



JavaScript

STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA

HOMEWORK



1. Make a car object

- Hardcode a car object
- The object should have properties: acceleration and maxSpeed
- The object should have a method: getSpeed that accepts time in seconds as parameter
- The method should return us speed that the car will have after the time

example.js

```
1
2 const car = {
3   acceleration: 5,
4   maxSpeed: 200,
5   getSpeed(time) {
6     const calculateSpeed = this.acceleration * time;
7     return calculateSpeed > this.maxSpeed ? this.maxSpeed : calculateSpeed;
8   }
9 }
10
```

2. Make a car function

- Create function to produce car objects
- Created object should have properties: acceleration and maxSpeed
- The object should have a method: getSpeed that accepts time in seconds as parameter
- The method should be available to every created object

example.js

```
1
2 function Car(acceleration, maxSpeed) {
3   this.acceleration = acceleration;
4   this.maxSpeed = maxSpeed;
5 }
6
7 Car.prototype.getSpeed = function (time) {
8   const calculatedSpeed = this.acceleration * time;
9   return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
10 };
11
```

3. Make a car class

- Write a class that will produce car objects
- Created object should have fields: acceleration, maxSpeed, wheels
- Created object should have method getSpeed

```
1
2 class Car {
3   acceleration;
4   maxSpeed;
5   wheels = 4;
6
7   constructor(acceleration, maxSpeed) {
8     this.acceleration = acceleration;
9     this.maxSpeed = maxSpeed;
10  }
11
12  getSpeed(time) {
13    const calculatedSpeed = this.acceleration * time;
14    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
15  }
16 }
17
```


4. Add features to the class

- Add new fields:
 - id
 - price
 - production date
 - and a method `changePrice`

```
1
2 class Car {
3   wheels = 4;
4
5   constructor(acceleration, maxSpeed, price, productionDate) {
6     this.id = Date.now(); // correct way is to use uuid
7     this.acceleration = acceleration;
8     this.maxSpeed = maxSpeed;
9     this.price = price;
10    this.productionDate = productionDate
11      ? new Date(productionDate)
12      : new Date();
13  }
14
15  getSpeed(time) {
16    const calculatedSpeed = this.acceleration * time;
17    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
18  }
19
20  changePrice(newPrice) {
21    if (typeof newPrice !== "number") {
22      throw new Error("Price should be a number!");
23    }
24    this.price = newPrice;
25  }
26 }
27
```

5. Add static method

- Add static method that for a given object, will check if the object is a car

```
1
2 class Car {
3   wheels = 4;
4
5   constructor(acceleration, maxSpeed, price, productionDate) {
6     this.id = Date.now(); // correct way is to use uuid
7     this.acceleration = acceleration;
8     this.maxSpeed = maxSpeed;
9     this.price = price;
10    this.productionDate = productionDate
11      ? new Date(productionDate)
12      : new Date();
13  }
14
15  getSpeed(time) {
16    const calculatedSpeed = this.acceleration * time;
17    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
18  }
19
20  changePrice(newPrice) {
21    if (typeof newPrice !== "number") {
22      throw new Error("Price should be a number!");
23    }
24    this.price = newPrice;
25  }
26
27  static isCar(inputObj) {
28    return inputObj instanceof Car;
29  }
30 }
31
```

6. Add another feature

- Add new field: status
- Create Car status object with acceptable statuses (NEW, USED, REFUND)
- Add method changeStatus that accepts new status as a parameter
- Add static method to check if car is after return

```
example.js

1
2 const CAR_STATUS = require("../const");
3
4 class Car {
5   wheels = 4;
6   status = CAR_STATUS.NEW;
7
8   constructor(acceleration, maxSpeed, price, productionDate) {
9     this.id = Date.now();
10    this.acceleration = acceleration;
11    this.maxSpeed = maxSpeed;
12    this.price = price;
13    this.productionDate = productionDate
14      ? new Date(productionDate)
15      : new Date();
16  }
17
18  getSpeed(time) {
19    const calculatedSpeed = this.acceleration * time;
20    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
21  }
22
23  changePrice(newPrice) {
24    if (typeof newPrice !== "number") {
25      throw new Error("Price should be a number!");
26    }
27    this.price = newPrice;
28  }
29
30  changeStatus(newStatus) {
31    const acceptableStatuses = Object.values(CAR_STATUS);
32    if (!acceptableStatuses.includes(newStatus)) {
33      throw new Error("Incorrect status");
34    }
35    this.status = newStatus;
36  }
37
38  static isCar(inputObj) {
39    return inputObj instanceof Car;
40  }
41
42  static isRefund(car) {
43    if (!Car.isCar(car)) {
44      throw new Error("Input should be a car!");
45    }
46
47    return car.status === CAR_STATUS.REFUND;
48  }
49 }
50
```

