Started on	Wednesday, 6 September 2023, 7:52 AM
State	Finished
Completed on	Wednesday, 6 September 2023, 9:06 AM
Time taken	1 hour 14 mins
Marks	7.00/7.00
Grade	**10.00** out of 10.00 (**100**%)

QUESTION 1

Correct

Mark 1.00 out of 1.00

Create a *Movie* class, the constructor of which accepts 3 parameters: movie name *name*, movie genre *category* and start year of *startShow*.

The class has a *watchMovie()* method that returns a phrase and adds a movie name *name* parameter to it at the end. For example, "I watch the movie Titanic!"

Create an instance of the *movie1* class with the title of the movie "Titanic", the genre "drama" and 1997 release.

Create an instance of the *movie2* class with the title of the movie "Troya", the genre "historical" and the 2004 release.

* For correct passing of all tests don't use console.log() method in your code.

Answer: (penalty regime: 0 %)

Reset answer

```
class Movie {
    constructor(name, category, startShow){
        this.name = name;
        this.category = category;
        this.startShow = startShow;
    }
    watchMovie(){
        return `I watch the movie ${this.name}!`;
    }
}

const movie1 = new Movie('Titanic', 'drama', 1997);
const movie2 = new Movie('Troya', 'historical', 2004);
```

	Test	Expected	Got	
✓	`console.log(Checker.classHasObject(Movie, movie1));`	true	true	✓
✓	`console.log(Checker.classHasObject(Movie, movie2));`	true	true	✓
✓	`console.log(Checker.isFieldValue(movie1, "name"));`	Titanic	Titanic	✓
✓	`console.log(Checker.isFieldValue(movie1, "category"));`	drama	drama	✓
✓	`console.log(Checker.isFieldValue(movie1, "startShow"));`	1997	1997	✓
✓	`console.log(Checker.isFieldValue(movie2, "name"));`	Troya	Troya	✓
✓	`console.log(Checker.isFieldValue(movie2, "category"));`	historical	historical	✓
✓	`console.log(Checker.isFieldValue(movie2, "startShow"));`	2004	2004	✓
✓	`console.log(movie1.watchMovie());`	I watch the movie Titanic!	I watch the movie Titanic!	✓
✓	`console.log(movie2.watchMovie());`	I watch the movie Troya!	I watch the movie Troya!	✓
✓	`const movie3 = new Movie('Mavka. The Forest Song', 'fantasy', 2023); console.log(Checker.isFieldValue(movie3, "category"));`	fantasy	fantasy	✓
✓	`console.log(Checker.isFieldValue(movie3, "startShow"));`	2023	2023	✓
✓	`console.log(movie3.watchMovie());`	I watch the movie Mavka. The Forest Song!	I watch the movie Mavka. The Forest Song!	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION 2

Correct

Mark 1.00 out of 1.00

Implement the *Student* class, the constructor of which accepts 2 parameters *fullName* - the name and surname of the student, *direction* - the direction in which he studies and save them in privete fields.

Create a *getFullName()* method that returns the student's name and surname

Create a *nameIncludes(data)* method that, using the *showFullName()* method, checks for the text *data* argument in the student's name and returns true if a match is found or false if not found.

Create a static *studentBuilder()* method that returns a new instance of the class named 'Ihor Kohut' and the direction 'qc'.

Create a getter and setter *direction()* to read and specify the *direction* field.

Create an instance of class *stud1* named 'Ivan Petrenko' and direction 'web'.

Create an instance of class *stud2* named 'Sergiy Koval' and direction 'python'.

* For correct passing of all tests don't use console.log() method in your code.

Note, that code redactor can display private fields as error but it should be work.

