PROXY

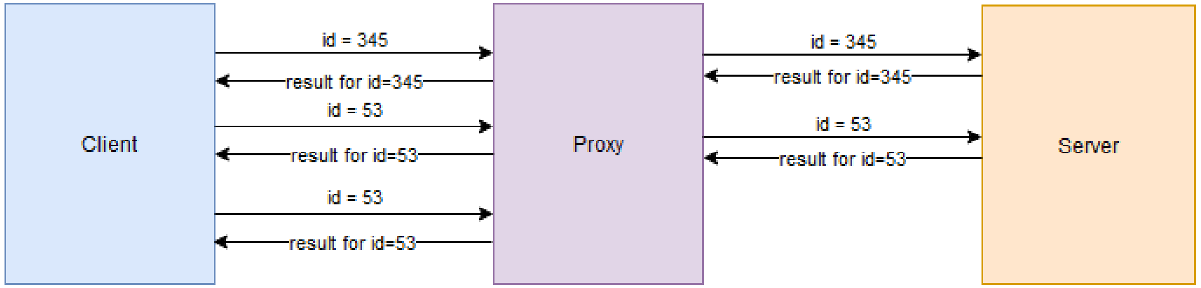
Przemysław Maćkowiak

Agenda

- PROXY

Structural design pattern: PROXY

- Rather than reference to a target directly, a client talks with its representative; a proxy
- Proxy works like an interface to a target object. It provides the same interface. Proxy manages the access to an object
- A client may even be not aware that it works with a proxy (e.g. a factory returns a proxy rather than a target object)
- There are situation where an object cannot be freely used (e.g. access control needed or when the initialization of the target object takes much time)

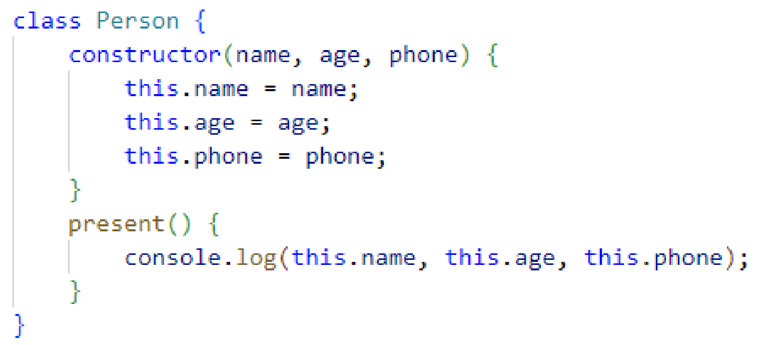


Usage

- Debugging purposes, analytics - logging data to a console/file/db/cloud/…
- Formatting – to assure consistent representation of data (”DD:MM:YYYY”, ”YYYY:MM:DD”)
- Validation – checking data in terms of allowed values (NaN, Infinity) and data types (*string* vs *number*)
- Caching – improves performance, avoid the same calculation, use stored result instead, especially useful if computation is expensive
- Other performance aspects
  - limiting a server load: joining a few HTTP requests into one
  - limiting a database load: restriction of the query number (no more than MAX)
- Implementation of access control - limited access to an object, depending on time or user rights
- Lazy initialization – defer the target object creation for the better time (when really needed)

An example

- A regular Person object on which Proxy pattern wil be presented

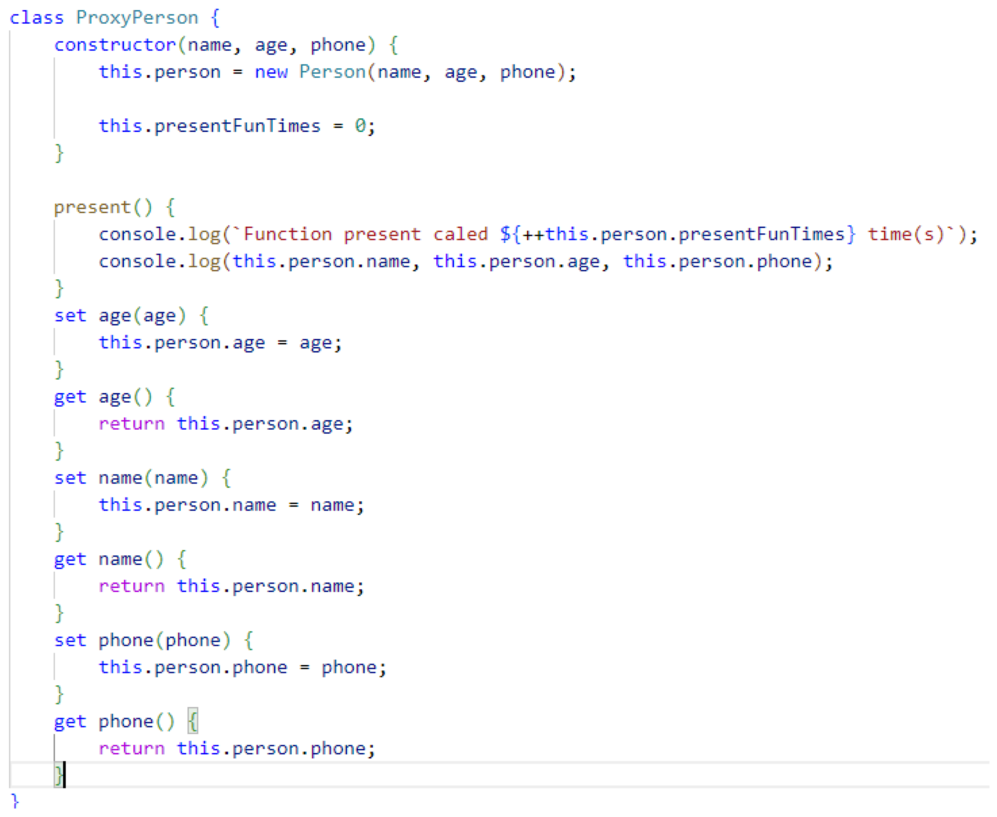


```
class Person {
    constructor(name, age, phone) {
        this.name = name;
        this.age = age;
        this.phone = phone;
    }
    present() {
        console.log(this.name, this.age, this.phone);
    }
}
```

- On the next silde we will see how the pattern may be applied for debugging purposes. To implement it, a Person representative is created that works on the behalf of an Person instance

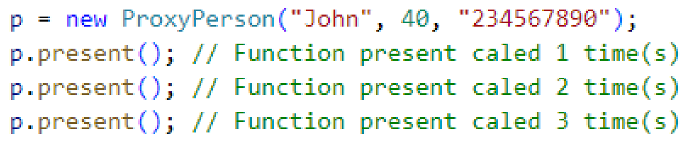
Proxy - debugging

Example: debugging



```
class ProxyPerson {
    constructor(name, age, phone) {
        this.person = new Person(name, age, phone);

        this.presentFunTimes = 0;
    }

    present() {
        console.log(`Function present caled ${++this.person.presentFunTimes} time(s)`);
        console.log(this.person.name, this.person.age, this.person.phone);
    }
    set age(age) {
        this.person.age = age;
    }
    get age() {
        return this.person.age;
    }
    set name(name) {
        this.person.name = name;
    }
    get name() {
        return this.person.name;
    }
    set phone(phone) {
        this.person.phone = phone;
    }
    get phone() {
        return this.person.phone;
    }
}
```

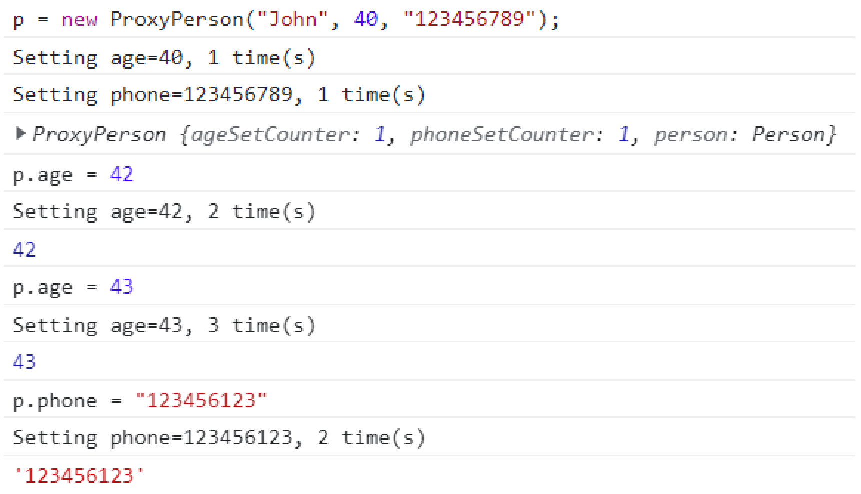
- Note, that the proxy works on the behalf of the target object
- It has the same props and a function



```
p = new ProxyPerson("John", 40, "234567890");
p.present(); // Function present caled 1 time(s)
p.present(); // Function present caled 2 time(s)
p.present(); // Function present caled 3 time(s)
```

Example: debugging

- Task
  - Take care right now about the setters of *age* and *phone*. Setting a value to these params should run a logger with info how many times the given param was set. See an example below:



```
p = new ProxyPerson("John", 40, "123456789");
Setting age=40, 1 time(s)
Setting phone=123456789, 1 time(s)
▶ProxyPerson {ageSetCounter: 1, phoneSetCounter: 1, person: Person}
p.age = 42
Setting age=42, 2 time(s)
42
p.age = 43
Setting age=43, 3 time(s)
43
p.phone = "123456123"
Setting phone=123456123, 2 time(s)
'123456123'
```

Solution

```
class ProxyPerson {
    constructor(name, age, phone) {
        this.ageSetCounter = 0;
        this.phoneSetCounter = 0;

        this.person = new Person();
        this.person.name = name;
        this.age = age;
        this.phone = phone;
    }
    present() {
        this.person.present();
    }
    set age(age) {
        console.log(`Setting age=${age}, ${++this.ageSetCounter} time(s)`);
        this.person.age = age;
    }
    get age() {
        return this.person.age
    }
    set phone(phone) {
        console.log(`Setting phone=${phone}, ${++this.phoneSetCounter} time(s)`);
        this.person.phone = phone;
    }
    get phone() {
        return this.person.phone;
    }
};
```

