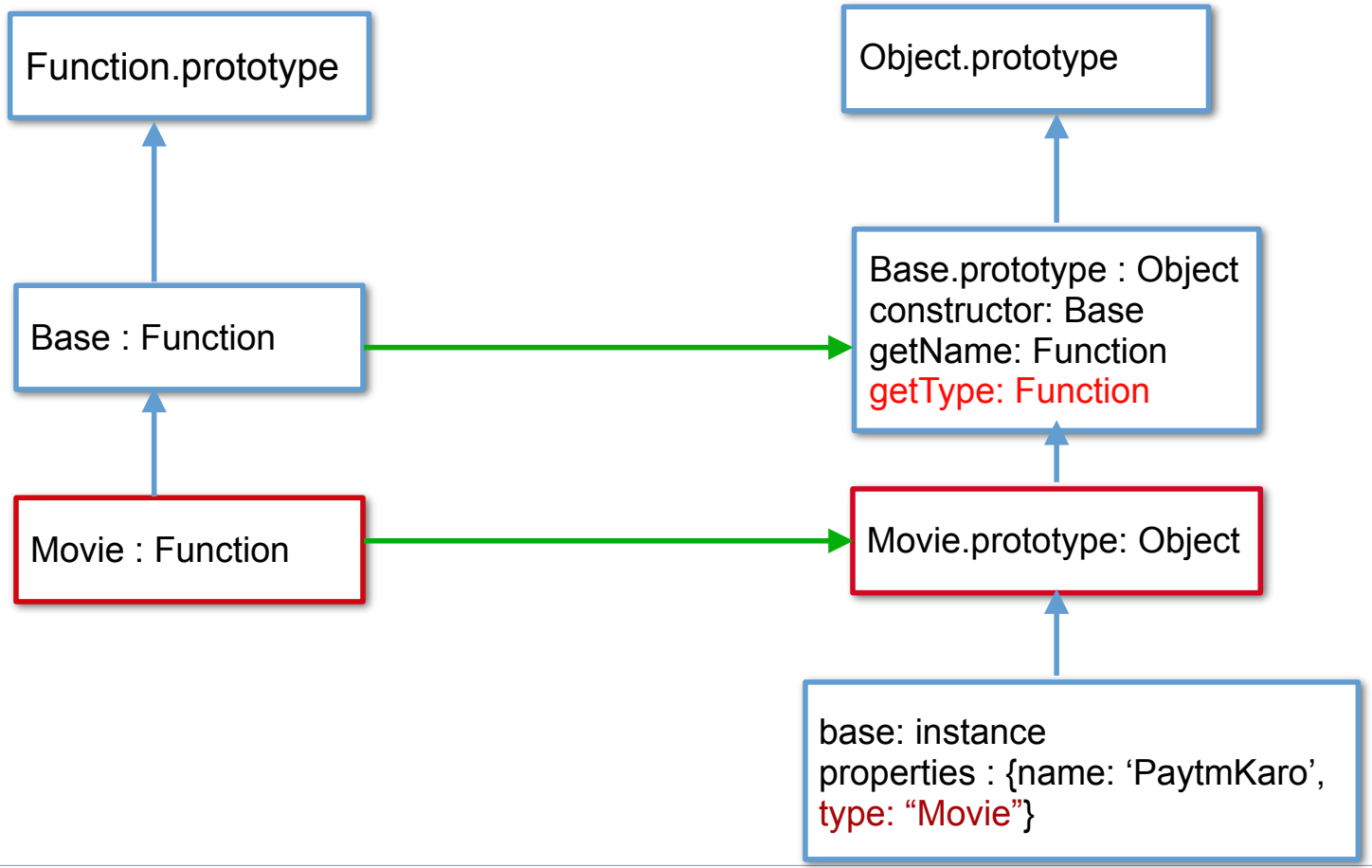
Inheritance

```
var util = require("util");
function Base(properties){
    this.properties = properties;
}

Base.prototype = {
    constructor: Base,
    getName: function getName(){
        return this.properties.name;
    },
    getType: function getType(){
        return this.properties.type;
    }
};

function Movie(properties){
    properties.type = "Movie";
    Base.call(this, properties);
}
util.inherits(movieObject, baseObject);
```

```
class Base{
    constructor(properties){
        this.properties = properties;
    }

    getName(){
        return this.properties.name;
    }

    getType(){
        return this.properties.type;
    }
}

class Movie extends Base{
    constructor(properties){
        properties.type = "Movie";
        super(properties);
    }
}
```

```
var movie = new Movie({"name": "Paytm Karo!"});
console.log("Object Type is = ",movie.getType());
```