For example:

Test	Result

`console.log(stud1)`
`console.log(stud1.getFullName());`
`console.log(stud1.direction);`
	`Student {}`
	`Ivan Petrenko`
	`web`

`const stud3 = Student.studentBuilder();`
`console.log(stud3)`
`console.log(stud3.getFullName());`
`console.log(stud3.direction);`
	`Student {}`
	`Ihor Kohut`
	`qc`

`console.log(stud1.nameIncludes('Ivan'));`	`true`
`console.log(stud1.nameIncludes('Ihor'));`	`false`

Answer: (penalty regime: 0 %)

Reset answer

```
class Student {
    #fullName;
    #direction;
  constructor(fullName, direction) {
    this.#fullName = fullName;
    this.#direction = direction;
  }

  getFullName() {
    return this.#fullName;
  }

  nameIncludes(data) {
    return this.getFullName().includes(data);
  }

  static studentBuilder() {
    return new Student('Ihor Kohut', 'qc');
  }

  get direction() {
    return this.#direction;
  }

  set direction(direction) {
    this.#direction = direction;
  }
}

const stud1 = new Student('Ivan Petrenko', 'web');
const stud2 = new Student('Sergiy Koval', 'python');

```

	Test	Expected	Got	

✓	`console.log(stud1)`
	`console.log(stud1.getFullName());`
	`console.log(stud1.direction);`
		`Student {}`
		`Ivan Petrenko`
		`web`
			`Student {}`
			`Ivan Petrenko`
			`web`
				✓

✓	`console.log(stud2)`
	`console.log(stud2.getFullName());`
	`console.log(stud2.direction);`
		`Student {}`
		`Sergiy Koval`
		`python`
			`Student {}`
			`Sergiy Koval`
			`python`
				✓

✓	`const stud3 = Student.studentBuilder();`
	`console.log(stud3)`
	`console.log(stud3.getFullName());`
	`console.log(stud3.direction);`
		`Student {}`
		`Ihor Kohut`
		`qc`
			`Student {}`
			`Ihor Kohut`
			`qc`
				✓

✓	`console.log(stud1.nameIncludes('Ivan'));`	`true`	`true`	✓
✓	`console.log(stud1.nameIncludes('Ihor'));`	`false`	`false`	✓
✓	`console.log(stud1.nameIncludes('Petrenko'));`	`true`	`true`	✓
✓	`console.log(stud2.nameIncludes('Sergiy'));`	`true`	`true`	✓
✓	`console.log(stud2.nameIncludes('Koval'));`	`true`	`true`	✓
✓	`console.log(stud2.nameIncludes('Ivan'));`	`false`	`false`	✓
✓	`console.log(stud3.nameIncludes('Ihor'));`	`true`	`true`	✓
✓	`console.log(stud3.nameIncludes('Kohut'));`	`true`	`true`	✓
✓	`console.log(stud3.nameIncludes('Petrenko'));`	`false`	`false`	✓
✓	`console.log(Checker.classHasObject(Student, stud1));`	`true`	`true`	✓
✓	`console.log(Checker.classHasObject(Student, stud2));`	`true`	`true`	✓
✓	`console.log(Checker.classHasObject(Student, stud3));`	`true`	`true`	✓

✓	`stud2.direction = "Java";`
	`console.log(stud2.direction);`
		`Java`	`Java`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION **3**

Correct

Mark 1.00 out of 1.00

Please, fill in missed lines of code.

Product constructor should provide a generation of unique product id within the application no matter how many products are created.

Distributor can store information about products in its *products* property and has an ability to add and remove a product.

addProduct adds a new property to *products* with name of product id and value - product name.

removeProduct removes a property with specified id from *products*

Please, use Symbol data type.

Answer: (penalty regime: 0 %)

Reset answer

```
class Distributor {
  constructor() {
    this.products = {};
  }

  addProduct(id, name) {
    this.products[id] = name;
  }

  removeProduct(id) {
    if (this.products.hasOwnProperty(id)) {
      delete this.products[id];
    }
  }
}

const localDistributor = new Distributor();

class MyProduct {
  constructor(name) {
    this.id = Symbol(name);
    this.name = name;
  }

  distribute(distributor) {
    distributor.addProduct(this.id, this.name);
  }
}

```

	Test	Expected	Got	

✓	`const product1 = new MyProduct('butter');`
	`product1.distribute(localDistributor)`
	`console.log(localDistributor.products);`
		`{ [Symbol(butter)]: 'butter' }`	`{ [Symbol(butter)]: 'butter' }`	✓

