JavaScript

Šesnaesti dio

Pregled

- OOP, nastavak
  - Object konstruktor
  - Prototype chain
  - Inheritance
    - Factory function pristup
    - Konstruktor (pseudoclassical) pristup
    - Class pristup
- arguments

Obnavljanje

- Copy/paste strogo zabranjen
- Projekat, domaći 5 i 6, test 4

```
function UserCreator(name, score){
  this.name = name;
  this.score = score;
}

UserCreator.prototype.increment = function(){
  this.score++;
};
UserCreator.prototype.login = function(){
  console.log("login");
};

const user1 = new UserCreator(“Eva”, 9)

user1.increment()
```

```
class UserCreator {
  constructor (name, score){
      this.name = name;
      this.score = score;
  }
  increment (){
    this.score++;
  }
  login (){
    console.log("login");
  }
}

const user1 = new UserCreator("Eva", 9);

user1.increment();
```

- call(), apply(), bind()

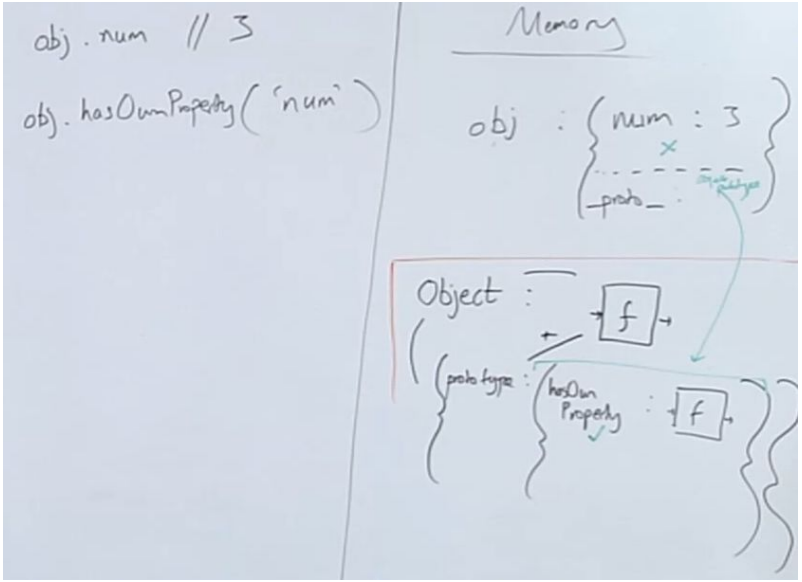
Object konstruktor funkcija

- JS koristi **proto** link da bi imali dodatne funkcionalnosti za objekte, funkcije i nizove. Svi objekti, po default-u imaju _ _ proto _ _

```
const obj = {
    num : 3
}

obj.num // 3
obj.hasOwnProperty("num") // ? Where's this method?

Object.prototype // {hasOwnProperty: FUNCTION}
```

- Sa Object.create brišemo default _ _ proto _ _ referencu ka Object.prototype i replacujemo sa functionStore
- Ali functionStore je objekat tako da ima _ _ proto _ _ referencu ka Object.prototype

