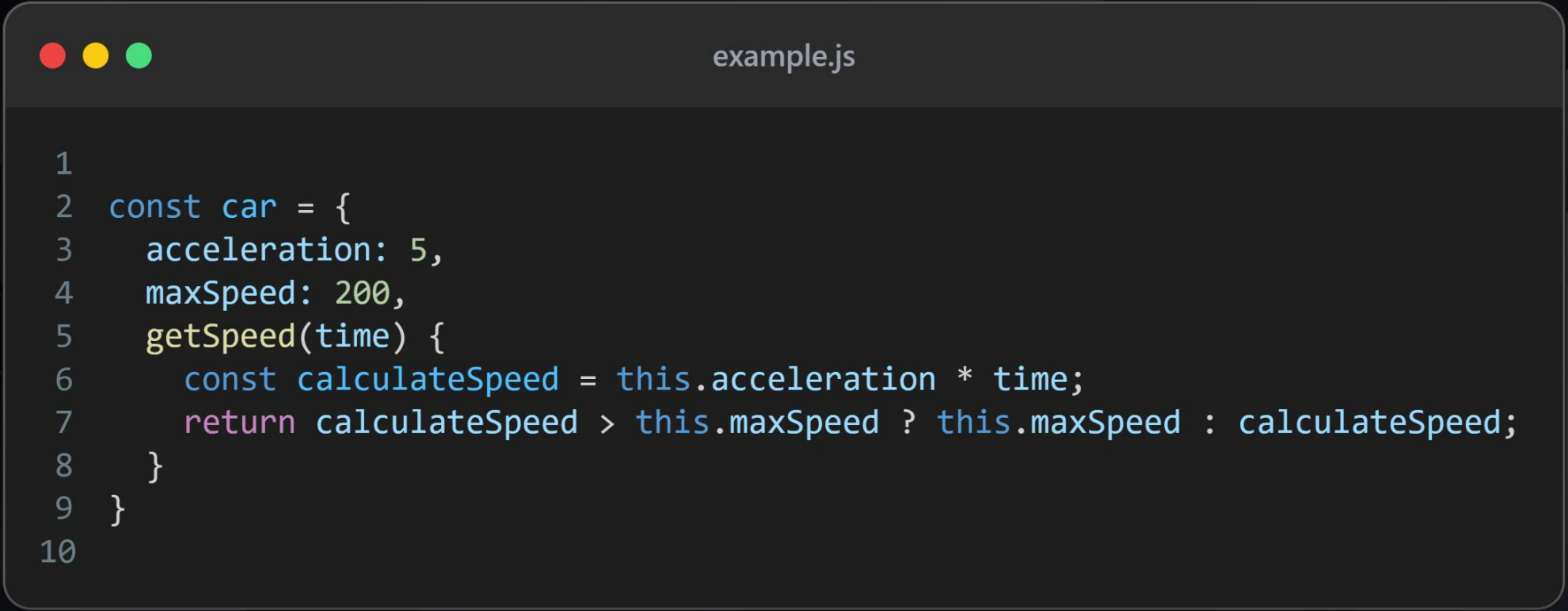
JavaScript

STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA

HOMEWORK

1. Make a car object

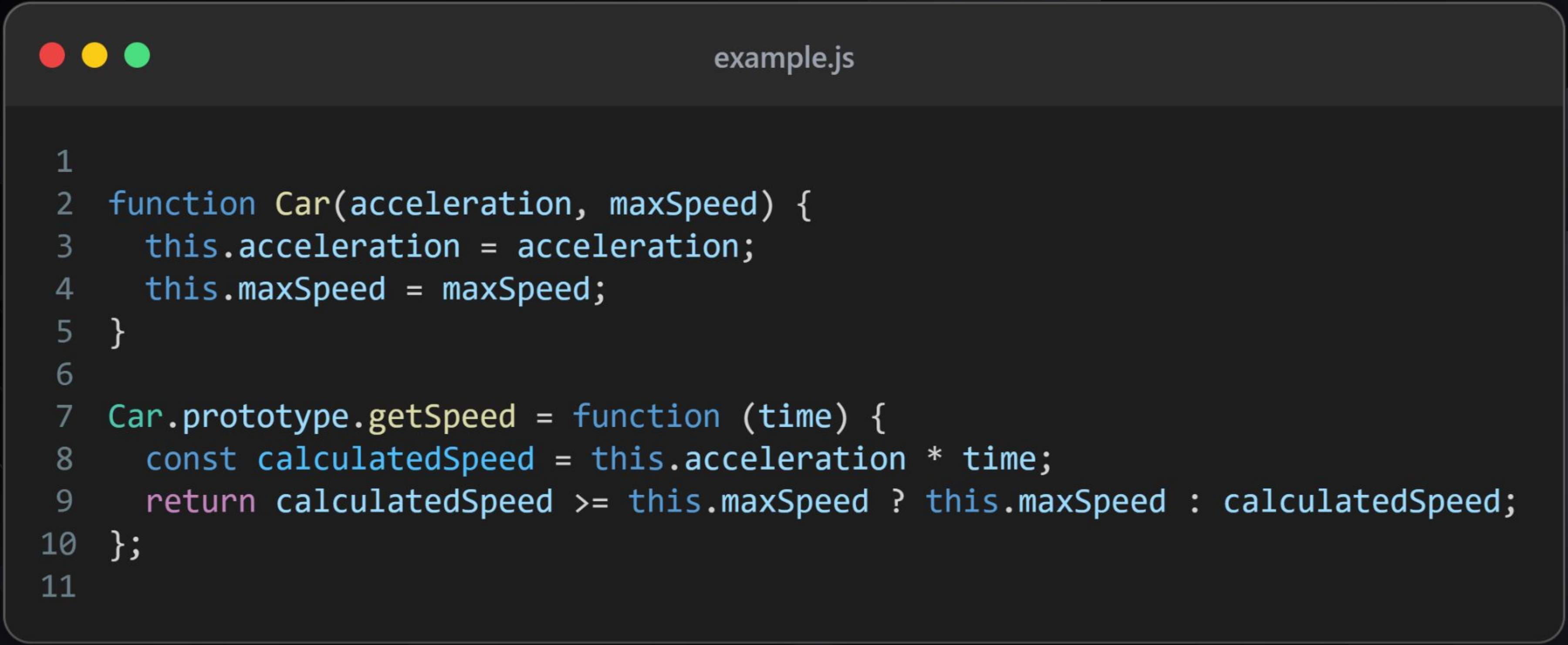
- Hardcode a car object
- The object should have properties: acceleration and maxSpeed
- The object should have a method: getSpeed that accepts time in seconds as parameter
- The method should return us speed that the car will have after the time



```javascript

const car = {
  acceleration: 5,
  maxSpeed: 200,
  getSpeed(time) {
    const calculateSpeed = this.acceleration * time;
    return calculateSpeed > this.maxSpeed ? this.maxSpeed : calculateSpeed;
  }
}