Scenario with all loggers

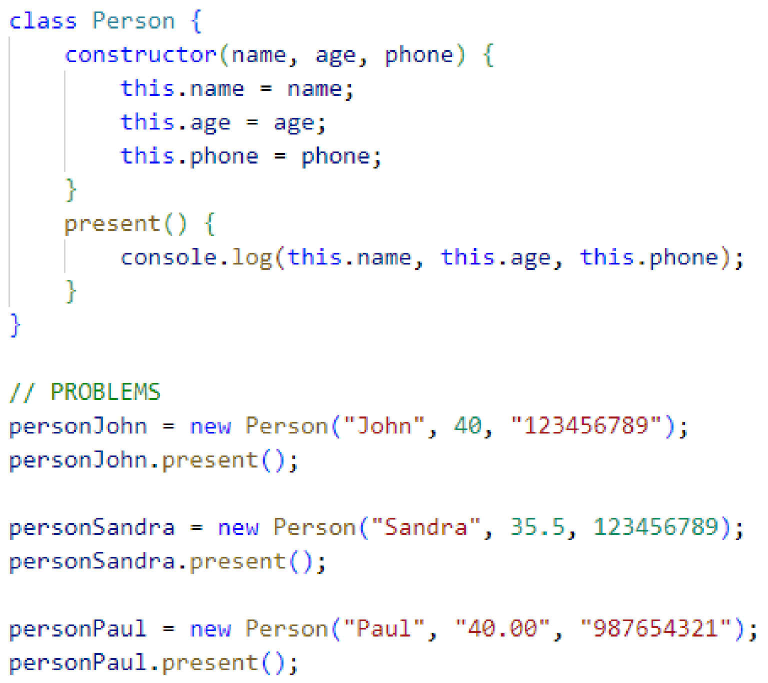
- Display info how many times the given prop was set or a function was called

```javascript
class ProxyPerson {
    constructor(name, age, phone) {
        this.person = new Person(name, age, phone);

        this.nameSetCounter = this.ageSetCounter = 0;
        this.phoneSetCounter = this.presentFuncCounter = 0;
    }
    present() {
        console.log(`present function called ${++this.presentFuncCounter} time(s)`);
        this.person.present();
    }
    set name(name) {
        console.log(`Setting name=${name}, ${++this.nameSetCounter} time(s)`);
        this.person.age = name;
    }
    get name() {
        return this.person.name;
    }
    set age(age) {
        console.log(`Setting age=${age}, ${++this.ageSetCounter} time(s)`);
        this.person.age = age;
    }
    get age() {
        return this.person.age;
    }
    set age(age) {
        console.log(`Setting age=${age}, ${++this.ageSetCounter} time(s)`);
        this.person.age = age;
    }
    get age() {
        return this.person.age;
    }
    set phone(phone) {
        console.log(`Setting phone=${phone}, ${++this.phoneSetCounter} time(s)`);
        this.person.phone = phone;
    }
    get phone() {
        return this.person.phone;
    }
};

personSandra = new ProxyPerson("Sandra", 35, "123456789");
personSandra.present();// present function called 1 time(s)

personSandra.age = 36; // Setting age=36, 1 time(s)
personSandra.age = 40; // Setting age=40, 2 time(s)
personSandra.phone = "572572572"; // Setting phone=572-572-572, 1 time(s)
personSandra.present(); // present function called 2 time(s),
```

Proxy - formatting

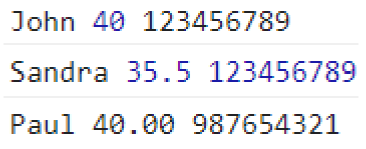
Usage - formatting

- Formatting issues, incorrect data type or inconssistent representation



```
class Person {
    constructor(name, age, phone) {
        this.name = name;
        this.age = age;
        this.phone = phone;
    }
    present() {
        console.log(this.name, this.age, this.phone);
    }
}

// PROBLEMS
personJohn = new Person("John", 40, "123456789");
personJohn.present();

personSandra = new Person("Sandra", 35.5, 123456789);
personSandra.present();

personPaul = new Person("Paul", "40.00", "987654321");
personPaul.present();
```

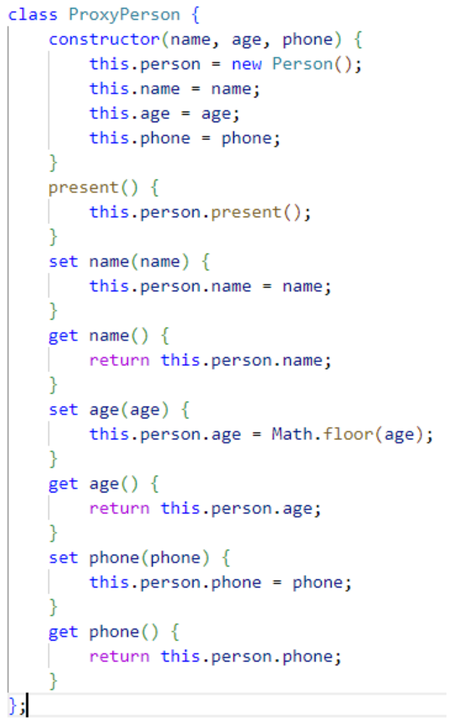
Another problem is a birthday YYYY:MM:DD, DD:MM:YYYY



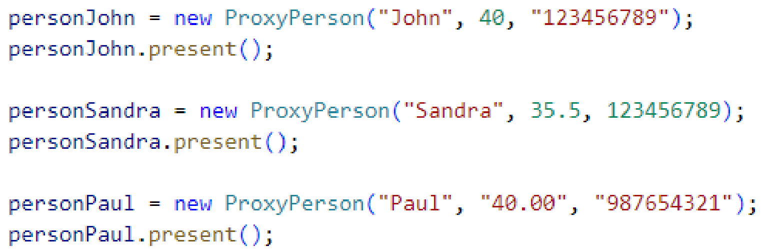
```
John 40 123456789
Sandra 35.5 123456789
Paul 40.00 987654321
```

Usage - formatting

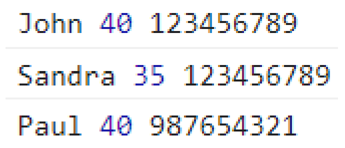
- See how the below code resolves the formatting issues related to *age* param



```
class ProxyPerson {
    constructor(name, age, phone) {
        this.person = new Person();
        this.name = name;
        this.age = age;
        this.phone = phone;
    }
    present() {
        this.person.present();
    }
    set name(name) {
        this.person.name = name;
    }
    get name() {
        return this.person.name;
    }
    set age(age) {
        this.person.age = Math.floor(age);
    }
    get age() {
        return this.person.age;
    }
    set phone(phone) {
        this.person.phone = phone;
    }
    get phone() {
        return this.person.phone;
    }
};
```



```
personJohn = new ProxyPerson("John", 40, "123456789");
personJohn.present();

personSandra = new ProxyPerson("Sandra", 35.5, 123456789);
personSandra.present();

personPaul = new ProxyPerson("Paul", "40.00", "987654321");
personPaul.present();
```

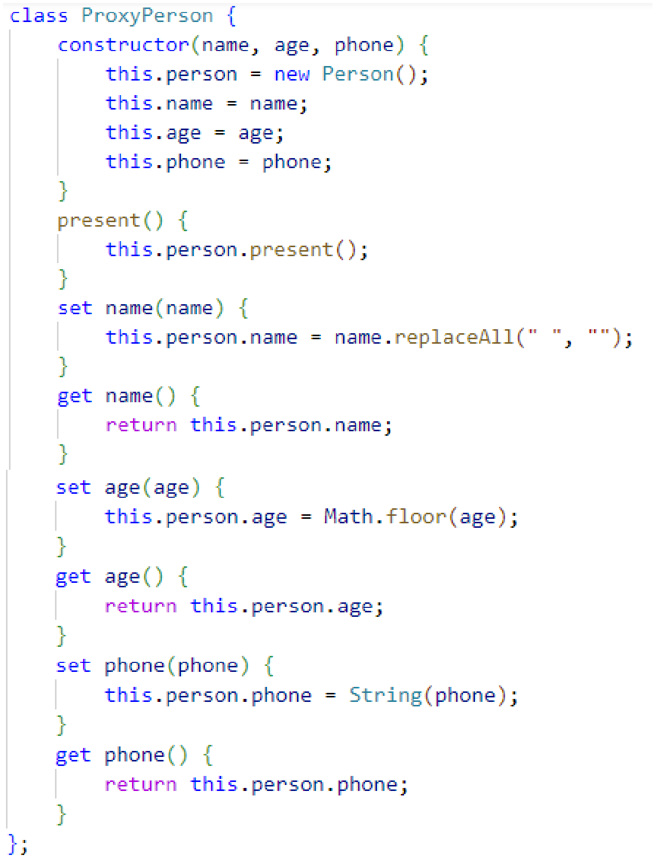


```
John 40 123456789
Sandra 35 123456789
Paul 40 987654321
```

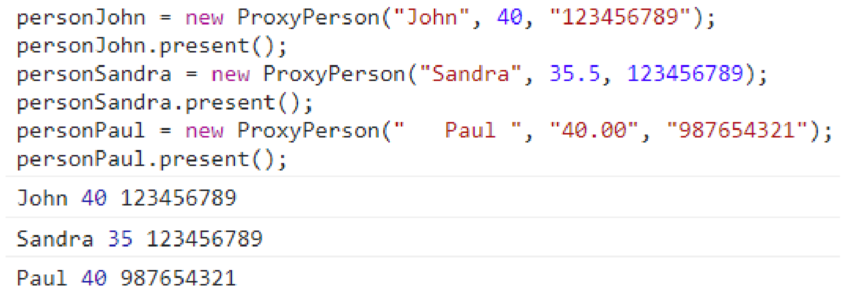
Usage - formatting

- Task
  - Introduce formatting functionality for:
    - name → remove accidental space character
    - phone → common representation as a string

