Exercises – ES2015 – ES2016

# Lab 1 – SET UP

## Install Browser

Install the Google Chrome Browser (recommended) on your machine (if not already there) and make it your default browser.

Proceed to Step 2 if you do not plan to use Visual Studio 2015 Update 1 or to Step 3 if you are going to use it.

## Without Visual Studio 2015 Update 1 – Install Node, Gulp, Babel

<https://travismaynard.com/writing/getting-started-with-gulp>

### Install Node

Visit [http://nodejs.org](http://nodejs.org/) and select the “Download” button for the latest stable available version.

Once your download completes, run that application

### Create Your Project Directory and initialize node

Create your project directory

Open the command prompt and navigate to your project directory

For example:  
cd \

md myproject

cd \myproject

Run npm command to initialize the package.json file

npm init

This will prompt you to answer a few small questions regarding your project. Accept the default values if you are unsure about what to answer.  
Once completed it will create a package.json file in the project's root directory which will contain information about the project and help manage dependencies.

### Install gulp

In your command-line application, type:

npm install -g gulp

Install gulp locally

npm install --save-dev gulp

### Set Up Your Gulpfile and Babel

Install Required Plugins  
In your command-line application, type:

npm install gulp-concat gulp-dest gulp-babel babel-core babel-polyfill babel-preset-es2015 rimraf --save-dev

Create gulpfile  
In the root directory of your project create a new file and name it gulpfile.js and paste the following code inside.

"use strict";

var gulp = require("gulp"),

rimraf = require("rimraf"),

dest = require('gulp-dest'),

babel = require("gulp-babel");

var paths = {

webroot: "./wwwroot/"

};

paths.js = paths.webroot + "js/\*\*/\*.js";

paths.minJs = paths.webroot + "js/\*\*/\*.min.js";

gulp.task(**"clean"**, function (cb) {

rimraf(paths.minJs, cb);

});

gulp.task(**"min"**, function () {

return gulp.src([paths.js, "!" + paths.minJs], { base: "." })

.pipe(babel())

.pipe(dest('.', { ext: '.min.js' }))

.pipe(gulp.dest('./'));

});

gulp.task(**'watch'**, function () {

gulp.watch('js/\*.js', ['min']);

});

// Default Task

gulp.task(**'default'**, ['clean', 'min', 'watch']);

#### Set Up Babel

Switch back over to your command-line and type:

notepad .babelrc

in your notepad type

{

"presets": [ "es2015" ]

}

Save the file .babelrc

### Test

Under the root of your project create a folder wwwroot

Under wwwroot create a folder js

Under the folder js create a file site.js

Open the file site.js and type

"use strict"

class Person {

constructor(name, surname) {

this.name = name;

this.surname = surname;

}

sayHi() {

console.log(`Hi my name is ${this.name} ${this.surname}!`);

}

}

Save the file site.js

Switch back over to your command-line and type:

gulp

You should now see a site.min.js with the following content:

"use strict";

var \_createClass = (function () { function defineProperties(target, props) { for (var i = 0; i < props.length; i++) { var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value" in descriptor) descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } } return function (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(Constructor.prototype, protoProps); if (staticProps) defineProperties(Constructor, staticProps); return Constructor; }; })();

function \_classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call a class as a function"); } }

var Person = (function () {

function Person(name, surname) {

\_classCallCheck(this, Person);

this.name = name;

this.surname = surname;

}

\_createClass(Person, [{

key: "sayHi",

value: function sayHi() {

console.log("Hi my name is " + this.name + " " + this.surname + "!");

}

}]);

return Person;

})();

## If you have Visual Studio 2015 Update 1 – Install Babel and configure Gulp

## Create an ASP.NET 5 project

Open Visual Studio 2015 Update 1

Click on New Project

Select Web – Asp.NET Web Application

Under ASP.NET 5 Templates , select Web Application

Click on “Change Authentication” and select “No Authentication”

Uncheck “Host In The Cloud”

Click On Ok

## Install Babel

In the Solution Explorer, open Dependencies

Right click npm and select “open package.json”

Under devDependencies add:

"gulp-babel": "^6.1.2",

"gulp-dest": "^0.2.3",

"babel-core": "^6.4.5",

"babel-polyfill": "^6.3.14",

"babel-preset-es2015": "^6.3.13"

