Alex Samos Muñoz

1. Create as many classes as you need to store information about trucks, cars, motorcycles, vans and caravans. Information about manufacturer, model, year, type of engine (electric of combustion), source of power (battery, diesel, gasoline, gas…), power (measured in Kw), cubic centimeters, top speed and driver license needed should be stored. Those vehicles should be able to ignite and stop the engine, refill the reservoir and increase and decrease speed.

```
class Vehiculo{
    constructor(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir){
        this.manufactura = manufactura;
        this.modelo = modelo;
        this.year = year;
        this.tipoIngenieria = tipoIngenieria;
        this.poder = poder;
        this.cm3 = cm3;
        this.velocidadMax = velocidadMax;
        this.licenciaConducir = licenciaConducir;
        this.motor = false;
        this.velocidad = 0;
        this.combustible = 0;
    }

    encenderMotor(){
        if (this.motor) {
            return console.log("El motor ya esta encendido");
        }
        this.motor = true;
        return console.log("El motor esta encendido");
    }

    apagarMotor(){
        if (!this.motor) {
            return console.log("El motor ya esta apagado");
        }
        this.motor = false;
        return console.log("El motor esta apagado");
    }

    rellenarDeposito(litros){
        if (litros <= 0) {
            return console.log("No se puede rellenar el deposito con: "
+ litros + " litros");
        }
        this.combustible += litros;
```

```
        return console.log("El nuevo combustible del vehiculo es: " +
this.combustible + " litros");
    }

    aumentarVelocidad(nuevaVelocidad){
        if (nuevaVelocidad <= 0) {
            return console.log("No se puede aumentar la velocidad");
        }else if (nuevaVelocidad > this.velocidadMax) {
            return console.log("La velocidad nueva excede la velocidad
maxima del vehiculo");
        }
        this.velocidad = nuevaVelocidad;
        return console.log("La nueva velocidad del vehiculo es: " +
this.velocidad + " Km/h");
    }

    disminuirVelocidad(nuevaVelocidad){
        if (nuevaVelocidad < 0) {
            return console.log("No se puede disminuir esa velocidad");
        }else if (nuevaVelocidad > this.velocidad) {
            return console.log("No se puede disminuir esa velocidad");
        }
        this.velocidad -= nuevaVelocidad;
        return console.log("La nueva velocidad del vehiculo es: " +
this.velocidad + " Km/h");
    }
}

class Camion extends Vehiculo{
    constructor(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir){
        super(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir);
        this.tipoVehiculo = "camion";
    }

    claseVehiculo(){
        return console.log("El vehiculo es un: " + this.tipoVehiculo);
    }
}

class Coche extends Vehiculo{
```

```
    constructor(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir){
        super(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir);
        this.tipoVehiculo = "coche";
    }

    claseVehiculo(){
        return console.log("El vehiculo es un: " + this.tipoVehiculo);
    }
}

class Moto extends Vehiculo{
    constructor(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir){
        super(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir);
        this.tipoVehiculo = "moto";
    }

    claseVehiculo(){
        return console.log("El vehiculo es una: " + this.tipoVehiculo);
    }
}

class Furgoneta extends Vehiculo{
    constructor(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir){
        super(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir);
        this.tipoVehiculo = "furgoneta";
    }

    claseVehiculo(){
        return console.log("El vehiculo es una: " + this.tipoVehiculo);
    }
}

class Caravana extends Vehiculo{
    constructor(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir){
        super(manufactura, modelo, year, tipoIngenieria, poder, cm3,
velocidadMax, licenciaConducir);
```

```
        this.tipoVehiculo = "caravana";
    }

    claseVehiculo(){
        return console.log("El vehiculo es una: " + this.tipoVehiculo);
    }
}

let BMW = new Coche("maquinas", "X3", 2003, "combustion", 187.9, 4.7,
200, "B");

BMW.claseVehiculo();
BMW.encenderMotor();
BMW.rellenarDeposito(60);
BMW.aumentarVelocidad(80);
BMW.disminuirVelocidad(30);
BMW.apagarMotor();
```

