# Example One

## Pulsation Units

var PropulsionUnitsNS = (function () {

    function PropulsionUnit() {

        var getAcceleration = function () { };

        return {

            getAcceleration: getAcceleration

        };

    }

    //Wheel End

    function Wheel(radiusOfWheel) {

        if (!radiusOfWheel || radiusOfWheel <= 0 || isNaN(radiusOfWheel)) {

            throw {

                message: "Wheel should have positive radius",

                name: "Vehicle Error"

            };

        }

        var radius = parseInt(radiusOfWheel, 10);

        this.getAcceleration = function () {

            var acceleration = 2 \* radius;

            return acceleration;

        };

    }

    Wheel.inherit(PropulsionUnit);

    //Wheel End

    //Nozzle Start

    function PropellingNozzle(powerOfNozzle) {

        if (!powerOfNozzle || powerOfNozzle <= 0 || isNaN(powerOfNozzle)) {

            throw {

                message: "Nozzle should have positive power",

                name: "Vehicle Error"

            };

        }

        var power = parseInt(powerOfNozzle, 10);

        var afterBurner = false;

        this.getAcceleration = function () {

            var acceleration;

            if (afterBurner) {

                acceleration = 2 \* power;

            }

            else {

                acceleration = power;

            }

            return acceleration;

        };

        this.toggleAfterburner = function () {

            afterBurner = !afterBurner;

        };

    }

    PropellingNozzle.inherit(PropulsionUnit);

    //Nozzle End

    //Propeller Start

    function Propeller(numberOfFins, spinningDirection) {

        if (!numberOfFins || numberOfFins <= 0 || isNaN(numberOfFins)) {

            throw {

                message: "Propeller should have positive number of Fins",

                name: "Vehicle Error"

            };

        }

        if (!spinningDirection) {

            throw {

                message: "Propeller should have direction of spin",

                name: "Vehicle Error"

            };

        }

        if (spinningDirection !== "clockwise" && spinningDirection !== "counterClockwise") {

            throw {

                message: "Direction of spin should be 'clockwise' or 'counterClockwise'",

                name: "Vehicle Error"

            };

        }

        var numberFins = parseInt(numberOfFins, 10);

        var directionOfSpinning = spinningDirection;

        this.getAcceleration = function () {

            var acceleration = numberFins;

            if (directionOfSpinning === "counterClockwise") {

                acceleration = -acceleration;

            }

            return acceleration;

        };

        this.reverseSpinning = function () {

            if (directionOfSpinning === "clockwise") {

                directionOfSpinning = "counterClockwise";

            }

            else {

                directionOfSpinning = "clockwise";

            }

        };

    }

    Propeller.inherit(PropulsionUnit);

    //Propeller end

    ///visible outiside

    return {

        Wheel: Wheel,

        PropellingNozzle: PropellingNozzle,

        Propeller: Propeller

    };

})();

## VechiclesNS

Function.prototype.inherit = function (parent) {

    this.prototype = new parent();

    this.prototype.constructor = parent;

};

Function.prototype.extend = function (parent) {

    for (var i = 1; i < arguments.length; i++) {

        var propertyName = arguments[i];

        this[propertyName] = parent[propertyName];

    }

    return this;

};

var VehicleNS = (function () {

    var LAND\_VEHICLE\_WHEELS = 4;

    //base vehicle - not visible to outside - kind of abstract

    function Vehicle (vehicleSpeed) {

        if(!vehicleSpeed){

            vehicleSpeed = 0;

        }

        var speed = vehicleSpeed;

        var propulsionUnits = [];

        this.accelerate = function () {

            var i;

            for (i = 0; i < propulsionUnits.length; i++) {

                speed += propulsionUnits[i].getAcceleration();

            }

        };

        this.addPropulsion = function (propulsionUnit) {

            propulsionUnits.push(propulsionUnit);

        };

        this.getSpeed = function () {

            return speed;

        };

        //this is a method that exposes the methods of the propulsion

        //units without exposing the array they're in

        this.togglePropulsion = function (actionToToggle) {

            var i;

            for (i = 0; i < propulsionUnits.length; i++) {

                if (propulsionUnits[i][actionToToggle]) {

                    propulsionUnits[i][actionToToggle]();

                }

            }

        };

    }

    //Land Vehicle region 'wheels' should hold 4 wheels

    function LandVehicle(wheels) {

        if (!wheels || !areWheelsOK(wheels)) {

            throw {

                message: "LandVehicle should have " + LAND\_VEHICLE\_WHEELS +

                    " wheels. You passed somehing that doesen't hold that many wheels.",

                name: "Vehicle Error"

