* **課前問卷**

<https://forms.gle/6PaM3sq6XgeEuqMo9>

* **講義下載**

<https://reurl.cc/gZbpLR>

* **CDX-VM登記**

<https://forms.gle/UWifQ7KpkuWv3kEr5>

* **CDX-VM分配**

<https://reurl.cc/3OApvX>

* **CDX平台**(首次登入記得修改密碼)

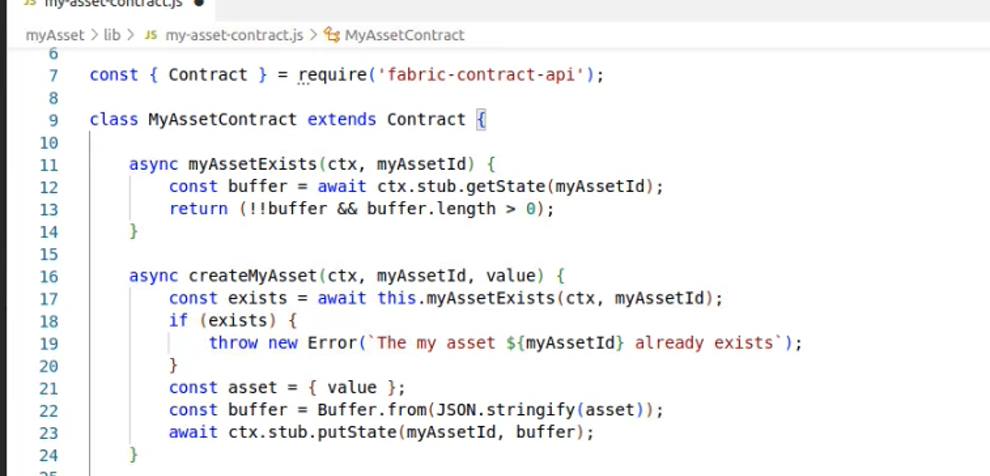
<https://cdx.nchc.org.tw/>

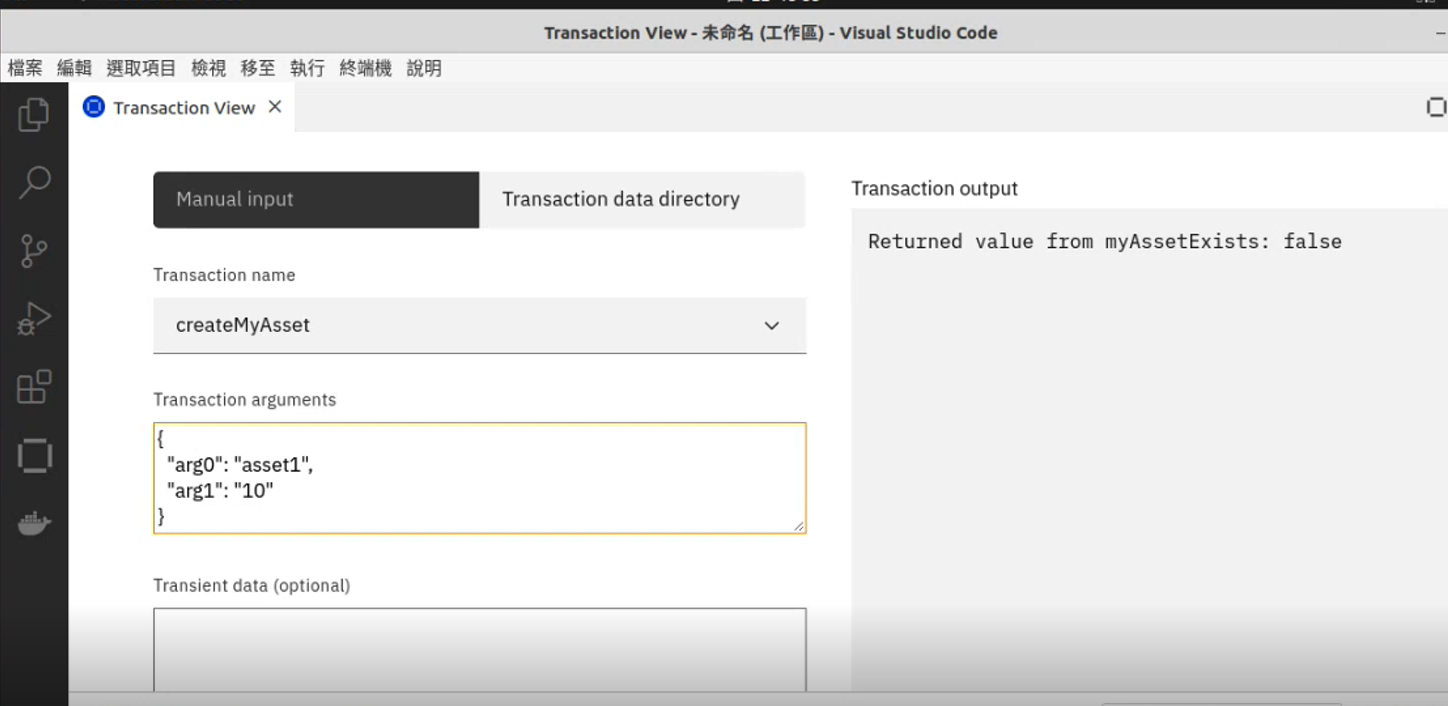
* **範例程式**

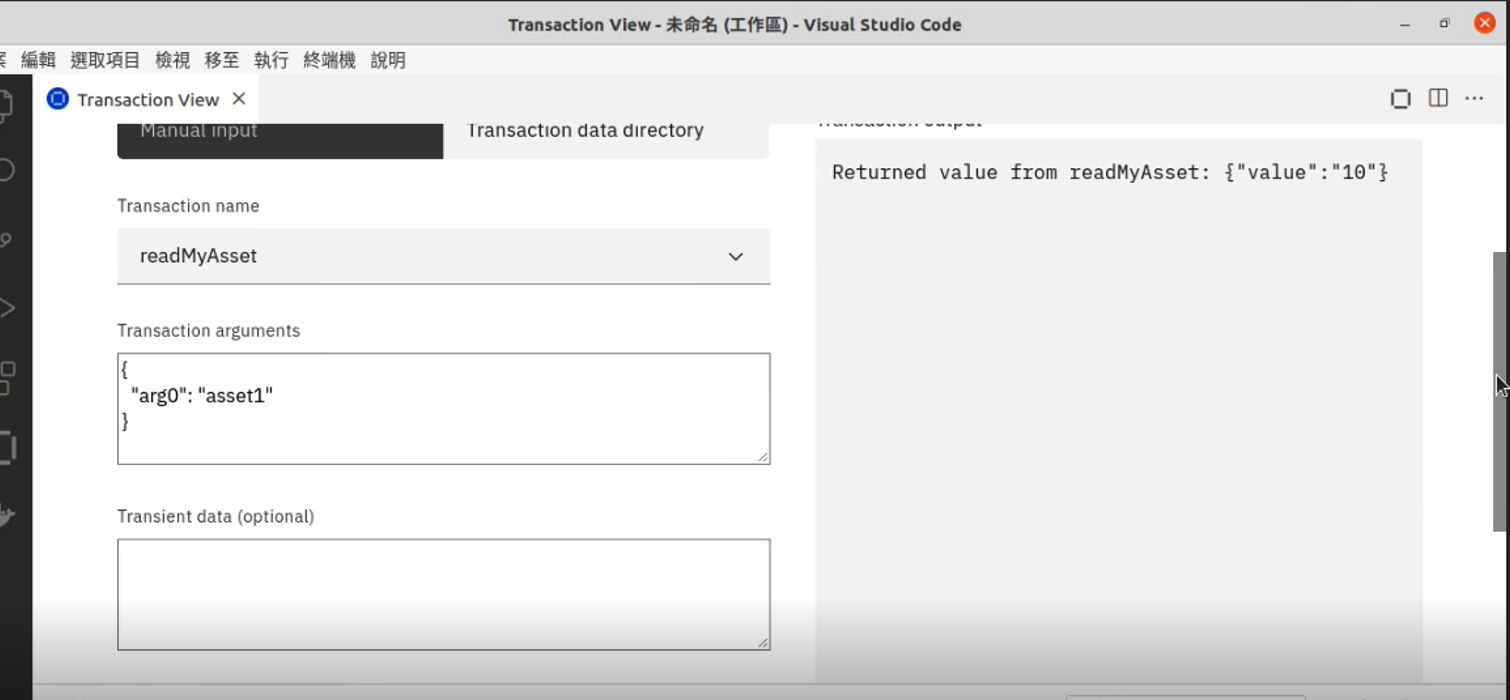
<https://reurl.cc/8qA8vo>

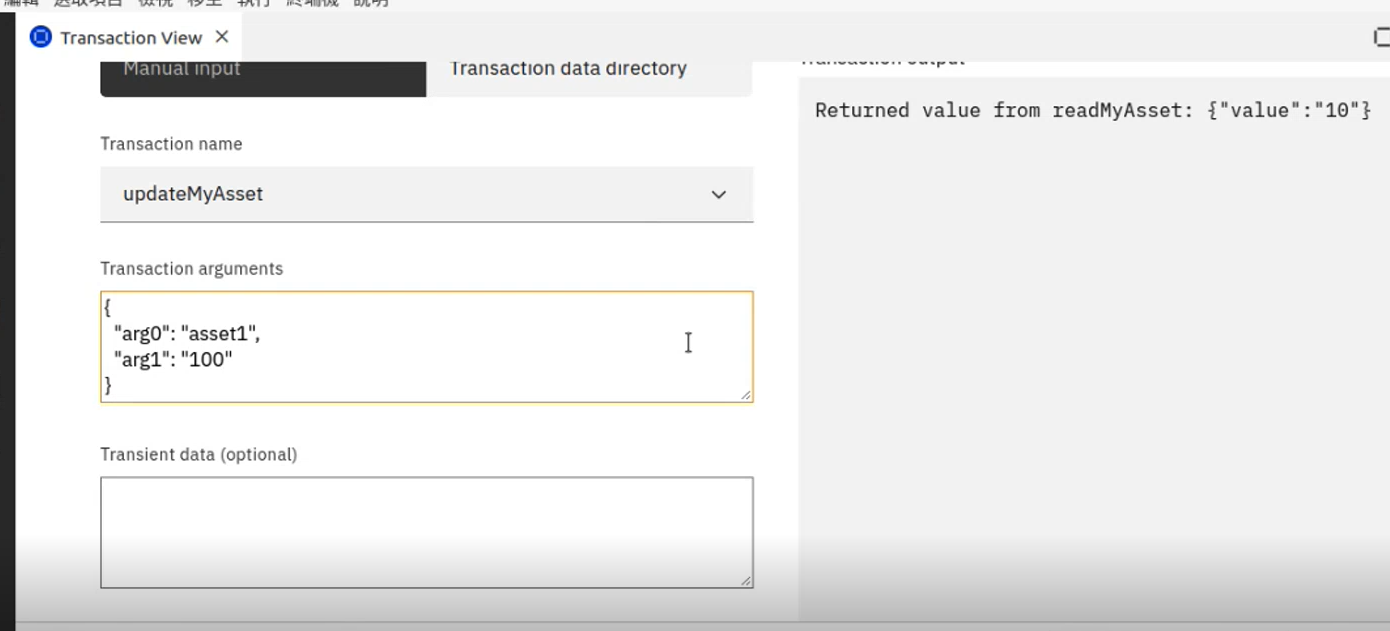
* **課後問卷**

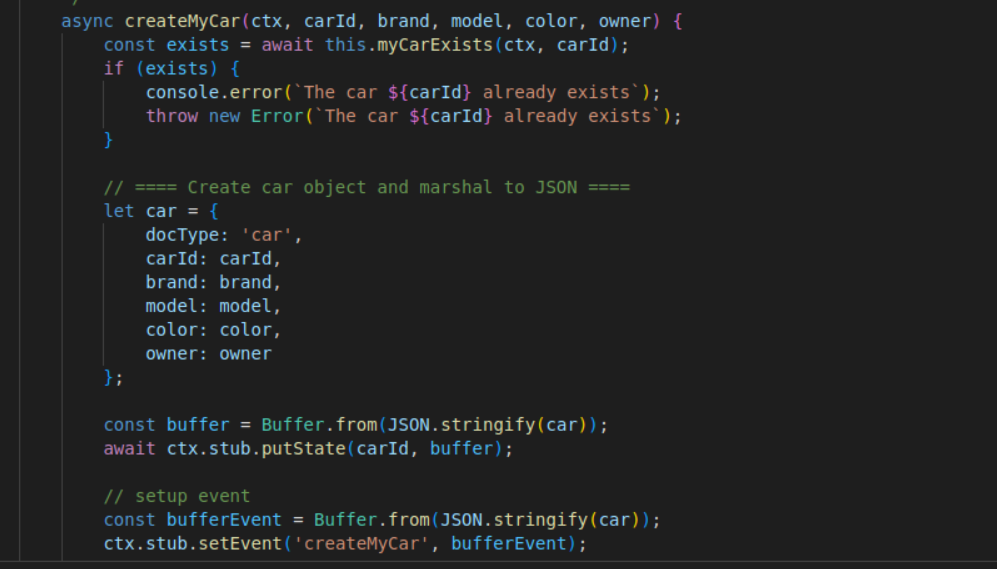
<https://forms.gle/UeEb9hPAUagSzma6A>