7. Car Dealer

- Create a class – Car Dealer
- Class should have field: name
- It should have a private field where cars are stored
- A getter availableCars to see all available cars for sell

example.js

```
1
2 class CarDealer {
3   #carsStorage;
4
5   constructor(name) {
6     this.#carsStorage = [];
7     this.name = name;
8   }
9
10  getAvailableCars() {
11    return this.#carsStorage;
12  }
13
14  get availableCars() {
15    return this.#carsStorage;
16  }
17 }
18
```


8. Car Dealer

- Add new methods
- A method to add a car (that would check if the input object is a car)
- A method to remove a car by ID
- A getter that would return the total price of all cars in the storage

```
example.js
1
2 const Car = require("../car-class");
3
4 class CarDealer {
5   #carsStorage;
6
7   constructor(name) {
8     this.#carsStorage = [];
9     this.name = name;
10  }
11
12  addCar(newCar) {
13    if (!Car.isCar(newCar)) {
14      throw new Error("Input should be a car!");
15    }
16
17    this.#carsStorage.push(newCar);
18  }
19
20  removeCar(id) {
21    this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
22  }
23
24  getAvailableCars() {
25    return this.#carsStorage;
26  }
27
28  get availableCars() {
29    return this.#carsStorage;
30  }
31
32  get totalCarsPrice() {
33    return this.#carsStorage.reduce((sum, car) => {
34      return (sum += car.price);
35    }, 0);
36  }
37 }
```

9. Car Dealer

- Add new methods
- A method to accept car return, method should mark car as refund
- Add a static method, that would check if a given car is a refund car (it will have "REFUND" status)

```
example.js

1
2 const Car = require("./car-class");
3
4 class CarDealer {
5   #carsStorage;
6
7   constructor(name) {
8     this.#carsStorage = [];
9     this.name = name;
10  }
11
12  acceptCarReturn(car) {
13    car.changeStatus("REFUND");
14    this.#carsStorage.push(car);
15  }
16
17  addCar(newCar) {
18    if (!Car.isCar(newCar)) {
19      throw new Error("Input should be a car!");
20    }
21
22    this.#carsStorage.push(newCar);
23  }
24
25  removeCar(id) {
26    this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
27  }
28
29  getAvailableCars() {
30    return this.#carsStorage;
31  }
32
33  get availableCars() {
34    return this.#carsStorage;
35  }
36
37  get totalCarsPrice() {
38    return this.#carsStorage.reduce((sum, car) => {
39      return (sum += car.price);
40    }, 0);
41  }
42
43  static isCarAfterRefund(car) {
44    return Car.isRefund(car);
45  }
46 }
```

10. Car Dealer

- Add new method `orderFromFactory`
- Method should be asynchronous
- Method should accept cars amount to order as a parameter
- Handle error in this method in case something goes wrong

```
example.js
1
2 const Car = require("../car-class");
3 const { orderCars } = require("../fake-api");
4
5 class CarDealer {
6   #carsStorage;
7
8   constructor(name) {
9     this.#carsStorage = [];
10    this.name = name;
11  }
12
13  acceptCarReturn(car) {
14    car.changeStatus("REFUND");
15    this.#carsStorage.push(car);
16  }
17
18  addCar(newCar) {
19    if (!Car.isCar(newCar)) {
20      throw new Error("Input should be a car!");
21    }
22
23    this.#carsStorage.push(newCar);
24  }
25
26  removeCar(id) {
27    this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
28  }
29
30  getAvailableCars() {
31    return this.#carsStorage;
32  }
33
34  async orderCarsFromFactory(carsAmount) {
35    try {
36      const orderedCars = await orderCars(carsAmount);
37      orderedCars.forEach((car) => this.addCar(car));
38      return this.#carsStorage;
39    } catch (e) {
40      console.error(e);
41      return this.#carsStorage;
42    }
43  }
44
45  get availableCars() {
46    return this.#carsStorage;
47  }
48
49  get totalCarsPrice() {
50    return this.#carsStorage.reduce((sum, car) => {
51      return (sum += car.price);
52    }, 0);
53  }
54
55  static isCarAfterRefund(car) {
56    return Car.isRefund(car);
57  }
58 }
```