Solution



```
class ProxyPerson {
    constructor(name, age, phone) {
        this.person = new Person();
        this.name = name;
        this.age = age;
        this.phone = phone;
    }
    present() {
        this.person.present();
    }
    set name(name) {
        this.person.name = name.replaceAll(" ", "");
    }
    get name() {
        return this.person.name;
    }
    set age(age) {
        this.person.age = Math.floor(age);
    }
    get age() {
        return this.person.age;
    }
    set phone(phone) {
        this.person.phone = String(phone);
    }
    get phone() {
        return this.person.phone;
    }
};
```



```
personJohn = new ProxyPerson("John", 40, "123456789");
personJohn.present();
personSandra = new ProxyPerson("Sandra", 35.5, 123456789);
personSandra.present();
personPaul = new ProxyPerson("   Paul ", "40.00", "987654321");
personPaul.present();
John 40 123456789
Sandra 35 123456789
Paul 40 987654321
```

Note, that the additional props to be validated require setting more setters and getters

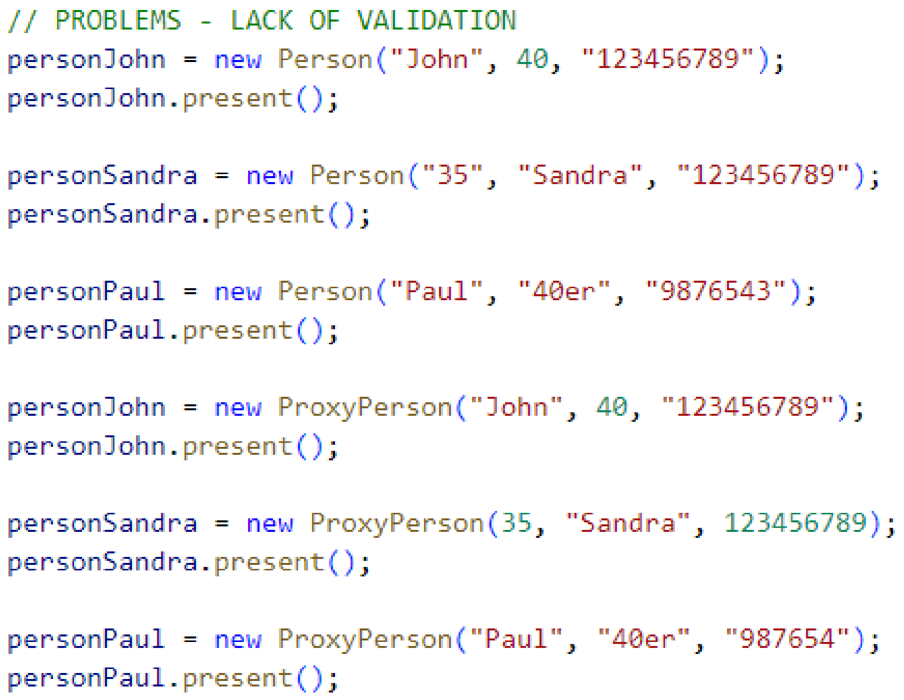
Usage - formatting

- Other use case of formatting: writing consistent data to a storage (data base / file / cloud)
- Rather than change of a source code of an imported library; 1) make a proxy and 2) implement the correspondig *write* function

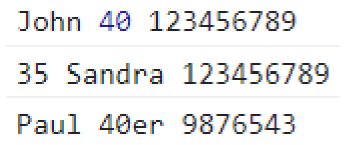
Proxy - validation

Usage - validation

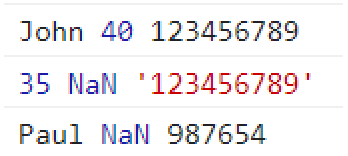
- Data should be validated before its content will be stored in an instance of *Person*



```
// PROBLEMS - LACK OF VALIDATION
personJohn = new Person("John", 40, "123456789");
personJohn.present();

personSandra = new Person("35", "Sandra", "123456789");
personSandra.present();

personPaul = new Person("Paul", "40er", "9876543");
personPaul.present();

personJohn = new ProxyPerson("John", 40, "123456789");
personJohn.present();

personSandra = new ProxyPerson(35, "Sandra", 123456789);
personSandra.present();

personPaul = new ProxyPerson("Paul", "40er", "987654");
personPaul.present();
```



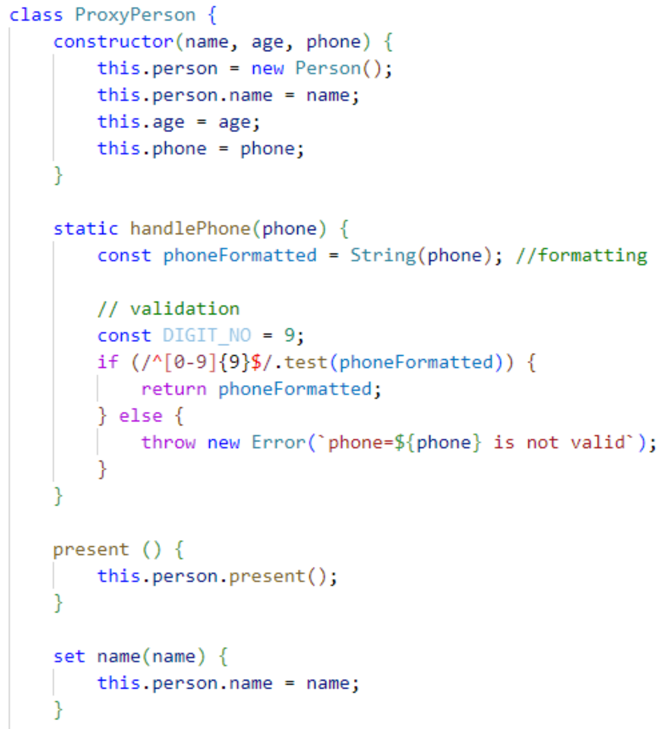
```
John 40 123456789
35 Sandra 123456789
Paul 40er 9876543
```



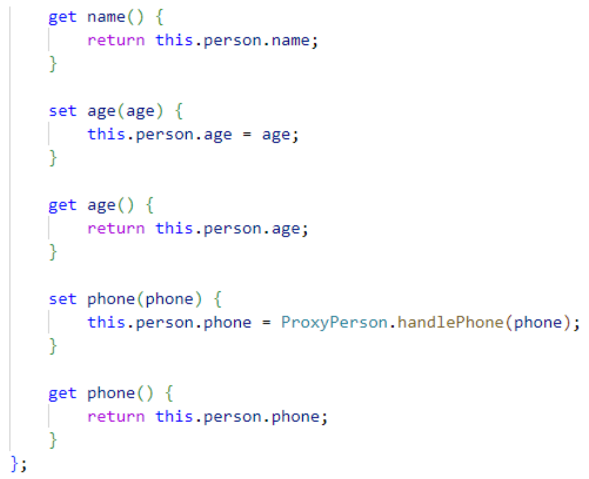
```
John 40 123456789
35 NaN '123456789'
Paul NaN 987654
```

Usage - validation

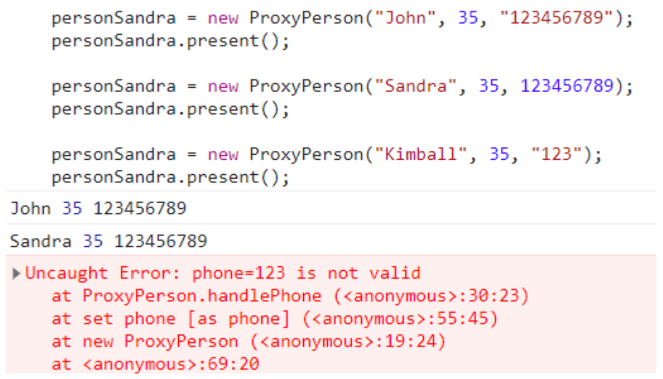
- Note how *phone* param is validated



```
class ProxyPerson {
    constructor(name, age, phone) {
        this.person = new Person();
        this.person.name = name;
        this.age = age;
        this.phone = phone;
    }

    static handlePhone(phone) {
        const phoneFormatted = String(phone); //formatting

        // validation
        const DIGIT_NO = 9;
        if (/^[0-9]{9}$/.test(phoneFormatted)) {
            return phoneFormatted;
        } else {
            throw new Error(`phone=${phone} is not valid`);
        }
    }

    present () {
        this.person.present();
    }

    set name(name) {
        this.person.name = name;
    }
```



```
    get name() {
        return this.person.name;
    }

    set age(age) {
        this.person.age = age;
    }

    get age() {
        return this.person.age;
    }

    set phone(phone) {
        this.person.phone = ProxyPerson.handlePhone(phone);
    }

    get phone() {
        return this.person.phone;
    }
};
```



```
personSandra = new ProxyPerson("John", 35, "123456789");
personSandra.present();

personSandra = new ProxyPerson("Sandra", 35, 123456789);
personSandra.present();

personSandra = new ProxyPerson("Kimball", 35, "123");
personSandra.present();
```

```
John 35 123456789
Sandra 35 123456789
▶Uncaught Error: phone=123 is not valid
    at ProxyPerson.handlePhone (<anonymous>:30:23)
    at set phone [as phone] (<anonymous>:55:45)
    at new ProxyPerson (<anonymous>:19:24)
    at <anonymous>:69:20
```

Usage - validation

- Task
  - Write validators for:
    - age → <1;130>
    - name → string that contains only letters

Solution

- Validators are welcomed

```
static handleAge(age) {
    const ageFormatted = Math.floor(age); // formatting

    //validation
    const MAX_AGE = 130, MIN_AGE = 1;
    if (!Number.isNaN(ageFormatted) && ageFormatted >= MIN_AGE &&
        ageFormatted <= MAX_AGE) {
        return age;
    } else {
        throw new Error(`age=${age} is not valid`);
    }
}
```