✓	`new MyProduct('bread').distribute(localDistributor);`
	`new MyProduct('bread').distribute(localDistributor);`
	`console.log(localDistributor.products);`
		`{`
		`  [Symbol(butter)]: 'butter',`
		`  [Symbol(bread)]: 'bread',`
		`  [Symbol(bread)]: 'bread'`
		`}`
			`{`
			`  [Symbol(butter)]: 'butter',`
			`  [Symbol(bread)]: 'bread',`
			`  [Symbol(bread)]: 'bread'`
			`}`
				✓

✓	`localDistributor.removeProduct(product1.id)`
	`console.log(localDistributor.products);`
		`{ [Symbol(bread)]: 'bread', [Symbol(bread)]: 'bread' }`	`{ [Symbol(bread)]: 'bread', [Symbol(bread)]: 'bread' }`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION **4**

Correct

Mark 1.00 out of 1.00

Implement the *getMin(arr)* function, which takes an array of numbers *arr* and returns the smallest number of the array. To solve the problem, you must use one of the methods to specify the context of this. It is forbidden to use any cycles.

* For correct passing of all tests don't use console.log() method in your code.

Answer: (penalty regime: 0 %)

Reset answer

```
function getMin(arr) {
    return Math.min.apply(null, arr);
}

// Або:
// function getMin(arr) {
//     return Math.min(...arr);
// }
```

	Test	Expected	Got	
✓	`console.log(getMin([12, 6, 22, 13, 7]));`	6	6	✓
✓	`console.log(getMin([8, 0, 11, 24, 14, 9]));`	0	0	✓
✓	`console.log(getMin([15, 26, 2, -3, 3, 33]));`	-3	-3	✓
✓	`console.log(getMin([15, 0.26, 12, 8, 44, 22]));`	0.26	0.26	✓
✓	`console.log(getMin([12, -10, 32, 0.5, -77, 0, -44]));`	-77	-77	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION 5

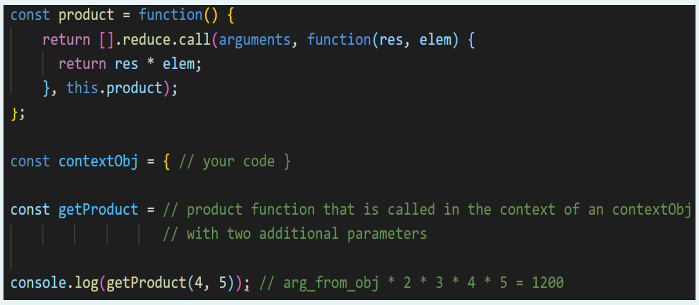
Correct

Mark 1.00 out of 1.00

We have the *product()* function, you can see it on the snapshot below. This *product()* function finds the product of its arguments and also uses *this* object for the initial value of the product.

Please, create a new function *getProduct()* that, no matter how it is called, will be always bound to a particular *this* value. *getProduct()* should be created from the original *product()* function and work with the same logic, but should pass two additional arguments – 2 and 3 – to the original function, every time *getProduct()* is called.

Object *this* for *getProduct()* function you should also define by yourself. Look at snapshot for clues what it should be.



```
const product = function() {
    return [].reduce.call(arguments, function(res, elem) {
      return res * elem;
    }, this.product);
};

const contextObj = { // your code }

const getProduct = // product function that is called in the context of an contextObj
                   // with two additional parameters

console.log(getProduct(4, 5)); // arg_from_obj * 2 * 3 * 4 * 5 = 1200
```

* For correct passing of all tests don't use console.log() method in your code.

