Objects

Przemysław Maćkowiak

Agenda

Creation

- object literal
- *new Object*
- factory
- prototype
- constructor

ES6 (class)

Object.assign

Object creation patterns

Factory function

Constructor pattern

Prototype pattern

Rzut okiekm

Obiekt jest tworzony na zasadzie klucz : wartość

```
let article = {
    journal: null,
    year: null,
    title: null,
    getDescr() {
        return `${this.journal}, ${this.title}, ${this.year}`;
    },
    displayYearInConsole() {
        console.log(`Journal published in ${this.year}`);
    }
};
```

właściwości mogą reprezentować funkcje

```
> typeof article
< 'object'
>
```

. operator
[ ] operator

```
> article.journal
< null
> article.journal = "Math Journal"
< 'Math Journal'
> article.title = "Linear function"
< 'Linear function'
> article.year = 1990
< 1990
> article.getDescr()
< 'Math Journal, Linear function, 1990'
> article["year"]
< 1990
>
```

Tworzenie obiektów: literals

Obiekt jest tworzony przez określenie kluczy i wartości

```
let person = {
    name: "Pol",
    surname: "Smith",
    hair: "brown",
    eye: "brown",
    age: 40,
}
```

```
let elephant = {
    origin: "Africa",
    noOfLegs: 4,
    age: 35,
    sex: "male",
};
```

```
let audi = {
    type: "car",
    year: 1990,
    engine: "3.0",
    status: "good",
};
```

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

```
let info = {
    journal: null,
    year: null,
    title: null,
};
```

Obiekt jako właściwość

```
let nestedObj = {
    title: "IEEE",
    authors: {
        names: ["Wiliam S.", "David K."],
        universities: ["Oxford", "MIT"],
    },
};
```

Tworzenie obiektów: literals

Jak tworzyć właściwości w prostszy sposób ?

```
let age = 40, surName = "Smith", city = "LA";
let details = {/*...*/};
let smithPerson = {
    age,
    surName,
    city,
    details,
}
```

```
let addInfo = "model";
let car = {
    type: "vehicle",
    info: age > 2015 ? "new" : "old",
    [addInfo]: "RMX 4.5",
    [`${addInfo}-europe`]: "RMX 4.5 europe",
    [`${addInfo}-usa`]: "RMX 4.5 usa",
};
```

Dynamiczne właściwości i wartości

Tworzenie obiektów: literals

Właściwości obiektu mogą odnosić się do funkcji

```
let article = {
    journal: null,
    year: null,
    title: null,
    getDescr() {
        return `${this.journal}, ${this.title}, ${this.year}`;
    },
    displayYearInConsole() {
        console.log(`Journal published in ${this.year}`);
    }
};
```

```
person = {
    name: "Pol",
    surname: "Smith",
    hair: "brown",
    eye: "brown",
    age: 40,
    presentNameAndSurname () {
        return `Person: ${this.name} ${this.surname}`;
    }
};
```

Tworzenie obiektów: literals

Zauważ, że funkcje strzałkowe mogą być związane z problemami

```
person = {
    name: "Pol",
    surname: "Smith",
    hair: "brown",
    eye: "brown",
    age: 40,
    presentNameAndSurname: () => `Person: ${this.name} ${this.surname}`,
};
```

```
> person.presentNameAndSurname()
<· 'Person: undefined undefined'
>
```

this jest związany z globalnym obiektem

Tworzenie obiektów: literals

Obiekty nie są właścicielami funkcji, sprawdź poniższe przykłady

```
let customer = {
    age: null,
    setAge(newAge) {
        this.age = newAge;
    }
    //...
}

let setAge = customer.setAge;
setAge(30);
customer.setAge(40);
console.log(customer);
console.log(window.age);
```

```
let customer = {
    age: null,
    setAge(newAge) {
        this.age = newAge;
    }
    //...
}

let setAge = customer.setAge;
setAge(30);
customer.setAge(40);
console.log(customer);
console.log(window.age);
```

```
▶ {age: 40, setAge: ƒ}
30
```

Tworzenie obiektów: new Object

Składnia *new Object* jest również używana do tworzenia obiektu, jakkolwiek nie jest to popularne podejście

```
> new Object("time")
◂ ▶ String {'time'}
> new Object(true)
◂ ▶ Boolean {true}
> new Object(null)
◂ ▶ {}
> new Object(undefined)
◂ ▶ {}
```

