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(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida) {
       this.fabricante = fabricante;
       this.modelo = modelo;
       this.año = año;
       this.tipoMotor = tipoMotor;
       this.fuenteEnergia = fuenteEnergia;
       this.potencia = potencia;
       this.centimetrosCubicos = centimetrosCubicos;
       this.velocidadMaxima = velocidadMaxima;
       this.licenciaRequerida = licenciaRequerida;
       this.estadoMotor = 'apagado';
       this.velocidad = 0;
       this.nivelDeposito = 100;
   }

   encender() {
       if (this.estadoMotor === 'apagado') {
           console.log(`El motor de ${this.fabricante} ${this.modelo}
esta ahora encendido.`);
           this.estadoMotor = 'encendido';
       } else {
           console.log(`El motor de ${this.fabricante} ${this.modelo}
ya esta encendido.`);
       }
   }

   apagarMotor() {
       if (this.estadoMotor === 'encendido') {
           console.log(`El motor de ${this.fabricante} ${this.modelo}
esta ahora apagado.`);
           this.estadoMotor = 'apagado';
       } else {
           console.log(`El motor de ${this.fabricante} ${this.modelo}
ya esta apagado.`);
       }
```

```
    }

    rellenarDeposito() {
        console.log(`Rellenando el deposito de ${this.fabricante}
${this.modelo}.`);
        this.nivelDeposito = 100;
    }

    aumentarVelocidad() {
        if (this.estadoMotor === 'encendido') {
            this.velocidad += 10;
            console.log(`La velocidad de ${this.fabricante}
${this.modelo} aumento a ${this.velocidad} km/h.`);
        } else {
            console.log(`No se puede aumentar la velocidad. El motor de
${this.fabricante} ${this.modelo} esta apagado.`);
        }
    }

    disminuirVelocidad() {
        if (this.estadoMotor === 'encendido' && this.velocidad > 0) {
            this.velocidad -= 10;
            console.log(`La velocidad de ${this.fabricante}
${this.modelo} disminuyo a ${this.velocidad} km/h.`);
        } else if (this.velocidad === 0) {
            console.log(`${this.fabricante} ${this.modelo} ya esta
detenido.`);
        } else {
            console.log(`No se puede disminuir la velocidad. El motor de
${this.fabricante} ${this.modelo} esta apagado.`);
        }
    }
}

class Camion extends Vehiculo {
    constructor(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida,
capacidadCarga) {
        super(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida);
        this.capacidadCarga = capacidadCarga;
    }
}
```

```
class Coche extends Vehiculo {
   constructor(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida,
pasajeros) {
       super(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida);
       this.pasajeros = pasajeros;
   }
}

class Motocicleta extends Vehiculo {
   constructor(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida,
tieneCasco) {
       super(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida);
       this.tieneCasco = tieneCasco;
   }
}

class Furgoneta extends Vehiculo {
   constructor(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida,
volumenCarga) {
       super(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida);
       this.volumenCarga = volumenCarga;
   }
}

class Caravana extends Vehiculo {
   constructor(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida,
camas) {
       super(fabricante, modelo, año, tipoMotor, fuenteEnergia,
potencia, centimetrosCubicos, velocidadMaxima, licenciaRequerida);
       this.camas = camas;
   }
}

const miCoche = new Coche('Toyota', 'Camry', 2022, 'Combustion',
'Gasolina', 150, 2000, 180, 'Clase C', 5);
```

```
miCoche.encender();
miCoche.aumentarVelocidad();
miCoche.disminuirVelocidad();
miCoche.apagarMotor();

const miMoto = new Motocicleta('Harley-Davidson', 'Sportster', 2021,
'Combustion', 'Gasolina', 60, 1200, 160, 'Clase M', true);
miMoto.encender();
miMoto.aumentarVelocidad();
miMoto.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, tieneCola)
{
        this.grupo = grupo;
        this.nombre = nombre;
        this.edad = edad;
        this.formaDeMoverse = formaDeMoverse;
        this.habitat = habitat;
        this.tieneCola = tieneCola;
        this.estáDespierto = false;
   }

   dormir() {
        if (this.estáDespierto) {
            console.log(`${this.nombre} ahora está durmiendo.`);
            this.estáDespierto = false;
        } else {
            console.log(`${this.nombre} ya está dormido.`);
        }
   }