#### *Configure Gulp*

In the Solution Explorer, open gulpfile.js

Replace the content of your gulpfile.js with

/// <binding Clean='**clean**' />

"use strict";

var gulp = require("**gulp**"),

rimraf = require("**rimraf**"),

dest = require('**gulp-dest**'),

babel = require("**gulp-babel**");

var paths = {

webroot: "**./wwwroot/**"

};

paths.js = paths.webroot + "**js/\*\*/\*.js**";

paths.minJs = paths.webroot + "**js/\*\*/\*.min.js**";

gulp.task("**clean**", function (cb) {

rimraf(paths.minJs, cb);

});

gulp.task("**min**", function () {

return gulp.src([paths.js, "**!**" + paths.minJs], { base: "**.**" })

.pipe(babel())

.pipe(dest('**.**', { ext: '**.min.js**' }))

.pipe(gulp.dest('**./**'));

});

## Configure babel

In the Solution Explorer, in the root folder of your project, right click Add -> new item

Select -> Client side -> JSON File

Name the file .babelrc

Replace the content of the file with

{

"presets": [ "es2015" ]

}

## Test

In the Solution Explorer, open wwwroot/js/site.js

Type

"use strict"

class Person {

constructor(name, surname) {

this.name = name;

this.surname = surname;

}

sayHi() {

console.log(`Hi my name is ${this.name} ${this.surname}!`);

}

}

Click on the Menu View -> Other Windows -> Task Runner Explorer

Under Tasks double click on min

In the Solution Explorer you should now see a wwwroot/js/site.min.js with the following content

"use strict";

var \_createClass = (function () { function defineProperties(target, props) { for (var i = 0; i < props.length; i++) { var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value" in descriptor) descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } } return function (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(Constructor.prototype, protoProps); if (staticProps) defineProperties(Constructor, staticProps); return Constructor; }; })();

function \_classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call a class as a function"); } }

var Person = (function () {

function Person(name, surname) {

\_classCallCheck(this, Person);

this.name = name;

this.surname = surname;

}

\_createClass(Person, [{

key: "sayHi",

value: function sayHi() {

console.log("Hi my name is " + this.name + " " + this.surname + "!");

}

}]);

return Person;

})();

# Lab 2 – Let and closures

* Create an HTML page with 4 buttons with id
  + button1, button2, button3, button4
* Write a loop from 1 to 4 that
  + finds a button(i)
  + Adds to the click event a listener function than logs on the console “You clicked on Button “ + i
* Use the var keyword to declare the counter for the loop
  + What happens when you click the buttons? Why?
* Fix it using the let keyword

# Lab 3 – Strings

* Create an array with at least 10 surnames, such as

"Jansen",

"De Vries",

"Van den Berg",

"Van Dijk",

"Bakker",

"Janssen",

"Visser",

"Smit",

"Meijer",

"De Boer",

"Mulder",

"De Groot",

"Bos",

"Vos",

"Peters",

"Hendriks",

"Van Leeuwen",

"Dekker",

"Brouwer",

"De Wit",

"Dijkstra",

"Smits",

"De Graaf",

"Van der Meer"

* Log to the console only the surnames starting with an A, in the following form

[surname] found on position [p]  
replacing [surname] and [p] with actual values

* Use new string methods and the string literals

# Lab 4 – functions

Write a function with an arbitrary number of parameters:

function sum(/\*arbitrary number of parameters\*/)

// returns the sum of all parameters

Write a function with an optional parameter:

function sayHello(/\*optional: name\*/)

// writes ‘Hello you’ to the console,

// or ‘Hello <name>’ if a name is provided

Rewrite them using arrow functions.

# Lab 5 – objects

Write a CreateCustomer function that accepts the following parameters:

CreateCustomer(id, name, city)

The functions returns an object that has the following properties:

id, name, city, nrOfUnpaidBills

and the methods:

buyStuff(), payBill()

When a Customer is created, the nrOfUnpaidBills equals 0. Each time buyStuff() is called, the nrOfUnpaidBills is increased by 1, and each time payBill() is called, the nrOfUnpaidBills is decreased by 1.

Add a method

badPayer(int n)  
that returns true, if the nrOfUnpaidBills is n or more

and returns false, otherwise.

