Object Composition

Closures, Revealing Module Pattern, Object Inheritance, Prototypes

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Content

SoftUni Foundation

Have a Question?

SoftUni Foundation

sli.do

#JS-CORE

Object Composition

Objects Holding Other Objects

What is Object Composition?

Combining simple objects or data types into more **complex ones**

```
let student = {
  firstName: 'Maria',
  lastName: 'Green',
  age: 22,
  location: { lat: 42.698, lng: 23.322 }
}
console.log(student);
console.log(student.location.lat);
```

Composing Objects

```
let name = "Sofia";
let population = 1325744;
let country = "Bulgaria";
let town = { name, population, country };
console.log(town);
// Object {name: "Sofia", population: 1325744,
country: "Bulgaria"}
```

Combine variables into object

```
town.location = { lat: 42.698, lng: 23.322 };
console.log(town); // Object {…, location: Object}
```

Combining Data with Functions

```
let rect = {
  width: 10,
  height: 4,
  grow: function(w, h) {
    this.width += w; this.height += h;
  },
  print: function() {
    console.log(`[${this.width} x ${this.height}]`);
  }
};
rect.grow(2, 3);
rect.print(); // [12 x 7]
```

Printing Objects: toString() Function

SoftUni Foundation

```
let rect = {
  width: 10,
  height: 4,
  toString: function() {
    return `rect[${this.width} x ${this.height}]`;
  }
};

console.log(rect); // Object {width: 10, height: 4}
// This will invoke toString() to convert the object to String
console.log('' + rect); // rect[12 x 7]
```

Problem: Order Rectangles

Create **objects** to represent the rectangles. Should additionally have two functions:

- **area()** - that returns the area of the rectangle
- **compareTo(**other**)** - compares the current rectangle with another

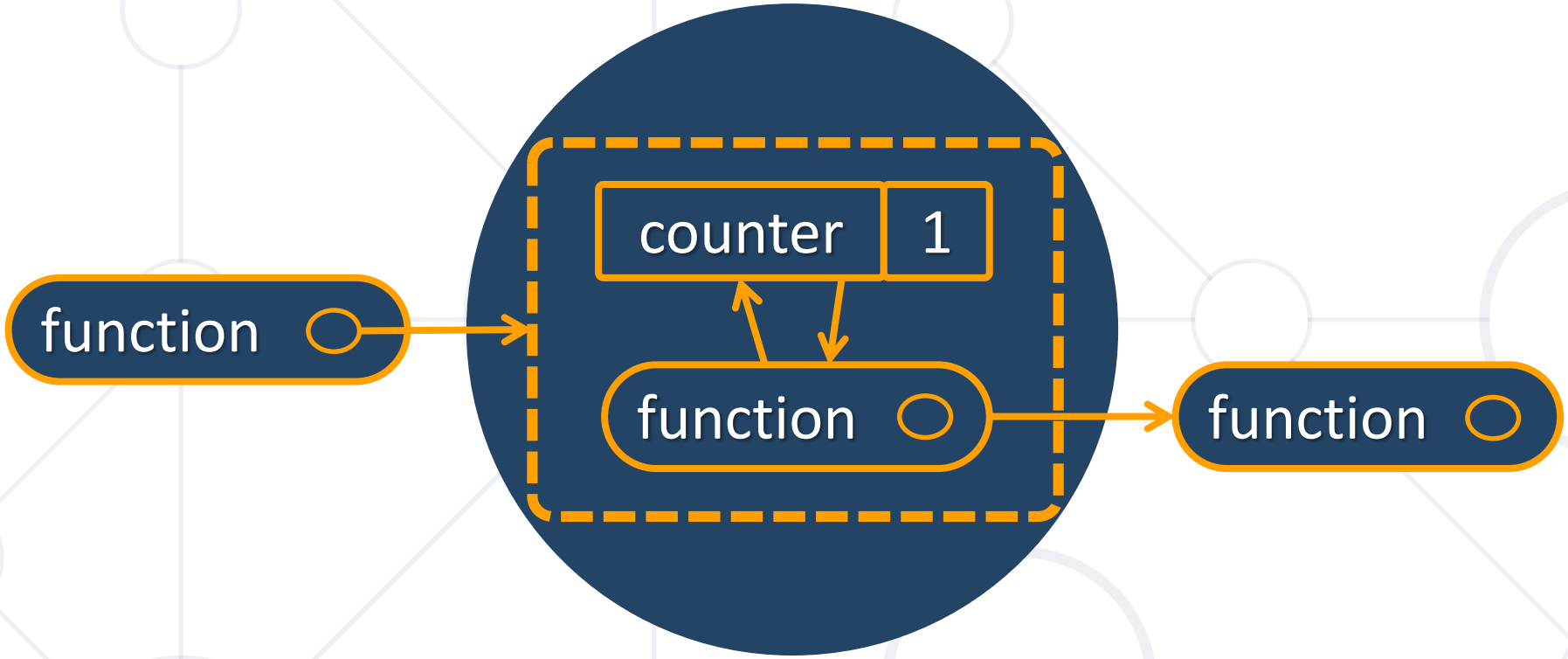
Input	Output
`[[10,5],[5,12]]`	`[{width:5, height:12, area:function(), compareTo:function(other)},` `{width:10, height:5, area:funciton(),compareTo:function(other)}]`