2. Create the following classes to model some animal: mammal, reptiles and fish. They all perform the following actions: sleep, awake, move and stop and they have the following properties: group, name, age, number of paws or fins, way of moving, habitat and presence or absence of tail. Instantiate classes to model a shark, a cat and a snake.

```
class Animal {
    constructor(grupo, nombre, edad, formaDeMoverse, habitat) {
      this.grupo = grupo;
      this.nombre = nombre;
      this.edad = edad;
      this.formaDeMoverse = formaDeMoverse;
      this.habitat = habitat;
      this.tieneCola = tieneCola;
    }

    dormir() {
      console.log(`${this.nombre} está durmiendo.`);
    }

    despertar() {
      console.log(`${this.nombre} está despierto.`);
    }

    moverse() {
      console.log(`${this.nombre} se mueve ${this.formaDeMoverse}.`);
    }
```

```
    detenerse() {
      console.log(`${this.nombre} se detuvo.`);
    }
  }

  class Mamifero extends Animal {
    constructor(nombre, edad, formaDeMoverse, habitat, numeroDePatas,
tieneCola) {
      super('mamífero', nombre, edad, formaDeMoverse, habitat);
      this.numeroDePatas = numeroDePatas;
      this.tieneCola = tieneCola;
    }
  }

  class Reptil extends Animal {
    constructor(nombre, edad, formaDeMoverse, habitat, numeroDePatas,
tieneCola) {
      super('reptil', nombre, edad, formaDeMoverse, habitat);
      this.numeroDePatas = numeroDePatas;
      this.tieneCola = tieneCola;
    }
  }

  class Pez extends Animal {
    constructor(nombre, edad, formaDeMoverse, hábitat, numeroDeAletas,
tieneCola) {
      super('pez', nombre, edad, formaDeMoverse, hábitat);
      this.numeroDeAletas = numeroDeAletas;
      this.tieneCola = tieneCola;
    }
  }

  let tiburon = new Pez("tiburon", 50, "nadando", "oceano", 3, "si");
  let gato = new Mamifero("gato",3,"andando","casa",4,"si");
  let serpiente = new
Reptil("serpiente",7,"arrastrarse","campo",0,"si");


  tiburon.moverse();
  gato.dormir();
  serpiente.despertar();
```

Alex Samos Muñoz

3. Implement a puzzle game in which you have sqares with numbers on it and an empty space. Board must be randomly created with unordered numbers. The goal is to order them by moving sqares adjacent to the empty space. The following interaction is mandatory:

1. choose the number of rows and columns at start
2. choose the number of empty spaces at start
3. restart the game
4. move the adjacent number Information about number of movements performed and time spent solving it must be stored.

```
class tablero{
    constructor(filas, columnas){
        this.filas = filas;
        this.columnas = columnas;
        this.espacioBlanco;
        this.tablero = [];
        this.numeros = [];
        this.movimientos = 0;
        this.inicioJuego = null;
        this.finJuego = null;
    }


    generarTablero(){
        let contador = 0;
        for (let i = 1; i < this.filas * this.columnas; i++){
            this.numeros.push(i);
        }
        this.numeros.push(" ");
        this.numeros.sort(() => Math.random() - 0.5);

        for (let  i = 0; i < this.filas; i++) {
            this.tablero[i] = [];
            for (let j = 0; j < this.columnas; j++) {
                    this.tablero[i][j] = this.numeros[contador];
                    contador ++;
            }
        }
    }

    movimiento() {
            let numerosMovibles = this.movimientoValido();
```

```
            let listaNumeros = " ";
            let mover;
            let num;
            let encontrado = false;
            let espacioBlanco = this.espacioBlanco;
            let filaEspacioBlanco = espacioBlanco.fila;
            let columnaEspacioBlanco = espacioBlanco.columna;

            for(let i = 0; i< numerosMovibles.length; i++){
                listaNumeros += numerosMovibles[i] + " ";
            }

            console.log("Los numeros que se pueden mover son: " +
listaNumeros);

            while(!encontrado){
                num = parseInt(prompt("Que numero quiere mover: "));
                for(let i = 0; i< numerosMovibles.length; i++){
                    if(num == numerosMovibles[i]){
                        encontrado = true;
                        break;
                    }
            }
            }


            for (let i = 0; i < this.filas; i++) {
                for (let j = 0; j < this.columnas; j++) {
                    if (this.tablero[i][j] === num) {
                        mover = {fila: i, columna: j};
                    }
                }
            }


            let posNumero = this.tablero[mover.fila][mover.columna];

            this.tablero[mover.fila][mover.columna] = " ";
            this.tablero[filaEspacioBlanco][columnaEspacioBlanco] =
posNumero;

            this.movimientos++;
```

```
            this.espacioBlanco = {fila: mover.fila, columna:
mover.columna};
            this.verificarFinJuego();

    }


    encontrarEspacioBlanco() {
        for (let i = 0; i < this.filas; i++) {
            for (let j = 0; j < this.columnas; j++) {
                if (this.tablero[i][j] === " ") {
                    this.espacioBlanco = { fila: i, columna: j };
                    return this.espacioBlanco;
                }
            }
        }
    }

   movimientoValido() {
        let espacioBlanco = this.espacioBlanco;
        let movimientosPosibles = [];

            if((espacioBlanco.fila - 1) != -1){

movimientosPosibles.push(this.tablero[espacioBlanco.fila
-1][espacioBlanco.columna]);
            }
            if((espacioBlanco.fila + 1) != (this.filas)){

movimientosPosibles.push(this.tablero[espacioBlanco.fila
+1][espacioBlanco.columna]);
            }
            if((espacioBlanco.columna - 1) != -1){

movimientosPosibles.push(this.tablero[espacioBlanco.fila][espacioBlanco
.columna - 1]);
            }
            if((espacioBlanco.columna + 1) != (this.columnas)){

movimientosPosibles.push(this.tablero[espacioBlanco.fila][espacioBlanco
.columna + 1]);
            }
        return movimientosPosibles;
```

```
    }

        verificarFinJuego(){
            let contador = 1;
            for (let i = 0; i < this.filas; i++) {
                for (let j = 0; j < this.columnas; j++) {
                    if(contador != (this.filas * this.columnas)){
                        if(this.tablero[i][j] != contador){
                            return false;
                        }else{
                            contador++;
                        }
                    }
                }
            }

            this.finJuego = new Date();
            return true;
        }

    reiniciar(){
        this.generarTablero();
        this.desordenarTablero();
        console.log(this.tablero);
    }

    estadisticas(){
        let movimientos = this.movimientos;
        let tiempoJuego = ((this.finJuego - this.inicioJuego) / 1000) /
60;

        return console.log("Has tenido: " + movimientos + " movimientos
y has tardado en acabar " + tiempoJuego + " minutos");
    }

    estadoDelJuego(){
        let tablero = "";
        for (let i = 0; i < this.filas; i++) {
            for (let j = 0; j < this.columnas; j++) {
                tablero += "| " + this.tablero[i][j] + " | ";
            }
            console.log(tablero);
            tablero = "";
```

```
        }
    }
}


  let juego = new tablero(3, 3);

  juego.generarTablero();
  juego.encontrarEspacioBlanco();

  while(!juego.verificarFinJuego()){
    juego.estadoDelJuego();
    juego.movimiento();
  }

  juego.estadisticas();
```