   despertar() {
        if (!this.estáDespierto) {
            console.log(`${this.nombre} ahora está despierto.`);
            this.estáDespierto = true;
        } else {
```

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

    moverse() {
        if (this.estáDespierto) {
            console.log(`${this.nombre} ahora se está moviendo.`);
        } else {
            console.log(`${this.nombre} no puede moverse mientras
duerme.`);
        }
    }

    detener() {
        console.log(`${this.nombre} se ha detenido.`);
    }
}

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

class Reptil extends Animal {
    constructor(nombre, edad, formaDeMoverse, habitat, tieneCola) {
        super('Reptil', nombre, edad, formaDeMoverse, habitat,
tieneCola);
    }
}

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

// Ejemplo de uso
```

```
const miGato = new Mamífero('Peludo', 3, 'Caminando', 'Doméstico',
true, 4);
miGato.despertar();
miGato.moverse();
miGato.detener();
miGato.dormir();

const miSerpiente = new Reptil('Deslizante', 5, 'Arrastrándose',
'Selva', true);
miSerpiente.despertar();
miSerpiente.moverse();
miSerpiente.detener();
miSerpiente.dormir();

const miTiburón = new Pez('Aletafilosa', 8, 'Nadando', 'Océano', true,
2);
miTiburón.despertar();
miTiburón.moverse();
miTiburón.detener();
miTiburón.dormir();
```

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

```
//No se porque ni encuentro el motivo, pero en algunas ocasiones
//El juego se come un numero del puzle y no lo muestra en el tablero.
class Puzle {
   constructor (filas, columnas, huecos) {
       this.filas = filas;
       this.columnas = columnas;
       this.huecos = huecos;
       this.tablero = this.tablero();
```

```
        this.movimientos = 0;
        this.tiempo = null;
    }

    // Método para generar el tablero del puzle utilizando el numero de
filas y columnas y huecos
    // pasados por el usuario
    // Devuelve un array bidimensional con el tablero generado
    tablero () {
        let numbers = Array.from(
            {length: this.filas * this.columnas - this.huecos},
            (_, i) => i +1); //función de mapeo que solo usa el segundo
índice
        let emptyTile = 0;
        let board = [];

        for (let i = 0; i < this.filas; i++) {
            let fila = [];
            for (let j = 0; j < this.columnas; j++) {
                let indice = Math.floor(Math.random() * numbers.length);
                fila.push(numbers.splice(indice, 1)[0]);
            }
            board.push(fila);
        }

        for (let i = 0; i < this.huecos; i++) {
            let filaVacia = Math.floor(Math.random() * this.filas);
            let columnaVacia = Math.floor(Math.random() *
this.columnas);
            board[filaVacia][columnaVacia] = emptyTile;

        }

        return board;
    }

    //Método el cual se encarga de dibujar el tablero en la consola
    dibujaTablero () {
        console.clear();
        console.log("Movimientos: " + this.movimientos);
        console.log("Tiempo: " + this.calcularTiempo() + " segundos");

        for (let i = 0; i < this.filas; i++) {
```

```
            console.log(this.tablero[i].join('\t'));
        }
    }

    //Método que se encarga de calcular el tiempo que ha tardado el
usuario en resolver el puzle
    calcularTiempo () {
        if (!this.tiempo) return 0;
        let tiempoActual = new Date();
        return Math.floor((tiempoActual - this.tiempo) / 1000);
    }

    //Método que se encarga de mover las fichas del puzle y
    //comprobar si el usuario ha resuelto el puzle con su movimiento.
    mover (fila, columna) {
        let filaVacia = this.encuentraCelda().fila;
        let columnaVacia = this.encuentraCelda().columna;

        if (this.esValido(fila, columna, filaVacia, columnaVacia)) {
            this.tablero[filaVacia][columnaVacia] =
this.tablero[fila][columna];
            this.tablero[fila][columna] = 0;
            this.movimientos++;
            this.dibujaTablero();

            if (this.solucion()) {
                console.log("Felicidades! has resuelto el puzle");
                console.log("Movimientos totales: " + this.movimientos);
                console.log("Tiempo total: " + this.calcularTiempo() + "
segundos");
                this.restart();
            }
        } else {
            console.log("Movimiento no valido, Intentalo de nuevo");
        }
    }

    //Método que se encarga de comprobar si el movimiento que ha
realizado el usuario es valido
    esValido (fila, columna, filaVacia, columnaVacia) {
        return (
            (fila === filaVacia && Math.abs(columna - columnaVacia) ===
1) ||
```

```
            (columna === columnaVacia && Math.abs(fila - filaVacia) ===
1)
        );
    }
    //Método que se encarga de encontrar la celda vacia en el tablero
    encuentraCelda () {
        for (let i = 0; i < this.filas; i++) {
            for (let j = 0; j < this.columnas; j++) {
                if (this.tablero[i][j] === 0) {
                    return {fila: i,  columna: j};
                }
            }
        }
    }

    //Método que se encarga de comprobar si el usuario ha resuelto el
puzle
    solucion () {
        let flatBoard = this.tablero.flat();
        for (let i = 0; i < flatBoard.length; i++) {
            if (flatBoard[i] !== i + 1){
                return false;
            }
        }
        return true;
    }

    //Método que se encarga de iniciar el puzle
    start () {
        this.tiempo = new Date();
        this.dibujaTablero();

        while (!this.solucion()) {
            let fila = parseInt(prompt("Introduce el numero de la fila a
la que moverte: ")) - 1;
            let columna = parseInt(prompt("Introduce el numero de la
columna a la que moverte: ")) - 1;

            if (fila < 0 || fila >= this.filas || columna < 0 || columna
>= this.columnas) {
                console.log("Valor no valido, por favor introduzca una
posicion valida");
                continue;
```

```
            }

            this.mover(fila,columna);
        }
    }

    //Método que se encarga de reiniciar el puzle
    restart () {
        let nuevaFila = parseInt(prompt("Introduce el numero de filas
para el tablero:"));
        let nuevaColumna = parseInt(prompt("Introduzca el numero de
columnas del tablero:"));
        let nuevosHuecos = parseInt(prompt("Introduzca el nuevo numero
de huecos:"));
    }
}

let fila = parseInt(prompt("Introduce el numero de filas para el
tablero:"));
let columna = parseInt(prompt("Introduzca el numero de columnas del
tablero:"));
let huecos = parseInt(prompt("Introduzca el nuevo numero de huecos:"));

let puzle = new Puzle(fila, columna, huecos);
puzle.start();
```