Solution: Order Rectangles

```
let rects = [];
for (let [width, height] of data) {
  let rect = comparator(width, height);
  rects.push(rect);
}
rects.sort((a, b) => a.compareTo(b));
return rects;
// Continue on the next slide
```

Solution: Order Rectangles (2)

```
function comparator(w, h) {
  let rect = {
    width: w,
    height: h,
    area: () => rect.width * rect.height,
    compareTo: function (other) {
      let result = other.area() - rect.area();
      return result || (other.width - rect.width);
    }
  };
  return rect;
}
```



Closures

Enclosing Object State in a Function

What is Closure?

State maintained (closed) inside a function

- Hidden from the outside world
- Example: counter with closures

```
function counterClosure() {
  let counter = 0;
  function getNextCount() {
    console.log(++counter);
  };
  return getNextCount;
}
```

```
let count =
counterClosure();
count(); // 1
count(); // 2
count(); // 3
count(); // 4
count(); // 5
```

Closures - Shorter Syntax with IIFE

SoftUni Foundation

```
let counter = (function() {
  let num = 0;
  return function() {
    console.log(++num);
  };
})();
counter(); // 1
counter(); // 2
counter(); // 3
```

```
let counter = (() => {
  let num = 0;
  return () =>
    console.log(++num);
})();
counter(); // 1
counter(); // 2
counter(); // 3
counter(); // 4
```

Problem: Fibonacci

Write a function that when called

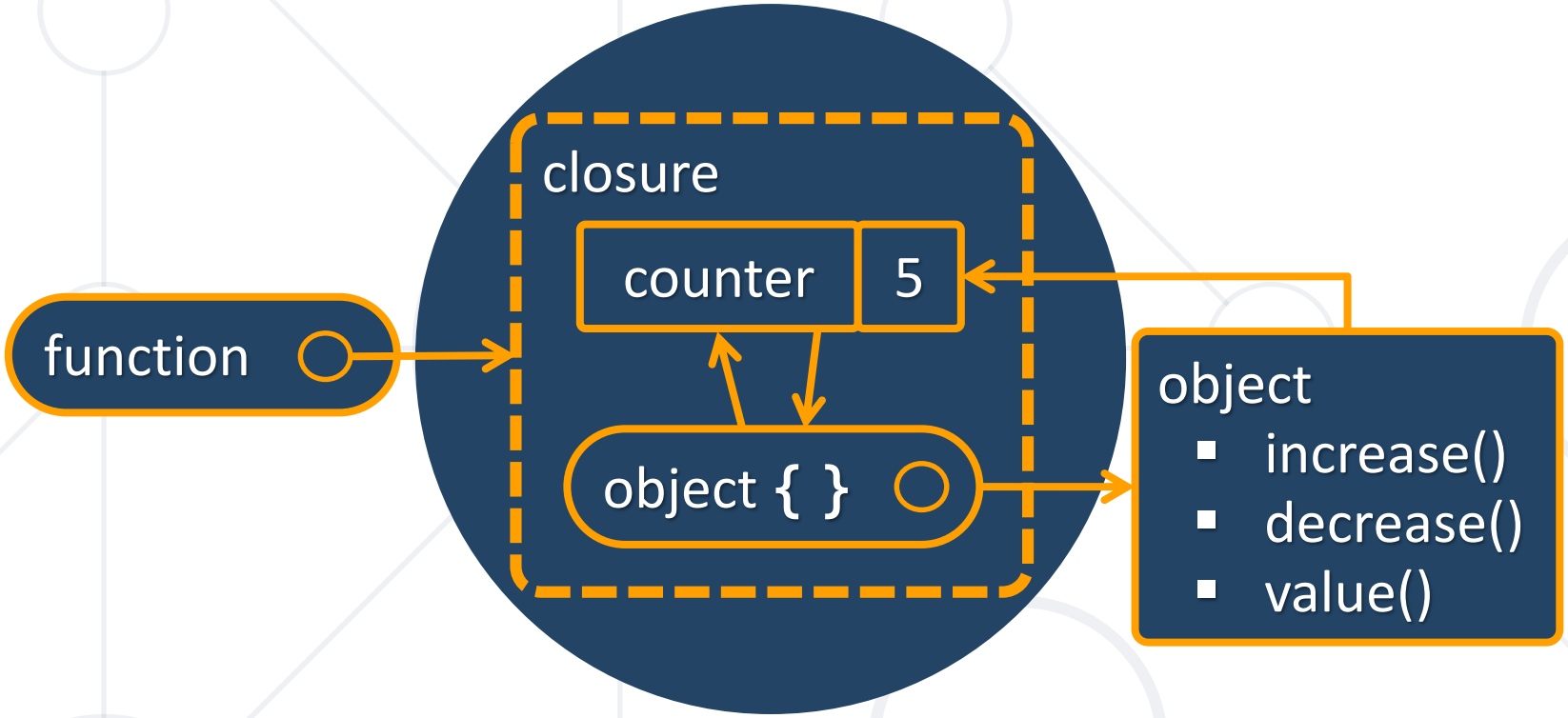
- **returns** the next **Fibonacci number**, starting at 0, 1
- use a **closure** to keep the current number