```

2. Make a car function

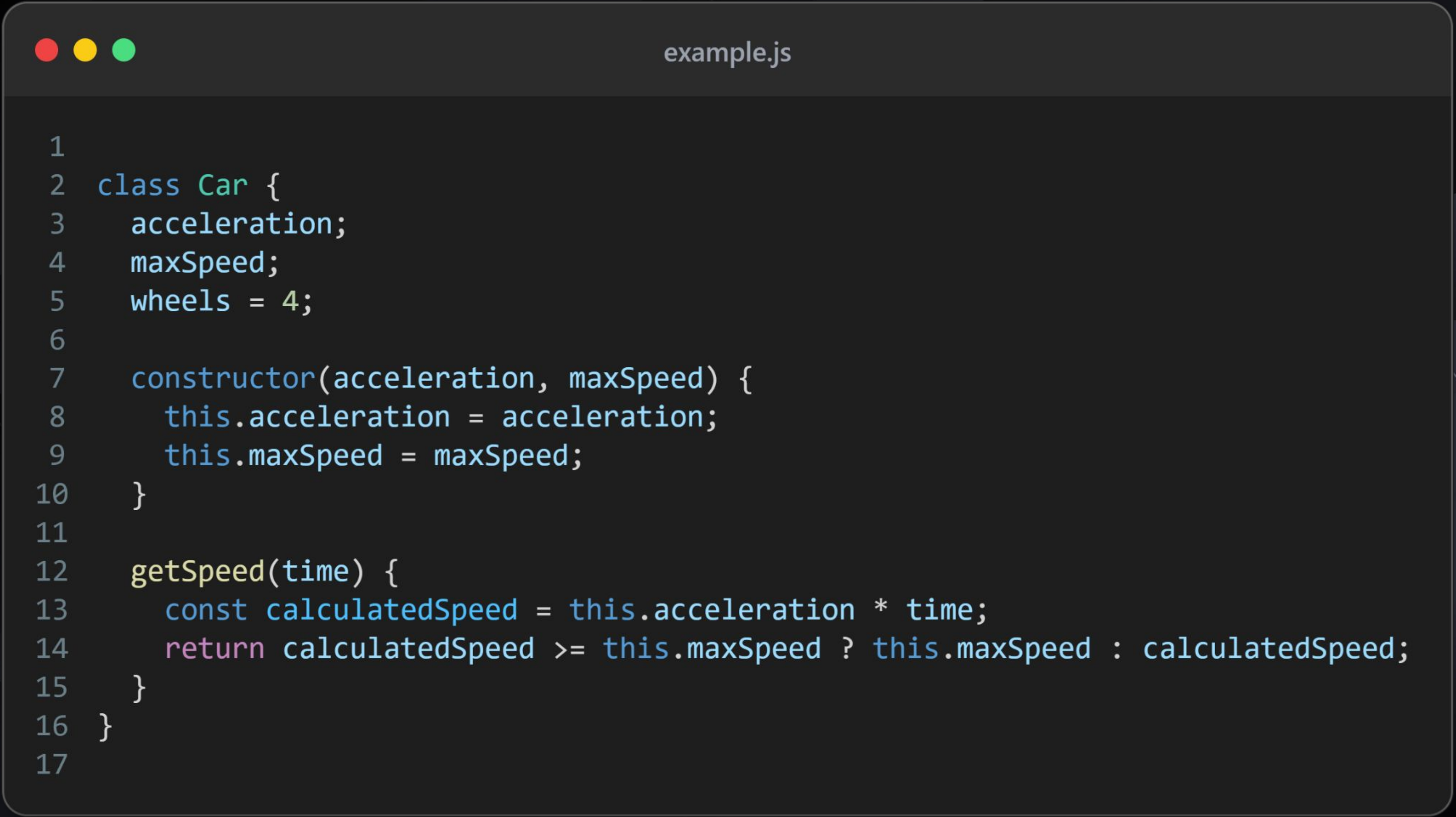
- Create function to produce car objects
- Created object should have properties: acceleration and maxSpeed
- The object should have a method: getSpeed that accepts time in seconds as parameter
- The method should be available to every created object


example.js

```javascript

function Car(acceleration, maxSpeed) {
  this.acceleration = acceleration;
  this.maxSpeed = maxSpeed;
}

Car.prototype.getSpeed = function (time) {
  const calculatedSpeed = this.acceleration * time;
  return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
};