Override the toString() method for Customer objects. It should return id, name, and city, formatted like:  
 "(id) name - city"

# Lab 6 – Destructuring

## Exercise 1

You receive from your server an object with the following data:

course = {

title: 'ES2015 and 2016',

description: 'New features of ES2015 and 2016',

editions: [

{

trainer: 'Matt Smith',

dates: {

start: '01/01/2016',

end : '05/01/2016'

},

location: {

address1: 'One Way Street',

city : 'New York'

}

},

{

dates: {

start: '03/05/2016',

end: '08/05/2016'

},

location: {

address1: 'Two Blocks Road',

address2: '1234 AB',

city: 'Las Vegas'

}

},

{

trainer: 'John Doe',

dates: {

start: '10/10/2016',

end : '15/10/2016'

},

location: {

address1: 'One Way Street',

city : 'New York'

}

},

]

};

Write a function firstEdition() that uses destructuring to extract the data of the first edition.

Return an object with the following structure:

{title, trainer, startDate, city}

Insert the value “TBD” into the trainer, should that not be present.

Invoke the function and save the trainer and city properties of the returned object into two variables t and c, using destructuring.

Print the values to the console.

## Exercise 2

Write a function addEdition accepting 5 parameters: title, trainer, startDate, duration, city.

The title argument must always be the first.

The other four arguments may or may not be passed, in any possible order.

The trainer parameter should have a “TBD” default value, unless otherwise specified by the caller.

The startDate parameter should have a current Date default value, unless otherwise specified by the caller.

The duration parameter should have a 5 default value, unless otherwise specified by the caller.

The city parameter should have a “TBD” default value, unless otherwise specified by the caller.

The function should log the values of the 5 parameters to the console.

Invoke the function without any argument.

Invoke the function with a the following arguments:

* title : “Course 1”
* trainer: “John”
* startDate : 1st of April 2016
* duration: 3
* city: “Ney York”

Invoke the function the following arguments:

* title : “Course 2”
* startDate : 1st of April 2016

Invoke the function the following argument:

* title : “Course 3”

# Lab 7 - Symbols

With the help of the Symbol.toPrimitive property (used as a function value), an object can be converted to a primitive value. The function is called with a string argument hint, which specifies the preferred type of the result primitive value. The hint argument can be one of "number", "string", and "default".

Create an empty object obj1.

Convert obj1 to a number using the unary plus (+) operator. Log the result to the console.

Log to the console the value of obj1 using a string literal.

Compare obj1 with 0 using the == operator. Log the result to the console.

Create an object obj2.

Override the Symbol.toPrimitive function to return:

10 if the hint is “number”

“hello” if hint is “string”

true otherwise

Convert obj2 to a number using the unary plus (+) operator. Log the result to the console.

Log to the console the value of obj2 using a string literal.

Compare obj2 with 0 using the == operator. Log the result to the console.

WARNING: do not transpile the script using Babel, since it does not support Symbol.toPrimitive. Use Chrome,Firefox or Edge instead to test the JavaScript 6 syntax.

# Lab 8 - Maps

You receive the following array from your server:

[{

email: "user1@gmail.com",

name: "User1",

address: "Street 1"

}, {

email: "user2@hotmail.com",

name: "User2",

address: "Street 2"

}, {

email: "user3@yahoo.com",

name: "User3",

address: "Street 3"

}, {

email: "user4@outlook.com",

name: "User4",

address: "Street 4"

}]

You want to be able to find a user via email and in order to do that you decide to use a Map.

* Initialize a **users** variable with the result of an immediately invoked arrow function expression that creates a new Map. Fill the Map with the array items and use the email property as key.
* Log the Map entries to the console
  + Use a for – of loop to loop through the Map entries
  + Use Array destructuring to retrieve the key and the value
  + Use string literals to concatenate the key and the value

You want to customize the output string used by the string literal of a user object. In order to do this you decide to fill the map with objects that override the Symbol.toPrimitive method instead of the original array items.

* Create a function createUser accepting an object with three properties email, name, address.
  + Use parameter destructuring to put the arguments values into three parameters email, name, address
  + Return a new object, using:
    - the property initializer shorthand for the three properties email, name and address
    - the concise method syntax to override the Symbol.toPrimitive method
* Modify the immediately invoked arrow function expression you defined in the previous exercise by filling the Map with the results of the invocation of the createUser method
* Log to the console the value of the Map entry associated with the user2@hotmail.com key

# Lab 9 – Iterators and generators

A Customer has an id, a name, a city and set of bills, divided in two piles: the paid ones and the unpaid ones.

When the customer gets a new bill, the bill goes to the unpaid pile.

When the customer pays a bill, the bill gets transferred to the unpaid pile.

Each bill has a unique identifier and an amount to be paid.