Sample execution

```
let fib = getFibonator();
console.log(fib()); // 1
console.log(fib()); // 1
console.log(fib()); // 2
console.log(fib()); // 3
console.log(fib()); // 5
console.log(fib()); // 8
console.log(fib()); // 13
```

Solution: Fibonacci

```
function getFibonator() {
  let sum = 0;
  let first = 0;
  let second = 1;
  return function () {
      sum = first + second;
      first = second;
      second = sum
      return first;
  }
}
```



Module and Revealing Module Patterns

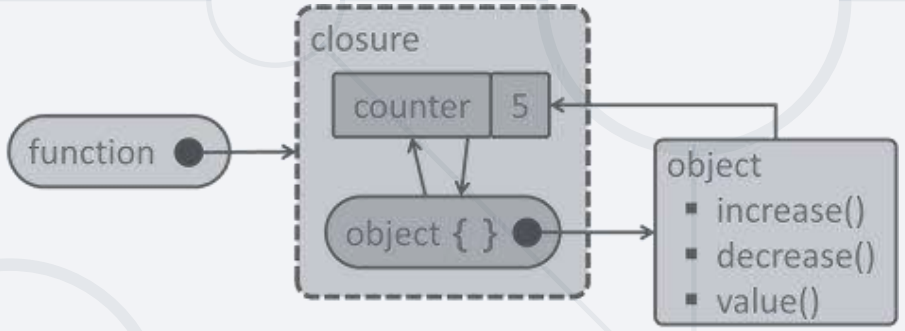
"Module" Pattern (with Object Literal)

SoftUni Foundation

```
let moduleObj = {
  count: 0, // public
  increase: function(num) { return this.count += num },
  decrease: function(num) { return this.count -= num },
  value: function() { return this.count }
};
moduleObj.count = 2; // the counter is accessible
console.log(moduleObj.value()); // 2
console.log(moduleObj.increase(5)); // 7
console.log(moduleObj.decrease(1)); // 6
```

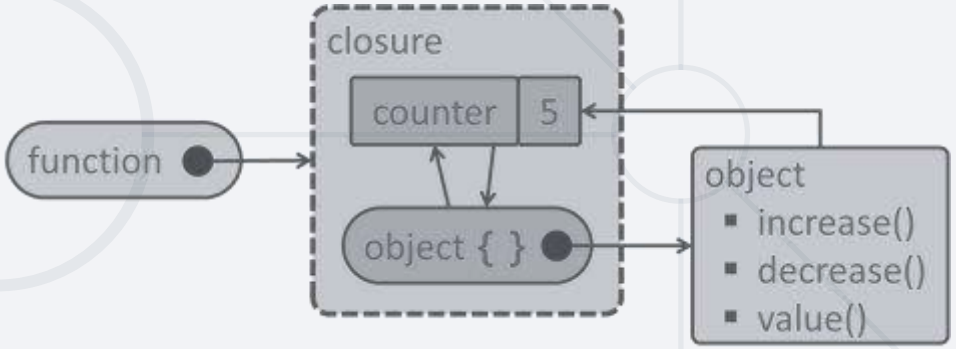
"Module" Pattern (with Closure)

```
let module = (function() {
  let count = 0; // private
  return {
    increase: (num) => count += num,
    decrease: (num) => count -= num,
    value: () => count,
  };
})();
console.log(module.value()); // 0
console.log(module.increase(5)); // 5
console.log(module.decrease(2)); // 3
console.log(module.count); // undefined (not accessible)
```



"Revealing Module" Pattern (with Closure)

```
let revModule = (function() {
  let count = 0; // private
  function change(amount) { return count += amount; }
  function increase(num) { return change(num); }
  function decrease(num) { return change(-num); }
  function value() { return count; }
  return { increase, decrease, value };
})();
console.log(revModule.value()); // 0
console.log(revModule.increase(5)); // 5
console.log(revModule.decrease(2)); // 3
console.log(revModule.count); // undefined (not accessible)
```



Problem: List processor

Using a closure, create an inner object to process list commands

- **add** - adds the following string in an inner collection
- **remove** - removes all occurrences
- **print** - prints all elements of the inner collection joined by **","**

Input	Output
`['add hello', 'add again', 'remove hello', 'add again', 'print']`	`again, again`