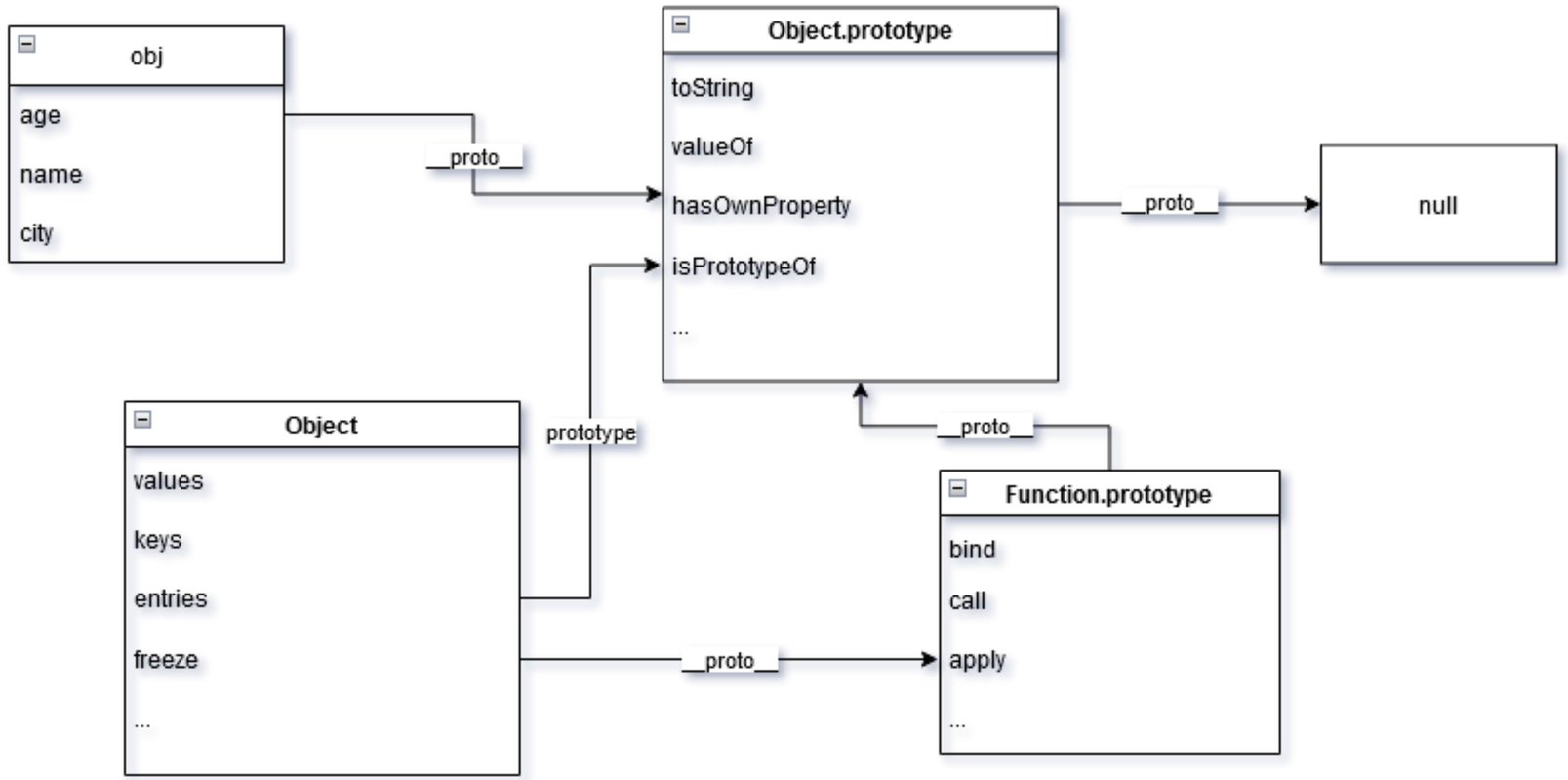
```
> new Object(56)
◂ ▶ Number {56}
> new Object(56n)
◂ ▶ BigInt {56n}
> new Object({age:40})
◂ ▶ {age: 40}
```

Jak widzimy, tworzone jest opakowanie wokół primitive data type. Są to odpowiadające wbudowane obiekty.