Answer: (penalty regime: 0 %)

Reset answer

```
const product = function () {
  return [].reduce.call(
    arguments,
    function (res, elem) {
      return res * elem;
    },
    this.product
  );
};

const contextObj = { product: 10 };

const getProduct = product.bind(contextObj, 2, 3);
```

	Test	Expected	Got	
✓	console.log(getProduct(4, 5));	1200	1200	✓
✓	console.log(getProduct(6, 7));	2520	2520	✓
✓	console.log(getProduct(0, 7));	0	0	✓
✓	console.log(getProduct(6, 0));	0	0	✓
✓	console.log(getProduct(-5, 5));	-1500	-1500	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION 6

Correct

Mark 1.00 out of 1.00

Implement the *Plane* class, the constructor of which accepts 3 parameters *model* - model of the plane, *fuelSupply* - capacity of a stock of fuel of the plane, *fuelConsumption* - average fuel consumption in liters on 100 km of flight.

Create a method of class *calcFlightRange(),* which determines the range of the plane by the formula *fuelSupply / fuelConsumption* * 100 and returns it.

Create a static method of class *showSortedByFlightRange(planesArray)*, which takes an array of instances of classes *planesArray*, sorts the flight range of plane in ascending order and outputs the result to the console in the format plane_model: range.

Create a *TransportPlane* class, which is inherited from the *Plane* class, the constructor of which takes 5 parameters *model* - plane model, *fuelSupply* - the amount of fuel, *fuelConsumption* - the average fuel consumption in liters per 100 km, *cargo* - maximum tonnage, *addTank* - about additional tanks of the plane In this class, you need to override the *calcFlightRange()* method to take into account that the *fuelSupply* has increased the amount of fuel added by the *addTank*.

Create a class *PassengerPlane*, which is inherited from the class *Plane*, whose constructor accepts 5 parameters *model*, *fuelSupply*, *fuelConsumption*, *passengers* - the maximum number of passengers, *refueling* - the amount of additional fuel received in the refueling. In this class, you need to override the *calcFlightRange()* method to take into account that the *fuelSupply* has increased refueling.

Create a *WarPlane* class, which is inherited from the *Plane* class, the constructor of which accepts 5 parameters *model*, *fuelSupply*, *fuelConsumption*, *missiles* - the number of missile weapons, *aerodynamicsKoef* - the coefficient of aerodynamics of the plane. In this class, you need to override the *calcFlightRange()* method in such a way as to take into account that the flight range of the plane increases in proportion to the value of the aerodynamics coefficient of *aerodynamicsKoef*.

For example:

Test	Result

`const plane1 = new TransportPlane("An-225 Mriya", 105000, 5000, 500, 300000);`
``console.log(`${plane1.model}: ${plane1.calcFlightRange()}`);``
	`An-225 Mriya: 8100`

`const plane2 = new PassengerPlane("Boeing-747", 193000, 2500, 410, 90000);`
``console.log(`${plane2.model}: ${plane2.calcFlightRange()}`);``
	`Boeing-747: 11320`

`const plane3 = new WarPlane("F-22 Raptor", 8200, 320, 20, 1.2);`
``console.log(`${plane3.model}: ${plane3.calcFlightRange()}`);``
	`F-22 Raptor: 3075`

`const planesArray = [plane1, plane2, plane3];`
`console.log("Sorted range of planes:");`
`Plane.showSortedByFlightRange(planesArray);`
	`Sorted range of planes:`
	`F-22 Raptor: 3075`
	`An-225 Mriya: 8100`
	`Boeing-747: 11320`

Answer: (penalty regime: 0 %)