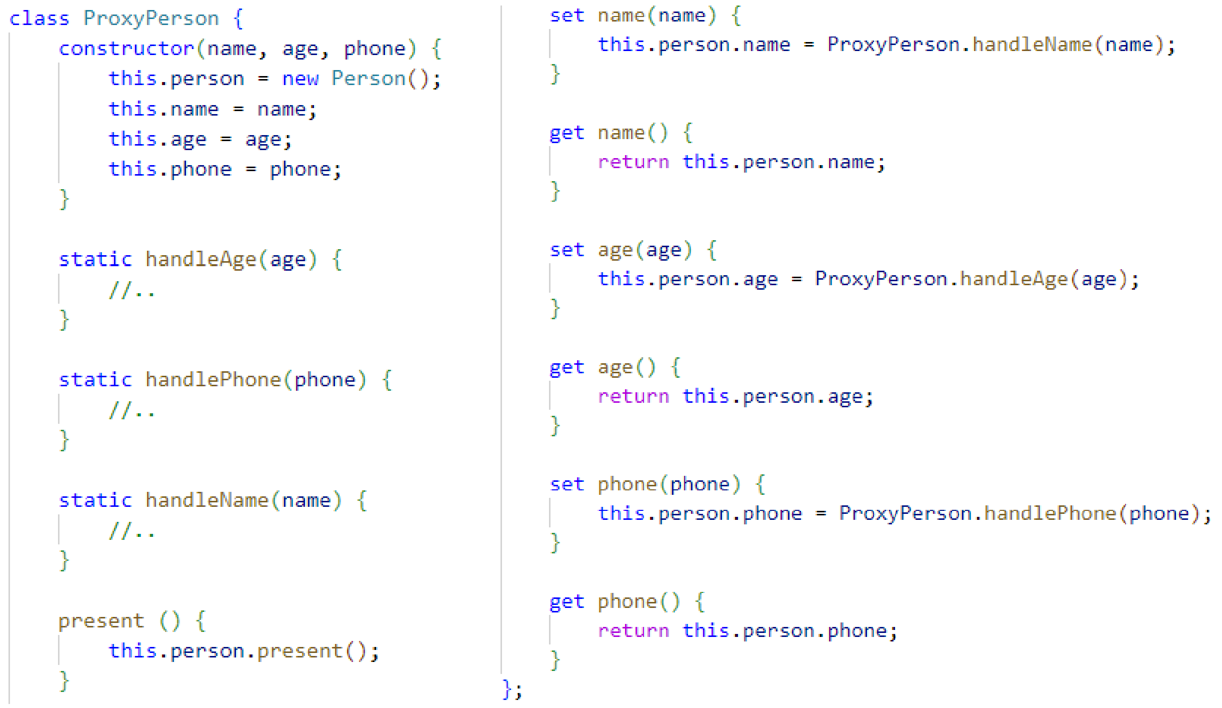
```
static handlePhone(phone) {
    const phoneFormatted = String(phone); //formatting

    // validation
    const DIGIT_NO = 9;
    if (/^[0-9]{9}$/.test(phoneFormatted)) {
        return phone;
    } else {
        throw new Error(`phone=${phone} is not valid`);
    }
}
```

```
static handleName(name) {
    if (typeof name === "String" &&
        /[a-zA-Z]+/.test(name)) {
        return name;
    } else {
        throw new Error(`name=${name} must be string, reg: [a-zA-Z]+`);
    }
}
```

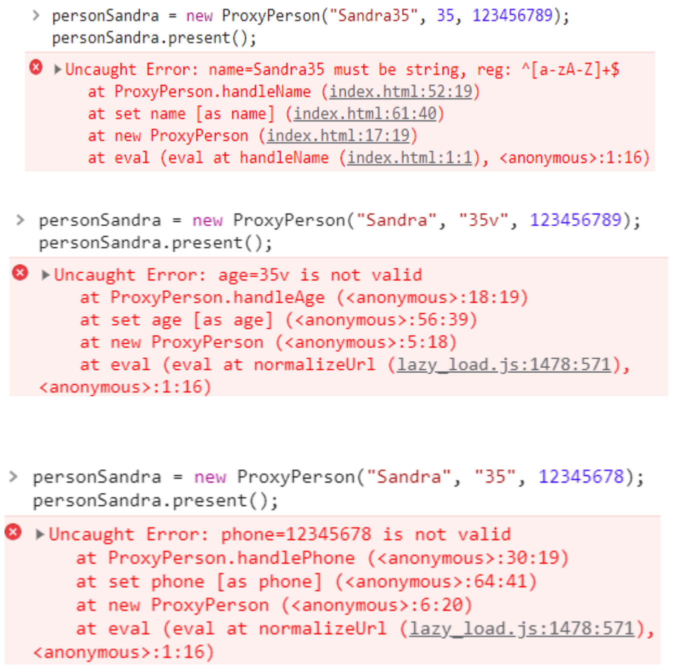
An exception is thrown in case of unwanted data

Usage - validation

- Below, the final code with validators and formatters



```
class ProxyPerson {
    constructor(name, age, phone) {
        this.person = new Person();
        this.name = name;
        this.age = age;
        this.phone = phone;
    }

    static handleAge(age) {
        //..
    }

    static handlePhone(phone) {
        //..
    }

    static handleName(name) {
        //..
    }

    present () {
        this.person.present();
    }

    set name(name) {
        this.person.name = ProxyPerson.handleName(name);
    }

    get name() {
        return this.person.name;
    }

    set age(age) {
        this.person.age = ProxyPerson.handleAge(age);
    }

    get age() {
        return this.person.age;
    }

    set phone(phone) {
        this.person.phone = ProxyPerson.handlePhone(phone);
    }

    get phone() {
        return this.person.phone;
    }
};
```

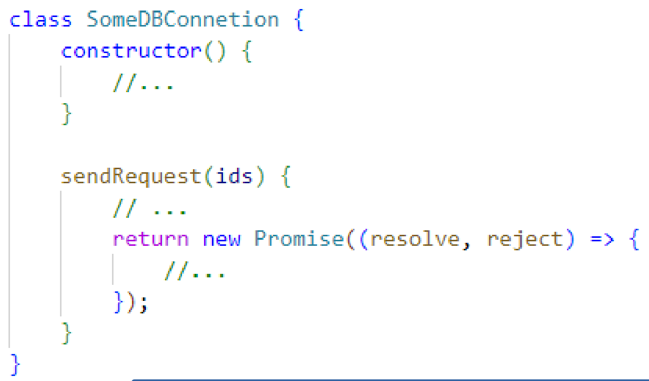


```
> personSandra = new ProxyPerson("Sandra35", 35, 123456789);
  personSandra.present();
⊗ ▸Uncaught Error: name=Sandra35 must be string, reg: ^[a-zA-Z]+$
      at ProxyPerson.handleName (index.html:52:19)
      at set name [as name] (index.html:61:40)
      at new ProxyPerson (index.html:17:19)
      at eval (eval at handleName (index.html:1:1), <anonymous>:1:16)

> personSandra = new ProxyPerson("Sandra", "35v", 123456789);
  personSandra.present();
⊗ ▸Uncaught Error: age=35v is not valid
      at ProxyPerson.handleAge (<anonymous>:18:19)
      at set age [as age] (<anonymous>:56:39)
      at new ProxyPerson (<anonymous>:5:18)
      at eval (eval at normalizeUrl (lazy_load.js:1478:571),
  <anonymous>:1:16)

> personSandra = new ProxyPerson("Sandra", "35", 12345678);
  personSandra.present();
⊗ ▸Uncaught Error: phone=12345678 is not valid
      at ProxyPerson.handlePhone (<anonymous>:30:19)
      at set phone [as phone] (<anonymous>:64:41)
      at new ProxyPerson (<anonymous>:6:20)
      at eval (eval at normalizeUrl (lazy_load.js:1478:571),
  <anonymous>:1:16)
```

Proxy – performance aspects
grouping HTPP requests into one

Usage - performance purposes

- A client queries a datbase about movies, either one by one or passing multiple ids
- A batch request is better from performance point of view

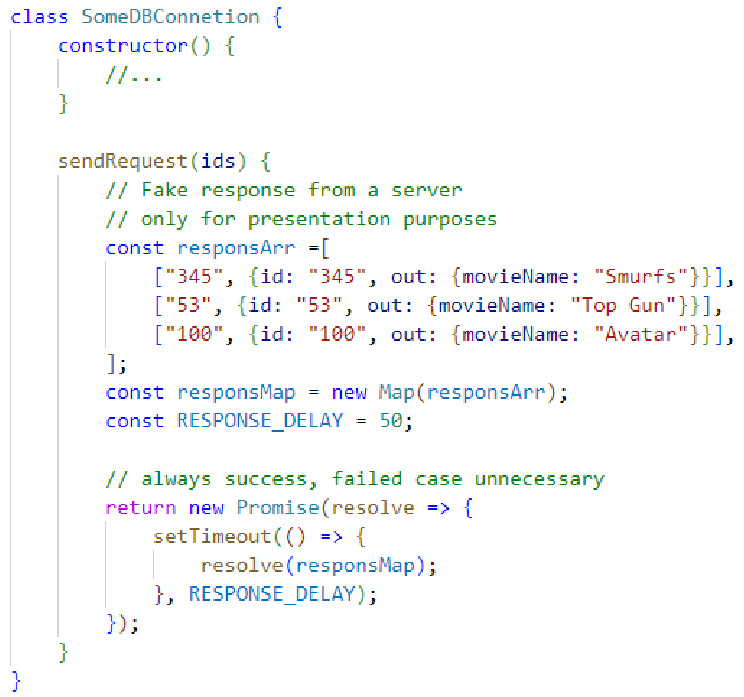


```
class SomeDBConnetion {
    constructor() {
        //...
    }

    sendRequest(ids) {
        // ...
        return new Promise((resolve, reject) => {
            //...
        });
    }
}
```

One can send a request one by one:
req1 = someDBConnection.sendRequest(”345”);
req2 = someDBConnection.sendRequest(”53”);
req3 = someDBConnection.sendRequest(”100”);