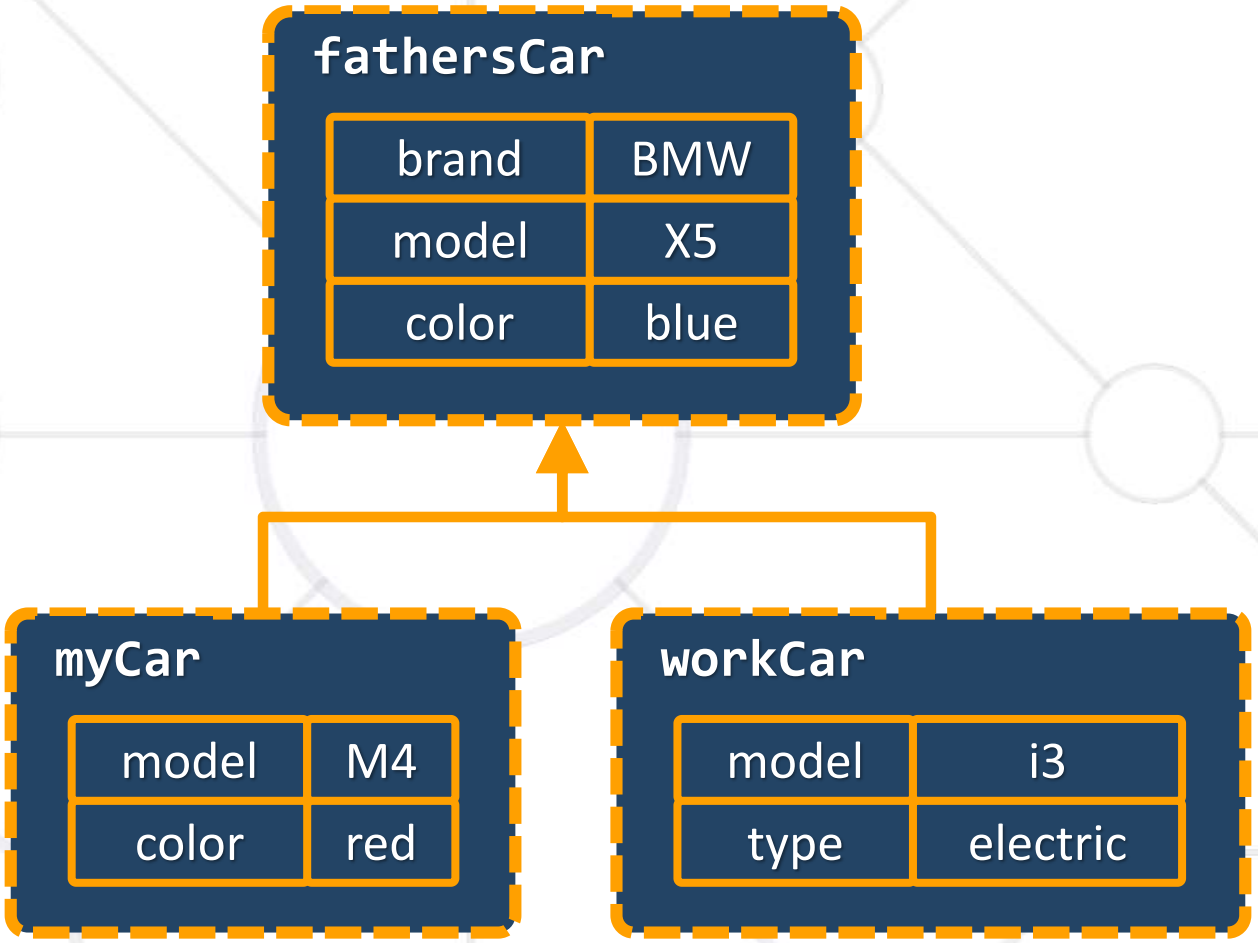
Solution: List processor

```
let arr = []
let command = {
  add: function (arr, str) {
    arr.push(str)
    return arr;
  },
  remove: function (arr, str) {
    arr = arr.filter(e => e !== str);
    return arr
  },
  print: function (str) {
    console.log(arr.join(','));
    return arr;
  }
};        // Continue on the next slide
```

Solution: List processor (2)

```
return function (input) {

  for (let e of input) {
    e = e.split(' ');
    arr = command[e[0]](arr, e[1])
  }
}
```



