

OLOO style of coding

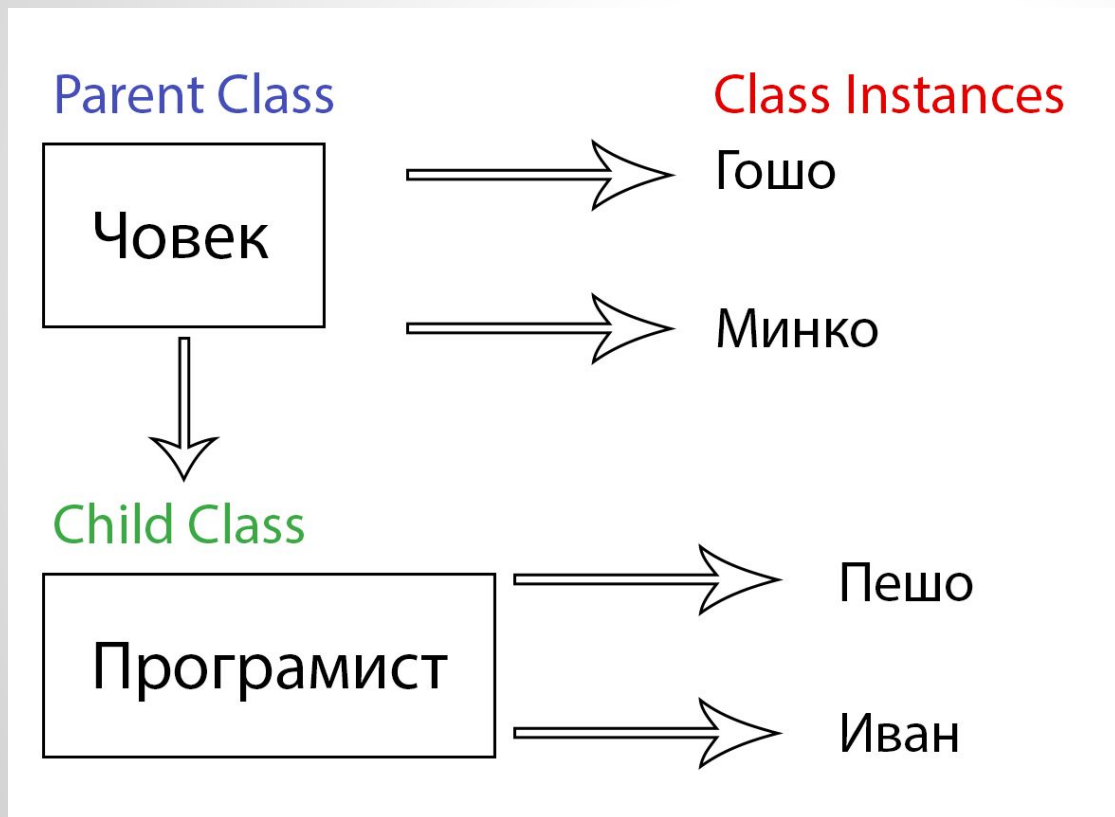
Samuil Gospodinov



Today's Plan

- Inheritance
- Prototypal Inheritance
- Behaviour Delegation Pattern
- OOLO
- Object Pooling Design Pattern
- Quiz

Classical Inheritance - COPY DOWN



Prototypal Inheritance

It's not an orange, it's an orange apple...

Inheritance means COPY

Prototypal ~~Inheritance~~ delegation

Person.prototype

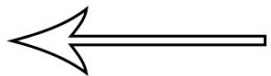


Programmer.prototype



Prototype Links

Гошо



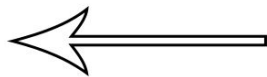
Минко



Пешо



Иван



Prototypal ~~Inheritance~~ delegation

JS has ~~inheritance~~

~~nope~~, JS has Behaviour delegation

- It's a design pattern!

Prototypal delegation

```
1 function Chovek(ime) {  
2     this.ime = ime;  
3 }  
4 Chovek.prototype.koisSumAz = function() {  
5     return "Az sum " + this.ime;  
6 };  
7  
8 var pesho = new Chovek('Pesho');  
9 pesho.koisSumAz(); // 'Az sum Pesho'
```

Prototypal delegation

Keep it simple stupid!