```

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

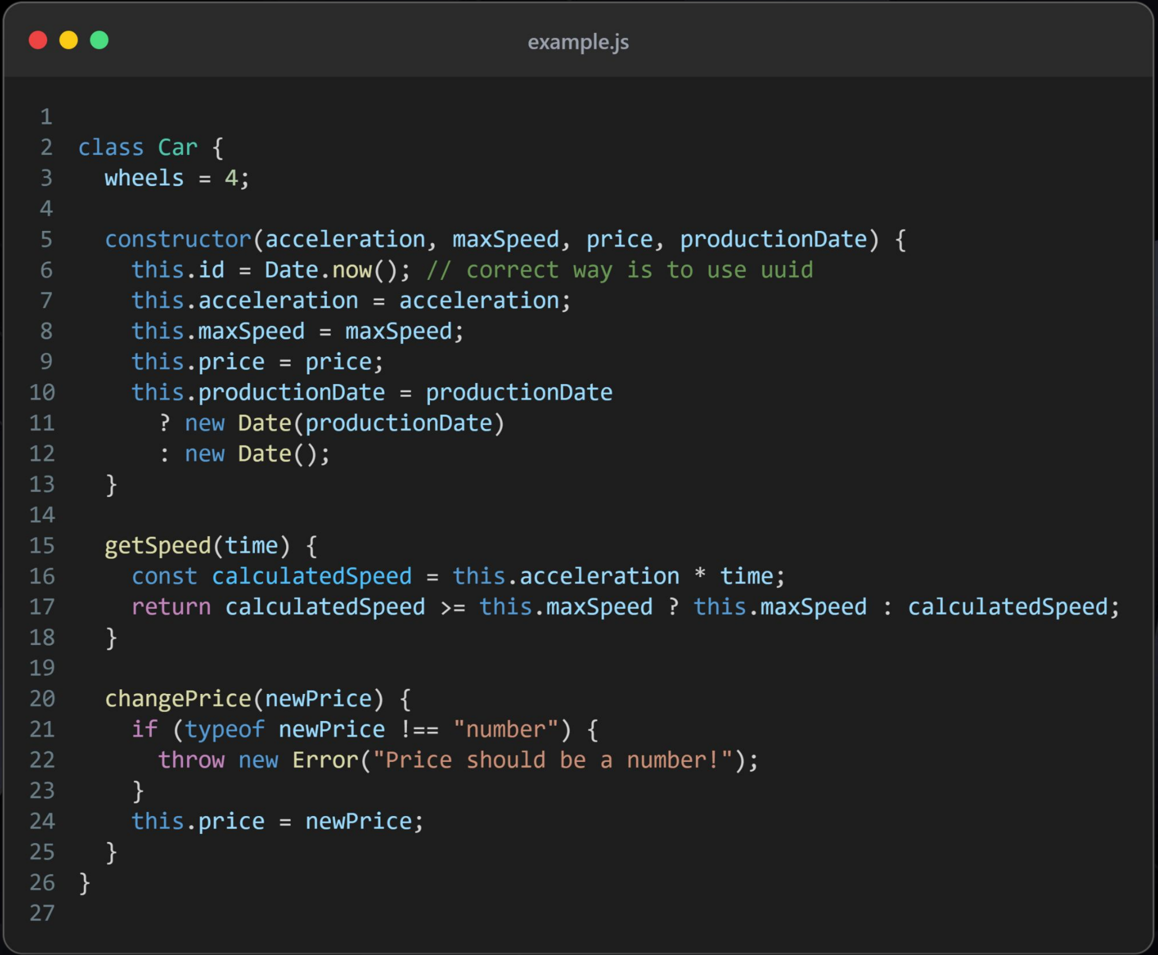


example.js

```javascript

class Car {
  acceleration;
  maxSpeed;
  wheels = 4;

  constructor(acceleration, maxSpeed) {
    this.acceleration = acceleration;
    this.maxSpeed = maxSpeed;
  }

  getSpeed(time) {
    const calculatedSpeed = this.acceleration * time;
    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
  }
}

```

4. Add features to the class

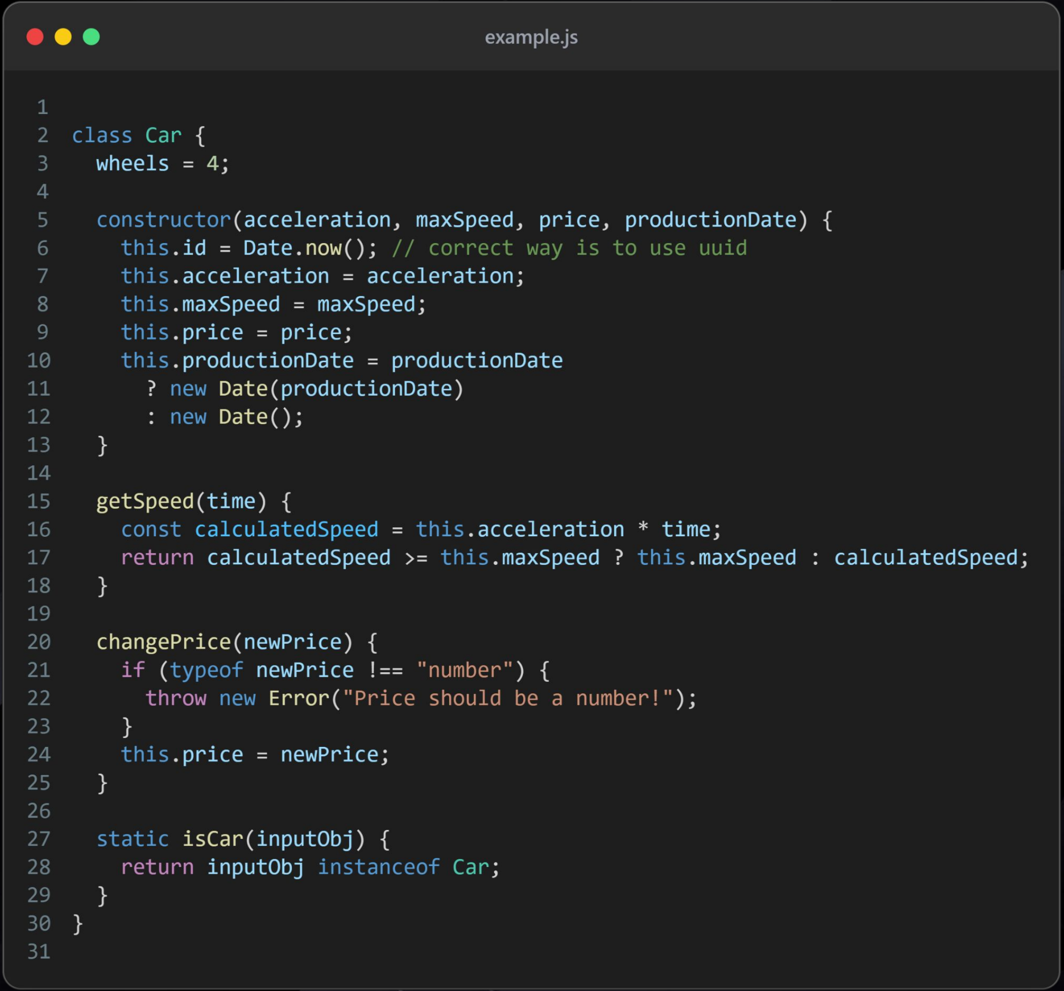
- Add new fields:
  - id
  - price
  - production date
  - and a method changePrice


example.js

```javascript

class Car {
  wheels = 4;

  constructor(acceleration, maxSpeed, price, productionDate) {
    this.id = Date.now(); // correct way is to use uuid
    this.acceleration = acceleration;
    this.maxSpeed = maxSpeed;
    this.price = price;
    this.productionDate = productionDate
      ? new Date(productionDate)
      : new Date();
  }

  getSpeed(time) {
    const calculatedSpeed = this.acceleration * time;
    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
  }

  changePrice(newPrice) {
    if (typeof newPrice !== "number") {
      throw new Error("Price should be a number!");
    }
    this.price = newPrice;
  }
}