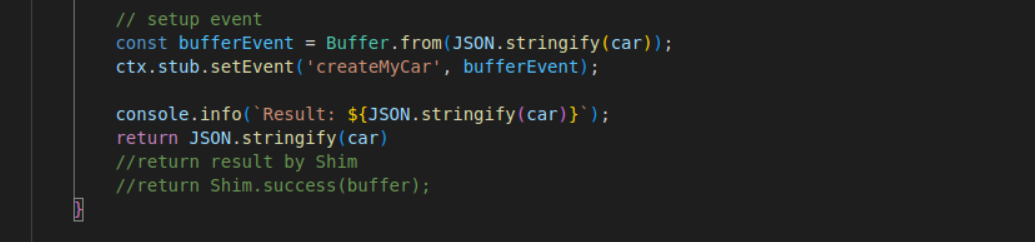


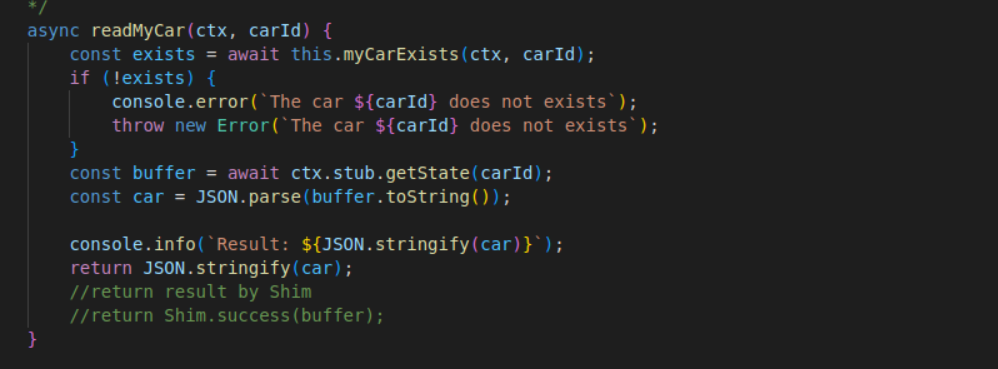


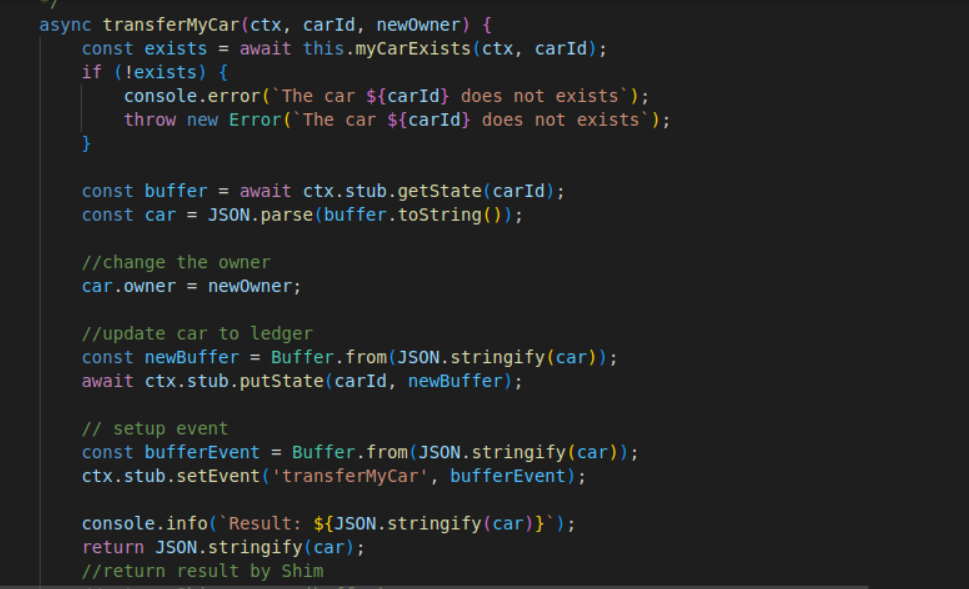


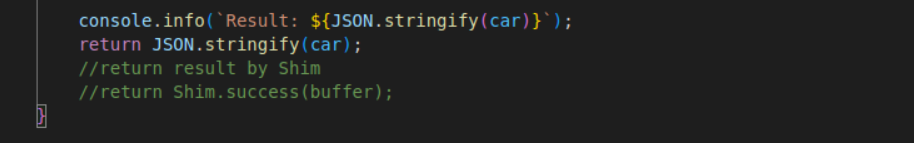


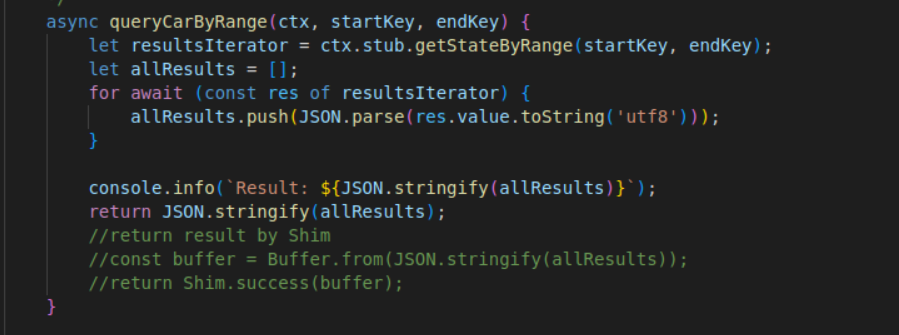












'use strict';

const { Contract } = require('fabric-contract-api');

const Shim = require('fabric-shim');

class MyCarContract extends Contract {

    /\*\*

     \* 確認car物件是否存在於Ledger

     \* @param {\*} ctx: Transation Context

     \* @param {String} carId: Car ID

     \* @returns true/false

     \* @example myCarExists(ctx, "car1")