Reset answer

```
class Plane {
  constructor(model, fuelSupply, fuelConsumption) {
    this.model = model;
    this.fuelSupply = fuelSupply;
    this.fuelConsumption = fuelConsumption;
  }

  calcFlightRange() {
    let range = (this.fuelSupply / this.fuelConsumption) * 100;
    return range;
  }

  static showSortedByFlightRange(planesArray) {
    const copiedArray = planesArray.slice();
    const sortedPlanes = copiedArray.sort(
      (a, b) => a.calcFlightRange() - b.calcFlightRange()
    );
     sortedPlanes.map((plane) => {
      console.log(`${plane.model}: ${plane.calcFlightRange()}`);
    });
  }
}

class TransportPlane extends Plane {
  constructor(model, fuelSupply, fuelConsumption, cargo, addTank) {
    super(model, fuelSupply, fuelConsumption);
    this.cargo = cargo;
    this.addTank = addTank;
  }

  calcFlightRange() {
    let increasedFuelSupply = this.fuelSupply + this.addTank;
    let range = (increasedFuelSupply / this.fuelConsumption) * 100;
    return range;
  }
```

```
}

class PassengerPlane extends Plane {
  constructor(model, fuelSupply, fuelConsumption, passengers, refueling) {
    super(model, fuelSupply, fuelConsumption);
    this.passengers = passengers;
    this.refueling = refueling;
  }
  calcFlightRange() {
    let increasedFuelSupply = this.fuelSupply + this.refueling;
    let range = (increasedFuelSupply / this.fuelConsumption) * 100;
    return range;
  }
}

class WarPlane extends Plane {
  constructor(model, fuelSupply, fuelConsumption, missiles, aerodynamicsKoef) {
```

	Test	Expected	Got	

✓	`const plane1 = new TransportPlane("An-225 Mriya", 105000, 5000, 500, 300000);`
	``console.log(`${plane1.model}: ${plane1.calcFlightRange()}`);``
		`An-225 Mriya: 8100`	`An-225 Mriya: 8100`	✓

✓	`const plane2 = new PassengerPlane("Boeing-747", 193000, 2500, 410, 90000);`
	``console.log(`${plane2.model}: ${plane2.calcFlightRange()}`);``
		`Boeing-747: 11320`	`Boeing-747: 11320`	✓

✓	`const plane3 = new WarPlane("F-22 Raptor", 8200, 320, 20, 1.2);`
	``console.log(`${plane3.model}: ${plane3.calcFlightRange()}`);``
		`F-22 Raptor: 3075`	`F-22 Raptor: 3075`	✓

✓	`const planesArray = [plane1, plane2, plane3];`
	`console.log("Sorted range of planes:");`
	`Plane.showSortedByFlightRange(planesArray);`
		`Sorted range of planes:`
		`F-22 Raptor: 3075`
		`An-225 Mriya: 8100`
		`Boeing-747: 11320`
			`Sorted range of planes:`
			`F-22 Raptor: 3075`
			`An-225 Mriya: 8100`
			`Boeing-747: 11320`
				✓

✓	`console.log(planesArray.map(plane => plane.model));`	`[ 'An-225 Mriya', 'Boeing-747', 'F-22 Raptor' ]`	`[ 'An-225 Mriya', 'Boeing-747', 'F-22 Raptor' ]`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION 7

Correct

Mark 1.00 out of 1.00

Implement the *PizzaMaker* class, which allows you to create pizza of different types, with different ingredients, calculate the price and calorie content of pizza.

The pizza comes in 3 sizes: S, M and L.

There are 4 types of pizza available: meat, fish, cheese and vegetarian.

When creating a new pizza, be sure to specify the size and appearance.

Additional ingredients are available that can be added to the pizza at the customer's request: tomatoes, peppers, bacon and olives.

Each element that makes up pizza has its own name, price and calorie content. All of this data is contained in the *pizzaMenu* object.

The *PizzaMaker* class has a number of methods for generating pizza:

- *addIngredient(ingredient)* method adds an additional ingredient to the pizza. A new ingredient is added if it is not included in the pizza, and the message "ingredient_name has been added" is displayed in the console. If such an ingredient has already been added, the message "Such an ingredient already exists!" Is generated.

- *deleteIngredient(ingredient)* method removes the specified ingredient from the list of existing ingredients, displays the message "ingredient_name has been deleted" to the console.

- *getIngredients()* method returns a list of the attached ingredients with their name, price, calorie content.

- *getSize()* method returns the size of the pizza.

- *getKind()* method returns the type of pizza.

- *calculatePrice()* method calculates and returns the total cost of a pizza, which consists of the sum of the values of all its components.

- *calculateCalories()* method calculates and returns the total calorie content of a pizza, which consists of the sum of the calories of all its components.