Skica za prethodni primjer



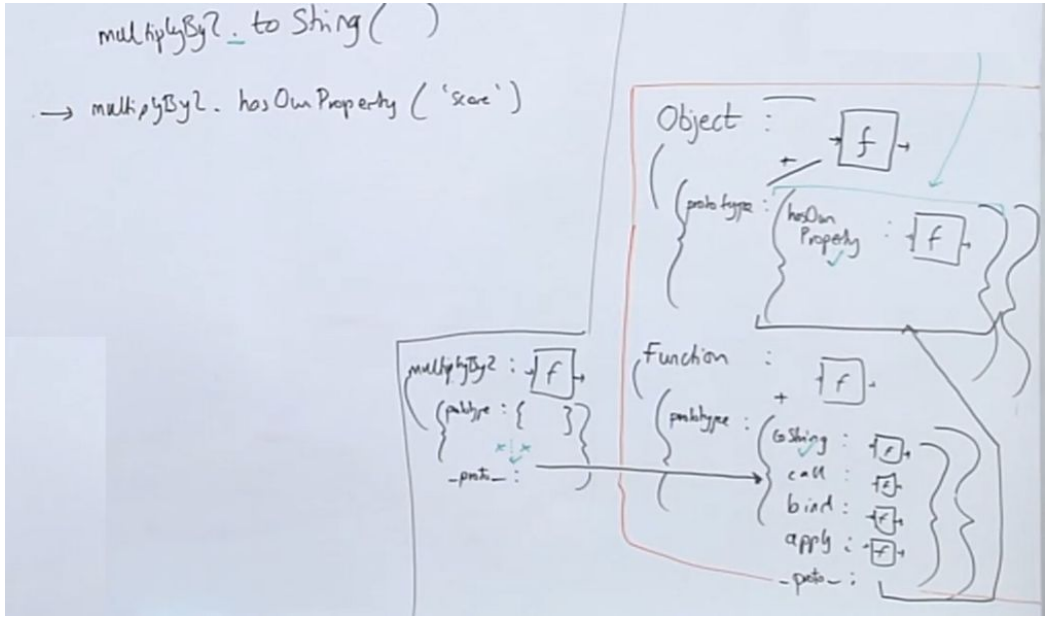
Funkcije i nizovi, prototype

- Su takođe objekti koji imaju pristup svim funkcijama u Object.prototype

```
function multiplyBy2(num){
  return num*2
}

multiplyBy2.toString() //Where is this method?

Function.prototype // {toString : FUNCTION, call : FUNCTION, bind : FUNCTION}

multiplyBy2.hasOwnProperty("score") // Where's this function?

Function.prototype.__proto__ // Object.prototype  {hasOwnProperty: FUNCTION}
```

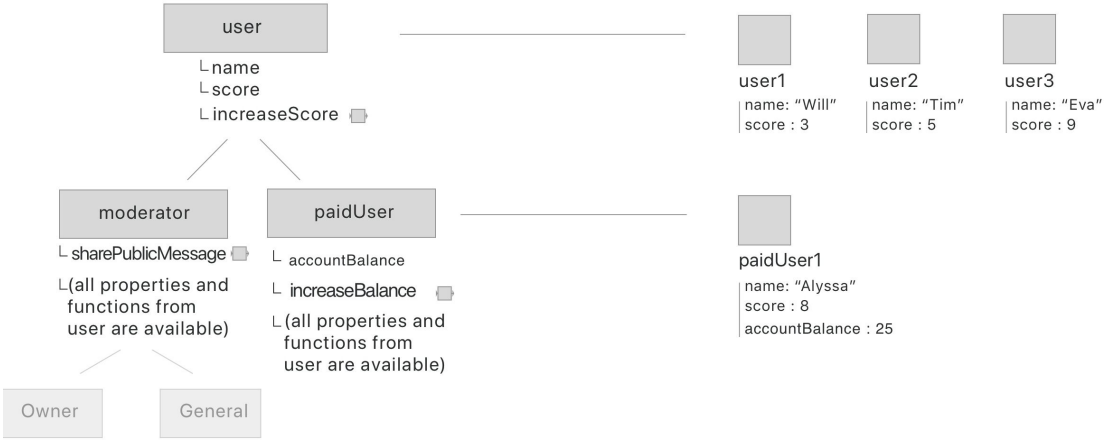
- Šta ako pozovemo na kraju npr. multiplyBy2.increase() ?

Skica za prethodni primjer, prototype chain



OOP, nasleđivanje

- Vidjeli smo u prethodnim primjerima već nešto što se zove nasleđivanja, a do sada nismo ni bili svjesni da to radimo



