**Syed examples**

**Syed\_shared folder**

foo.js
module.exports = {

    something: 123

};

app.js

var foo = require('./foo');

console.log('initial something:', foo.something); // 123

// Now modify something:

foo.something = 456;

// Now load bar:

var bas = require('./bar');

bar.js
var foo = require('./foo');

console.log('in another module:', foo.something); // 456

**Run app**

PS C:\myNode\Syed\_shared> node app

initial something: 123

in another module: 456

**Notice that within this folder, foo is shared and so when we chan**g**e the value of something, that change persists, even in other modules (here in bar.js)**

**BUT – if we have foo export a function, then each time the function is called we will instantiate a new object!**

**Syed\_factory folder**

**foo.js - notice that foo now exports a function (not an object)**module.exports = function () {

    return {

        something: 123

    };

};

**app.js which now uses the function that foo exported to create an object**

var foo = require('./foo');

//Note that foo is a function, not an object

//create a new object

var obj = foo();

//use it

console.log(obj.something);

// Now modify something:

obj.something = 456;

// Now load bar:

var bas = require('./bar');

**bar.js also uses the function which foo exported**

var foo = require('./foo');

var newObj = foo();

console.log(newObj.something);

**When we run app - the change in the value of obj.something does not affect the value of what we created in bar.js ---- that is the constructor function is shared, but each time we run it we get another obect!**

PS C:\myNode\Syed\_factory> node app

123

123

**MORAL: When you export an object that object is shared. If one of the properties in the object is a VALUE, then the VALUE is shared across files in that project/folder. When the property is a FUNCTION, then each time you run the FUNCTION you instantiate a new object.**

**Exercise:** Write a new version of foo.js that exports

function() {return
 { aValue:123,
 aFunction: function() {
 console.log(this.aValue;
 return this.aValue} //end of aFunction
 } //end of the object which this function returns
} //end of the anonymous function.

Write new versions of app.js and bar.js which do the following:
 app.js should require foo, and use it to create an object appObj.
 (That is appObj = foo(); )
 It should log the value returned by appObj.aFunction and then change appObj.aValue to 456.
 Finally it should require bar.js (which will run bar.js)
Meanwhile, bar.js should also use the function returned by foo to create barObj = foo().
What will happen when app.js calls bar.js?