KISS

Prototypal delegation - OLOO


Objects Linked to Other Object

Prototypal delegation - OLOO

```
1 function Chovek(ime) {
2     this.ime = ime;
3 }
4 Chovek.prototype.koisSumAz = function() {
5     return "Az sum " + this.ime;
6 };
7
8 function Programist(ime) {
9     Chovek.call(this, ime);
10 }
11
12 Programist.prototype = Object.create(Chovek.prototype);
13
14 Programist.prototype.kazvamSe = function () {
15     console.log('Privet! ' + this.koisSumAz() + '.');
16 };
17
18 var pesho = new Programist('Pesho');
19 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```

Prototypal delegation - OLOO

```
1 function Chovek(ime) {  
2     this.ime = ime;  
3 }  
4 Chovek.prototype.koisSumAz = function() {  
5     return "Az sum " + this.ime;  
6 };  
7  
8 function Programist(ime) {  
9     Chovek.call(this, ime);  
10 }  
11  
12 Programist.prototype = Object.create(Chovek.prototype);  
13  
14 Programist.prototype.kazvamSe = function () {  
15     console.log('Privet! ' + this.koisSumAz() + '.');  
16 };  
17  
18 var pesho = new Programist('Pesho');  
19 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```



Prototypal delegation - OLOO

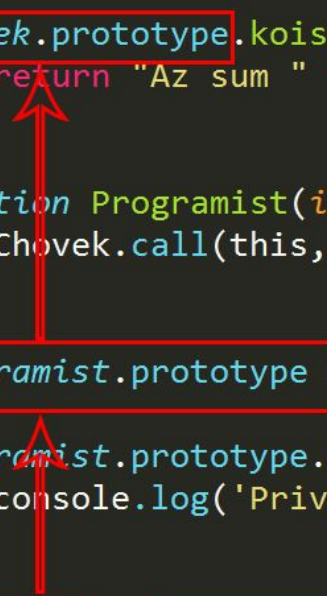
```
1 function Chovek(ime) {
2     this.ime = ime;
3 }
4 Chovek.prototype.koisSumAz = function() {
5     return "Az sum " + this.ime;
6 };
7
8 function Programist(ime) {
9     Chovek.call(this, ime);
10 }
11
12 Programist.prototype = Object.create(Chovek.prototype);
13
14 Programist.prototype.kazvamSe = function () {
15     console.log('Privet! ' + this.koisSumAz() + '.');
16 };
17
18 var pesho = Object.create(Programist.prototype);
19 pesho.call(pesho, 'Pesho');
20 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```

Prototypal delegation - OLOO

```
1 function Chovek(ime) {
2     this.ime = ime;
3 }
4 Chovek.prototype.koisSumAz = function() {
5     return "Az sum " + this.ime;
6 };
7
8 function Programist(ime) {
9     Chovek.call(this, ime);
10 }
11
12 Programist.prototype = Object.create(Chovek.prototype);
13
14 Programist.prototype.kazvamSe = function () {
15     console.log('Privet! ' + this.koisSumAz() + '.');
16 };
17
18 var pesho = Object.create(Programist.prototype);
19 pesho.call(pesho, 'Pesho');
20 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```

Prototypal delegation - OLOO

```
1 function Chovek(ime) {
2   this.ime = ime;
3 }
4 Chovek.prototype.koisSumAz = function() {
5   return "Az sum " + this.ime;
6 };
7
8 function Programist(ime) {
9   Chovek.call(this, ime);
10 }
11
12 Programist.prototype = Object.create(Chovek.prototype);
13
14 Programist.prototype.kazvamSe = function () {
15   console.log('Privet! ' + this.koisSumAz() + '.');
16 };
17
18 var pesho = Object.create(Programist.prototype);
19 pesho.call(pesho, 'Pesho');
20 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```



Prototypal delegation - OLOO

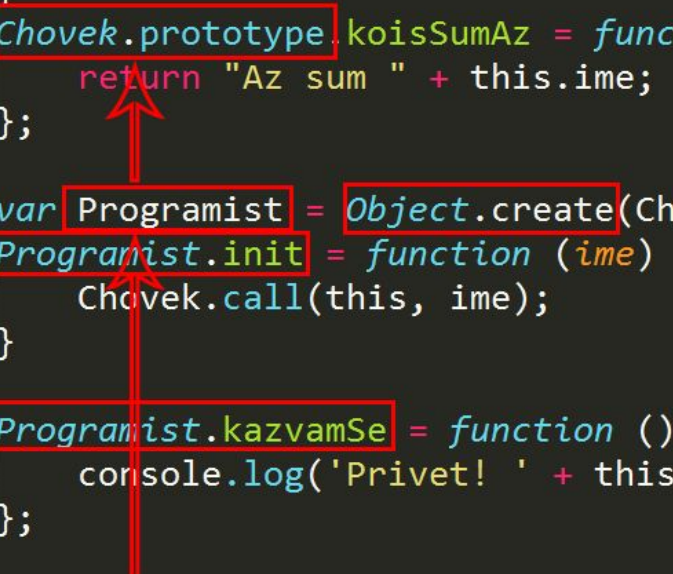
```
1 function Chovek(ime) {
2     this.ime = ime;
3 }
4 Chovek.prototype.koisSumAz = function() {
5     return "Az sum " + this.ime;
6 };
7
8 var Programist = Object.create(Chovek.prototype)
9 Programist.init = function (ime) {
10     Chovek.call(this, ime);
11 }
12
13 Programist.kazvamSe = function () {
14     console.log('Privet! ' + this.koisSumAz() + '.');
15 };
16
17 var pesho = Object.create(Programist);
18 pesho.call(pesho, 'Pesho');
19 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```