```

5. Add static method

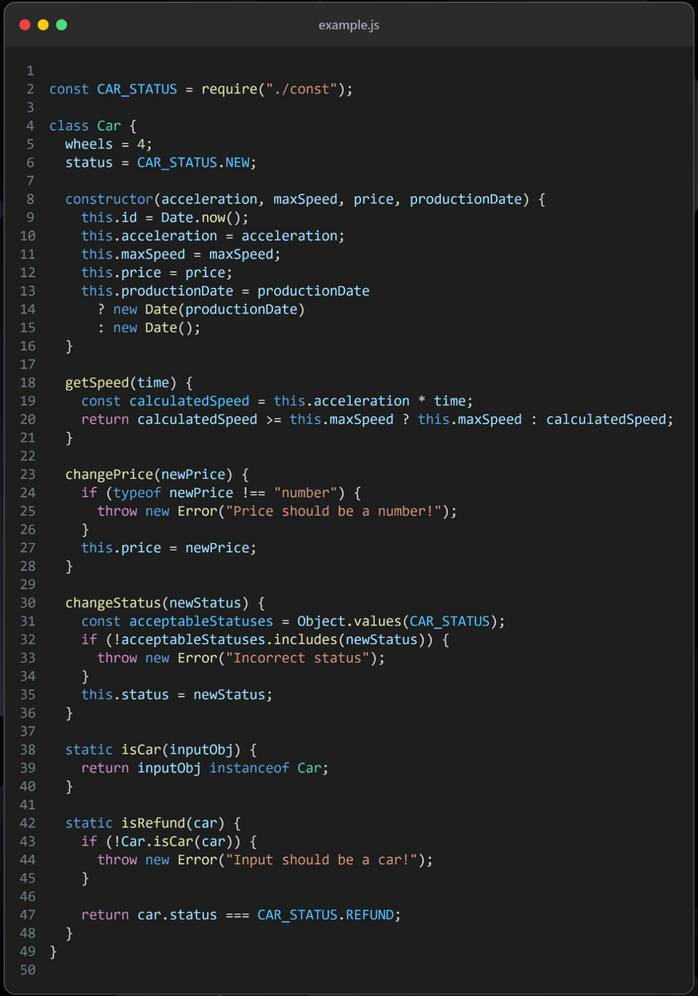
- Add static method that for a given object, will check if the object is a car


example.js

```
class Car {
  wheels = 4;

  constructor(acceleration, maxSpeed, price, productionDate) {
    this.id = Date.now(); // correct way is to use uuid
    this.acceleration = acceleration;
    this.maxSpeed = maxSpeed;
    this.price = price;
    this.productionDate = productionDate
      ? new Date(productionDate)
      : new Date();
  }

  getSpeed(time) {
    const calculatedSpeed = this.acceleration * time;
    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
  }

  changePrice(newPrice) {
    if (typeof newPrice !== "number") {
      throw new Error("Price should be a number!");
    }
    this.price = newPrice;
  }

  static isCar(inputObj) {
    return inputObj instanceof Car;
  }
}
```

6. Add another feature

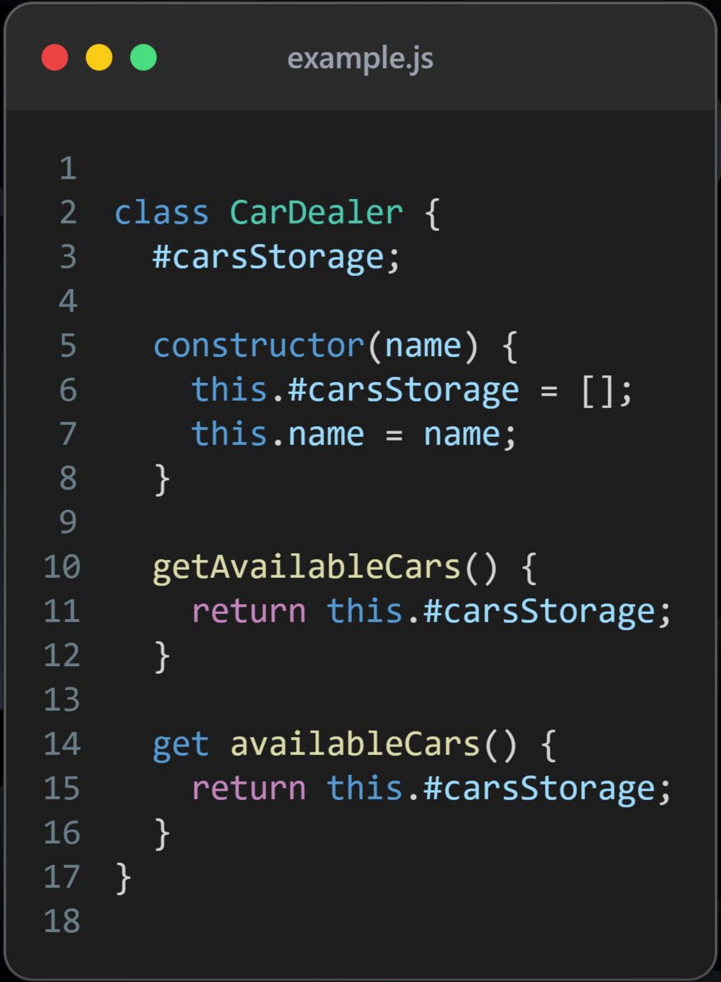
- Add new field: status
- Create Car status object with acceptable statuses (NEW, USED, REFUND)
- Add method changeStatus that accepts new status as a parameter
- Add static method to check if car is after return


example.js

```
const CAR_STATUS = require("./const");

class Car {
  wheels = 4;
  status = CAR_STATUS.NEW;

  constructor(acceleration, maxSpeed, price, productionDate) {
    this.id = Date.now();
    this.acceleration = acceleration;
    this.maxSpeed = maxSpeed;
    this.price = price;
    this.productionDate = productionDate
      ? new Date(productionDate)
      : new Date();
  }

  getSpeed(time) {
    const calculatedSpeed = this.acceleration * time;
    return calculatedSpeed >= this.maxSpeed ? this.maxSpeed : calculatedSpeed;
  }

  changePrice(newPrice) {
    if (typeof newPrice !== "number") {
      throw new Error("Price should be a number!");
    }
    this.price = newPrice;
  }

  changeStatus(newStatus) {
    const acceptableStatuses = Object.values(CAR_STATUS);
    if (!acceptableStatuses.includes(newStatus)) {
      throw new Error("Incorrect status");
    }
    this.status = newStatus;
  }

  static isCar(inputObj) {
    return inputObj instanceof Car;
  }

  static isRefund(car) {
    if (!Car.isCar(car)) {
      throw new Error("Input should be a car!");
    }

    return car.status === CAR_STATUS.REFUND;
  }
}
```