            };

        }

        function areWheelsOK(wheels) {

            var i;

            var wheelsAreOK = true;

            if (!(wheels instanceof Array)) {

                wheelsAreOK = false;

            }

            var len = wheels.length;

            if (len !== 4) {

                wheelsAreOK = false;

            }

            for (i = 0; i < wheels.length; i++) {

                if (!(wheels[i] instanceof PropulsionUnitsNS.Wheel)) {

                    wheelsAreOK = false;

                    break;

                }

            }

            return wheelsAreOK;

        }

        Vehicle.apply(this);

        for (var i = 0; i < LAND\_VEHICLE\_WHEELS; i++) {

            this.addPropulsion(wheels[i]);

        }

    }

    LandVehicle.inherit(Vehicle);

    //end of Land Vehicle

    //Air Vehicle

    function AirVehicle(propellingNozzle) {

        if (!propellingNozzle || !(propellingNozzle instanceof PropulsionUnitsNS.PropellingNozzle)) {

            throw {

                message: "Air Vehicles have one propelling nozzle - you passed something that's not a nozzle!",

                name: "Vehicle Error"

            };

        }

        Vehicle.apply(this);

        this.addPropulsion(propellingNozzle);

        this.toggleAfterburner = function () {

            this.togglePropulsion("toggleAfterburner");

        };

    }

    AirVehicle.inherit(Vehicle);

    //Air Vehicle End

    //SeaWehicle

    function SeaVehicle(propellerCount) {

        if (!propellerCount || isNaN(propellerCount)) {

            throw {

                message: "Sea vehicles must have pozitive number of propellers. You passed something else.",

                name: "Vehicle Error"

            };

        }

        var propellerSpotsAvailable = parseInt(propellerCount, 10);

        Vehicle.call(this);

        var oldPropulsionAdd = this.addPropulsion;

        this.addPropulsion = function (propeller) {

            if (propellerSpotsAvailable > 0) {

                oldPropulsionAdd(propeller);

                propellerSpotsAvailable--;

            }

            else {

                throw {

                    message: "The capacity for propellers of this boat is full. No more propellers can be added!",

                    name: "Vehicle Error"

                };

            }

        };

        this.reverseDirectionOfPropeller = function () {

            this.togglePropulsion("reverseSpinning");

        };

    }

    SeaVehicle.inherit(Vehicle);

    //SeaWehicle End

    function AmphibianVehicle(fourWheels, propeller, mode) {

        if (!mode || (mode !== "sea" && mode !== "land")) {

            throw exception;

        }

        Vehicle.apply(this);

        var boat = new SeaVehicle(1);

        boat.addPropulsion(propeller);

        this.reverseDirectionOfPropeller = function () {

            boat.reverseDirectionOfPropeller();

        };

        var car = new LandVehicle(fourWheels);

        var vehicleMode = mode;

        this.changeMode = function (mode) {

            if (mode === "sea") {

                this.getSpeed = boat.getSpeed;

                this.accelerate = boat.accelerate;

            }

            else if (mode === "land") {

                this.getSpeed = car.getSpeed;

                this.accelerate = car.accelerate;

            }

            else {

                throw exception;

            }

        };

        this.changeMode(vehicleMode);

        var exception = {

            message: "Amphibian has modes 'sea' and 'land'. You failed to specifiy which to use.",

            name: "Vehicle Error"

        };

    }

    //theese are visible to outside

    return {

        LandVehicle: LandVehicle,

        AirVehicle: AirVehicle,

        SeaVehicle: SeaVehicle,

        AmphibianVehicle: AmphibianVehicle

    };

})();

# Example Two

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <title>ConsoleModule</title>

</head>

<body>

    <script>

        // Add property inherit to Function

        Function.prototype.inherit = function (parent) {

            this.prototype = new parent();

            this.prototype.constructor = parent;

        }

        // Create object of type PropulsionUnit

        function PropulsionUnit() {

            this.acceleration = 0;

        }

        // Create object of type Weel

        function Weel(radius) {

            PropulsionUnit.apply(this, arguments);

            this.radius = radius;

            this.acceleration = 2 \* Math.PI \* this.radius;

        }

        Weel.inherit(PropulsionUnit);

        // Create object of type PropellingNozzle

        function PropellingNozzle(power) {

            PropulsionUnit.apply(this, arguments);

            this.power = power;

            this.afterburnerSwitch = false;

            this.acceleration = this.afterburnerSwitch ? 2 \* this.power : this.power;

        }

        PropellingNozzle.inherit(PropulsionUnit);

