|  |
| --- |
| WEB-015 EcmaScript 2015 |
| Workbook  (version 1.0, 23.09.2016) |

Employees application 3

Task 1: basic classes 3

Task 2: using arrow functions and map/reduce 3

Task 3: using promises 3

Task 4: using generators 4

Task 5: using co library: promises + generators 4

Task 6: adding exception handling to promises 4

Task 7: using async/await syntax (ES2017 feature) 4

Detailed guide 5

Task 1: configuration and basic classes 5

Task 2: using arrow functions and map/reduce 9

Task 3: using promises 10

Task 4: using generators 10

Task 5: using co library: promises + generators 11

Task 6: adding exception handling to promises 12

Task 7: using async/await syntax (ES2017 feature) 13

# Employees application

## Task 1: basic classes

1. Define class Person, with constructor taking name as parameter
   1. Implement getInfo() function using string interpolation: it should return String representation of Person
   2. Implement get/set name; set name should validate name length: it should be not less than 3
2. Define Employee class extending Person, with adding properties salary and position, and overriding getInfo()
3. Define Employees class with encapsulated list of employees with static metods:
   1. add() to add employee to hidden employees array; it should include type check and throw exception if added value is not Employee
   2. list() which returns a copy of all employees array
   3. remove(employee) to remove employee from the \_employees array
4. Create main.js which should:
   1. create several employees and add to Employees using add() function
   2. print list of employees with use of getInfo() method
5. Create employees.html which should use main.js and show all information

## Task 2: using arrow functions and map/reduce

1. Add static method averageSalary() to Employees class which calculates average salary of all employees (use map/reduce)
2. Update main.js which should print average salary of employees

## Task 3: using promises

1. Add method bonus() to Employee which should return Promise having randomly generated bonus in range 0…1000; bonus should be calculated after 1000ms timeout (it imitates long server-side request)
2. Add method total() to Employee which should calculate sum of bonus and salary and return new Promise
3. Print total income of every employee

## Task 4: using generators

1. Add these static functions to Employees:
   1. iterator which allows to iterate over all employees using for (let e of Employees) (**Hint**: function name should be \*[Symbol.iterator] )
   2. generator names() which allows to iterate over all employee names

**Hint:** you should add require("babel-polyfill") to allow generators support in browser

1. Update main.js which should:
2. iterate over Employees using for…of and print info by calling getInfo()
3. print all employees names separated by comma

## Task 5: using co library: promises + generators

1. Install co library (npm i co --save-dev), add it to main.js by using require
2. Use co() to show list of all employees with their salaries, bonuses and totals; list should be shown gradually as it is evaluated (each employee should be shown immediately after bonus is retrieved); use generator+promise technique

## Task 6: adding exception handling to promises

1. Change bonus() method in Employee class so that it reject Promise if bonus is more than 700
2. Change total() method in Employee which handles exception in bonus() and rejects Promise as well
3. Update printing list of employees in main.js (Promise verision) by adding catch block which will print «Bonus is impossibly big» for the employee in case of exception
4. Do the same for co version of printing employees

## Task 7: using async/await syntax (ES2017 feature)

1. Configure webpack to use async/await syntax (add syntax-async-functions to plugins)
2. Create .babelrc with async/await support
3. Use async/await in main.js to print list of employees and bonuses

# Detailed guide

## Task 1: configuration and basic classes

1. Configure environment: create package.json with necessary modules

{  
 **"name"**: **"employees"**,  
 **"version"**: **"1.0.0"**,  
 **"devDependencies"**: {  
 **"babel-cli"**: **"^6.0.0"**,  
 **"babel-polyfill"**: **"^6.13.0"**,  
 **"babel-preset-es2015"**: **"^6.13.0"**,  
 **"webpack"**: **"\*"** }  
}

1. Execute **npm install** to download and install modules
2. Create webpack.config.js with webpack configuration

**module**.**exports** = {  
 **entry**: **"./main"**,  
 **output**: {  
 **path**: **"out"**,  
 **filename**: **"[name].js"** },  
  
 **module**: {  
 **loaders**: [  
 {  
 **test**: /.js$/,  
 **exclude**: /node-modules/,  
 **loader**: **'babel-loader'**,  
 **query**: {  
 **presets**: [**'es2015'**]  
 }  
 }  
 ]  
 },  
  
 **watch**: **true**,  
 **devtool**: **'source-map'**}

1. Define class Person, with constructor taking name as parameter
   1. Implement getInfo() function using string interpolation: it should return String representation of Person
   2. Implement get/set name; set name should validate name length: it should be not less than 3