Instancja obiektu oraz jego prototyp

```
obj  = {
    age: 40,
    name: "Jospeh",
    city: "Poznan",
};
▶ {age: 40, name: 'Jospeh', city: 'Poz
obj.age
40
```

```
age
city
name
__defineGetter__
__defineSetter__
__lookupGetter__
__lookupSetter__
__proto__
constructor
hasOwnProperty
isPrototypeOf
propertyIsEnumerable
toLocaleString
toString
valueOf
```



Tworzenie obiektów: fabryka

Factory

```
function createBall (size, color) {
    let ball = {
        type: "soccer",
        preasure: "4.5",
        color,
        size,
    }

    return ball;
}

let myBall = createBall(5, "blackAndWhite");
```

Tworzenie obiektów: fabryka

Factory, aspekt pamięciowy

```
function createBall (size, color) {
    let ball = {
        type: "soccer",
        preasure: "4.5",
        color,
        size,
        getDescription() {
            console.log(`That is a ${this.type} ball\n` +
                        `Preasure value: ${this.preasure}\n` +
                        `Color ${color}\nSize: ${size}`);
        }
    }

    return ball;
}

let myBall = createBall(5, "blackAndWhite");
```

Za każdym razem kedy funkcja *createBall* jest wołana, tworzona jest funkcja *getDescription*

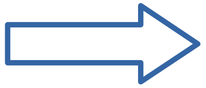
Tworzenie obiektów: fabryka

Factory, aspekt pamięciowy

```
function createBall (size, color) {
    let ball = {
        type: "soccer",
        preasure: "4.5",
        color,
        size,
        getDescription() {
            console.log(`That is a ${this.type} ball\n` +
                        `Preasure value: ${this.preasure}\n` +
                        `Color ${color}\nSize: ${size}`);
        }
    }

    return ball;
}

let myBall = createBall(5, "blackAndWhite");
```



```
function createBall (size, color) {
    let baseProtoype = {
        getDescription() {
        console.log(`That is a ${this.type} ball\n` +
                    `Preasure value: ${this.preasure}\n` +
                    `Color ${color}\nSize: ${size}`);
        }
    };
    let ball = Object.create(baseProtoype);

    return Object.assign(ball, {
        type: "soccer",
        preasure: "4.5",
        color,
        size,
    });
}

myBall = createBall(5, "blackAndWhite");
```

Tworzenie obiektów: operator new

new YourObject

```
function Ball(color, size) {
    this.color = "color";
    this.size = size;
    this.type = "soccer";
    this.preasure = "4.5";
    this.getDescription = () => {
        console.log(`That is a ${this.type} ball\n` +
                    `Preasure value: ${this.preasure}\n` +
                    `Color ${this.color}\nSize: ${this.size}`);
    }
}

myBall = new Ball("brown", "5");
```

getDescription jest tworzona wraz z nową instancją *Ball*

```
function Ball(color, size) {
    this.color = "color";
    this.size = size;
    this.type = "soccer";
    this.preasure = "4.5";
}
Ball.prototype.getDescription = function () {
    console.log(`That is a ${this.type} ball\n` +
                `Preasure value: ${this.preasure}\n` +
                `Color ${this.color}\nSize: ${this.size}`);
}

myBall = new Ball("brown", "5");
```

Ćwiczenia

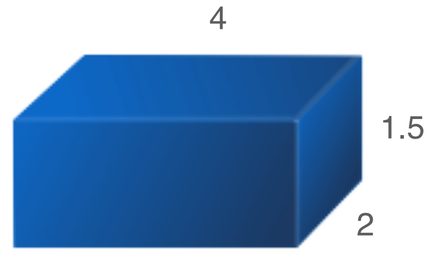
```
padding-bottom: 10px;
border-width: ▶ 90px;
margin: ▶ 70px;
border-color: ▶ black;
border-style: ▶ double;
```

1. Zaimplementuj obiekt, który odpowiada obrazkowi po prawej stronie

2. Napisz funkcję (klasę) która opisuje komputer. Zaimplementuj właściwośc *getInfo*. Użyj operatora *new* by powołać przykładową instancję obiektu

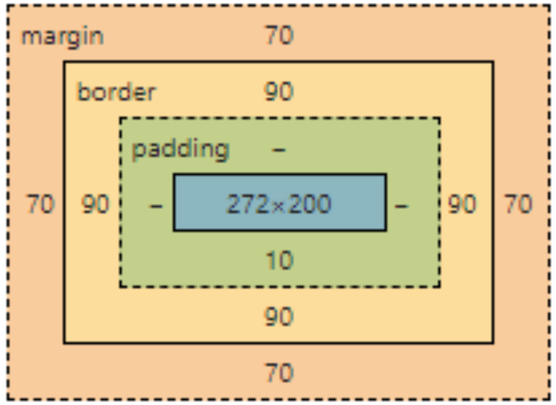
3. Stwórz obiekt opisujący przyładowy artykuł. Dodaj funkcje takie jak *getAuthor* oraz *getPageNo*. Ta druga zwraca string ”A book *articleTitle* has *pageNo”* a pierwsza:”An author of an article *articleTitle* is *articleAuthor*”. Jakie inne funkcje byłyby przydatne ?