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

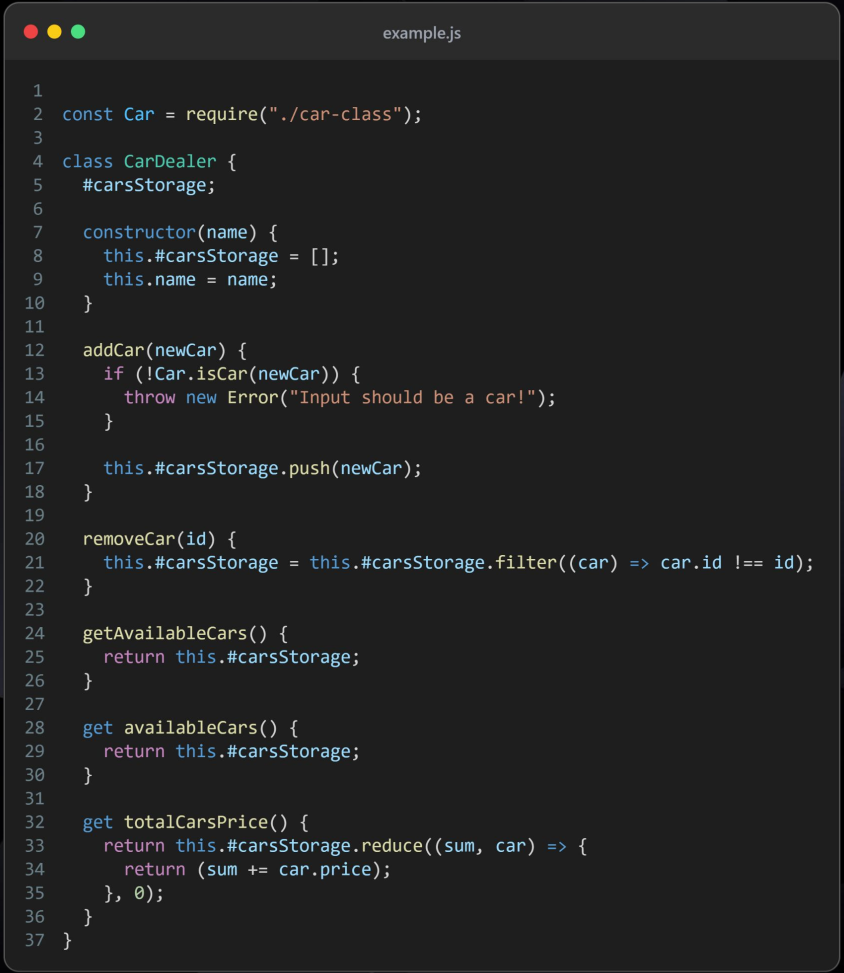


example.js

```

class CarDealer {
  #carsStorage;

  constructor(name) {
    this.#carsStorage = [];
    this.name = name;
  }

  getAvailableCars() {
    return this.#carsStorage;
  }

  get availableCars() {
    return this.#carsStorage;
  }
}

```

8. Car Dealer

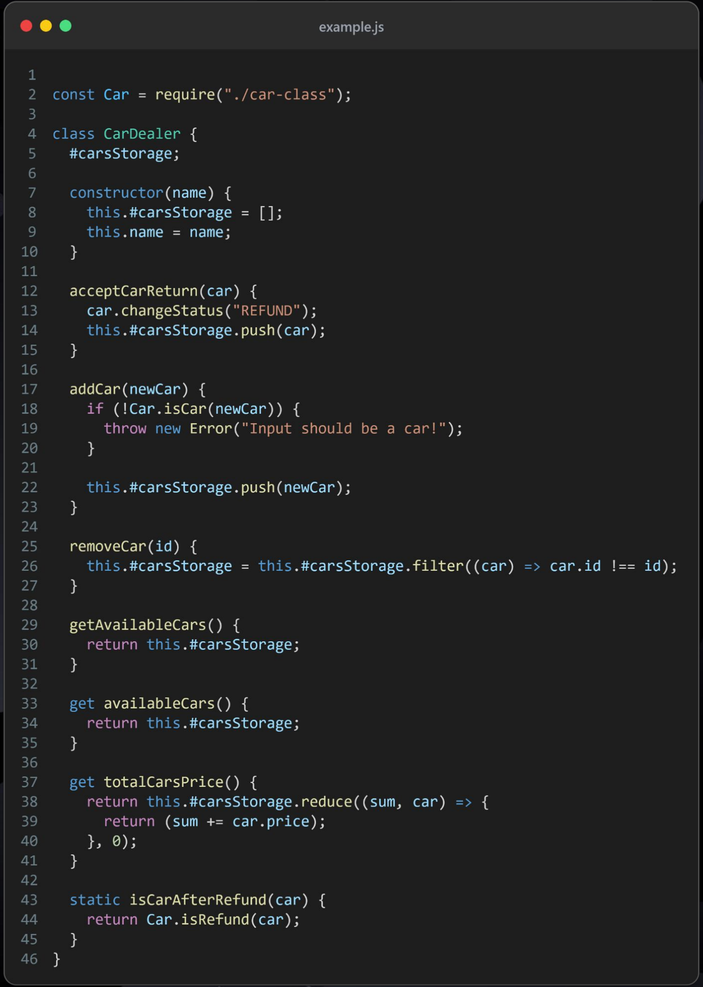
- Add new methods
- A method to add a car (that would check if the input object is a car)
- A method to remove a car by ID
- A getter that would return the total price of all cars in the storage


example.js

```javascript

const Car = require("./car-class");

class CarDealer {
  #carsStorage;

  constructor(name) {
    this.#carsStorage = [];
    this.name = name;
  }

  addCar(newCar) {
    if (!Car.isCar(newCar)) {
      throw new Error("Input should be a car!");
    }

    this.#carsStorage.push(newCar);
  }

  removeCar(id) {
    this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
  }

  getAvailableCars() {
    return this.#carsStorage;
  }

  get availableCars() {
    return this.#carsStorage;
  }

  get totalCarsPrice() {
    return this.#carsStorage.reduce((sum, car) => {
      return (sum += car.price);
    }, 0);
  }
}
```

