SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Re-accredited by NAAC with A+ grade, Accredited by NBA(CSE, IT, ECE, EEE & Mechanical)
Approvedy by AICTE, New Delhi, Recognized by UGC, Affiliated to Anna University, Chennai

Department of MCA

Subclass and Superclass

Course: 19CAT901 – Web Programming Essentials

Unit IV : Advanced Java Script

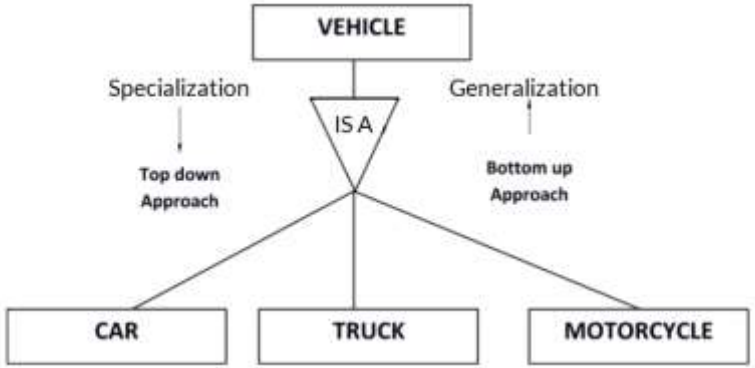
I Semester / I MCA

Subclasses, Superclasses, and Inheritance

- The Enhanced Entity Relationship Model contains all the features of the Entity Relationship model. In addition to all that, it also contains features of Subclasses, Superclasses and Inheritance.

Subclasses

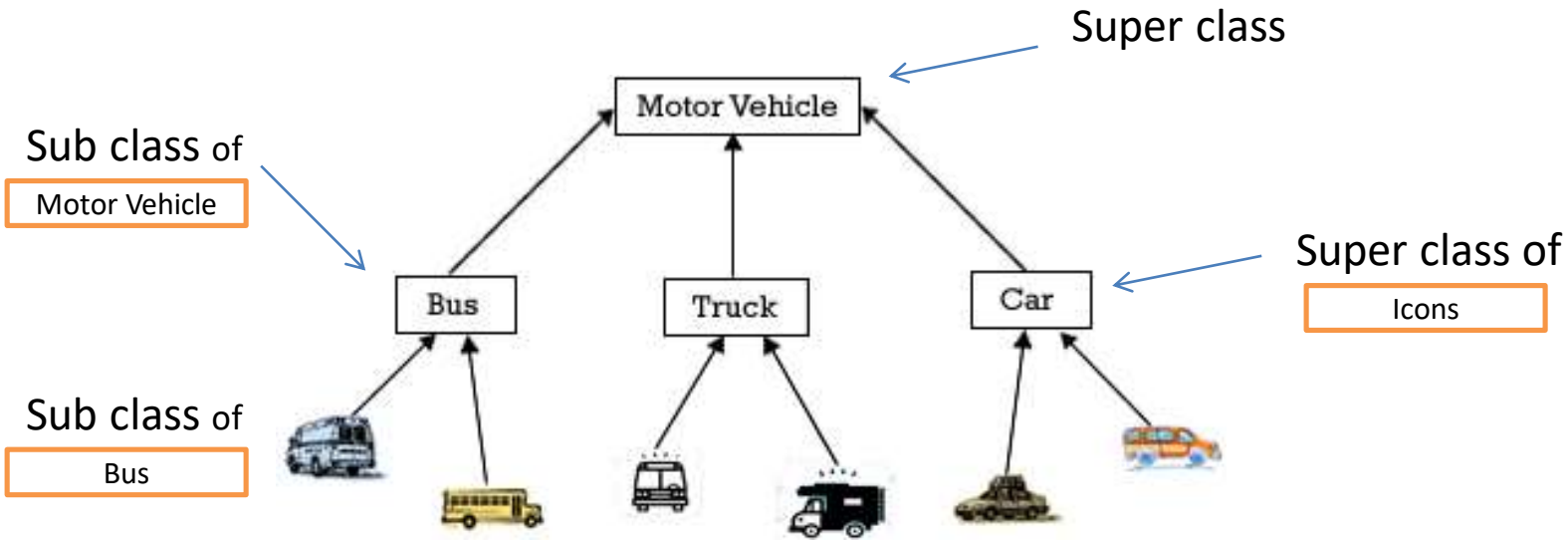
A subclass is a class derived from the superclass. It inherits the properties of the superclass and also contains attributes of its own. An example is:



Inheritance

Inheritance is basically the process of basing a class on another class i.e to build a class on a existing class. The new class contains all the features and functionalities of the old class in addition to its own.
The class which is newly created is known as the subclass or child class and the original class is the parent class or the superclass.

Sub and Super classes



Subclasses and Superclasses

In Java and C++, there is an explicit concept of the class hierarchy. i.e. Every class can have a super class from which it inherits properties and methods. Any class can be extended, or sub-classed so the resulting subclass can inherit its parent's behavior. As we have seen, JavaScript supports prototype inheritance instead of class based. It's possible for inheritance to happen other ways, however.
The following is an example of inheritance through functions.

Subclasses and Superclasses

```
<script language="javascript" type="text/javascript">
<!--

// thanks to webreference

function superClass() {
  this.supertest = superTest; //attach method superTest
}

function subClass() {
  this.inheritFrom = superClass;
  this.inheritFrom();
  this.subtest = subTest; //attach method subTest
}

function superTest() {
  return "superTest";
}

function subTest() {
  return "subTest";
}

  var newClass = new subClass();

  alert(newClass.subtest()); // yields "subTest"
  alert(newClass.supertest()); // yields "superTest"

//-->
</script>
```

JavaScript Class Inheritance

Class Inheritance

To create a class inheritance, use the extends keyword.

A class created with a class inheritance inherits all the methods from another class:

Example

Create a class named "Model" which will inherit the methods from the "Car" class:

JavaScript Class Inheritance

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

Getters and Setters

Classes also allows you to use getters and setters.

It can be smart to use getters and setters for your properties, especially if you want to do something special with the value before returning them, or before you set them.

To add getters and setters in the class, use the get and set keywords.

Example

Create a getter and a setter for the "carname" property:

Getters and Setters

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
  set cnam(x) {
    this.carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;
```

Getters and Setters

Note: even if the getter is a method, you do not use parentheses when you want to get the property value.

The name of the getter/setter method cannot be the same as the name of the property, in this case carname.

Many programmers use an underscore character _ before the property name to separate the getter/setter from the actual property:

Example

You can use the underscore character to separate the getter/setter from the actual property:

Getters and Setters

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.carname;
```

Object Properties

Inheritance

Override

Constructor chaining

Concrete class

Method chaining

Abstract class

Example

```
class Animal
{
 constructor(name)
 {
  this.speed = 0;
  this.name = name;
 }
 run(speed)
 {
   this.speed = speed;
  alert(`${this.name} runs with speed ${this.speed}.`);
}
stop()
{
this.speed = 0;
alert(`${this.name} stands still.`);
}
}
let animal = new Animal("My animal")
```

```
class Rabbit extends Animal
{
   hide()
  {
    alert(`${this.name} hides!`);
  }
}
let rabbit = new Rabbit("White Rabbit");
rabbit.run(5);
rabbit.hide()
```

References

- Thomas A. Powell, “HTML & CSS: The Complete Reference”, Fifth Edition, 2010
- https://www.w3schools.com/js/js_object_constructors.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/constructor
- https://css-tricks.com/understanding-javascript-constructors/

Problem Space