1) Define class Person in Person.js, export it

**export class** Person {  
 constructor(name) {  
 **this**.**\_name** = name;  
 }  
}

2) Define getInfo() method:

getInfo() {  
 **return `person:** ${**this**.name}**`**}

3) Define getter and setter for name property

**set** name(name) {  
 **if** (name.**length**<3) **throw `incorrect name** ${name}**`  
 this**.**\_name** = name  
}  
  
**get** name() {  
 **return this**.**\_name**}

1. Define Employee class extending Person, with adding properties salary and position, and overriding getInfo()

1) Create Employee.js and import Person

**import** {Person} **from './Person'**

2) Define Employee class which extends Person:

**export class** Employee **extends** Person {  
  
 constructor(name, position, salary) {  
 **super**(name);  
 **this**.**position** = position;  
 **this**.**salary** = salary;  
 }

}

3) Override getInfo() and call getInfo() from superclass

getInfo() {  
 **return super**.getInfo()+

**`** ${**this**.**position**} ${**this**.**salary**}**`**}

1. Define Employees class with encapsulated list of employees with static metods:
   1. add() to add employee to hidden employees list; it should include type check and throw exception if added value is not Employee
   2. list() which returns a copy of all employees list

1) Create Employees.js and import Employee

**import** { Employee } **from "./Employee"**

2) Create exported class Employees

**export class** Employees { }

3) Define module variable which will keep list of employees:

**let** \_employees = []

4) Add static method add() which adds new employee to \_employees array and check if argument is Employee

**static** *add*(employee) {  
 **if** (!employee **instanceof** Employee)

**throw "Can add only employees"** \_employees.push(employee)  
}

5) Add static method list() which returns copy of employees list

**static** *list*() {  
 **return** [...\_employees]  
}

6) Add static method remove(employee) which removes employee from the \_employees array

**static** *remove*(employee) {  
 **const** idx = \_employees.findIndex(employee);  
 \_employees.splice(idx,1)  
}

1. Create main.js which should:
   1. create several employees and add to Employees using add() function
   2. print list of employees with use of getInfo() method

1) Create main.js and import Employee and Employees

**import** {Employees} **from "./Employees"  
import** {Employee} **from "./Employee"**

2) Create several employees and add it with use of Employees.add()

Employees.*add*(**new** Employee(**"John"**,**"manager"**,1000));  
Employees.*add*(**new** Employee(**"Bill"**,**"developer"**,5000));  
Employees.*add*(**new** Employee(**"James"**,**"director"**,4000));

3) Retrieve list of employees

**let** employees = Employees.*list*()

4) Create variable as html placeholder

**let** html=**""**

5) Iterate over employees to add html representation

**for** (**let** e **of** employees) {  
 html += e.getInfo()+**"<br>"**}

6) Put resulting html to the web page

document.getElementById(**"employees"**).**innerHTML** = html;

1. Execute webpack: **node\_modules/.bin/webpack** (on windows) or **webpack** (on unix)

Note: on Windows you can add node\_modules/.bin to PATH and then execute **webpack** directly

Webpack will generate out/main.js with JavaScript 5 code and out/main.js.map with source map

1. Create employees.html which should use main.js and show all information

<**body**>  
 <**div id="employees"**></**div**>  
</**body**>  
<**script src="out/main.js"**></**script**>

1. Open employees.html in browser. It should show list of employees with information generated by getInfo()

## Task 2: using arrow functions and map/reduce

1. Add static method averageSalary() to Employees class which calculates average salary of all employees (use map/reduce)

Add method averageSalary() to Employees class in Employees.js:

**static** *averageSalary*() {  
 **return Math**.round(  
 \_employees.map(e=>e.**salary**).reduce((a,b)=>a+b)  
 /\_employees.**length**)  
}

1. Update main.js which should print average salary of employees

html += **`Average salary:** ${Employees.*averageSalary*()} **<p>`**;

## Task 3: using promises

1. Add method bonus() to Employee which should return Promise having randomly generated bonus in range 0…1000; bonus should be calculated after 1000ms timeout (it imitates long server-side request)

bonus() {  
 **return new** Promise(resolve=>  
 setTimeout(()=>resolve(**Math**.round(**Math**.random()\*1000))  
 ,1000))  
}

1. Add method total() to Employee which should calculate sum of bonus and salary and return new Promise

total() {  
 **return new** Promise(resolve=>  
 **this**.bonus().then(bonus=>  
 resolve(**this**.**salary**+bonus)));  
}

Note that we are able to access **this**.**salary** because of lexical scoping of **this** in arrow function.

For not arrow function we would need to use bind() or this renaming.