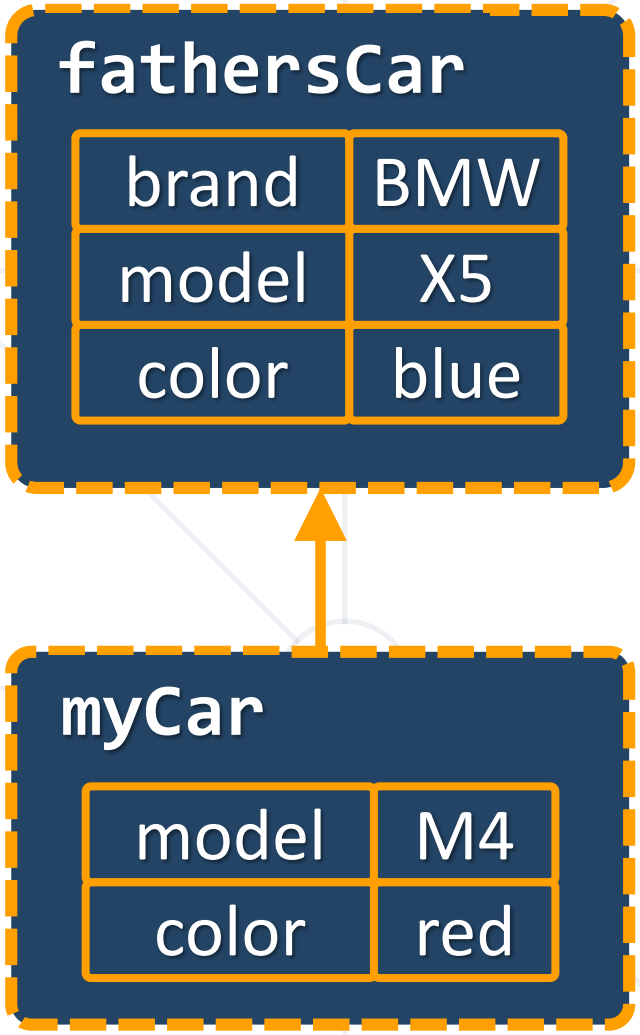
Object Inheritance

Object Inheritance

```
let fatherCar = { brand: 'BMW',
  model: 'X5', color: 'blue',
  toString: function() { return `[brand:
    ${this.brand}, model: ${this.model},
    color: ${this.color}]`; }
};
console.log('' + fatherCar);
```

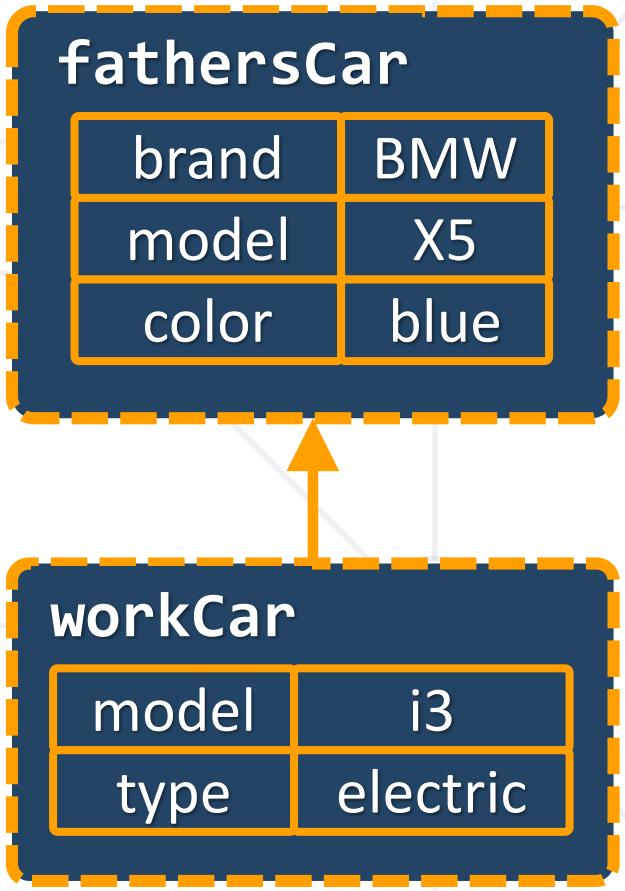
```
let myCar = Object.create(fatherCar);
myCar.model = 'M4';
myCar.color = 'red';
console.log('' + myCar);
```