For example:

Test	Result

`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
`console.log("Size:", pizza.getSize());`
	`Size: SIZE_M`

`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
`console.log("Kind:", pizza.getKind());`
	`Kind: KIND_MEAT`

`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
`console.log("calculatePrice:", pizza.calculatePrice());`
	`calculatePrice: 145`

`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
`console.log("calculateCalories:", pizza.calculateCalories());`
	`calculateCalories: 680`

`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
`console.log("getIngredients:", pizza.getIngredients());`
	`getIngredients: []`

`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
	`INGREDIENT_BACON has been added`
	`Such an ingredient already exists!`

`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
`pizza.addIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
`pizza.deleteIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
	`INGREDIENT_TOMATOES has been added`
	`INGREDIENT_BACON has been added`
	`INGREDIENT_TOMATOES has been deleted`

Answer: (penalty regime: 0 %)

Reset answer

```
const pizzaMenu = {
  SIZE_S: {param: "SIZE_S", price: 60, calorie: 300},
  SIZE_M: {param: "SIZE_M", price: 90, calorie: 450},
  SIZE_L: {param: "SIZE_L", price: 110, calorie: 600},
  KIND_MEAT: {param: "KIND_MEAT", price: 55, calorie: 230},
  KIND_FISH: {param: "KIND_FISH", price: 70, calorie: 150},
  KIND_CHEESE: {param: "KIND_CHEESE", price: 50, calorie: 200},
  KIND_VEGETARIAN: {param: "KIND_VEGETARIAN", price: 35, calorie: 50},
  INGREDIENT_TOMATOES: {param: "INGREDIENT_TOMATOES", price: 15, calorie: 5},
  INGREDIENT_PEPPER: {param: "INGREDIENT_PEPPER", price: 18, calorie: 5},
  INGREDIENT_BACON: {param: "INGREDIENT_BACON", price: 25, calorie: 40},
  INGREDIENT_OLIVES: {param: "INGREDIENT_OLIVES", price: 20, calorie: 0}
};

class PizzaMaker {
  constructor(size, kind) {
    this.size = size;
    this.size = size;
    this.kind = kind;
    this.ingredients = [];
  }

  addIngredient(ingredient) {
    if (!this.ingredients.includes(ingredient.param)) {
      this.ingredients.push(ingredient.param);
      console.log(`${ingredient.param} has been added`);
    } else {
      console.log(`Such an ingredient already exists!`);
    }
  }

  deleteIngredient(ingredient) {
    const index = this.ingredients.indexOf(ingredient.param);
    this.ingredients.splice(index, 1);
    console.log(`${ingredient.param} has been deleted`);

  }

  getIngredients() {
    const ingredientList = this.ingredients.map(
      (ingredientParam) => pizzaMenu[ingredientParam]
    );
    return ingredientList;
  }

  getSize() {
    return this.size.param;
  }

  getKind() {
    return this.kind.param;
  }

```

	Test	Expected	Got	

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`console.log("Size:", pizza.getSize());`
		`Size: SIZE_M`	`Size: SIZE_M`	✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`console.log("Kind:", pizza.getKind());`
		`Kind: KIND_MEAT`	`Kind: KIND_MEAT`	✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`console.log("calculatePrice:", pizza.calculatePrice());`
		`calculatePrice: 145`	`calculatePrice: 145`	✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`console.log("calculateCalories:", pizza.calculateCalories());`
		`calculateCalories: 680`	`calculateCalories: 680`	✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`console.log("getIngredients:", pizza.getIngredients());`
		`getIngredients: []`	`getIngredients: []`	✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
		`INGREDIENT_TOMATOES has been added`	`INGREDIENT_TOMATOES has been added`	✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
		`INGREDIENT_BACON has been added`
		`Such an ingredient already exists!`
			`INGREDIENT_BACON has been added`
			`Such an ingredient already exists!`
				✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
		`INGREDIENT_TOMATOES has been added`
		`INGREDIENT_BACON has been added`
			`INGREDIENT_TOMATOES has been added`
			`INGREDIENT_BACON has been added`
				✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
	`console.log("getIngredients:", pizza.getIngredients());`
		`INGREDIENT_TOMATOES has been added`
		`INGREDIENT_BACON has been added`
		`getIngredients: [`
		`  { param: 'INGREDIENT_TOMATOES', price: 15, calorie: 5 },`
		`  { param: 'INGREDIENT_BACON', price: 25, calorie: 40 }`
		`]`
			`INGREDIENT_TOMATOES has been added`
			`INGREDIENT_BACON has been added`
			`getIngredients: [`
			`  { param: 'INGREDIENT_TOMATOES', price: 15, calorie: 5 },`
			`  { param: 'INGREDIENT_BACON', price: 25, calorie: 40 }`
			`]`
				✓