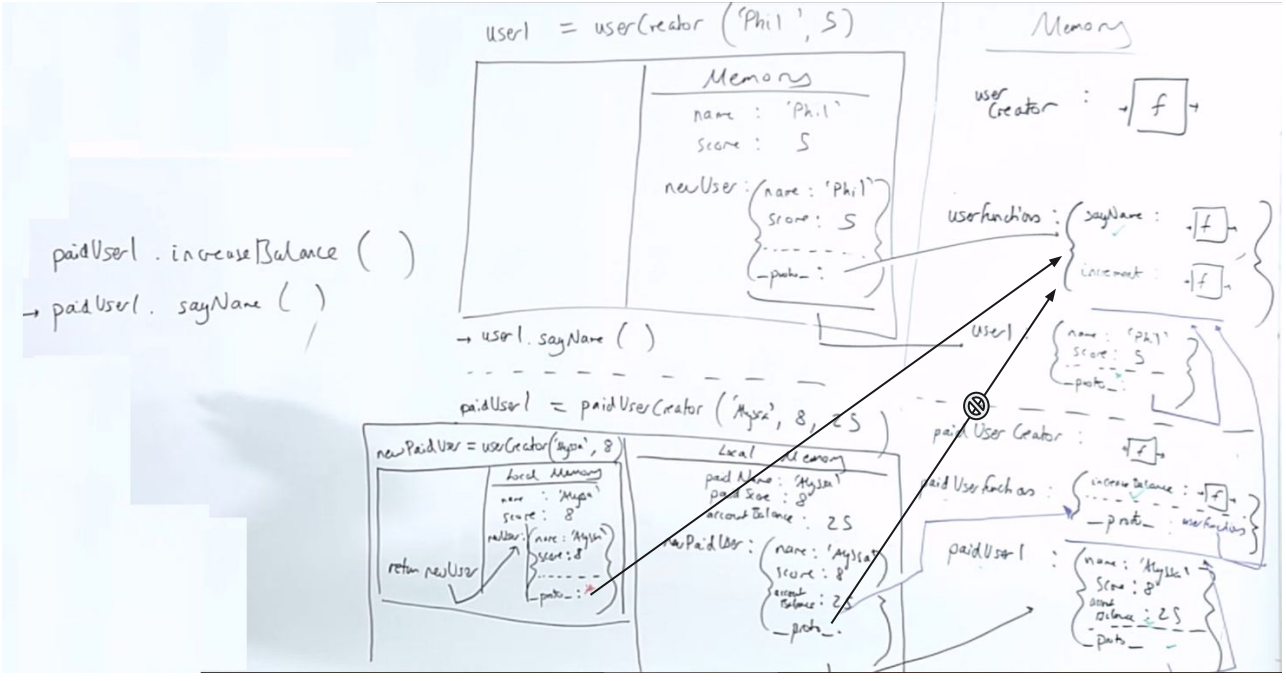
Factory function pristup

- userCreator() je factory funkcija

```
function userCreator(name, score){
  const newUser = Object.create(userFunctions);
  newUser.name = name;
  newUser.score = score;
  return newUser;
}

userFunctions = {
  sayName: function (){
      console.log("I'm " + this.name);
  },
  increment: function(){
      this.score++;
  }
}

const user1 = userCreator("Phil",5);

user1.sayName(); // "I am Phil"
```

```
function paidUserCreator(paidName, paidScore, accountBalance){
  const newPaidUser = userCreator(paidName, paidScore);
  Object.setPrototypeOf(newPaidUser, paidUserFunctions);
  newPaidUser.accountBalance = accountBalance;
  return newPaidUser;
}

const paidUserFunctions = {
  increaseBalance : function (){
    this.accountBalance++;
  }
};

Object.setPrototypeOf(paidUserFunctions, userFunctions)

const paidUser1 = paidUserCreator("Alyssa", 8, 25);

paidUser1.increaseBalance();

paidUser1.sayName(); // "I'm Alyssa"
```