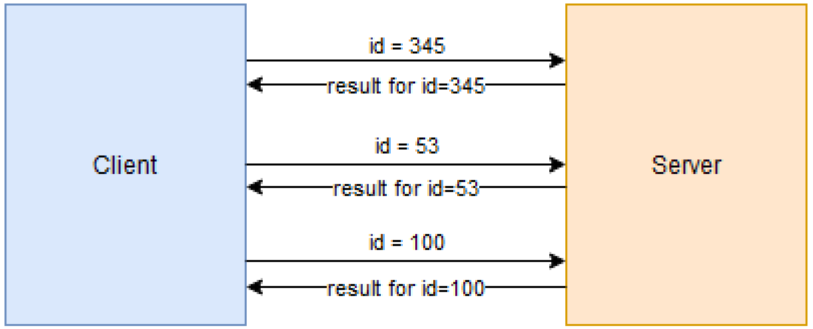
Or a single batch:
req = someDBConnection.sendRequest(”345, 53, 100”);

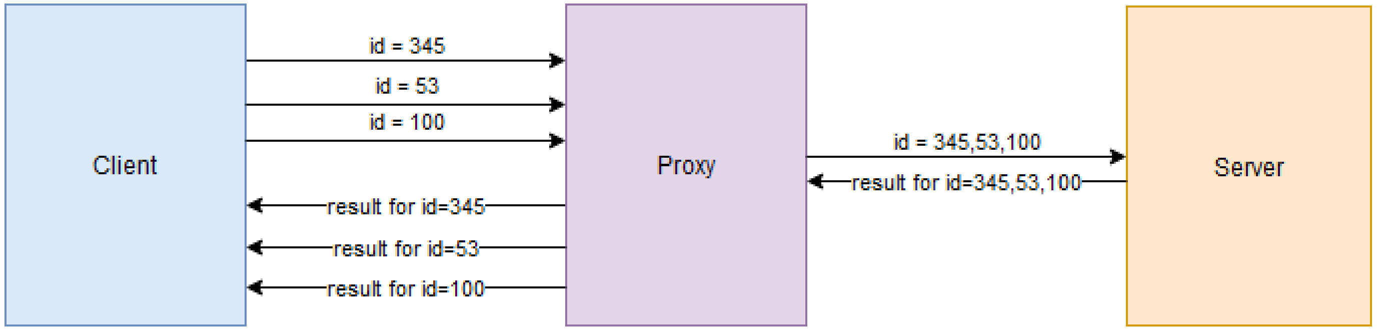
- There is some database whose fake API in JS is implemented below



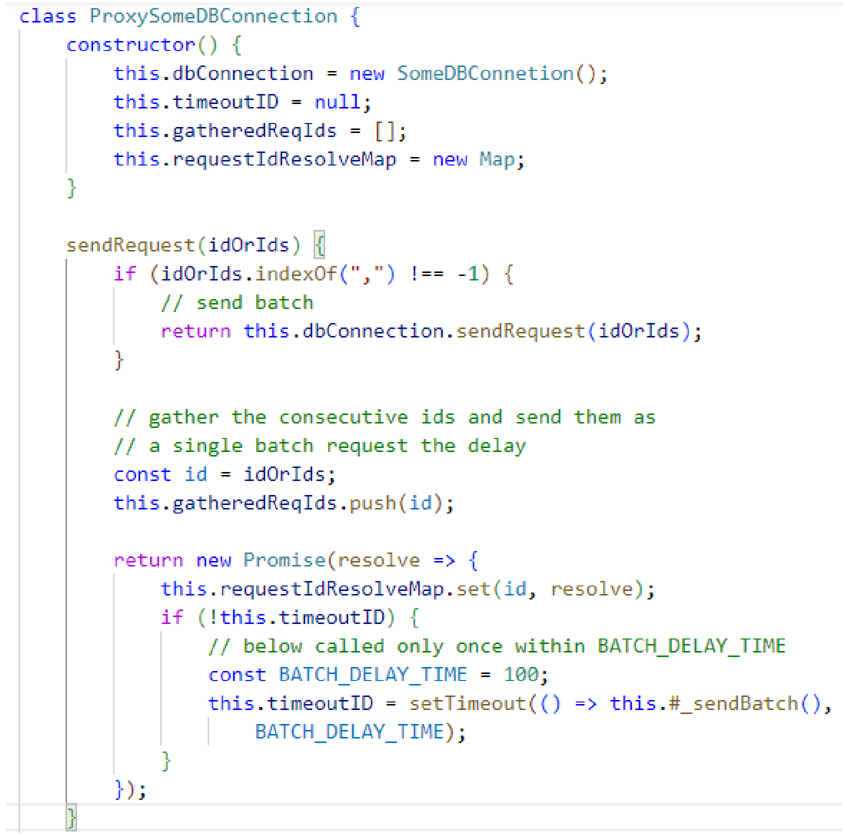
```
class SomeDBConnetion {
    constructor() {
        //...
    }

    sendRequest(ids) {
        // Fake response from a server
        // only for presentation purposes
        const responsArr =[
            ["345", {id: "345", out: {movieName: "Smurfs"}}],
            ["53", {id: "53", out: {movieName: "Top Gun"}}],
            ["100", {id: "100", out: {movieName: "Avatar"}}],
        ];
        const responsMap = new Map(responsArr);
        const RESPONSE_DELAY = 50;

        // always success, failed case unnecessary
        return new Promise(resolve => {
            setTimeout(() => {
                resolve(responsMap);
            }, RESPONSE_DELAY);
        });
    }
}
```

Usage - performance purposes

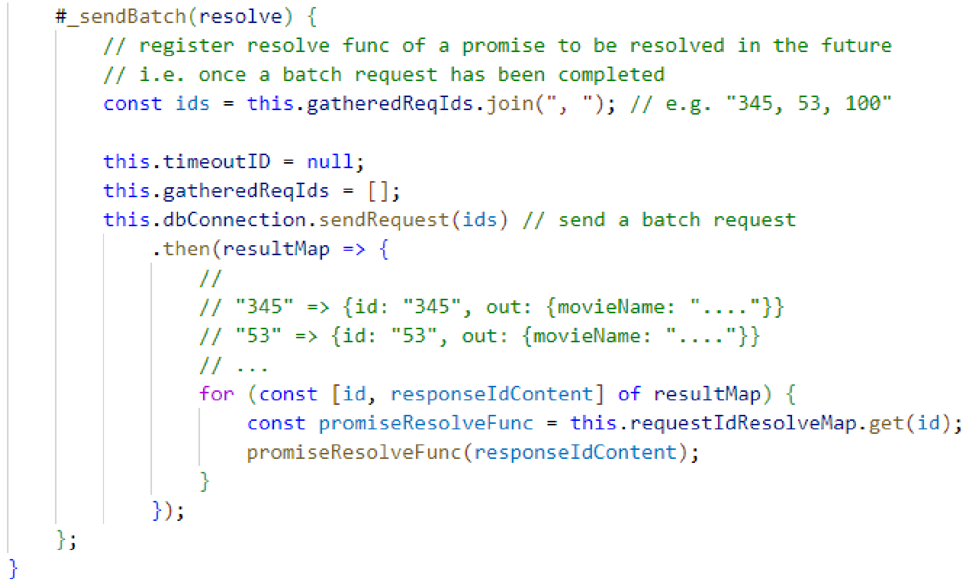




Usage - performance purposes



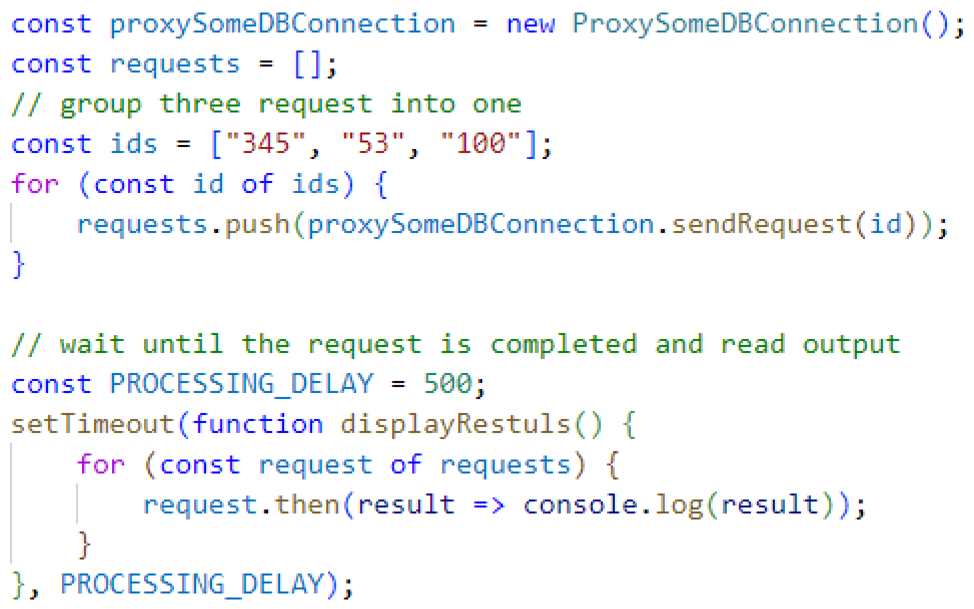
```
class ProxySomeDBConnection {
    constructor() {
        this.dbConnection = new SomeDBConnetion();
        this.timeoutID = null;
        this.gatheredReqIds = [];
        this.requestIdResolveMap = new Map;
    }

    sendRequest(idOrIds) {
        if (idOrIds.indexOf(",") !== -1) {
            // send batch
            return this.dbConnection.sendRequest(idOrIds);
        }

        // gather the consecutive ids and send them as
        // a single batch request the delay
        const id = idOrIds;
        this.gatheredReqIds.push(id);

        return new Promise(resolve => {
            this.requestIdResolveMap.set(id, resolve);
            if (!this.timeoutID) {
                // below called only once within BATCH_DELAY_TIME
                const BATCH_DELAY_TIME = 100;
                this.timeoutID = setTimeout(() => this.#_sendBatch(),
                    BATCH_DELAY_TIME);
            }
        });
    }
```