You have to be able to use your application like this:

let c1 = createCustomer(1,"Mario","Roma");

c1.bills.add({number: "ab123", amount : 123});

c1.bills.add({number: "cd456", amount : 456});

c1.bills.add({number: "ef789", amount : 789});

c1.bills.add({number: "gh012", amount : 128});

c1.bills.add({number: "ij386", amount : 946});

console.log("\*\*\*\*\*\*\*\*\*unpaid\*\*\*\*\*\*\*\*\*\*\*");

for(let b of c1.bills.unpaid()){

console.log(b.number, b.amount);

}

c1.bills.pay("ef789");

c1.bills.pay("cd456");

console.log("\*\*\*\*\*\*\*\*\*unpaid\*\*\*\*\*\*\*\*\*\*\*");

for(let b of c1.bills.unpaid()){

console.log(b.number, b.amount);

}

console.log("\*\*\*\*\*\*\*\*\*paid\*\*\*\*\*\*\*\*\*\*\*");

for(let b of c1.bills.paid()){

console.log(b.number, b.amount);

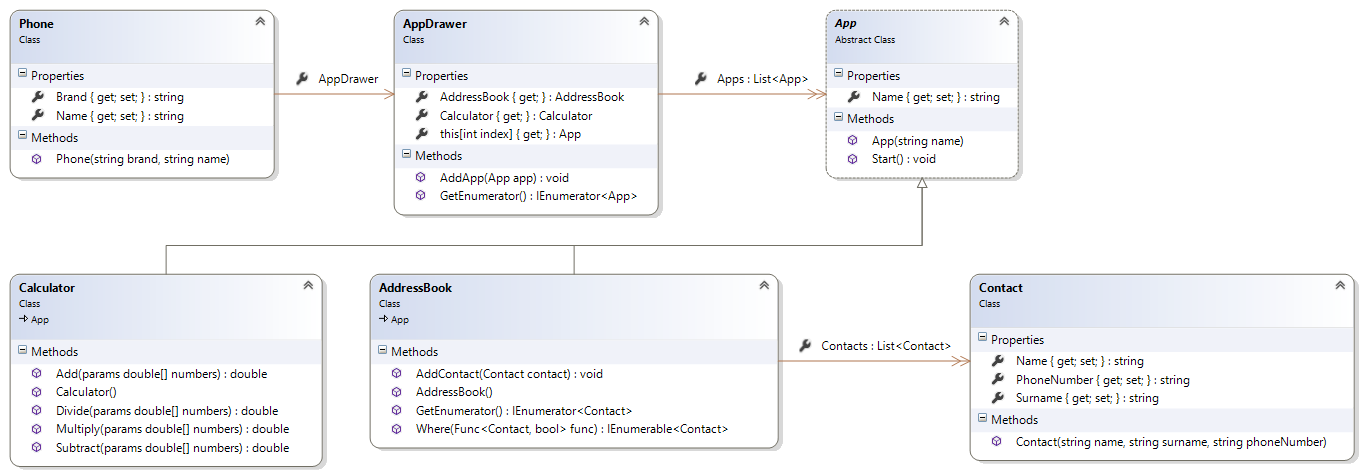
}

NOTE: You will need the polyfill.js file from a Babel polyfill installation. The easiest way to get this is to run npm install babel-polyfill (you already did this during lab1), then copy the polyfill.js file from the dist directory in the installed module.

Once you have polyfill.js, include it in your page via a script tag. In order for the polyfill to work, it must be included before any compiled ES2015 code from Babel.

# Lab 10 - Classes

Design a javascript application with the following structure

****

Phone:

* It has a model, a name and an App Drawer.
* Its constructor initializes the instance by
  + creating a new instance of an AppDrawer
  + Adding an instance of a Calculator and an instance of an AddressBook to its AppDrawer

AppDrawer:

* It has a collection of Apps
* It has a method to add an App   
  The method should check if the item type is App and if not it should throw an error.
* It can be iterated.
* It has a read only property that returns the Calculator instance
* It has a read only property that returns the AddressBook instance

App:

* It is an abstract class   
  It should throw an error if directly instantiated.
* It has a name property
* It has a start method.   
  The method should log the name of the app to the console.
* The constructor should accept a name and it should initialize the name property.

Calculator:

* It is an App.
* It initializes itself with a “Calculator” name, passing it to the constructor of its base class.
* It has four methods: Add, Subtract, Multiply and Divide.  
  These four methods accept a variable list of numbers and return the result of the corresponding operation performed on the parameters.