Object.create()
inherits an object

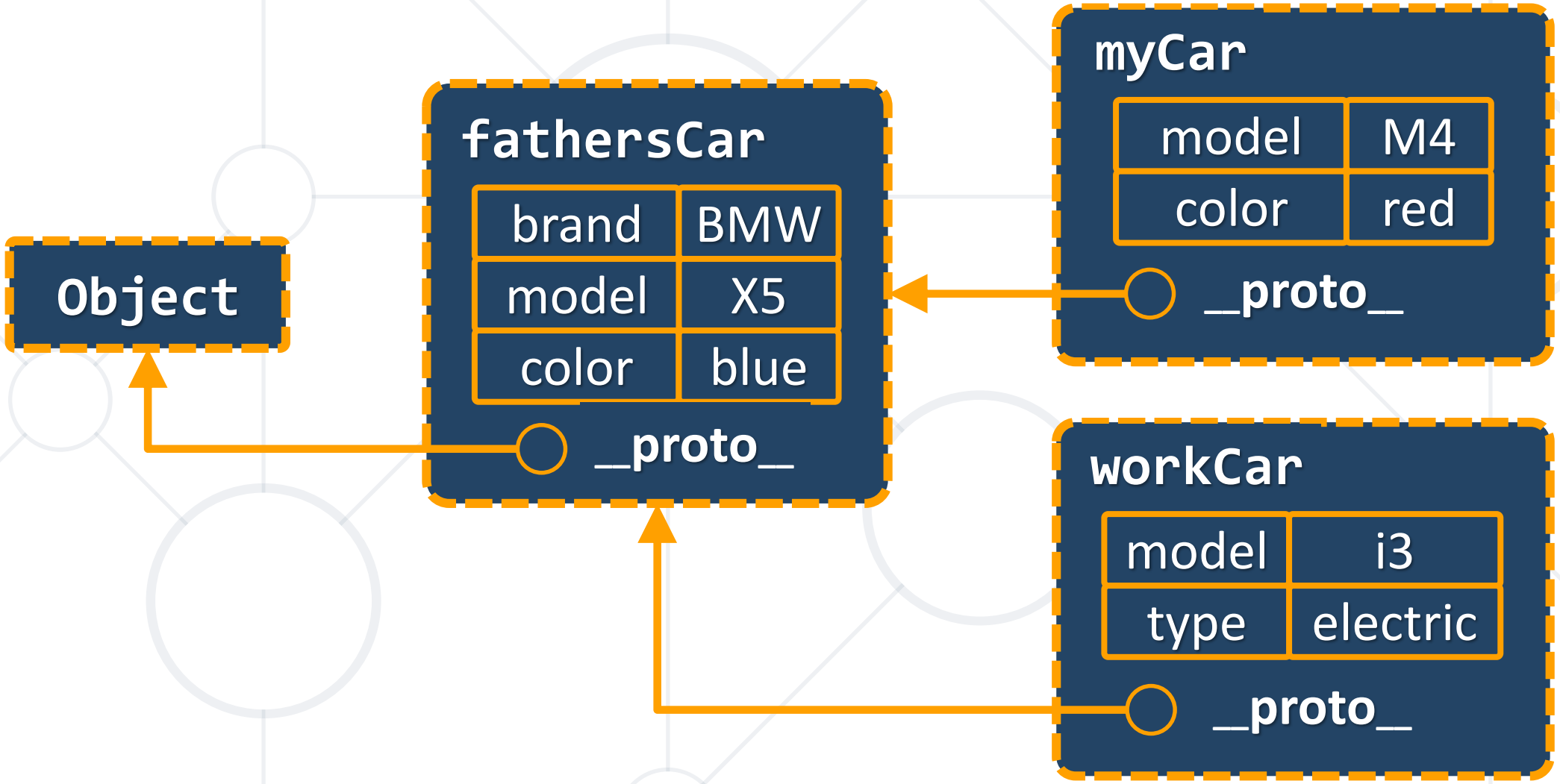


Object Inheritance (2)

```
let workCar =
  Object.create(fatherCar);
workCar.model = 'i3';
workCar.type = 'electric';
workCar.toString = function() {
  return `[brand: ${this.brand}, model:
    ${this.model}, color: ${this.color},
    type: ${this.type}]`;
}
console.log('' + workCar);
```