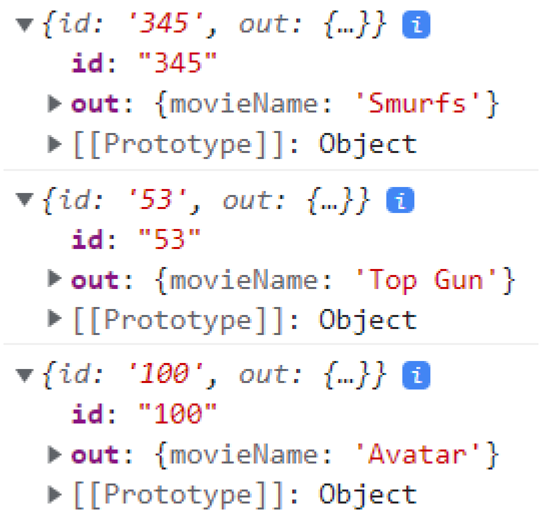
```
    #_sendBatch(resolve) {
        // register resolve func of a promise to be resolved in the future
        // i.e. once a batch request has been completed
        const ids = this.gatheredReqIds.join(", "); // e.g. "345, 53, 100"

        this.timeoutID = null;
        this.gatheredReqIds = [];
        this.dbConnection.sendRequest(ids) // send a batch request
            .then(resultMap => {
                //
                // "345" => {id: "345", out: {movieName: "...."}}
                // "53" => {id: "53", out: {movieName: "...."}}
                // ...
                for (const [id, responseIdContent] of resultMap) {
                    const promiseResolveFunc = this.requestIdResolveMap.get(id);
                    promiseResolveFunc(responseIdContent);
                }
            });
    };
}
```

Usage - performance purposes

- Requests are grouped into one. Every response in the code correposnds to a single promise. After some time it is resolved to the corresponding movie



```
const proxySomeDBConnection = new ProxySomeDBConnection();
const requests = [];
// group three request into one
const ids = ["345", "53", "100"];
for (const id of ids) {
    requests.push(proxySomeDBConnection.sendRequest(id));
}

// wait until the request is completed and read output
const PROCESSING_DELAY = 500;
setTimeout(function displayRestuls() {
    for (const request of requests) {
        request.then(result => console.log(result));
    }
}, PROCESSING_DELAY);
```

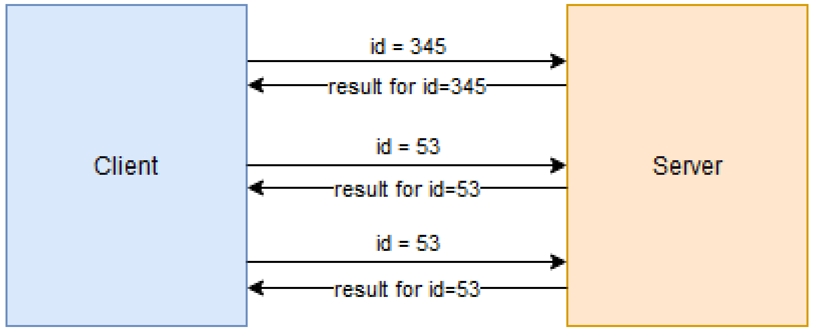


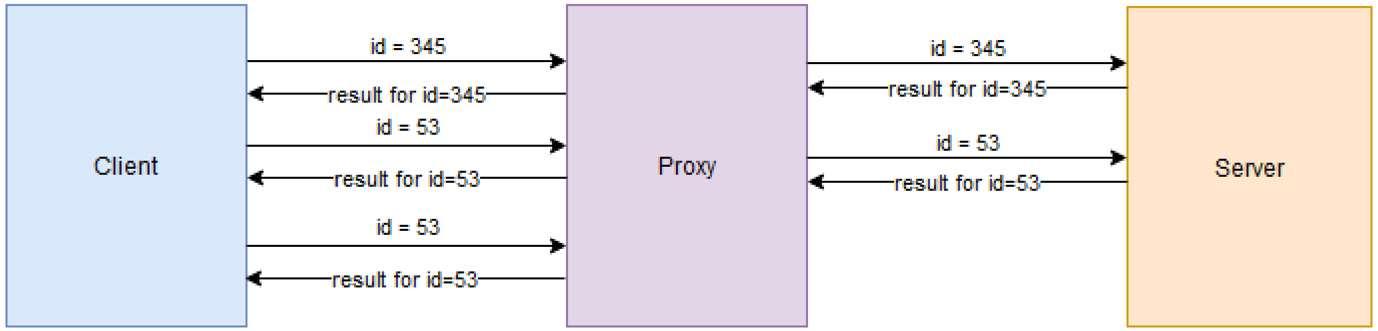
```
▼{id: '345', out: {…}} i
   id: "345"
 ▶out: {movieName: 'Smurfs'}
 ▶[[Prototype]]: Object
▼{id: '53', out: {…}} i
   id: "53"
 ▶out: {movieName: 'Top Gun'}
 ▶[[Prototype]]: Object
▼{id: '100', out: {…}} i
   id: "100"
 ▶out: {movieName: 'Avatar'}
 ▶[[Prototype]]: Object
```

Proxy - caching

Proxy and request caching

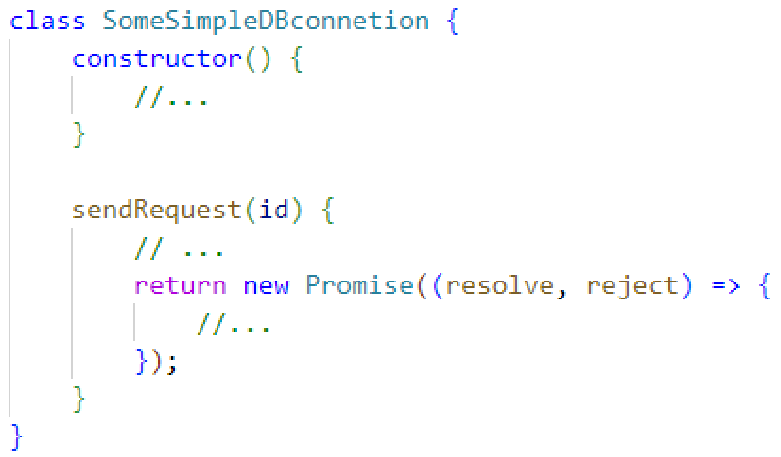
- The results are stored and reuse when the same request happens



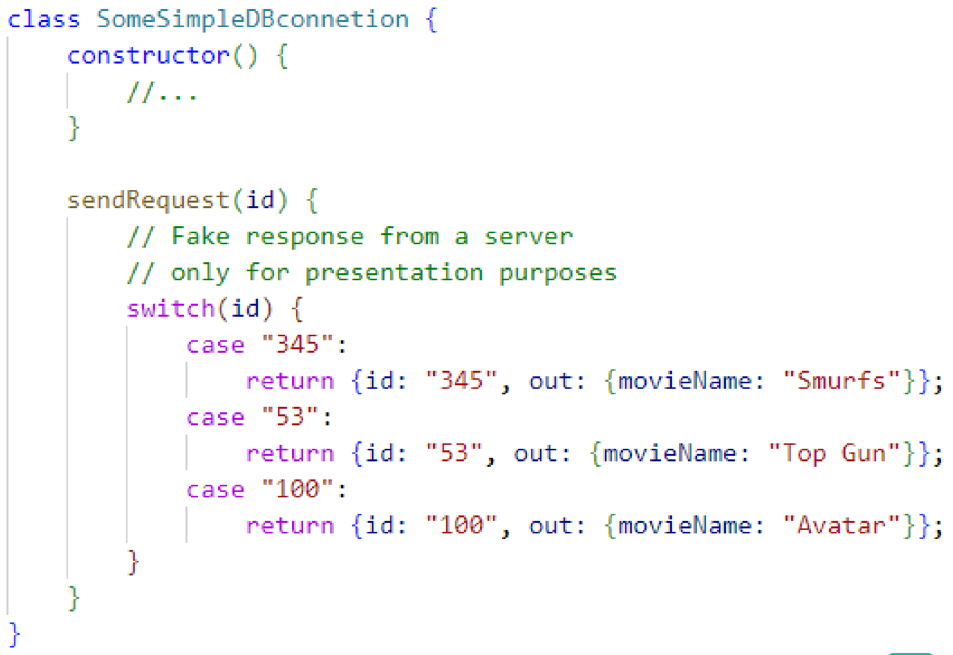


Usage - cache

- A fake interface on the right for presentation purposes



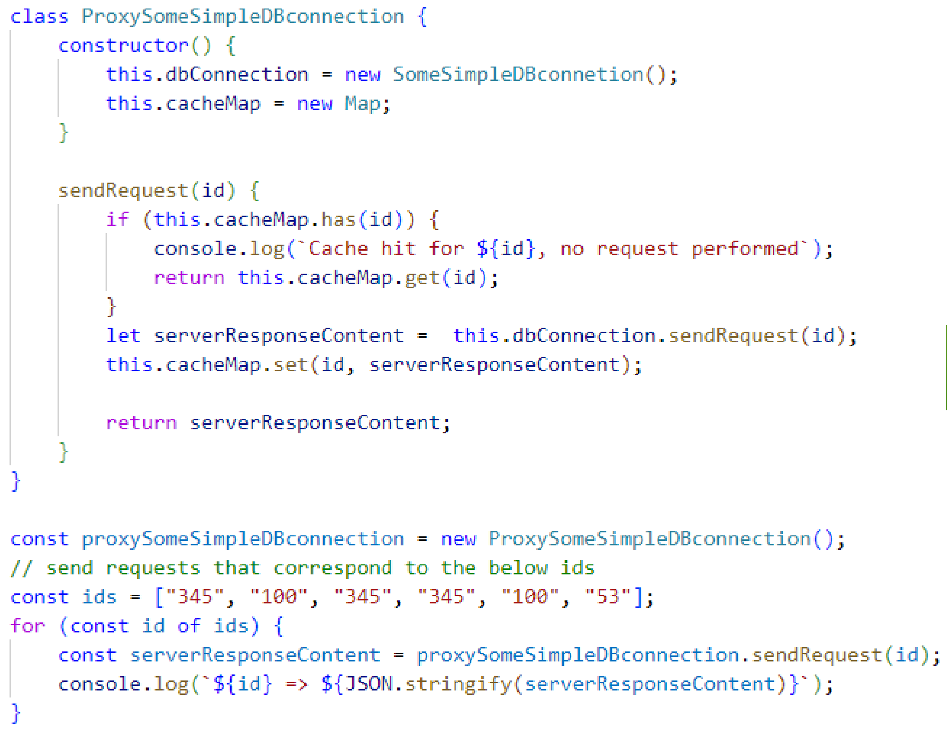
```
class SomeSimpleDBconnetion {
    constructor() {
        //...
    }

    sendRequest(id) {
        // ...
        return new Promise((resolve, reject) => {
            //...
        });
    }
}
```