     \* If car1 exist return true

     \* Else return false

    \*/

    async myCarExists(ctx, carId) {

        const buffer = await ctx.stub.getState(carId);

        return (!!buffer && buffer.length > 0);

    }

    /\*\*

     \* 新增Car物件

     \* @param {\*} ctx: Transation Context

     \* @param {String} carId: Car ID

     \* @param {String} brand: 廠牌

     \* @param {String} model: 型號

     \* @param {String} color: 顏色

     \* @param {String} owner: 擁有者

     \* @returns Car JSON String:

     \* @example createMyCar(ctx, "car1", "toyota", "altis", "white", "Tom")

     \* return {"brand":"toyota","carId":"car1","color":"white","docType":"car","model":"altis","owner":"Tom"}

    \*/

    async createMyCar(ctx, carId, brand, model, color, owner) {

        const exists = await this.myCarExists(ctx, carId);

        if (exists) {

            console.error(`The car ${carId} already exists`);

            throw new Error(`The car ${carId} already exists`);

        }

        // ==== Create car object and marshal to JSON ====

        let car = {

            docType: 'car',

            carId: carId,

            brand: brand,

            model: model,

            color: color,

            owner: owner

        };

        const buffer = Buffer.from(JSON.stringify(car));

        await ctx.stub.putState(carId, buffer);

        // setup event

        const bufferEvent = Buffer.from(JSON.stringify(car));

        ctx.stub.setEvent('createMyCar', bufferEvent);

        console.info(`Result: ${JSON.stringify(car)}`);

        return JSON.stringify(car)

        //return result by Shim

        //return Shim.success(buffer);