Prototype Chain



Multiple Inheritance

- Possible?
- Mixins

Polymorphism

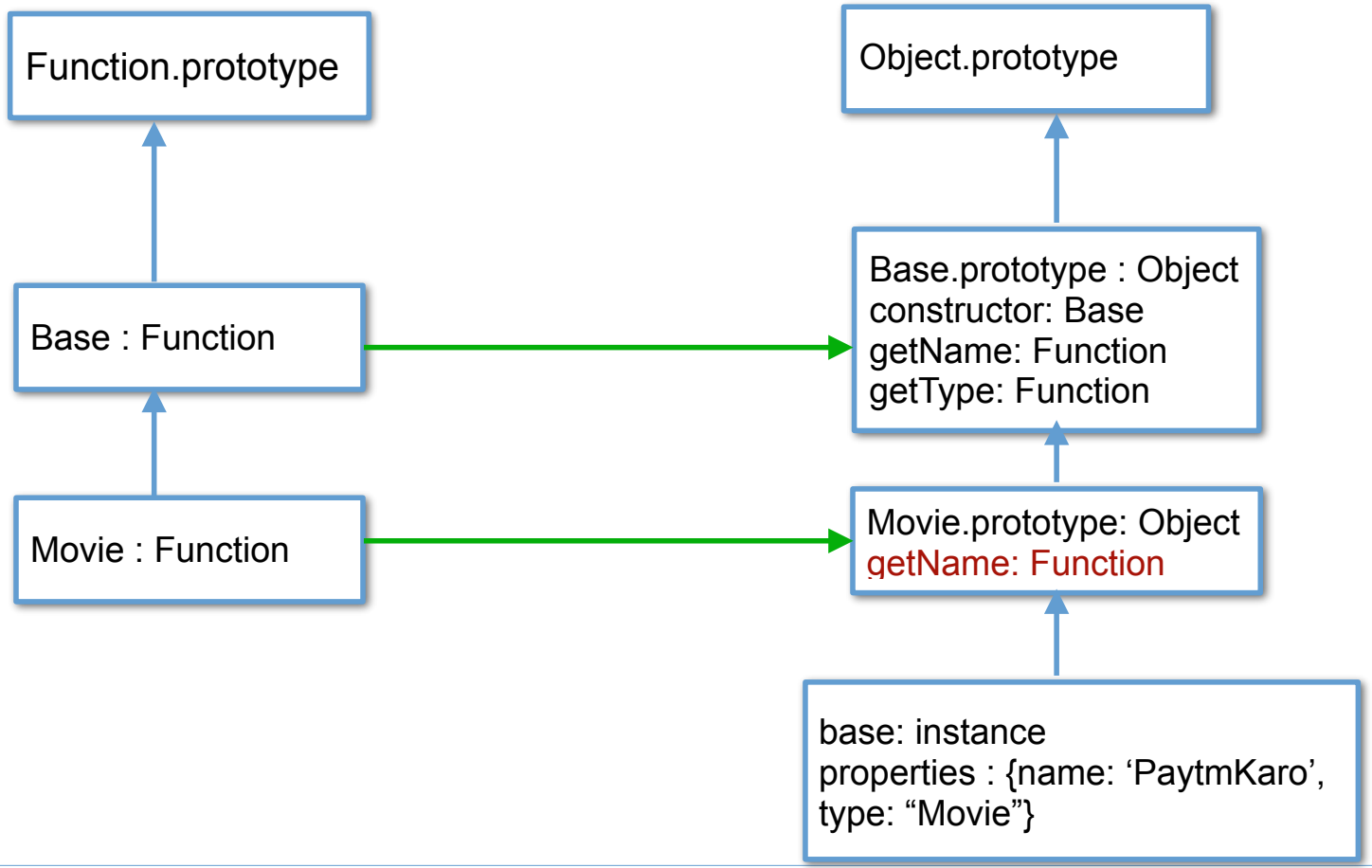
- Compile time - No
- Run Time
  - Method over-riding

- Run Time (method overriding)

```
Movie.prototype.getName =  function getName(){
    return "Movie: " +
Movie.super_.prototype.getName.apply(this);
};


var movie = new Movie({"name": "Paytm
Karo!"});
console.log("Object Type is =
",movie.getType());
console.log("Movie Name = ", movie.getName());
```

```
class Movie extends Base{
    constructor(properties){
        properties.type = "Movie"
        super(properties);
    }

    getName(){
        return "Movie: " + super.getName();
    }
}
```

Prototype Chain



Question-1

```
class Movie{
    constructor(properties){
        var Id = 1;
        this.properties = properties;
    }

    getName(){
        return this.properties.name;
    }
}
```

```
var movie = new Movie({name: "Paytm Karo!"});
console.log(movie.properties);
console.log(movie.Id);
```

Access Identifier

```
class Movie{
    constructor(properties){
        var Id = 1;

        this.properties = properties;
    }

    getName(){
        return this.properties.name;
    }
}
```

Private Variable

Public Variable

```
var movie = new Movie({name: "Paytm Karo!"});
console.log('Movie Name ' + movie.getName());
console.log(movie.properties);
```

