ECMAScript 6:
Classes & Modules

Persistent Interactive | Persistent University

Key Learning Points

- Property Shorthand
- Method Properties
- What are Modules?
- Export/Import
- Default/wild card values in modules
- Class Declaration
- Class Expression
- Getter/Setter
- Class inheritance
- Derived Classes from Expressions

Property Shorthand

- ES6 makes object literals more succinct by extending its syntax in many ways. It now eliminates the duplication that exists around property names and local variables by using property initializer shorthand syntax.
- Whenever an object property name is the same as the local variable name, then you just need to include the name without the colon and value.

Persistent

Property Shorthand…

- ES 5 and ES 6 ways to create objects

```
// ES5
function Human(name,age){
 return {
                name : name,
                age : age
                };
}
//ES6 way of creating object using property shorthand syntax
function Human(name,age){
 return {
                name ,
                age
                };
}
```

Method Properties

- ES6 also provides the improved syntax to assign methods to the object literals.
- Now you have to eliminate the colon and the function keyword for writing the method to an object.
- This is also called concise method syntax. For Example

```
var human={
            name:”Sachin”,
            dispName() {
                  console.log(this.name);
                  }
             };
```

What are Modules?

- An ES6 module is a JavaScript code that automatically runs in a strict mode.
- Modules have the capability to import or export only bindings that you need rather than everything in a file.
- Variables created in these modules are not automatically added to the shared global scope.
- A module has to export variables or functions that are required in the program.
- Modules can also import bindings from other modules.
- The value of ‘this’ in the top level of module is undefined.
- Modules does not allow HTML-style comments within the code.

Export/Import

- By using export keyword, it is possible to expose parts of the published code to other modules.
- Export can be used in front of any variable or class declaration to export it.
- You have to use the default keyword while trying to export the anonymous functions.
- You can also export references.
- Any variables or functions or classes that are not explicitly exported remain private to the module.

Export / Import…

- Functionality of an exported module can be accessed by importing that module by using ‘import’ keyword.
- You must specify the identifiers that you are importing and the module from which you are importing. For Example
- The list of bindings to import from a module looks like de-structured object but they are not actually the same.

```
import  {  identifier1, identifier2 } from  “./demo.js”;
```

Export/Import…

- Demo.js.

```
//demo.js
//export data
export var  item= "chair";
export let id= 777;

//export function
export function add(x , y){
                    return x + y;}
//main.js
//import function
import {add} from "./demo.js";
console.log(add(4,4));                                      //8
add=1;                                  //throws an error
```

Default/Wild Card Values in Modules

- The default value for a module is a single variable or function or class that you can export by specifying default keyword.
- Only one default export is possible per module. Using default keyword with multiple exports is a syntax error.
- Wild cards allows you to load the entire module and refer to the named exports using property notation.
- You can control what you want to expose from the module to be accessed.

```
//demo.js   -   Exporting default values
export default function square(x){ return x * x; }
export var x=5;

//main.js importing  default
import square from “./demo.js”;

console.log(square(6));

//main.js importing using wild card
import * as myMath from “./demo.js”;
Console.log(myMath.square(myMath.x));
```

Class Declaration

- ES6 class declaration is just a syntactic sugar on top of existing custom type class declaration.
- A basic class declaration in ES6 begin with the class keyword followed by the name of the class.
- ES6 class declaration allows you to define the constructor directly inside the class using constructor method.
- Class methods does not use the function keyword to define the methods.
- Own properties are also created inside a class constructor.
- Class declarations are not hoisted like function declarations.
- All the code inside the class runs in strict mode only.

Class Declaration…

- All methods of class are non-enumerable.

```
class Human {
        constructor( fname ) {
                this.fname = fname;
                }
        dispName(){
                console.log(this.fname);
                }
          }
let h1 = new Human(“Sachin”);
h1.dispName();              //Sachin
```

Class Expressions

- Classes in expression form does not require an identifier after class.
- Class expressions are designed to be used in variable declarations.
- These can be passed into functions as arguments.
- Class expressions are also not hoisted

Class Expressions…

- Example

```
let Human =  class{
        constructor(fname) {
                this.fname = fname;
                }
        dispName(){
                console.log(this.fname);
                }
        }
let h1 = new Human(“Sachin”);
h1.dispName();                                  //Sachin
```

Getter / Setter

- To create getter and setter in a class, get and set keywords are used followed by an identifier. For Example

```
class   Human {
        constructor(fname) {
                this.fname=fname;
        }
        get fname( ) {
                return this._fname;
        }
        set fname( newName){
                this._fname=newName;
        }
}
```

Getter / Setter

- Using class Human

```
let h1 = new Human(“Sachin”);
console.log(h1.fname);          //Sachin
h1.fname=“Rahul”;
console.log(h1.fname);          //Rahul
```

Class Inheritance

- Inheritance in ES6 is implemented by using the extends keyword.
- You can access the base class constructor by calling the super( ) method.
- Given below is the example to implement inheritance in ES6.

```
class Human{
        get fname( ){
            return this._fname;
        }
        set fname( newName){
            if(newName){
                return this._fname=newName;
                }
        } }
```

Class Inheritance…

- Extending class Human

```
class Person extends Human {
        get personId( ){
                return this.pId;
        }
        set personId(newId ){
                this.pId=newId;
        }
}
var p1 = new Person( );
p1.fname= " Sachin";          //Inherited from Human
p1.personId= 20;
```