AddressBook:

* It is an App.
* It has a collection of Contact
* It has a method to add a Contact  
  The method should throw an error if the item to add is not a contact
* It is iterable
* It has an iterable method “where” that accepts function parameter
  + The function should accept a Contact and return a Boolean
  + The Boolean should be true if the Contact has to be returned by the where method

For each contact, the where method should

* invoke the function passing the Contact as an argument
* yield the Contact if the function returns true.

Contact:

* It has a name
* It has a surname
* It has a phone number

Write a Javascript application that:

* Creates an instance of a Phone
* Adds 4 contacts to the address book of the phone
* Logs to the console all the contacts of the address book
* Logs to the console the contacts whose phone number includes a “06”
* Invokes the add method of the calculator app of the phone and logs the result to the console

NOTE: You will need the polyfill.js file from a Babel polyfill installation. The easiest way to get this is to run npm install babel-polyfill (you already did this during lab1), then copy the polyfill.js file from the dist directory in the installed module (you did this during lab 09).

Once you have polyfill.js, include it in your page via a script tag. In order for the polyfill to work, it must be included before any compiled ES2015 code from Babel.

# Lab 11 – Promises

Your server hosts a set of files. The first file contains the title of a story and the urls of all the chapters, in a json format. The other files contain the title and the content of each chapter of the story.

You want to:

* Indicate that your page is loading the story
* Fetch the JSON file for the story
* Add title of the story to the page
* Start fetching each chapter as fast as possible (in parallel instead of sequentially)
* Add each chapter of the story to the page as fast as possible, sequentially in the correct order (you don’t want chapter 3 to be shown before chapter 2)
* Indicate that the loading is finished

The file of the story has the following format:

{

"title": "Welcome to Promiseland",

"chapterUrls": [

"./Foreword.json",

"./Introduction.json",

"./Chapter1.json",

"./Chapter2.json",

"./Chapter3.json"

]

}

The other files have the following format:

{

"title": "chapter 1",

"content": "Once upon a time, a promise was made."

}

Note:

* If you use Chrome, FireFox or Edge you don’t need to Babelify your code.
* If you want to use async / await, you need to use Babel with babel-polyfill and the stage-3 preset

# Lab 12 – Proxies

## Data Validation

Given the following target structure

const target = {

name: "John",

surname: "Doe",

address: "One Way Street 1",

age: 40

}

Create a Proxy object enforcing the following validation rules:

The name and surname properties should always be of type string, with a maximum length of 50 characters.

When setting a value longer than 50 characters, truncate it to the first 50 letters.

The age property should always be of type number, with a value between 0 and 200.

When setting an out of range value, silently fail.

NOTE: You should not use Babel to transpile your code, since Babel does not support Proxies. Test your code with Google Chrome, Firefox or Microsoft Edge.

# Lab 12 – Modules

## Set Up WebPack

### Install the node packages

#### If you’re using node and not Visual Studio 2015

On the command line, make sure you are in the project root directory and install the babel-loader and webpack modules:

npm install babel-loader webpack webpack-stream gulp-webpack-build --save-dev

#### If you’re using Visual Studio 2015

In the Solution Explorer, go to your Project, then open the Dependencies node and right click on npm, then select Open package.json.

Add the following entries to devDependencies section

"babel-loader": "^6.2.1",

"gulp-webpack-build": "^0.13.1",

"webpack": "^2.0.5-beta",

"webpack-stream": "^3.1.0",

### Configure WebPack to work with Babel

In your project root directory, create a new file named webpack.config.js defined as follows:

var path = require('path');

var webpack = require('webpack');

module.exports = {

entry: {

lab02: ['./wwwroot/js/lab02.js'],

lab03: ['./wwwroot/js/lab03.js'],

lab04: ['./wwwroot/js/lab04.js'],

lab05: ['./wwwroot/js/lab05.js'],

lab06\_01: ['./wwwroot/js/lab06\_01.js'],

lab06\_02: ['./wwwroot/js/lab06\_02.js'],

lab07: ['./wwwroot/js/lab07.js'],

lab08: ['./wwwroot/js/lab08.js'],

lab09: ['babel-polyfill', './wwwroot/js/lab09.js'],

lab10: ['babel-polyfill', './wwwroot/js/lab10.js'],

lab11: ['babel-polyfill', './wwwroot/js/lab11.js'],

lab13: ['babel-polyfill', './wwwroot/js/lab13.js']

},

output: {

path: path.resolve(\_\_dirname, './wwwroot/js'),

filename: '[name].bundle.js'