Prototype Chain



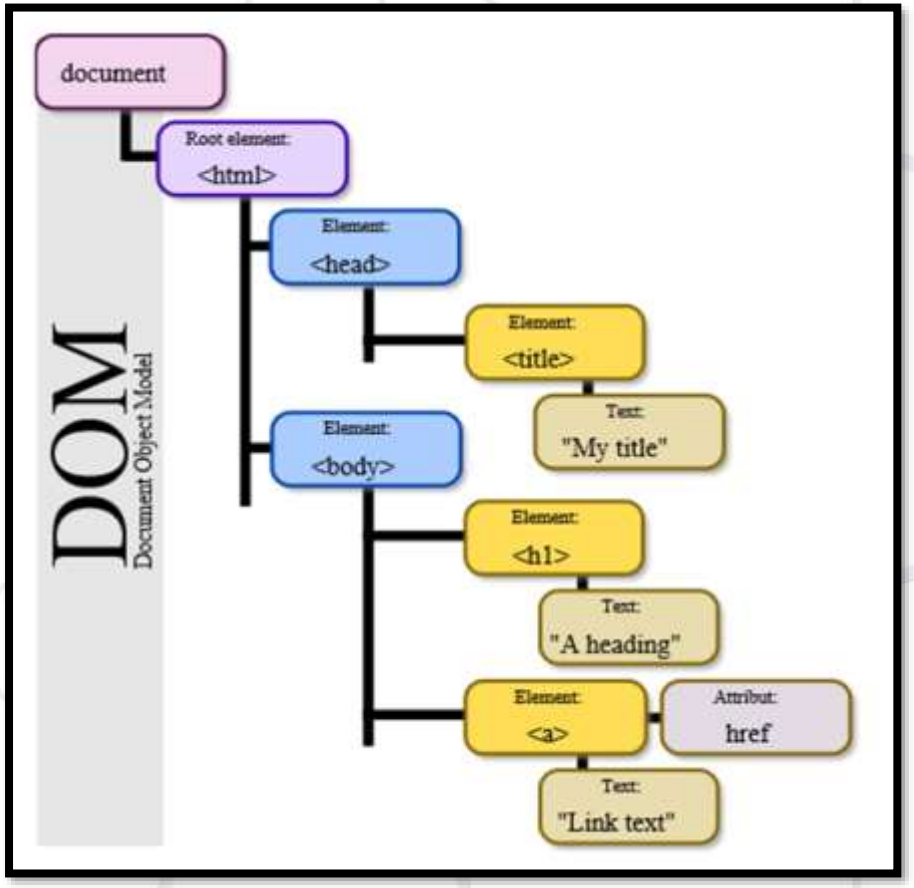
Prototype Chain

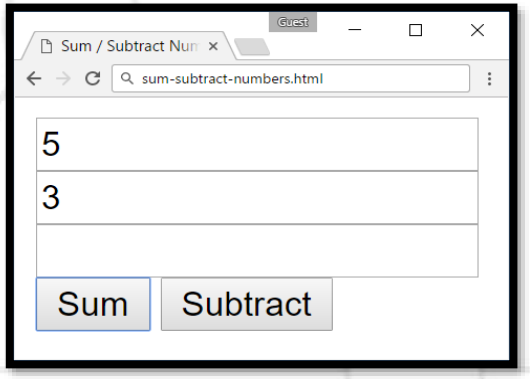
Objects have **prototype** (a parent object)

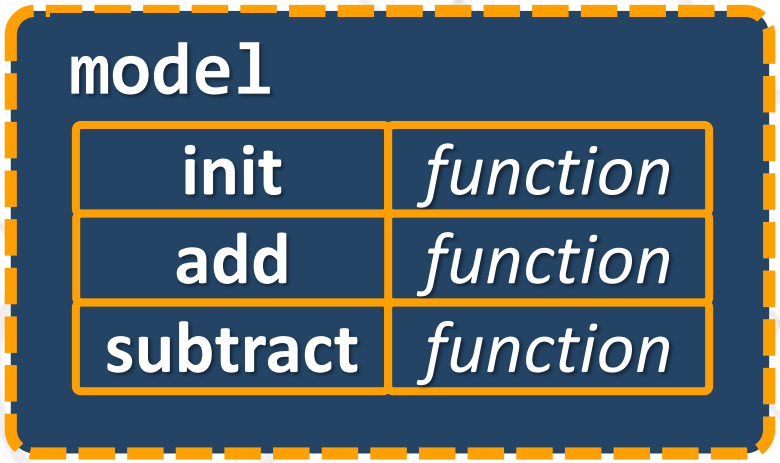
- Prototypes form a **prototype chain**

```
Object.getPrototypeOf(fatherCar);
// Object {}
Object.getPrototypeOf(myCar);
// Object {brand: "BMW", model: "X5", color: "blue"}
```

- If a property **is not found** in the object itself, it is searched in the **parent objects** (in the prototype chain)





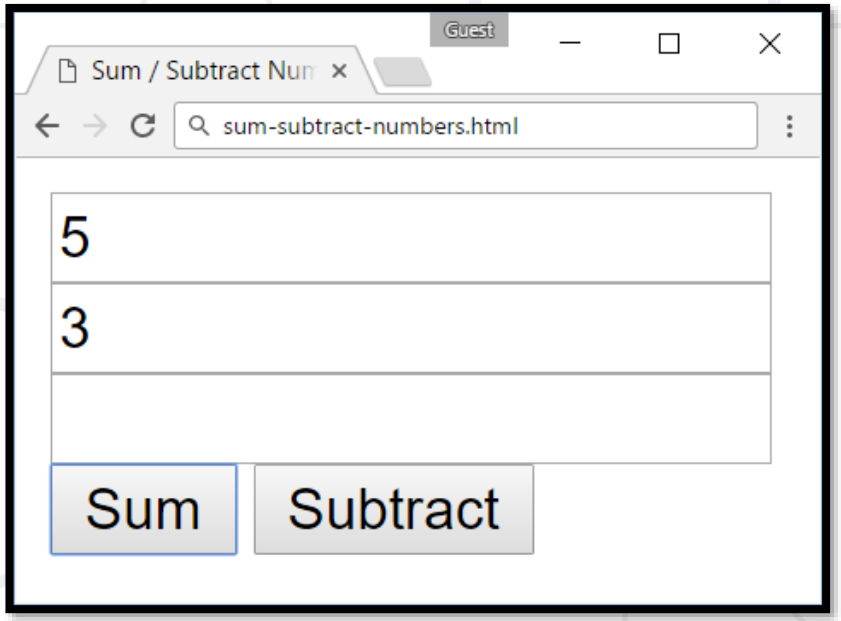


Objects Interacting with DOM

Example: Calculator

You are given the following HTML form

```
<input type="text" id="num1" />
<input type="text" id="num2" />
<input type="text" id="result"
readonly />
<br>
<button id="sumButton">
Sum</button>
<button id="subtractButton">
Subtract</button>
```