Prototypal delegation - OLOO

```
1 function Chovek(ime) {
2     this.ime = ime;
3 }
4 Chovek.prototype.koisSumAz = function() {
5     return "Az sum " + this.ime;
6 };
7
8 var Programist = Object.create(Chovek.prototype)
9 Programist.init = function (ime) {
10     Chovek.call(this, ime);
11 }
12
13 Programist.kazvamSe = function () {
14     console.log('Privet! ' + this.koisSumAz() + '.');
15 };
16
17 var pesho = Object.create(Programist);
18 pesho.call(pesho, 'Pesho');
19 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```


Prototypal delegation - OLOO

```
1 function Chovek(ime) {  
2     this.ime = ime;  
3 }  
4 Chovek.prototype.koisSumAz = function() {  
5     return "Az sum " + this.ime;  
6 };  
7  
8 var Programist = Object.create(Chovek.prototype)  
9 Programist.init = function (ime) {  
10     Chovek.call(this, ime);  
11 }  
12  
13 Programist.kazvamSe = function () {  
14     console.log('Privet! ' + this.koisSumAz() + '.');  
15 };  
16  
17 var pesho = Object.create(Programist);  
18 pesho.call(pesho, 'Pesho');  
19 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```



The diagram illustrates the prototype chain for the 'pesho' object. Red boxes highlight the following elements: 'Chovek.prototype.koisSumAz' (line 4), 'Object.create(Chovek.prototype)' (line 8), 'Programist.init' (line 9), 'Programist.kazvamSe' (line 13), and 'pesho' (line 17). Red arrows indicate the delegation path: one arrow points from 'Programist.kazvamSe' to 'Programist.init', and another points from 'Programist.init' to 'Chovek.prototype.koisSumAz'. This shows how 'pesho' delegates the 'kazvamSe' method to 'Programist', which then delegates the 'koisSumAz' method to 'Chovek'.

Prototypal delegation - OLOO

```
1 var Chovek = {
2   init: function (ime) {
3     this.ime = ime
4   },
5   koisSumAz: function() {
6     return "Az sum " + this.ime;
7   }
8 }
9
10 var Programist = Object.create(Chovek)
11
12 Programist.kazvamSe = function () {
13   console.log('Privet! ' + this.koisSumAz() + '.');
14 };
15
16 var pesho = Object.create(Programist);
17 pesho.init('Pesho');
18 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```

Prototypal delegation - OLOO

```
1 var Chovek = {  
2   init: function (ime) {  
3     this.ime = ime  
4   },  
5   koisSumAz: function() {  
6     return "Az sum " + this.ime;  
7   }  
8 }  
9  
10 var Programist = Object.create(Chovek)  
11  
12 Programist.kazvamSe = function () {  
13   console.log('Privet! ' + this.koisSumAz() + '.');  
14 };  
15  
16 var pesho = Object.create(Programist);  
17 pesho.init('Pesho');  
18 pesho.kazvamSe(); // 'Privet! Az sum Pesho.'
```

The diagram illustrates the prototype chain for the provided code. Red boxes highlight the objects `Chovek`, `Programist`, and `pesho`. Red arrows indicate the `__proto__` relationship: one arrow points from `Programist` to `Chovek`, and another points from `pesho` to `Programist`, showing that `Programist` inherits from `Chovek` and `pesho` inherits from `Programist`.

Quiz

1. What's the difference between JS `[[Prototype]]` links and traditional classical inheritance?
2. What is behaviour delegation?
3. How is behaviour delegation helpful?
4. What are the tradeoffs of behaviour delegation?

Object Pooling

Object Pooling design pattern