},

module: {

loaders: [

{

test: /\.js$/,

loader: 'babel-loader'

}

]

},

stats: {

colors: true

}

, devtool: 'source-map'

};

### Configure Gulp to work with Babel

Open your gulpfile.js

Replace all its content with

"use strict";

var gulp = require("gulp"),

rimraf = require("rimraf"),

webpackConfig = require("./webpack.config.js"),

gulpWebpack = require('webpack-stream'),

webpack = require('webpack');

var paths = {

webroot: "./wwwroot/"

};

paths.js = paths.webroot + "js/\*\*/\*.js";

paths.bundleJs = paths.webroot + "{js/\*\*/\*.min.js,js/\*\*/\*.bundle.js,js/\*\*/\*.bundle.js.map}";

gulp.task(**"clean"**, function (cb) {

rimraf(paths.bundleJs, cb);

});

gulp.task(**'webpack'**, [], function () {

return gulp.src('./wwwroot/js/lab12.js')

.pipe(gulpWebpack(webpackConfig, webpack))

.pipe(gulp.dest('./wwwroot/js'));

});

gulp.task(**'default'**, ['clean', 'webpack', 'watch']);

## Split your code into modules

Copy the code of the lab about classes.

Split it into different modules, one per each class plus one for the main application.

Import the modules where necessary.

Make sure that webpack creates a bundle starting from your main application.

Create an html page referencing the bundle.

NOTE: If you have not completed the labs about classes, use the following code as a starting point.

'use strict';

class App {

constructor(name) {

if (new.target === App)

throw new Error("This class cannot be instantiated.")

this.name = name;

}

start() {

console.log(`\*\*\*\*\*\*${this.name}\*\*\*\*\*\*`);

}

}

class AppDrawer {

constructor(){

this.apps = [];

}

addApp(app){

if(app instanceof App)

this.apps.push(app);

else

throw new Error("Cannot add item other than an App");

}

\*[Symbol.iterator](){

yield\* this.apps;

}

get addressBook(){

for (let app of this.apps) {

if(app instanceof AddressBook)

return app;

}

}

get calculator(){

for (let app of this.apps) {

if(app instanceof Calculator)

return app;

}

}

}

class Phone {

constructor(brand, model){

this.brand = brand;

this.model = model;

this.appDrawer = new AppDrawer();

this.appDrawer.addApp(new AddressBook());

this.appDrawer.addApp(new Calculator());

}

}

class Contact {

constructor(name, surname, phoneNumber) {

this.name = name;

this.surname = surname;

this.phoneNumber = phoneNumber;

}

}

class AddressBook extends App {

constructor() {

super("Address Book");

this.\_contacts = [];

}

start() {

super.start();

}

addContact(contact) {

if(contact instanceof Contact)

this.\_contacts.push(contact);

else

throw new Error("Cannot add item other than a Contact");

}

\*where(predicate) {

for(let contact of this.\_contacts){

if(predicate(contact)){

yield contact;

}

}

}

\*[Symbol.iterator](){

yield\* this.\_contacts;

}

}

class Calculator extends App{

constructor(){

super("Calculator");

}

add(...numbers){

let result = 0;

for(let n of numbers){

result +=n;

}

return result;

}

subtract(...numbers){

let result = 0;

for(let n of numbers){

result -=n;

}

return result;

}

multiply(...numbers){

let result = 1;

for(let n of numbers){

result \*=n;

}

return result;

}

divide(...numbers){

let result = 1;

for(let n of numbers){

result /=n;

}

return result;

}

}

let p1 = new Phone("Samsung", "Galaxy S5");

let ab = p1.appDrawer.addressBook;

ab.addContact(new Contact("Andrea", "Anderson", "06123456"));

ab.addContact(new Contact("Bob", "Builders", "018258392"));

ab.addContact(new Contact("Candice", "Clarkson", "0657890"));

ab.addContact(new Contact("David", "Danielson", "012356380"));

console.log("\*\*\*\*\*\*\*\*\*\*\*all contacts\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for(let c of ab){

console.log(c);

}

console.log("\*\*\*\*\*\*\*\*\*\*\*phone numbers including 06\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for(let c of ab.where(contact => contact.phoneNumber.includes("06"))){

console.log(c);

}

let calc = p1.appDrawer.calculator;

console.log(calc.add(1,2,3,4,5,6,7,8,9,10));

console.log(calc.multiply(1,2,3,4,5));