Advance Java Script

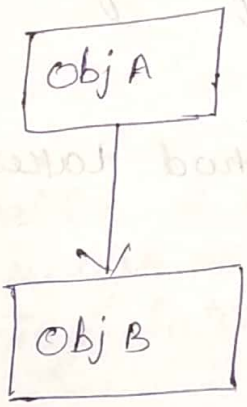
Inheritance

The process of defining an object and accessing the data of it (data members and member functions) within another object is called inheritance

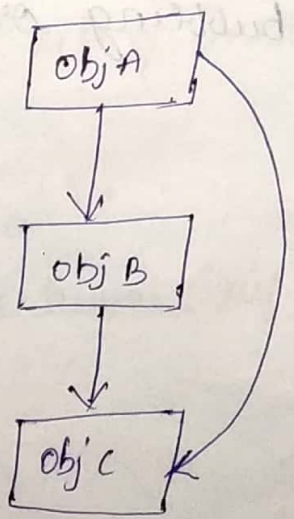
following are the types of inheritance being supported in

① Single level Inheritance
② Multi level Inheritance.

① Single level Inheritance



② Multi level Inheritance



Note: Due to data ambiguity issue js does not support multiple inheritance

following are the different ways between different types of object inheritance can be implemented.

① Inheritance b/w two static objects

② Inheritance b/w static object and dynamic object.

③ Inheritance b/w two dynamic objects.

① Inheritance b/w two static objects

* Every static object been created in js holds a pre-defined property by default '--proto--'.

* using '--proto--' we could able to extend the behaviour of any object to access another existing object with properties and behaviour.

① The main object, from which data gets derived is called parent object or base object.

② the object which is accessing the data or got derived from the parent object is called child object or derived object.

Inheritance b/w static object and dynamic object

"Object.create()" is a pre-defined way to create a dynamic object by inheriting data from a static object

Syntax:

```
Object.create(<existing static object>);
```

Eg:

```
var data = {
    ....
};
var childobj = Object.create(data);
```

childobj // holds its own properties along with it can point to existing data of "data" object.

Classes

Class indicates or represents a pre-defined structure of an object so that we can create any no. of objects having same structure but with different data. until ECMA-5 Classes were not directly supported in java script, using functions we could indirectly get features of classes.

* from ECMA-6, "class" is an pre-defined key word through which we could able to create classes directly.

ECMA 5 classes Syntax:

```
function <classname> (<optional params>) {
    this.key = '...';
    this.key2 = '...';
    ...
    this.method = function () {
        ...
    }
}
```

ECMA-6:

```
class <classname> {
    constructor (optional params) {
        this.key = '...';
        ...
    }
    method () {
        ....
    }
}
```

29/4/30

Exception Handling

Error / Exception: a set of instructions or a line of instructions making the execution of application to stop abruptly.

* The errors or exceptions might get raised while compiling the code or while executing the code

① Run-time Exceptions

② Compile time Exceptions

Compile Time Exceptions or syntactical Exceptions

The errors which gets raised while we compile the code is called compile time errors

Ex: Syntactical errors

* java script does not provide any way to handle compile time errors these errors are for sure need to be fixed to go further execution.

Run Time Exceptions

The Exceptions which gets raised while Executing the code is called runtime Exceptions or run time errors

Ex: 1. array index out of bound
2. file not found exception
3. referring a dom element which does not exist .. etc.

* In general, when a run time Exception gets raised the flow of execution gets stopped abruptly without going further Execution.

* java script provides a feature of handling Exceptions through which we could able to handle the Exceptions being raised and continue the execution flow without stop.

* following are the pre-defined keywords through which we could able to handle Exceptions.

① try
② catch
③ finally
④ throws

* using try, catch statement we could able to handle exceptions get raised at run time.

Try Block

The set of instructions in which there is a chance of getting run time error has to be placed under the try block

Catch Block

The set of instructions through which we could able to handle the Exception has to be placed under catch block.

① catch block should be the immediate block has to be placed after try block

② we cannot overload catch method in is, a single try block should have corresponding single catch method.

③ we cannot invoke the catch method manually it gets invoked automatically

when there is an exception under the try block.

(4) set of instructions through which we handle the error been raised at try block as to be placed under catch method block as to be placed under catch method

(5) catch method automatically gets involved holds any exception or error object with extra information about the current error been generated.

Syntax:
```
try {
    ... // set of instructions in which there is a chance of getting runtime error
    ...
} catch (error obj) {
    ...
    // set of instructions to handle error been raised
    ...
}
```

30/4/20 Finally Block

An optional block can be placed after the catch block which gets executed for sure irrelavent of whether there is a exception being raised within the try block or not

* The set of instructions which for sure need to be executed irrelavent of Exception been raised or not within the try block, has to be placed under finally block

Syntax:

```
try {
    .....
    ..... // set of instructions in which
    there is a chance of getting error
} catch (err) {
    ..... // code to handle error.
} finally {
    ..... // set of instructions need to Execute
    for sure, irrelavent of whether there
    is an exception raised or not.
}
```

Handling user-defined Exceptions

Java script provides a feature of handling user-defined Exceptions using "throws" Keyword. any time the controller reaches to the throw statement within the try block, it automatically treats that as an error and invokes corresponding catch block

Syntax:

```
try {
    .....
    throws "Exception desc"; // controller
    automatically jumps to catch block
    .....
} catch (err) {
```

... // instructions to handle error

}

01/5/20 Closures

Closure is a self invoked functions gets invoked automatically once the controller reaches to it.

* using closures we could able to bind set of java script instructions as a individual module
* The set of instructions within a closure module becomes a private data, cannot be accessible outside of the closure.
* within a closure we could able to bind set of js instructions like set of variables, methods, objects, classes etc.
* The data within a closure cannot be accessible outside of it even within the same page, it adds accessibility security to the data.

Syntax: (function () {

... // set of js instructions ...

})();

Steps to be followed to access closure data outside of the closure.

① create and assign a variable to the closure

(5) Return the data which need to be accessible outside of the closure. As like an object.

(3) using closure name, we could able to access closure returned object data outside of the closure.

Note: Not all the data within the closure can be accessible outside of it, only the data which is returned from the closure can be accessible through closure name.

Syntax:

```
Var closureName = (function () {
    ... // set of instructions
    return { // data within this object
        ...     can be accessable outside
    }           of closure.
})();
```

Eg:

```
Var userData = (function () {
    Var name = "..."; // cannot be accessible
                         outside.
    function sample () { // cannot be
                           accesable outside
        ....
    }
    ....
    return { // all the data under this
               object can be accesable
               outside of closure.
        Key_1 : Value;
        ...
        method : function {
```

```
    ....
        }
    }
})();
console.log(name); // throws error as name cannot be accessable outside of closure.
console.log(
    userdata.key-1); --> // can be accessable.
etc.
```