    }

    // 嚴謹寫法，透過try...catch...finally控制執行流程

    /\* async createMyCar(ctx, carId, brand, model, color, owner) {

        console.info('============= START: Create MyCar ===========');

        try {

            const exists = await this.myCarExists(ctx, carId);

            if (exists) {

                throw new Error(`The car ${carId} already exists`);

            }

            // ==== Create car object and marshal to JSON ====

            let car = {

                docType: 'car',

                carId: carId,

                brand: brand,

                model: model,

                color: color,

                owner: owner

            };

            const buffer = Buffer.from(JSON.stringify(car));

            const error = await ctx.stub.putState(carId, buffer);

            if (error.length == 0){

                console.info(`Result: ${JSON.stringify(car)}`);

                return Shim.success(buffer);

            } else {

                throw new Error(`Chaincode execute failed.\n ${error}`);

            }

        } catch (error) {

            console.error(`Chaincode execute with error.\n ${error}`);

            //console.error(error.message);

            //console.error(error.stack);

            throw new Error(`Chaincode execute with error.\n ${error}`);

        } finally {

            console.info('============= END: Create MyCar ===========');

        }

    } \*/

    /\*\*

     \* 查詢Car物件

     \* @param {\*} ctx: Transation Context

     \* @param {String} carId: Car ID

     \* @returns Car JSON String:

     \* @example readMyCar(ctx, "car1")

     \* return {"brand":"toyota","carId":"car1","color":"white","docType":"car","model":"altis","owner":"Tom"}

    \*/

    async readMyCar(ctx, carId) {

        const exists = await this.myCarExists(ctx, carId);

        if (!exists) {

            console.error(`The car ${carId} does not exists`);

            throw new Error(`The car ${carId} does not exists`);

        }

        const buffer = await ctx.stub.getState(carId);

        const car = JSON.parse(buffer.toString());

        console.info(`Result: ${JSON.stringify(car)}`);

        return JSON.stringify(car);

        //return result by Shim

        //return Shim.success(buffer);

    }

    /\*\*

     \* 查詢Car物件

     \* @description 只要函式名稱前面加上\_，此函式將不會被視為可被調用的智慧合約函式

     \* @param {\*} ctx: Transation Context

     \* @param {String} carId: Car ID

     \* @returns Car Object:

     \* @example readMyCar(ctx, "car1")

     \* return {docType:"car", carId:"car1", brand:"toyota", model:"altis", color:"white", owner:"Tom"}

    \*/

    async \_readMyCar(ctx, carId) {

        const exists = await this.myCarExists(ctx, carId);

        if (!exists) {

            console.error(`The car ${carId} does not exists`);

            throw new Error(`The car ${carId} does not exists`);

        }

        const buffer = await ctx.stub.getState(carId);

        const car = JSON.parse(buffer.toString());

        //return car Object

        return car

    }

    /\*\*

     \* 更新擁有者

     \* @param {\*} ctx: Transation Context

     \* @param {String} carId: Car ID

     \* @param {String} newOwner: 新擁有者

     \* @returns Car JSON String

     \* @example transferMyCar(ctx, "car1", "Mary")

     \* return {"brand":"toyota","carId":"car1","color":"white","docType":"car","model":"altis","owner":"Mary"}

    \*/

    async transferMyCar(ctx, carId, newOwner) {

        const exists = await this.myCarExists(ctx, carId);

        if (!exists) {

            console.error(`The car ${carId} does not exists`);

            throw new Error(`The car ${carId} does not exists`);

        }

        const buffer = await ctx.stub.getState(carId);

        const car = JSON.parse(buffer.toString());

        //change the owner

        car.owner = newOwner;

        //update car to ledger

        const newBuffer = Buffer.from(JSON.stringify(car));

        await ctx.stub.putState(carId, newBuffer);

        // setup event

        const bufferEvent = Buffer.from(JSON.stringify(car));

        ctx.stub.setEvent('transferMyCar', bufferEvent);

        console.info(`Result: ${JSON.stringify(car)}`);

        return JSON.stringify(car);

        //return result by Shim

        //return Shim.success(buffer);

    }

    /\*\*

     \* 刪除Car物件

     \* @param {\*} ctx: Transation Context

     \* @param {String} carId: Car ID

     \* @returns Empty

     \* @example deleteMyCar(ctx, "car1")