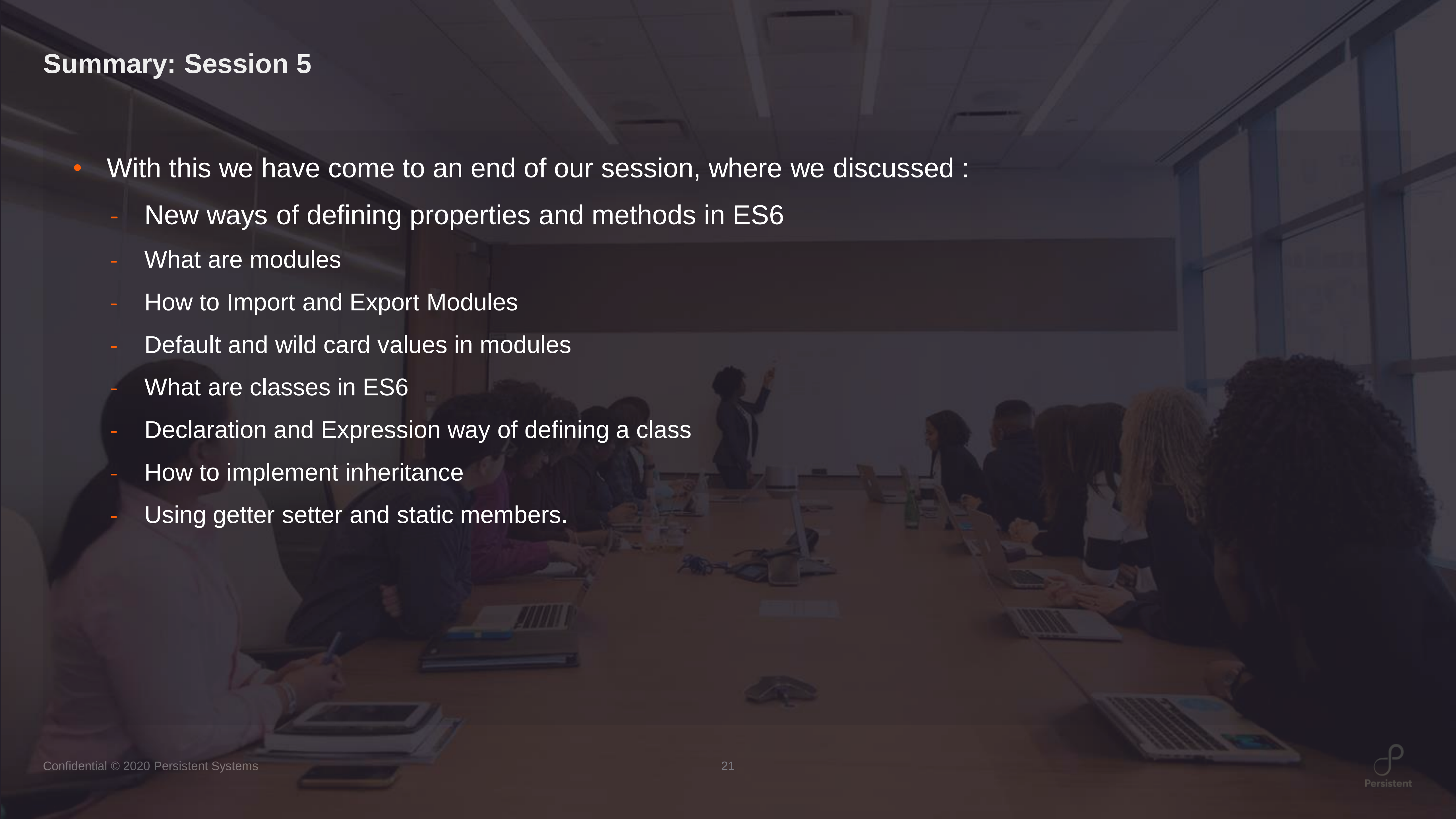
Derived Classes From Expressions

- Inheritance is possible using extends keyword with any class expression as long as the expressions resolves to a function with [[Construct]] and a prototype.
- Deriving classes from expressions provide a powerful possibility such as dynamically determining what to inherit directly from it.
- Not all expressions result in a valid class, specifically using null or a generator function after extends will cause errors. The reason behind this is that creating a new instance of the class will throw an error because there is no [[Construct]] to call.

Derived Classes From Expressions…

- For Example:

```
class Square extends getBase(){
// ES6 Class
                constructor(len){
                super(len, len);
                }
}
var  sq= new Square(4);
console.log(sq.calcArea());                                     //16
console.log(sq instanceof Rect)                                 //true
```



Summary: Session 5

- With this we have come to an end of our session, where we discussed :
  - New ways of defining properties and methods in ES6
  - What are modules
  - How to Import and Export Modules
  - Default and wild card values in modules
  - What are classes in ES6
  - Declaration and Expression way of defining a class
  - How to implement inheritance
  - Using getter setter and static members.

Appendix

References

Thank You

References

- http://odetocode.com/blogs/scott/archive/2014/11/25/class-inheritance-in-ecmascript-6.aspx
- http://es6-features.org
- Understanding ECMAScript 6 by Nicholas C. Zakas
- ECMAScript 6 Succinctly by Matthew Duffield

Persistent

Persistent

Thank you!

Persistent Interactive | Persistent University