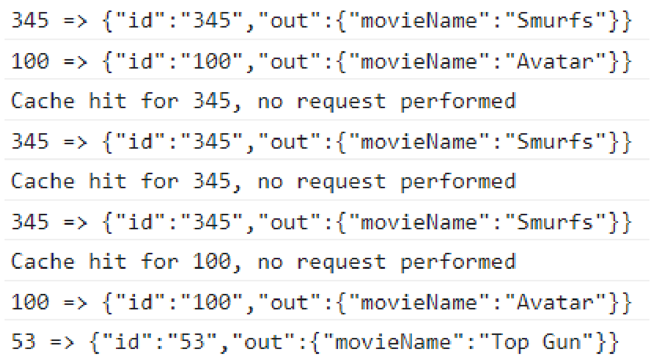
```
class SomeSimpleDBconnetion {
    constructor() {
        //...
    }

    sendRequest(id) {
        // Fake response from a server
        // only for presentation purposes
        switch(id) {
            case "345":
                return {id: "345", out: {movieName: "Smurfs"}};
            case "53":
                return {id: "53", out: {movieName: "Top Gun"}};
            case "100":
                return {id: "100", out: {movieName: "Avatar"}};
        }
    }
}
```

Usage - cache

- The results are stored and reuse when the same request happen



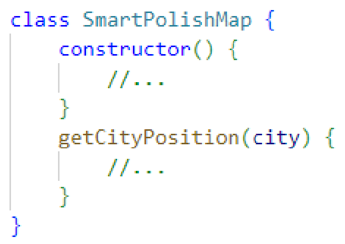
```
class ProxySomeSimpleDBconnection {
    constructor() {
        this.dbConnection = new SomeSimpleDBconnetion();
        this.cacheMap = new Map;
    }

    sendRequest(id) {
        if (this.cacheMap.has(id)) {
            console.log(`Cache hit for ${id}, no request performed`);
            return this.cacheMap.get(id);
        }
        let serverResponseContent =  this.dbConnection.sendRequest(id);
        this.cacheMap.set(id, serverResponseContent);

        return serverResponseContent;
    }
}

const proxySomeSimpleDBconnection = new ProxySomeSimpleDBconnection();
// send requests that correspond to the below ids
const ids = ["345", "100", "345", "345", "100", "53"];
for (const id of ids) {
    const serverResponseContent = proxySomeSimpleDBconnection.sendRequest(id);
    console.log(`${id} => ${JSON.stringify(serverResponseContent)}`);
}
```



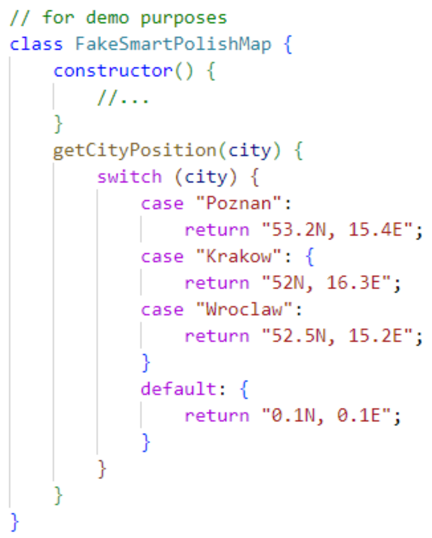
```
345 => {"id":"345","out":{"movieName":"Smurfs"}}
100 => {"id":"100","out":{"movieName":"Avatar"}}
Cache hit for 345, no request performed
345 => {"id":"345","out":{"movieName":"Smurfs"}}
Cache hit for 345, no request performed
345 => {"id":"345","out":{"movieName":"Smurfs"}}
Cache hit for 100, no request performed
100 => {"id":"100","out":{"movieName":"Avatar"}}
53 => {"id":"53","out":{"movieName":"Top Gun"}}
```

Usage - cache

- Task (the idea can be found here: https://www.dofactory.com/javascript/design-patterns/proxy)
  - There is a fake implementation of *SmartPolishMap* object. It is able to get the geographical location of cities. Propose a proxy that can cache resulsts to avoid doing the same requests to a server



```
class SmartPolishMap {
    constructor() {
        //...
    }
    getCityPosition(city) {
        //...
    }
}
```



```
// for demo purposes
class FakeSmartPolishMap {
    constructor() {
        //...
    }
    getCityPosition(city) {
        switch (city) {
            case "Poznan":
                return "53.2N, 15.4E";
            case "Krakow": {
                return "52N, 16.3E";
            case "Wroclaw":
                return "52.5N, 15.2E";
            }
            default: {
                return "0.1N, 0.1E";
            }
        }
    }
}
```

Solution

- The target object is not queried for the same requests

```
class ProxySmartPolishMap {
    constructor() {
        this.polishMap = new FakeSmartPolishMap();
        this.cityCache = new Map;
    }
    getCityPosition(city) {
        if (this.cityCache.has(city)) {
            console.log ("Query not performed, taking data from own cache");
            return this.cityCache.get(city);
        }
        const location = this.polishMap.getCityPosition(city);
        this.cityCache.set(city, location);

        return location;
    }
}
```

```
const smartProxyPolishMap = new ProxySmartPolishMap;
["Poznan", "Wroclaw", "Poznan", "Krakow", "Wroclaw"].forEach(city => {
    console.log(smartProxyPolishMap.getCityPosition(city));
});
```

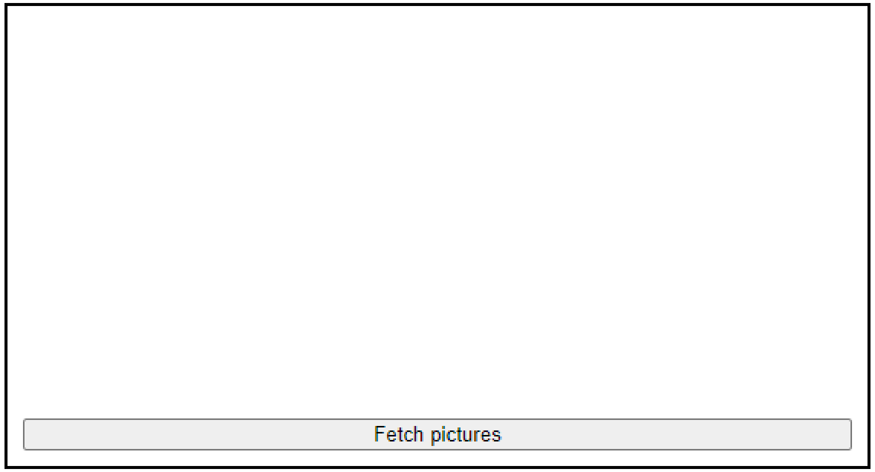
```
53.2N, 15.4E
52.5N, 15.2E
Query not performed, taking data from own cache
53.2N, 15.4E
52N, 16.3E
Query not performed, taking data from own cache
52.5N, 15.2E
```

Proxy - lazy initialization

Usage - lazy initialization

- In case of heavy objects, defer their creation to a moment when it is needed. Here, it is better to postpone the initialization, i.e. make it when *click* handler is executed

```
<body>
    <div id="container">
        <div id="pictureContainer"></div>
        <button>Fetch pictures</button>
    </div>


    <script src="http://-------.com/libs/pexelsAPI.js" ></script>
    <script>
        const cfg = {
            userID: "238sfSDFSF3",
        };

        // the below line is time consuming
        const pexels = new PexelsConnection(cfg);
        const fetchPictures = () => {
            // show spinner
            // ...
            const imgArr = pexels.getMyPictures();
            // hide spinner
            // ...
            // populate container with pictures
            //...
        };

        const btn = document.querySelector("button");
        btn.addEventListener("click", fetchPictures);
    </script>
```



propose a proxy that implements the lazy initialization (*click* event)

Solution

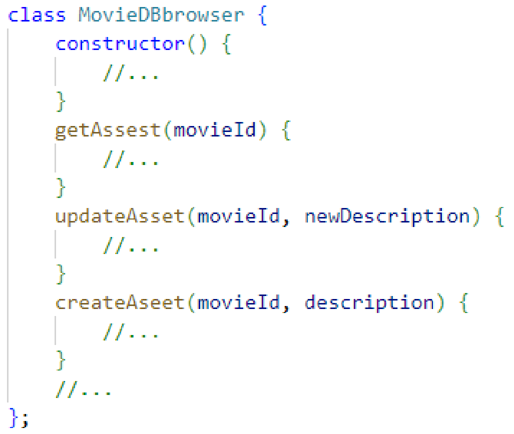
```
class ProxyPexelsConnection {
    constructor(cfg) {
        this.pexels = null;
        this.cfg = cfg;
    }
    init() {
        if (this.pexels === null) {
            this.pexels = new PexelsConnection(this.cfg);
        }
    }
    getMyPictures(){
        this.init();
        return this.pexels.getMyPictures();
    }
    //...
}
```

```
const cfg = {
    userID: "238sfSDFSF3",
};
const pexels = new ProxyPexelsConnection(cfg);
const fetchPictures = () => {
    // show spinner
    // ...
    const imgArr = pexels.getMyPictures();
    // hide spinner
    // ...
    // populate container with pictures
    //...
};

const btn = document.querySelector("button");
btn.addEventListener("click", fetchPictures);
```