4. Stwórz obiekt, który opisuje figurę o podanych wymiarach. Zaimplementuj funkcje *getArea, getPerimeter* i *getVolume*



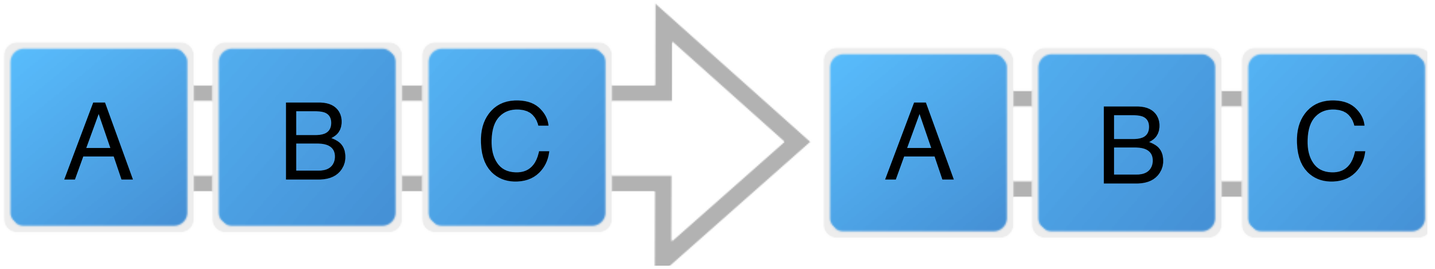
Ćwiczenia

5. Zaimplementuj obiekt opisujący wymiary, obramowanie, margines i dopełnienie (patrz rysunek)
   Dodaj następujące funkcje jako właściwości:
   getWidth zwraca czystą szerokośc bez dopełnienia i marginesu
   getTotalWidth zwraca powyższe wraz z obramowaniem, marginesem i dopełenieniem
   getHeight oraz *getTotalHeight* mają działaś w podobny sposób



Klonowanie

shallow copy




```
let ref = {
    age: 40,
    name: "Pol",
    origin: "Europe",
    city: "LA",
};
```

target

source

```
let refCopy = Object.assign({}, ref);
```

Kopiuje wartość po wartości używając operatora =

```
> ref
<· ▶ {age: 40, name: 'Pol', origin: 'Europe', city: 'LA'}
> refCopy
<· ▶ {age: 40, name: 'Pol', origin: 'Europe', city: 'LA'}
> refCopy.age=30;refCopy.origin="Africa";
<· 'Africa'
> ref
<· ▶ {age: 40, name: 'Pol', origin: 'Europe', city: 'LA'}
> refCopy
<· ▶ {age: 30, name: 'Pol', origin: 'Africa', city: 'LA'}
```

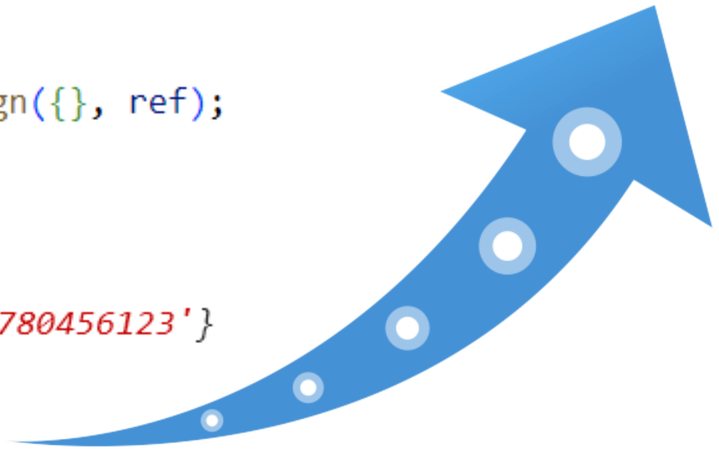
Klonowanie

Problemy

```
ref = {
    age: 40,
    cities: ["LA", "Chicago", "New York"],
    details: {
        company: "Samsung",
        phone: "780456123",
    },
};

refCopyObjectAssign = Object.assign({}, ref);
refCopyObjectAssign.details = {};
ref.details
```

```
▼{company: 'Samsung', phone: '780456123'}
    company: "Samsung"
    phone: "780456123"
  ▶[[Prototype]]: Object
>
```



```
refCopy = {
    age: ref.age,
    cities: [...ref.cities],
    details: {...ref.details},
};

refCopy.details = {};
ref.details;
```

```
▼{company: 'Samsung', phone: '780456123'}
    company: "Samsung"
    phone: "780456123"
  ▶[[Prototype]]: Object
>
```

Głębokie klonowanie jest potrzebne aby uniknąć zależności w stosunku do klonowanego obiektu

Dziękuję