	Test	Expected	Got	

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
	`pizza.deleteIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
		`INGREDIENT_TOMATOES has been added`
		`INGREDIENT_BACON has been added`
		`INGREDIENT_TOMATOES has been deleted`
			`INGREDIENT_TOMATOES has been added`
			`INGREDIENT_BACON has been added`
			`INGREDIENT_TOMATOES has been deleted`
				✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
	`pizza.deleteIngredient(pizzaMenu.INGREDIENT_TOMATOES);`
	`console.log("getIngredients:", pizza.getIngredients());`
		`INGREDIENT_TOMATOES has been added`
		`INGREDIENT_BACON has been added`
		`INGREDIENT_TOMATOES has been deleted`
		`getIngredients: [ { param: 'INGREDIENT_BACON', price: 25, calorie: 40 } ]`
			`INGREDIENT_TOMATOES has been added`
			`INGREDIENT_BACON has been added`
			`INGREDIENT_TOMATOES has been deleted`
			`getIngredients: [ { param: 'INGREDIENT_BACON', price: 25, calorie: 40 } ]`
				✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_PEPPER);`
	`console.log("getIngredients:", pizza.getIngredients());`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_PEPPER);`
		`INGREDIENT_PEPPER has been added`
		`getIngredients: [ { param: 'INGREDIENT_PEPPER', price: 18, calorie: 5 } ]`
		`Such an ingredient already exists!`
			`INGREDIENT_PEPPER has been added`
			`getIngredients: [ { param: 'INGREDIENT_PEPPER', price: 18, calorie: 5 } ]`
			`Such an ingredient already exists!`
				✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_L, pizzaMenu.KIND_FISH);`
	`console.log("Size:", pizza.getSize());`
	`console.log("Kind:", pizza.getKind());`
		`Size: SIZE_L`
		`Kind: KIND_FISH`
			`Size: SIZE_L`
			`Kind: KIND_FISH`
				✓

✓	`const pizza = new PizzaMaker(pizzaMenu.SIZE_M, pizzaMenu.KIND_MEAT);`
	`console.log("Size:", pizza.getSize());`
	`console.log("Kind:", pizza.getKind());`
	`console.log("calculatePrice:", pizza.calculatePrice());`
	`console.log("calculateCalories:", pizza.calculateCalories());`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_OLIVES);`
	`pizza.addIngredient(pizzaMenu.INGREDIENT_BACON);`
	`console.log("calculatePrice:", pizza.calculatePrice());`
	`console.log("calculateCalories:", pizza.calculateCalories());`
		`Size: SIZE_M`
		`Kind: KIND_MEAT`
		`calculatePrice: 145`
		`calculateCalories: 680`
		`INGREDIENT_OLIVES has been added`
		`INGREDIENT_BACON has been added`
		`calculatePrice: 190`
		`calculateCalories: 720`
			`Size: SIZE_M`
			`Kind: KIND_MEAT`
			`calculatePrice: 145`
			`calculateCalories: 680`
			`INGREDIENT_OLIVES has been added`
			`INGREDIENT_BACON has been added`
			`calculatePrice: 190`
			`calculateCalories: 720`
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

◄ Video. Call, apply, bind [13:00]

Jump to...

Quiz. Object-oriented design (OOD) / Object-oriented programming (OOP) ►