Skica za prethodni primjer



Konstruktor pristup

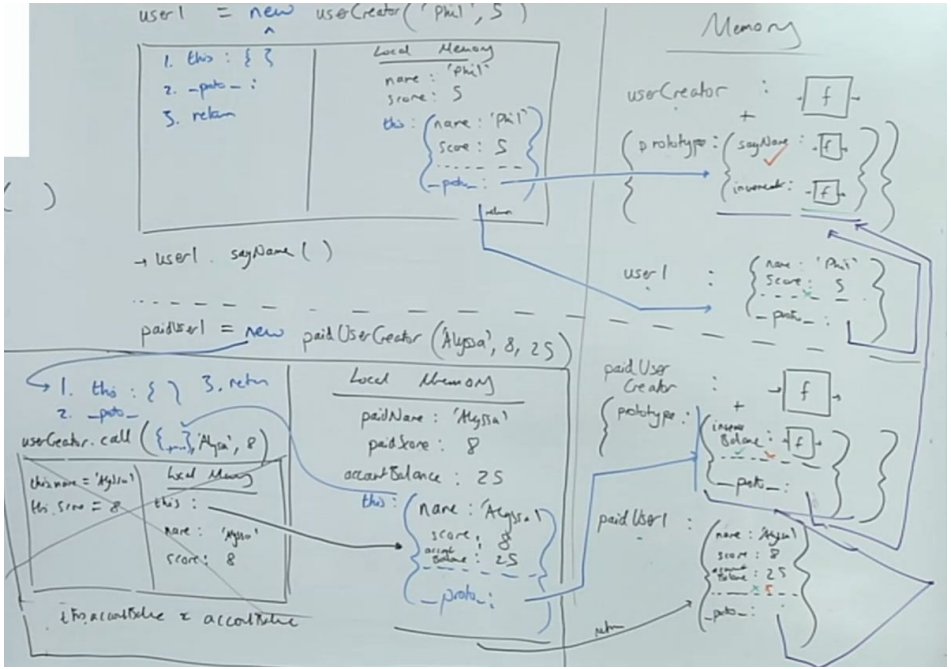
```
function userCreator (name, score){
  this.name = name
  this.score = score
}
```

```
userCreator.prototype.sayName = function (){
  console.log("I'm " + this.name);
}
userCreator.prototype.increment = function(){
  this.score++;
}
```

```
const user1 = new userCreator("Phil", 5);
const user2 = new userCreator("Tim", 4);

user1.sayName(); // "I'm Phil"
```

```
function paidUserCreator (paidName, paidScore, accountBalance){
  userCreator.call(this, paidName, paidScore);
  // userCreator.apply(this, [paidName, paidScore])
  this.accountBalance = accountBalance;
}

paidUserCreator.prototype = Object.create(userCreator.prototype);

paidUserCreator.prototype.increaseBalance = function (){
  this.accountBalance++;
};

const paidUser1 = new paidUserCreator("Alyssa", 8, 25);

paidUser1.increaseBalance()

paidUser1.sayName() // "I'm Alyssa"
```

Skica za prethodni primjer

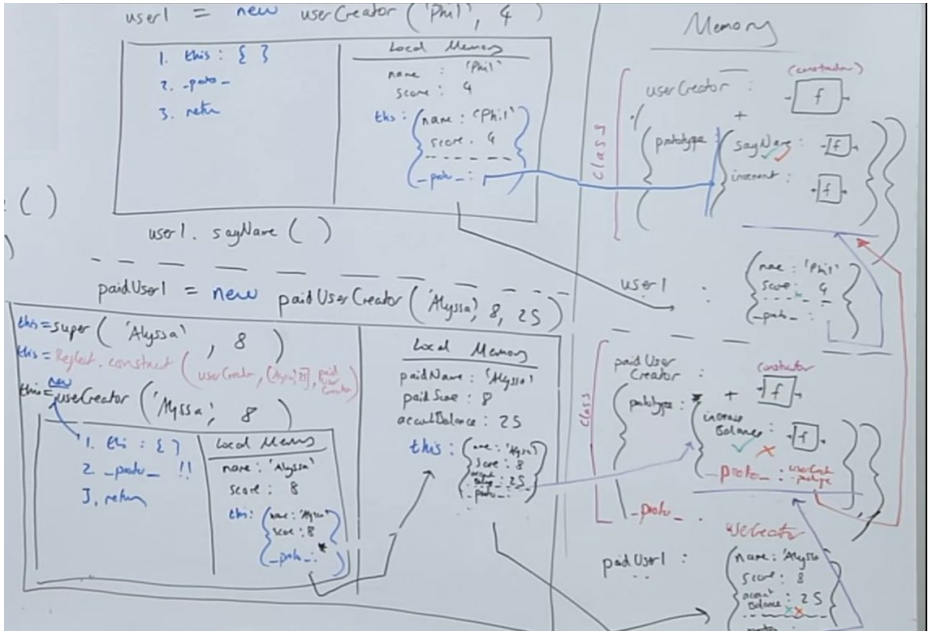


Class pristup

```
class userCreator{
  constructor  (name, score){
    this.name = name;
    this.score = score;
  }
  sayName (){
    console.log("I am " + this.name);
  }
  increment (){
    this.score++;
  }
}

const user1 = new userCreator("Phil", 4);
const user2 = new userCreator("Tim", 4);

user1.sayName()
```

```
class paidUserCreator extends userCreator {
  constructor(paidName, paidScore, accountBalance){
    super (paidName, paidScore);
    this.accountBalance = accountBalance;
  }
  increaseBalance (){
    this.accountBalance++;
  }
}

const paidUser1 = new paidUserCreator("Alyssa", 8, 25);

paidUser1.increaseBalance();

paidUser1.sayName();
```

Skica za prethodni primjer



arguments

- Pristup argumentima funkcije

Pair Programming

- http://csbin.io/promises

Pitanja