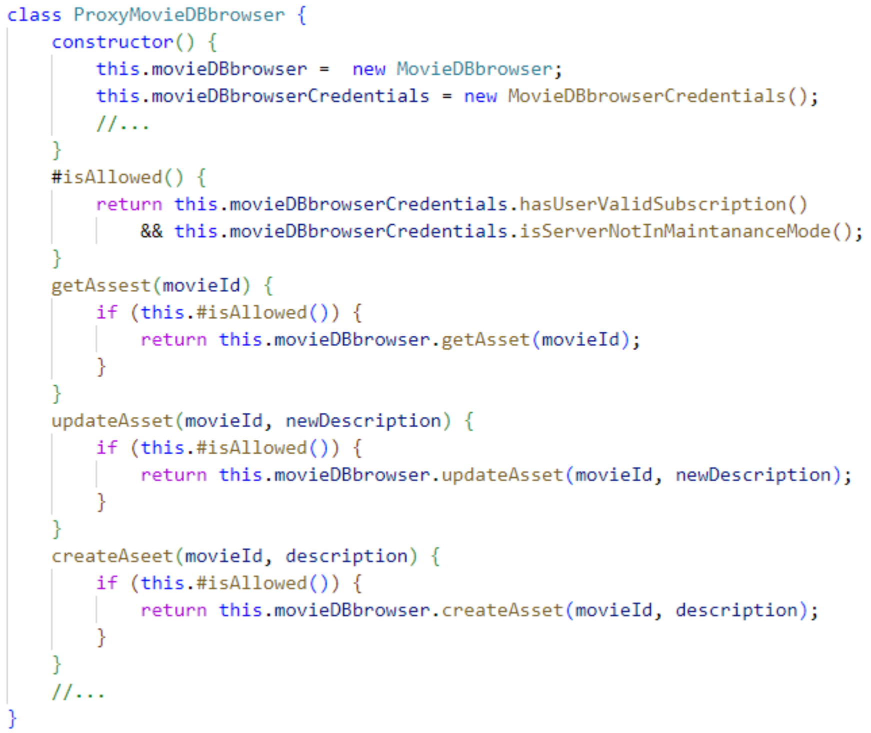
Proxy - access control

Usage – access control

- Limited access to an object. Some conditions need to be met first



```
class MovieDBbrowser {
    constructor() {
        //...
    }
    getAssest(movieId) {
        //...
    }
    updateAsset(movieId, newDescription) {
        //...
    }
    createAseet(movieId, description) {
        //...
    }
    //...
};
```



```
class ProxyMovieDBbrowser {
    constructor() {
        this.movieDBbrowser =  new MovieDBbrowser;
        this.movieDBbrowserCredentials = new MovieDBbrowserCredentials();
        //...
    }
    #isAllowed() {
        return this.movieDBbrowserCredentials.hasUserValidSubscription()
            && this.movieDBbrowserCredentials.isServerNotInMaintananceMode();
    }
    getAssest(movieId) {
        if (this.#isAllowed()) {
            return this.movieDBbrowser.getAsset(movieId);
        }
    }
    updateAsset(movieId, newDescription) {
        if (this.#isAllowed()) {
            return this.movieDBbrowser.updateAsset(movieId, newDescription);
        }
    }
    createAseet(movieId, description) {
        if (this.#isAllowed()) {
            return this.movieDBbrowser.createAsset(movieId, description);
        }
    }
    //...
}
```

Native implementation of Proxy in JavaScript (ES2015)

Thank you

Example - validation

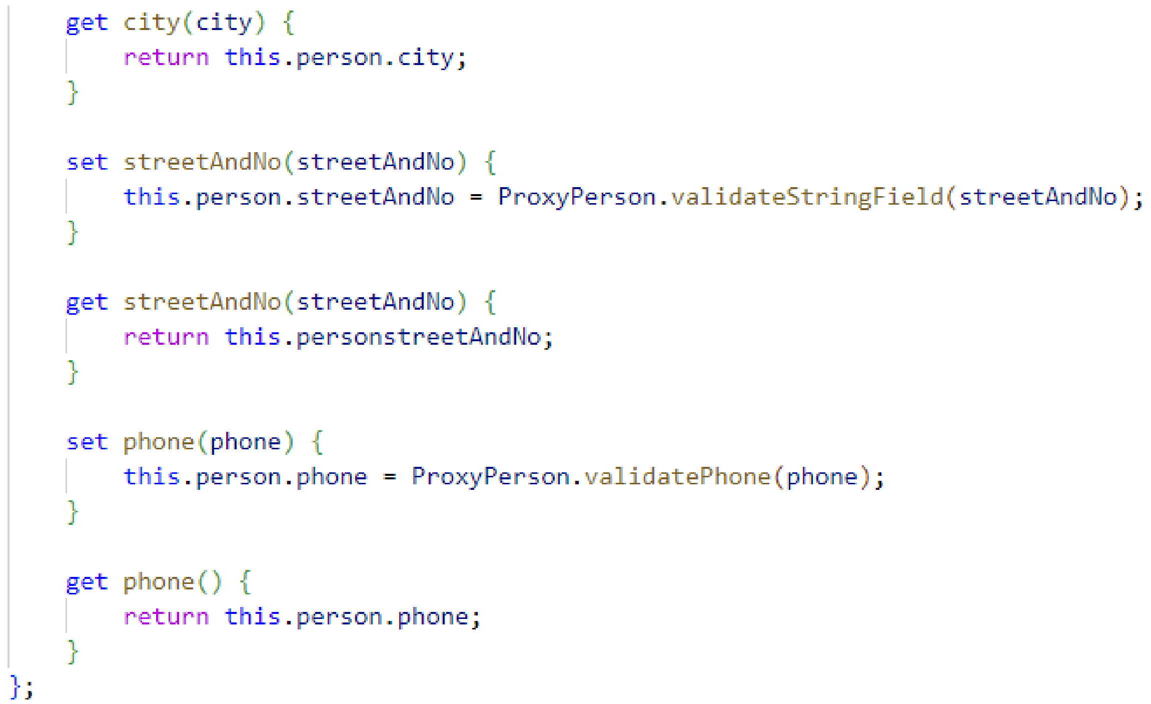
```
person = Proxy({}, {
    set (obj, prop, value) {
        function validateStringField(field) {
            if (typeof field === "String") {
                return field;
            } else {
                throw new Error(`${field} must be string`);
            }
        }

        function validatePhone(phone) {
            const DIGIT_NO = 9;
            if (typeof phone ==== String(phone) &&
                /^[0-9]{9}$/.test(phone)) {
                return phone;
            } else {
                throw new Error(`${phone} is not valid`);
            }
        }
        if (prop === "phone") {
            obj.phone = validatePhone(value);
        } else if (prop === name || prop === city ||
            prop === "streetAndNo") {
            obj[prop] = validateStringField(value);
        } else if (prop === "age") {
            obj.age = validateAge(value);
        } else {
            console.error("Cannot add a new property");
        }
    }
});
```

the proxy works on the empty object here

Example - validation

- Now it works on alreday created object

```javascript
class Person {
    constructor(name, age, city, streetAndNo, phone) {
        this.name = name;
        this.age = age;
        this.city = city;
        this.streetAndNo = streetAndNo;
        this.phone = phone;
    }
}

person = new Person("John", 40, "Poznan", "Freedom 45", 123456789);
person = Proxy(person, {
    set (obj, prop, value) {
        function validateStringField(field) {
            if (typeof field === "String") {
                return field;
            } else {
                throw new Error(`${field} must be string`);
            }
        }
        function validatePhone(phone) {
            const DIGIT_NO = 9;
            if (typeof phone ==== String(phone) &&
                /^[0-9]{9}$/.test(phone)) {
                return phone;
            } else {
                throw new Error(`${phone} is not valid`
            }
        }

        if (prop === "phone") {
            obj.phone = validatePhone(value);
        } else if (prop === name || prop === city ||
            prop === "streetAndNo") {
            obj[prop] = validateStringField(value);
        } else if (prop === "age") {
            obj.age = validateAge(value);
        } else {
            console.error("Cannot add a new property");
        }
    }
});
```



```
    get city(city) {
        return this.person.city;
    }

    set streetAndNo(streetAndNo) {
        this.person.streetAndNo = ProxyPerson.validateStringField(streetAndNo);
    }

    get streetAndNo(streetAndNo) {
        return this.personstreetAndNo;
    }

    set phone(phone) {
        this.person.phone = ProxyPerson.validatePhone(phone);
    }

    get phone() {
        return this.person.phone;
    }
};
```

- Every property requires a pair of a setter and getter
- Code length increases with the number of properties

Example - validation

- Now it works on alreday created object

```
class Person {
    constructor(name, age, city, streetAndNo, phone) {
        this.name = name;
        this.age = age;
        this.city = city;
        this.streetAndNo = streetAndNo;
        this.phone = phone;
    }
}

person = new Person("John", 40, "Poznan", "Freedom 45", 123456789);
person = Proxy(person, {
    set (obj, prop, value) {
        function validateStringField(field) {
            if (typeof field === "String") {
                return field;
            } else {
                throw new Error(`${field} must be string`);
            }
        }
```

```
        function validatePhone(phone) {
            const DIGIT_NO = 9;
            if (typeof phone ==== String(phone) &&
                /^[0-9]{9}$/.test(phone)) {
                return phone;
            } else {
                throw new Error(`${phone} is not valid`
            }
        }

        if (prop === "phone") {
            obj.phone = validatePhone(value);
        } else if (prop === name || prop === city ||
            prop === "streetAndNo") {
            obj[prop] = validateStringField(value);
        } else if (prop === "age") {
            obj.age = validateAge(value);
        } else {
            console.error("Cannot add a new property");
        }
    }
});
```

References

- https://www.patterns.dev/posts/proxy-pattern/
- https://www.dofactory.com/javascript/design-patterns/proxy
- https://stackoverflow.com/questions/7379732/what-is-a-javascript-proxy-pattern
- https://www.lambdatest.com/blog/comprehensive-guide-to-javascript-design-patterns/
-

Thank you