11. Car Dealer

- Add new method sell
- Method should accept id as a parameter and return found car
- Method should also remove car from storage, create transaction history entry in new private transactionsHistory field
- Method should include discount from base car price

```

1
2 const Car = require("./car-class");
3 const { orderCars } = require("./fake-api");
4
5 class CarDealer {
6   #carsStorage;
7   #transactionsHistory;
8
9   constructor(name) {
10     this.#carsStorage = [];
11     this.#transactionsHistory = [];
12     this.name = name;
13   }
14
15   acceptCarReturn(car) {
16     car.changeStatus("REFUND");
17     this.#carsStorage.push(car);
18   }
19
20   addCar(newCar) {
21     if (!Car.isCar(newCar)) {
22       throw new Error("Input should be a car!");
23     }
24
25     this.#carsStorage.push(newCar);
26   }
27
28   removeCar(id) {
29     this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
30   }
31
32   getAvailableCars() {
33     return this.#carsStorage;
34   }
35

```

```

36   async orderCarsFromFactory(carsAmount) {
37     try {
38       const orderedCars = await orderCars(carsAmount);
39       orderedCars.forEach((car) => this.addCar(car));
40       return this.#carsStorage;
41     } catch (e) {
42       console.error(e);
43       return this.#carsStorage;
44     }
45   }
46
47   sell(id) {
48     const carToSell = this.#carsStorage.find((car) => car.id === id);
49
50     if (!carToSell) {
51       throw new Error("Sorry this car has already been sold");
52     }
53
54     this.#useDicount(carToSell);
55     this.#transactionsHistory.push(carToSell);
56
57     return carToSell;
58   }
59
60   get availableCars() {
61     return this.#carsStorage;
62   }
63
64   get totalCarsPrice() {
65     return this.#carsStorage.reduce((sum, car) => {
66       return (sum += car.price);
67     }, 0);
68   }
69
70   static isCarAfterRefund(car) {
71     return Car.isRefund(car);
72   }
73
74   #useDicount(car) {
75     if (car.price >= 200_000) {
76       car.changePrice(car.price * 0.9);
77     } else if (car.price >= 150_000) {
78       car.changePrice(car.price * 0.95);
79     } else {
80       car.changePrice(car.price * 0.98);
81     }
82   }
83 }

```


Electric Car.

- Create a class - `ElectricCar` that will extend the `Car` class
- Add a new field: `batteryCapacity`

example.js

```
1
2 const Car = require("../car-class");
3
4 class ElectricCar extends Car {
5     constructor(acceleration, maxSpeed, price, productionDate, batteryCapacity) {
6         super(acceleration, maxSpeed, price, productionDate);
7         this.batteryCapacity = batteryCapacity;
8     }
9 }
10
```

Electric Car.

- Add getRemainingBattery method that will return % battery left after n amount of seconds
- This method should use a private method that will calculate battery drainage per second (let's say its acceleration * batteryCapacity / 100000)

```
1  const Car = require('./car.js');
2
3  class ElectricCar extends Car {
4    constructor(acceleration, maxSpeed, price, batteryCapacity) {
5      super(acceleration, maxSpeed, price);
6      this.batteryCapacity = batteryCapacity;
7    }
8
9    getRemainingBattery(time) {
10      return (
11        ((this.batteryCapacity - this.#calculateBatteryDrainagePerSec() * time) /
12          this.batteryCapacity) *
13        100
14      );
15    }
16
17    #calculateBatteryDrainagePerSec() {
18      return (this.batteryCapacity * this.acceleration) / 100000;
19    }
20  }
21
```

HOMework

- Josephus's Problem
- 1 EXAM tasks