9. Car Dealer

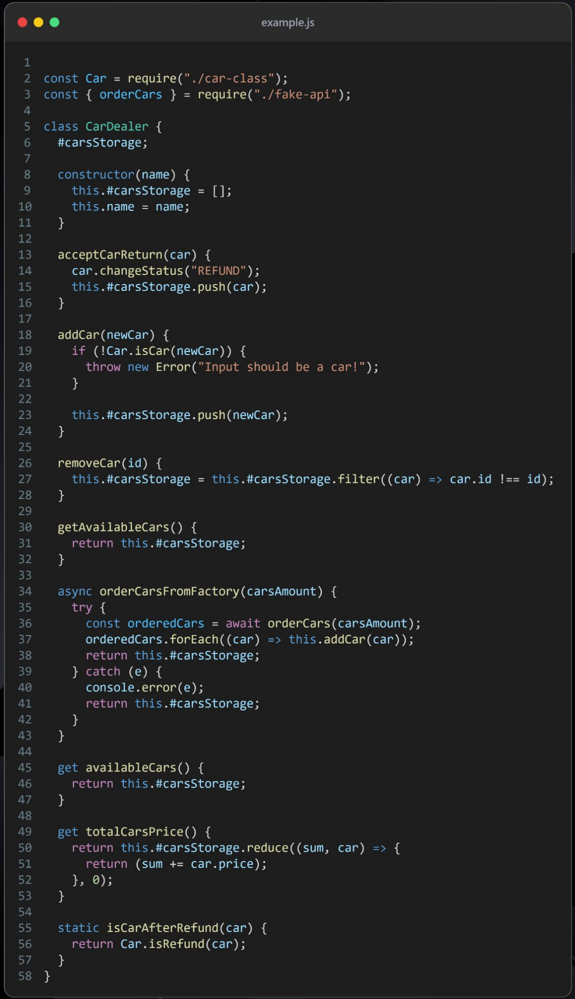
- Add new methods
- A method to accept car return, method should mark car as refund
- Add a static method, that would check if a given car is a refund car (it will have "REFUND" status)


example.js

```javascript

const Car = require("./car-class");

class CarDealer {
  #carsStorage;

  constructor(name) {
    this.#carsStorage = [];
    this.name = name;
  }

  acceptCarReturn(car) {
    car.changeStatus("REFUND");
    this.#carsStorage.push(car);
  }

  addCar(newCar) {
    if (!Car.isCar(newCar)) {
      throw new Error("Input should be a car!");
    }

    this.#carsStorage.push(newCar);
  }

  removeCar(id) {
    this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
  }

  getAvailableCars() {
    return this.#carsStorage;
  }

  get availableCars() {
    return this.#carsStorage;
  }

  get totalCarsPrice() {
    return this.#carsStorage.reduce((sum, car) => {
      return (sum += car.price);
    }, 0);
  }

  static isCarAfterRefund(car) {
    return Car.isRefund(car);
  }
}
```

10. Car Dealer

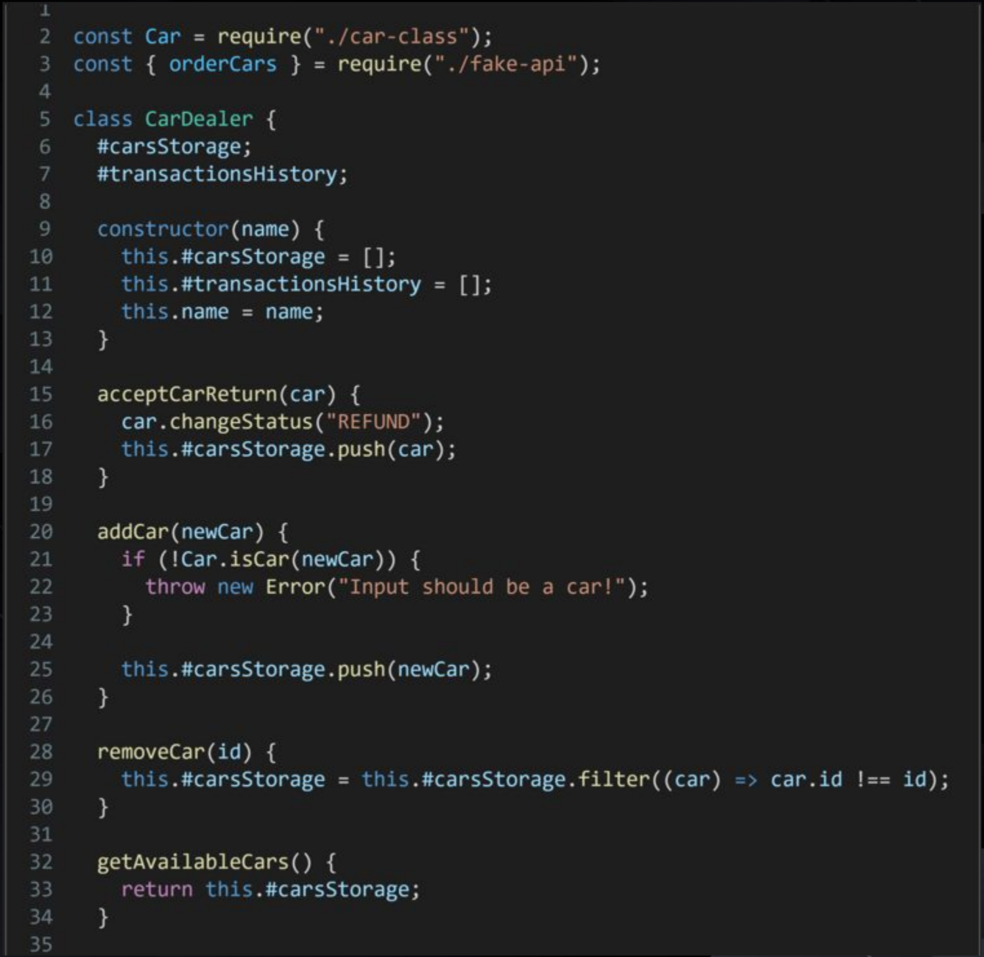
- Add new method orderFromFactory
- Method should be asynchronous
- Method should accept cars amount to order as a parameter
- Handle error in this method in case something goes wrong