1. In main.js move code which modifies html to separate function render:

**function** *render*() {  
 **document**.getElementById(**"employees"**).**innerHTML** = html;  
}

Add call to render() to the end of main.js

1. Print total income of every employee in main.js with use of promises:

**for** (**let** e **of** Employees) {  
 e.total().then(total=>{  
 html += **`**${e.**name**} **total:** ${total} **<br>`**;  
 *render*();  
 });  
}

## Task 4: using generators

1. Add these static functions to Employees:
   1. iterator which allows to iterate over all employees using for (let e of Employees) (**Hint**: function name should be \*[Symbol.iterator] )
   2. generator names() which allows to iterate over all employee names

**Hint:** you should add require("babel-polyfill") to allow generators support in browser

1) Define iterator method in Employees class:

**static** \*[Symbol.**iterator**]() {  
 **yield**\* \_employees;  
}

2) Define names() generator in Employees wich iterates over employees names:

**static** \**names*() {  
 **yield**\* \_employees.map(e=>e.**name**);  
}

3) Modify Employees.js by adding polyfill in the beginning (it adds support of generators):

**require**(**"babel-polyfill"**);

1. Update main.js which should:
2. iterate over Employees using for…of and print info by calling getInfo()
3. print all employees names separated by comma

1) Modify for loop in main.js with use of iterator:

**for** (**let** e **of** Employees) {  
 html += e.getInfo()+**"<br>"**}

2) Print all employees names separated by comma

**let** names = [...Employees.*names*()];  
html += **`Names:** ${names.join(**", "**)} **<p>`**

## Task 5: using co library: promises + generators

1. Install co library (execute **npm i co --save-dev**), add it to main.js by using require

Modify main.js by adding polyfill in the beginning (it adds support of generators):

**require**(**"babel-polyfill"**);

1. Use co() to show list of all employees with their salaries, bonuses and totals; list should be shown gradually as it is evaluated (each employee should be shown immediately after bonus is retrieved); use generator+promise technique

Access promises with use of **co** library (add to main.js):

co(**function** \*() {  
 html += **"<br>co version:<br>"**;  
 **for** (**let** e **of** Employees) {  
 **let** bonus = **yield** e.bonus();  
 html += **`**${e.**name**} **bonus:** ${bonus}

**total:** ${e.**salary**+bonus}**<br>`**;  
 *render*();  
 }  
})

It will print information about bonuses and render it gradually as it is evaluated.

## Task 6: adding exception handling to promises

1. Change bonus() method in Employee class so that it reject Promise if bonus is more than 700

bonus() {  
 **var** bonus = **Math**.round(**Math**.random()\*1000);  
 **return new** Promise((resolve,reject)=>  
 setTimeout(()=>bonus<700?resolve(bonus):reject(bonus),1000))  
}

1. Change total() method in Employee which handles exception in bonus() and rejects Promise as well

total() {  
 **return new** Promise((resolve,reject)=>  
 **this**.bonus()  
 .then(bonus=>resolve(**this**.**salary**+bonus))  
 .catch(bonus=>reject(bonus))  
 )  
}

1. Update printing list of employees in main.js (Promise verision) by adding catch block which will print «Bonus is impossibly big» for the employee in case of exception

**for** (**let** e **of** Employees) {  
 e.total()  
 .then(total=>  
 html += **`**${e.**name**} **total:** ${total} **<br>`**)  
 .catch(bonus=>  
 html += **`**${e.**name**} **bonus is too big**

**(**${bonus}**!) <br>`**)  
 .then(*render*)  
}

1. Do the same for co version of printing employees

co(**function** \*() {  
 html += **"<br>co version:<br>"**;  
 **for** (**let** e **of** Employees) {  
 **try** {  
 **let** bonus = **yield** e.bonus();  
 html += **`**${e.**name**} **bonus:**

${bonus} **total:** ${e.**salary**+bonus}**<br>`**;  
 } **catch**(ex) {  
 html += **`**${e.**name**} **bonus is too BIG (**${ex}**) <br>`**;  
 }  
 *render*();  
 }  
})

## Task 7: using async/await syntax (ES2017 feature)

1. Configure webpack to use async/await syntax (add syntax-async-functions to plugins)

Modify webpack.config.js by adding plugins in module.loaders.query section:

**plugins**: [  
 **"syntax-async-functions"**]

1. Create .babelrc with async/await support

Add file .babelrc with this contents:

{  
 **"presets"**: [**"es2015"**],  
 **"plugins"**: [**"syntax-async-functions"**,

**"transform-regenerator"**]  
}

1. Use async/await in main.js to print list of employees and bonuses

Add to main.js function with async/await syntax:

**async function** *printBonus*() {  
 html += **"<br>Async/await version:<br>"**;  
 **for** (**let** e **of** Employees) {  
 **let** bonus = **await** e.bonus();  
 html += **`**${e.**name**} **bonus:** ${bonus} **total:** ${e.**salary**+bonus}**<br>`**;  
 *render*();  
 }  
}

Execute it:

*printBonus*();

Now you can reload employees.html. You will notice that co() execution and printBonus() execution results are mixed.

1. Modify co() execution to execute printBonus **after** co() has been finished:

co( … ). then(printBonus);