Private Variable

- Environment of constructor
- Naming convention (prefix variable with _)
- WeakMaps [ES-6]
- Symbols [ES-6]

Paytm

```
// Constructor Environment
class Movie{
    constructor(i){
        var Id = i;
        this.getId = function(){
            return Id;
        };
    }
}
```

```
// Naming Convention
class Base{
    constructor(i){
        this._Id = i;
    }
    getId(){
        return this._Id;
    }
}
```

```
// WeakMap
const _Id = new WeakMap();
class Movie{
    constructor(i){
        var Id = i;
        _Id.set(this, Id);
    }
    getId(){
        return _Id.get(this);
    }
}
```

```
// Symbol
const _Id = Symbol("id");
class Movie{
    constructor(i){
        this[_Id] = i;
    }
    getId(){
        return this[_Id]
    }
}
```

Paytm

Question-2

```
class Base{
    constructor(properties){
        this.properties = properties;
    };
    getName(){
        return this.properties.name;
    }
    getType(cb){
        return this.properties.type;
    }
    getNameType1(cb){
        return this.getName()+"-"+this.getType();
    }
    getNameType2(cb){
        var getName = this.getName;
        var getType = this.getType;
        return getName()+"-"+getType();
    }
};
```

```
var properties = {};
properties.name = "Paytm Karo";
properties.type = "Movie";

var base = new Base(properties);
console.log(base.getName());
console.log(base.getNameType1());
console.log(base.getNameType2());
```

this

- It is unbound
- it is always the object before dot

Paytm

Context-Bind

```
class Base{
    constructor(properties){
        this.properties = properties;
    };
    getName(){
        return this.properties.name;
    }
    getType(cb){
        return this.properties.type;
    }
    getNameType1(cb){
        return this.getName()+"-"+this.getType();
    }
    getNameType2(cb){
        var getName = this.getName.bind(this);
        var getType = this.getType.bind(this);
        return getName()+"-"+getType();
    }
};
```

```
var properties = {};
properties.name = "Paytm Karo";
properties.type = "Movie";

var base = new Base(properties);
console.log(base.getName());
console.log(base.getNameType1());
console.log(base.getNameType2());
```

Exception Handling

- throw new “<message>”
- Extend Error class

Paytm

Patterns - Singleton

```
var _singleton = null;
class MovieValidator {
    constructor (configs) {
        if(!_singleton) {
            this.configs = configs;
            _singleton = this
        }
        else
            return _singleton;
    }

    validateMovie(movie){ ... }
}

module.exports = MovieValidator;
```

```
var movieValidator1 = new MovieValidator({});
var movieValidator2 = new MovieValidator({});
console.log(movieValidator1 === movieValidator2)
```

Paytm

Object/Connection Pool

```
class ConnectionPool{
    constructor(size){
        this.connections = []
        for(var i=0;i < size; i++){
            this.connections.push(new Connection())
        }
    }
    // return null if this.connections is empty
    // return connection instance and remove it from this.connections
    acquire(){ ... }

    // Add ConnectionInstance into this.connections
    release(ConnectionInstance){ ... }


class Connection {
    constructor(){ ... }
    // action to perform on connection
    action() { ... }
}
```

```
// initialize 10 object
var connections = ConnectionPool(10);
var connection = connections.acquire();
```

New in ES-6

- const
- arrow functions
- default parameter values
- string interpolation
- class definition
- class inheritance
  - can extend built-in Objects like Array
- Object.assign

Exercise

- Problem Statement:- Write a utility which supports following things
  - Parsing CSV file
  - Adding fields
  - Generating XML

```
/**
 * @param file  :- file to read
 * @param config  :- config required to
parse csv file
 * @param cb :- cb function which will
be called after parsing or in case of
error
 */
function readCSV(file, config, cb){
}

/**
 * @param jObject    :- jObject
 * @param field_path :- path at which
value will be added
 * @param value      :- value
 */
function addField(jObject, field_path,
value){
}

/**
 * Convert json object into xml
 * @param jObject
 */
function toXML(jObject){
}
```

```
/**
 * @param config
 * @constructor
 */
function CSV(config){
    this.config = config;
}

CSV.prototype = {
    constructor: CSV,
    /**
     * @param file
     * @param cb
     */
    readCSV: function readCSV(file, cb){},

    /**
     * @param field_path
     * @param value
     */
    addField: function addField(field_path,
value){},

    toXML: function toXML(){}
};
```

Take-Away

- When to create classes?
  - Logical separation of objects
  - Require multiple instances of an object
  - Common functionality
  - Need to maintain internal state at object level
  - Private scope

References

- http://javascript.info
- http://exploringjs.com/es6/ch_classes.html
- http://es6-features.org

??

Thank you!

Karan Jindal
+91 9619759212
karan.jindal@paytm.com