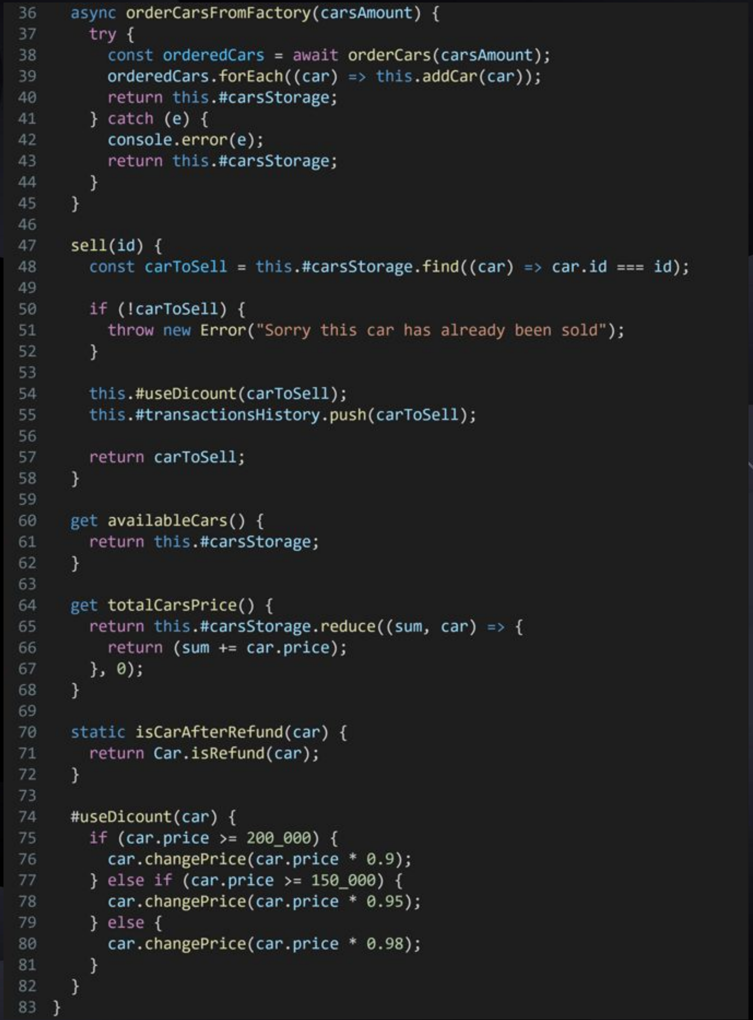
```javascript

const Car = require("./car-class");
const { orderCars } = require("./fake-api");

class CarDealer {
  #carsStorage;

  constructor(name) {
    this.#carsStorage = [];
    this.name = name;
  }

  acceptCarReturn(car) {
    car.changeStatus("REFUND");
    this.#carsStorage.push(car);
  }

  addCar(newCar) {
    if (!Car.isCar(newCar)) {
      throw new Error("Input should be a car!");
    }

    this.#carsStorage.push(newCar);
  }

  removeCar(id) {
    this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
  }

  getAvailableCars() {
    return this.#carsStorage;
  }

  async orderCarsFromFactory(carsAmount) {
    try {
      const orderedCars = await orderCars(carsAmount);
      orderedCars.forEach((car) => this.addCar(car));
      return this.#carsStorage;
    } catch (e) {
      console.error(e);
      return this.#carsStorage;
    }
  }

  get availableCars() {
    return this.#carsStorage;
  }

  get totalCarsPrice() {
    return this.#carsStorage.reduce((sum, car) => {
      return (sum += car.price);
    }, 0);
  }

  static isCarAfterRefund(car) {
    return Car.isRefund(car);
  }
}
```

11. Car Dealer

- Add new method sell
- Method should accept id as a parameter end 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



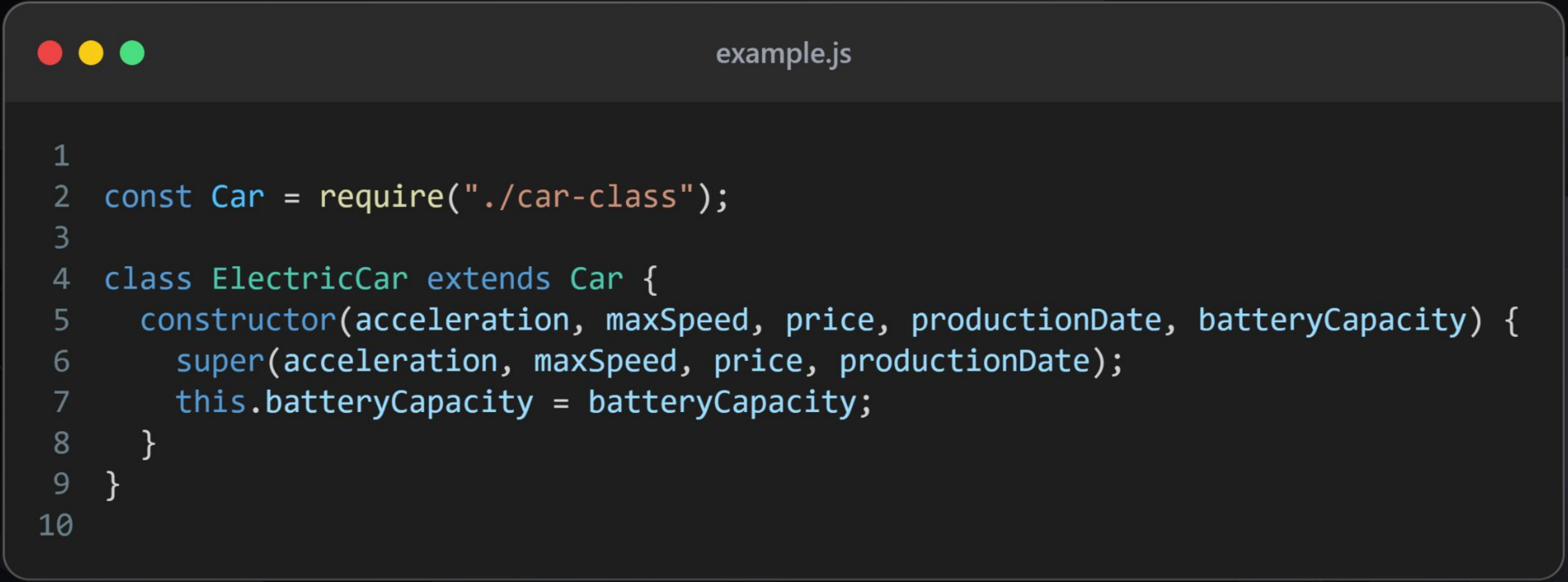
```javascript

const Car = require("./car-class");
const { orderCars } = require("./fake-api");

class CarDealer {
  #carsStorage;
  #transactionsHistory;

  constructor(name) {
    this.#carsStorage = [];
    this.#transactionsHistory = [];
    this.name = name;
  }

  acceptCarReturn(car) {
    car.changeStatus("REFUND");
    this.#carsStorage.push(car);
  }

  addCar(newCar) {
    if (!Car.isCar(newCar)) {
      throw new Error("Input should be a car!");
    }

    this.#carsStorage.push(newCar);
  }

  removeCar(id) {
    this.#carsStorage = this.#carsStorage.filter((car) => car.id !== id);
  }

  getAvailableCars() {
    return this.#carsStorage;
  }