Example: Calculator (2)

Write a JS function **getModel**() to return a JS object holding:

- function **init**() - initializes selectors for finding the fields **num1**, **num2** and **result** in the **DOM**
- function **add**() - calculates **result** = **num1** + **num2**
- function **subtract**() - calculates **result** = **num1** - **num2**

```
▼ Object {}
  ▶ add: function ()
  ▶ init: function (num1Sel, num2Sel, resultSel)
  ▶ subtract: function ()
```

Live Exercises

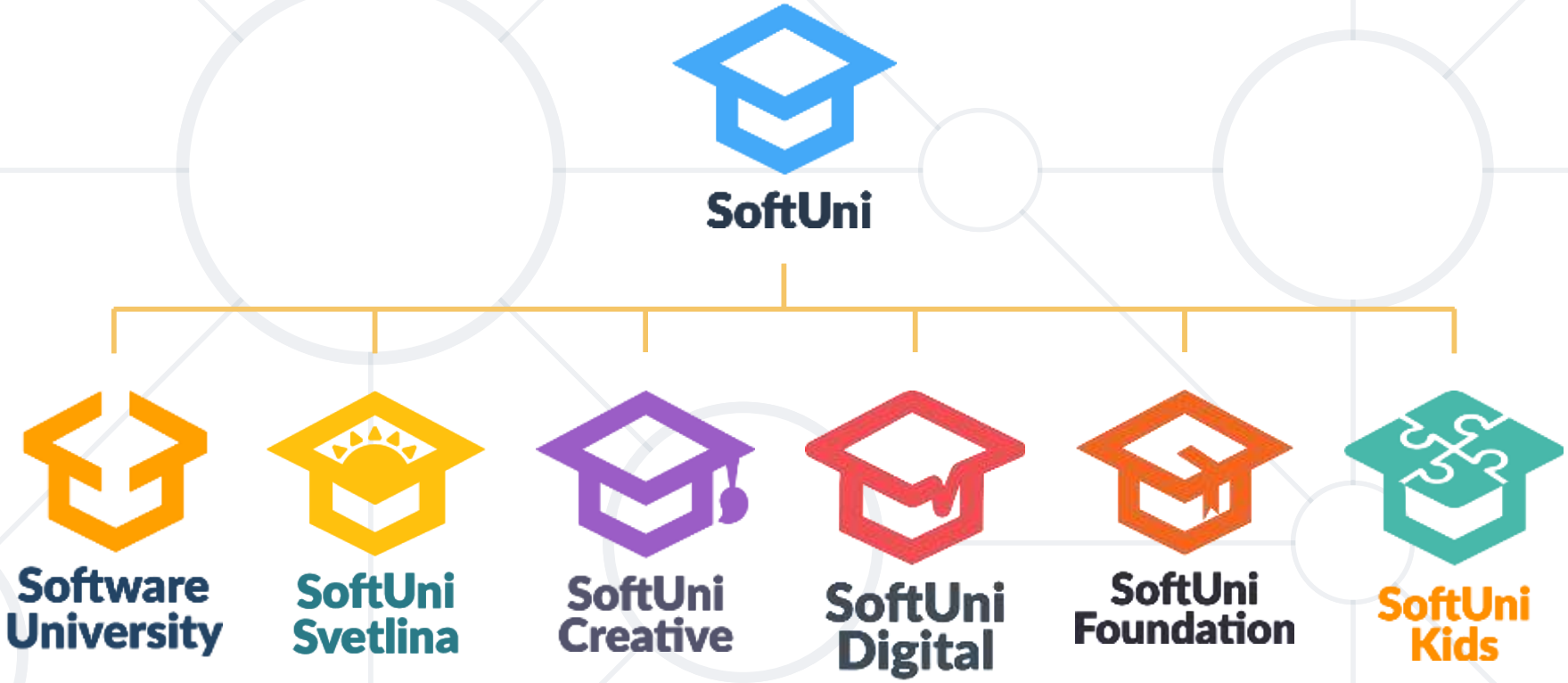
Summary

- Object composition combines **data** and **functions** into JS objects

```
let r = {w:5, h:3, grow:function() { … }}
```

- **The "Module" pattern** hides data into a function and reveals a JS object
- **The "Revealing Module" pattern** hides data and functions are reveals them as JS object
- Objects can **inherit** parent object by **Object.create(parent)**

Questions?



https://softuni.bg/courses/javascript-advanced

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

æternity

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

Lukanet.com

WORLD OF MYTHS

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license