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

```
class TresEnRaya {
    constructor() {
        this.tablero = [
        ['', '', ''],
        ['', '', ''],
        ['', '', '']
        ];
        this.jugador = 'X';
        this.ganador = null;
    }
    //Método que se encarga de dibujar el tablero en la consola
    tablero() {
        console.log('Tablero:');
        for (let i = 0; i < 3; i++) {
        console.log(this.tablero[i].join(' | '));
        if (i < 2) console.log('---------');
```

```
        }
        console.log('\n');
    }
    //Método que se encarga de poner x o o en el tablero
    //ademas de llamar a los metodos ganador y cambio
    movimiento() {
        let row, col;
        do {
        row = parseInt(prompt(`Player ${this.jugador}, enter row (0, 1,
or 2):`));
        col = parseInt(prompt(`Player ${this.jugador}, enter column (0,
1, or 2):`));
        } while (!this.esValido(row, col));

        this.tablero[row][col] = this.jugador;
        this.ganador();
        this.cambio();
        this.tablero();
    }

    //Metodo que se encarga de comprobar si se puede colocar la ficha en
la posicion indicada
    esValido(row, col) {
        return row >= 0 && row < 3 && col >= 0 && col < 3 &&
this.tablero[row][col] === '';
    }

    //Metodo que se encarga de cambiar de jugador
    cambio() {
        this.jugador = this.jugador === 'X' ? 'O' : 'X';
    }

    //Metodo que se encarga de comprobar si hay un ganador
    ganador() {
        for (let i = 0; i < 3; i++) {
            if (
                this.tablero[i][0] !== '' &&
                this.tablero[i][0] === this.tablero[i][1] &&
                this.tablero[i][1] === this.tablero[i][2]
            ) {
                this.ganador = this.tablero[i][0];
            }
            if (
```

```
                this.tablero[0][i] !== '' &&
                this.tablero[0][i] === this.tablero[1][i] &&
                this.tablero[1][i] === this.tablero[2][i]
            ) {
                this.ganador = this.tablero[0][i];
            }
            }
            if (
            this.tablero[0][0] !== '' &&
            this.tablero[0][0] === this.tablero[1][1] &&
            this.tablero[1][1] === this.tablero[2][2]
            ) {
                this.ganador = this.tablero[0][0];
            }
            if (
            this.tablero[0][2] !== '' &&
            this.tablero[0][2] === this.tablero[1][1] &&
            this.tablero[1][1] === this.tablero[2][0]
            ) {
                this.ganador = this.tablero[0][2];
            }
        if (this.ganador) {
        console.log(`EL jugador de la ${this.ganador} gano`);
        }
    }
}
 const partida = new TresEnRaya();
partida.tablero();
 while (!partida.winner) {
   partida.movimiento();
}
```