     \* return ""

    \*/

    async deleteMyCar(ctx, carId) {

    }

    /\*\*

     \* 查詢特定範圍的Car物件

     \* @param {\*} ctx: Transation Context

     \* @param {String} startKey: 開始Car ID

     \* @param {String} endKey: 結束Car ID

     \* @returns Car JSON String Array

     \* @example queryCarByRange(ctx, "car1", "car3")

     \* return [{"brand":"toyota","carId":"car1","color":"white","docType":"car","model":"altis","owner":"Tom"},{"brand":"honda","carId":"car2","color":"black","docType":"car","model":"fit","owner":"Alex"},{"brand":"bmw","carId":"car3","color":"red","docType":"car","model":"320","owner":"Bob"}]

    \*/

    async queryCarByRange(ctx, startKey, endKey) {

    }

    /\*\*

     \* 以擁有者查詢Car物件

     \* @param {\*} ctx: Transation Context

     \* @param {String} owner: 擁有者

     \* @returns Car JSON String Array

     \* @example queryCarByOwner(ctx, "Tom")

     \* return [{"brand":"toyota","carId":"car1","color":"white","docType":"car","model":"altis","owner":"Tom"}]

    \*/

    async queryCarByOwner(ctx, owner) {

    }

    /\*\*

     \* 從帳本查詢Car物件交易紀錄

     \* @param {\*} ctx: Transation Context

     \* @param {String} carId: Car ID

     \* @returnsKeyModification String Array

     \* @example readMyCar(ctx, "car7")

     \* return [{"timestamp":{"seconds":"1615047592","nanos":342000000},"txid":"f7b17b304ef75f2fee461ce393895fe540cb8ce088964fee18112dd3953255aa","isDelete":true,"data":"KEY DELETED"},{"timestamp":{"seconds":"1615047574","nanos":53000000},"txid":"864b5b495b3253b7aaffead6dee6a6320b066ed947a0cc8f44367b40bcdc145b","isDelete":false,"data":{"brand":"audi","carId":"car7","color":"brown","docType":"car","model":"A4","owner":"Alex"}},{"timestamp":{"seconds":"1615046815","nanos":624000000},"txid":"f5d347d653b547ffa16630a962fa48d32f58d29b7ead7b9edb4eaf4955bad6d3","isDelete":false,"data":{"docType":"car","carId":"car7","brand":"audi","model":"A4","color":"brown","owner":"Teddy"}}]

    \*/

    async queryCarHistory(ctx, carId) {

        const resultsIterator = ctx.stub.getHistoryForKey(carId);

        let allResults = [];

        for await (const keyMod of resultsIterator) {

            const resp = {

                timestamp: keyMod.timestamp,

                txid: keyMod.txId,

                isDelete: keyMod.isDelete

            }

            if (keyMod.isDelete) {

                resp.data = 'KEY DELETED';

            } else {

                resp.data = JSON.parse(keyMod.value.toString('utf8'));

            }

            allResults.push(resp);

        }

        console.info(`Result: ${JSON.stringify(allResults)}`);

        return JSON.stringify(allResults);

        //return result by Shim

        //const buffer = Buffer.from(JSON.stringify(allResults));

        //return Shim.success(buffer);

    }

    async InitLedger(ctx){

        const cars = [

            {

                carId: "car1",

                brand: "toyota",

                model: "altis",

                color: "white",

                owner: "Tom"

            },

            {

                carId: "car2",

                brand: "honda",

                model: "civic",

                color: "white",

                owner: "Ken"

            },

            {

                carId: "car3",

                brand: "bmw",

                model: "320",

                color: "blue",

                owner: "Bob"

            },

            {

                carId: "car4",

                brand: "benz",

                model: "A180",

                color: "red",

                owner: "Joe"

            },

            {

                carId: "car5",

                brand: "Ford",

                model: "focus",

                color: "black",

                owner: "Max"

            },

            {

                carId: "car6",

                brand: "Peugeot",

                model: "206",

                color: "violet",

                owner: "Grace"

            }

        ];

        for (const car of cars) {

            await this.createMyCar(ctx, car.carId, car.brand, car.model, car.color, car.owner);

        }

    }

}

module.exports = MyCarContract;