4. Implement the 3-in-a-row game

```
class TresEnRaya {
    constructor() {
        this.tablero = [
            [' ', ' ', ' '],
            [' ', ' ', ' '],
            [' ', ' ', ' ']
        ];
        this.turnoActual = 'X';
        this.ganador = null;
        this.movimientos = 0;
    }

    imprimirTablero() {
        let juego = "";
        for(let i = 0; i< this.tablero.length; i++){
            for(let j = 0; j<this.tablero[i].length; j++){
                juego += "| " + this.tablero[i][j] + " |";
            }
            juego += "\n"
        }
        return juego;
    }
```

```
    hacerMovimiento() {
        let posicionesLibres = [];
        for(let i = 0; i< this.tablero.length; i++){
            for(let j = 0; j<this.tablero[i].length; j++){
                if(this.tablero[i][j] == ' ') {
                    posicionesLibres.push({ fila: i, columna: j });
                }
            }
        }
         let posicionesRestantes = "Posiciones libres: ";
        for (let posicion of posicionesLibres) {
            posicionesRestantes +=
`[${posicion.fila},${posicion.columna}] `;
        }
        console.log(posicionesRestantes);
        let fila = parseInt(prompt("Inserte la fila: "));
        let columna = parseInt(prompt("Inserte la columna: "));
        if (this.tablero[fila][columna] == ' ') {
            this.tablero[fila][columna] = this.turnoActual;
            this.movimientos++;
            this.verificarGanador(fila, columna);
            this.cambiarTurno();
            return true;
        }
        return false;
    }

    cambiarTurno() {
        this.turnoActual = this.turnoActual == 'X' ? 'O' : 'X';
    }

    verificarGanador(fila, columna) {
        if(this.movimientos >= 3){
            if (
                this.verificarFila(fila) ||
                this.verificarColumna(columna) ||
                this.verificarDiagonales() ||
                this.verificarAntiDiagonales()
            ) {
                this.ganador = this.turnoActual;
            } else if (this.movimientos == 9) {
                this.ganador = 'empate';
```

```
            }
        }
    }

    verificarFila(fila) {
        return this.tablero[fila][0] == this.turnoActual &&
               this.tablero[fila][1] == this.turnoActual &&
               this.tablero[fila][2] == this.turnoActual;
    }

    verificarColumna(columna) {
        return this.tablero[0][columna] == this.turnoActual &&
               this.tablero[1][columna] == this.turnoActual &&
               this.tablero[2][columna] == this.turnoActual;
    }

    verificarDiagonales() {
        return this.tablero[0][0] == this.turnoActual &&
               this.tablero[1][1] == this.turnoActual &&
               this.tablero[2][2] == this.turnoActual;
    }

    verificarAntiDiagonales() {
        return this.tablero[0][2] == this.turnoActual &&
               this.tablero[1][1] == this.turnoActual &&
               this.tablero[2][0] == this.turnoActual;
    }

    obtenerGanador() {
        return this.ganador;
    }

    finjuego(){
        if(this.ganador != null){
            return true;
        }
        return false;
    }
}


let EnRalla = new TresEnRaya();
```

```
while(!EnRalla.finjuego()){
    console.log(EnRalla.imprimirTablero());
    EnRalla.hacerMovimiento();
}

 console.log(EnRalla.imprimirTablero());
let ganador = EnRalla.obtenerGanador();
if (ganador === 'empate') {
    console.log('¡Es un empate!');
} else if (ganador) {
    console.log(`¡El jugador ${ganador} ha ganado!`);
} else {
    console.log('El juego continúa...');
}
```