```



```javascript
  async orderCarsFromFactory(carsAmount) {
    try {
      const orderedCars = await orderCars(carsAmount);
      orderedCars.forEach((car) => this.addCar(car));
      return this.#carsStorage;
    } catch (e) {
      console.error(e);
      return this.#carsStorage;
    }
  }

  sell(id) {
    const carToSell = this.#carsStorage.find((car) => car.id === id);

    if (!carToSell) {
      throw new Error("Sorry this car has already been sold");
    }

    this.#useDicount(carToSell);
    this.#transactionsHistory.push(carToSell);

    return carToSell;
  }

  get availableCars() {
    return this.#carsStorage;
  }

  get totalCarsPrice() {
    return this.#carsStorage.reduce((sum, car) => {
      return (sum += car.price);
    }, 0);
  }

  static isCarAfterRefund(car) {
    return Car.isRefund(car);
  }

  #useDicount(car) {
    if (car.price >= 200_000) {
      car.changePrice(car.price * 0.9);
    } else if (car.price >= 150_000) {
      car.changePrice(car.price * 0.95);
    } else {
      car.changePrice(car.price * 0.98);
    }
  }
}
```

Electric Car.

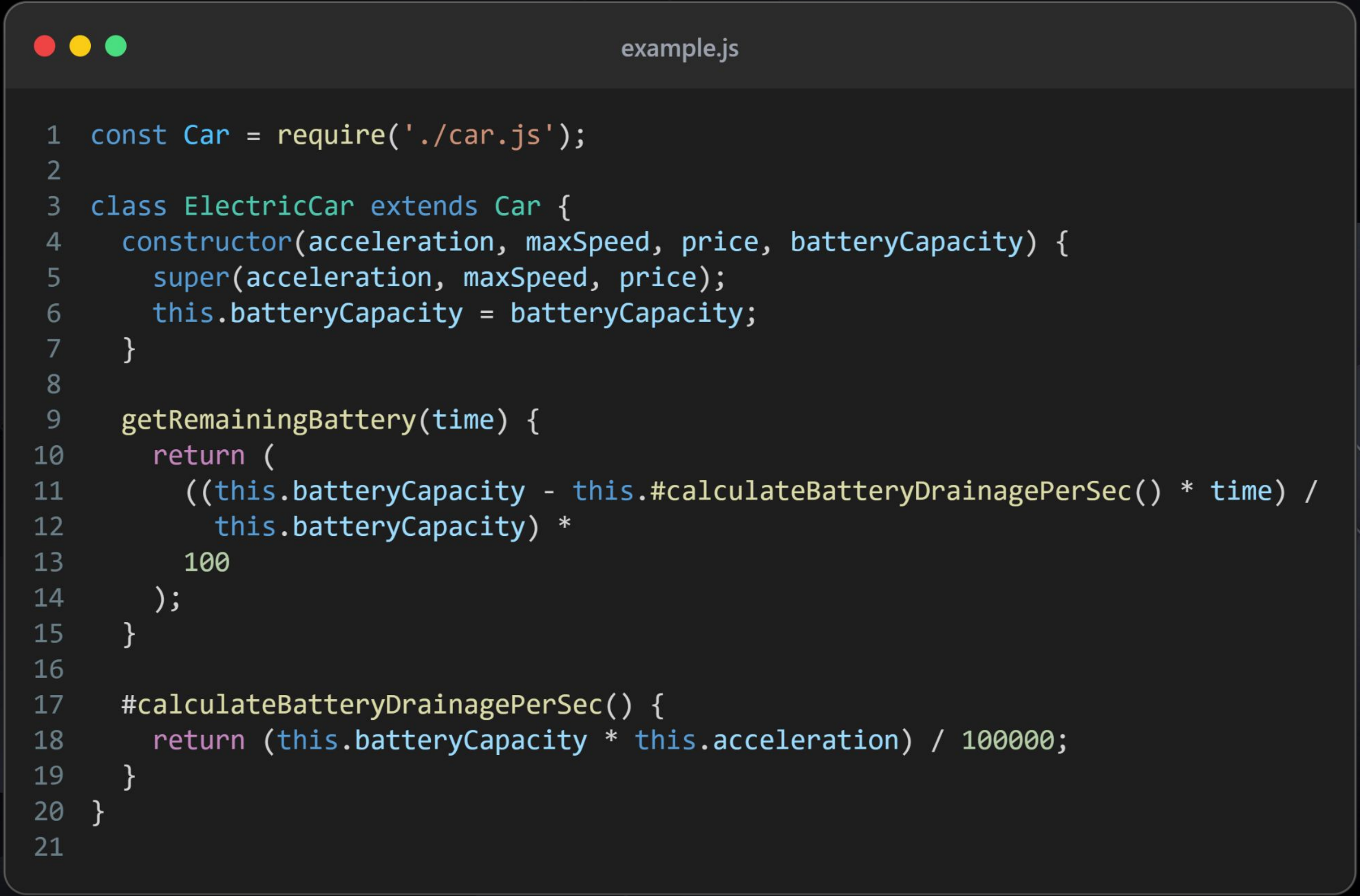
- Create a class - ElectricCar that will extend the Car class
- Add a new field: batteryCapacity


example.js

```javascript

const Car = require("./car-class");

class ElectricCar extends Car {
  constructor(acceleration, maxSpeed, price, productionDate, batteryCapacity) {
    super(acceleration, maxSpeed, price, productionDate);
    this.batteryCapacity = batteryCapacity;
  }
}

```

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)


example.js

```javascript
const Car = require('./car.js');

class ElectricCar extends Car {
  constructor(acceleration, maxSpeed, price, batteryCapacity) {
    super(acceleration, maxSpeed, price);
    this.batteryCapacity = batteryCapacity;
  }

  getRemainingBattery(time) {
    return (
      ((this.batteryCapacity - this.#calculateBatteryDrainagePerSec() * time) /
        this.batteryCapacity) *
      100
    );
  }

  #calculateBatteryDrainagePerSec() {
    return (this.batteryCapacity * this.acceleration) / 100000;
  }
}

```

HOMEWORK

- Josephus's Problem
- 1 EXAM tasks