        // Create object of type Propeller

        function Propeller(numberFins) {

            PropulsionUnit.apply(this, arguments);

            this.numberFins = numberFins;

            this.spinDirection = true;

            this.acceleration = this.spinDirection ? this.numberFins : -this.numberFins;

        }

        Propeller.inherit(PropulsionUnit);

        // Create object of type Vehicle

        function Vehicle(speed, propulsionUnits) {

            this.speed = speed;

            this.propulsionUnits = propulsionUnits;

        }

        // Add property accelerate to Vehicle

        Vehicle.prototype.accelerate = function () {

            var totalAcceleration = 0;

            var length = this.propulsionUnits.length;

            for (var count = 0; count < length; count++) {

                totalAcceleration += this.propulsionUnits[count].acceleration;

            }

            this.speed += totalAcceleration;

        }

        // Create object of type LandVehicle

        function LandVehicle(speed, propulsionUnits) {

            Vehicle.apply(this, arguments);

        }

        LandVehicle.inherit(Vehicle);

        // Create object of type AirVehicle

        function AirVehicle(speed, propulsionUnits) {

            Vehicle.apply(this, arguments);

        }

        AirVehicle.inherit(Vehicle);

        // Add property turn to AirVehicle

        AirVehicle.prototype.turn = function () {

            if (this.propulsionUnits[0].afterburnerSwitch) {

                this.propulsionUnits[0].afterburnerSwitch = false;

            }

            else {

                this.propulsionUnits[0].afterburnerSwitch = true;

            }

        }

        // Create object of type WaterVehicle

        function WaterVehicle(speed, propulsionUnits) {

            Vehicle.apply(this, arguments);

        }

        WaterVehicle.inherit(Vehicle);

        // Add property changeSpin to WaterVehicle

        WaterVehicle.prototype.changeSpin = function (position) {

            if (this.propulsionUnits[position].spinDirection) {

                this.propulsionUnits[position].spinDirection = false;

            }

            else {

                this.propulsionUnits[position].spinDirection = true;

            }

        }

        // Create object of type AmphibiousVehicle

        function AmphibiousVehicle(speed, landPropulsionUnits, waterPropulsionUnits) {

            Vehicle.apply(this, arguments);

            this.terrain = true;

            this.landPropulsionUnits = landPropulsionUnits;

            this.waterPropulsionUnits = waterPropulsionUnits;

            this.propulsionUnits = this.terrain ? this.landPropulsionUnits : this.waterPropulsionUnits;

        }

        AmphibiousVehicle.inherit(Vehicle);

        // Add property changeTerrain to AmphibiousVehicle

        AmphibiousVehicle.prototype.changeTerrain = function () {

            if (this.terrain) {

                this.terrain = false;

            }

            else {

                this.terrain = true;

            }

        }

    </script>

</body>

</html>

# Example Three

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>JS Console - Example</title>

</head>

<body>

    <div id="header">

    </div>

    <div id="wrapper">

    </div>

<script>

var vehicleNS = (function () {

Function.prototype.inherit = function (parent) {

this.prototype = new parent();

this.prototype.constructor = parent;

}

function Vehicle(speed, propulsionUnits) {

this.speed = speed;

this.propulsionUnits = propulsionUnits;

}

Vehicle.prototype.accelerate = function () {

for (var i = 0; i < this.propulsionUnits.length; i++) {

this.speed += this.propulsionUnits[i].produceAcceleration();

}

}

//vehicles

function LandVehicle(speed, wheels) {

Vehicle.apply(this, arguments);

}

LandVehicle.inherit(Vehicle);

function AirCraft(speed, nozzle) {

Vehicle.apply(this, arguments);

}

AirCraft.inherit(Vehicle);

AirCraft.prototype.swichAfterburner = function (isafterburnerSwichOn) {

for (var i = 0; i < this.propulsionUnits.length; i++) {

if (this.propulsionUnits[i] instanceof Nozzle) {

this.propulsionUnits[i].changeAfterburnerState(isafterburnerSwichOn);

}

}

}

function WaterCraft(speed, propellers) {

Vehicle.apply(this, arguments);

}

WaterCraft.inherit(Vehicle);

WaterCraft.prototype.changeSpinDirection = function (direction) {

for (var i = 0; i < this.propulsionUnits.length; i++) {

if (this.propulsionUnits[i] instanceof Propeller) {

this.propulsionUnits[i].changeSpinDirection(direction);

}

}

}

function Amphibious(speed, wheels, propelers, mode) {

var propulsionUnits = [];

this.mode = mode;

for (var i = 0; i < wheels.length; i++) {

propulsionUnits.push(wheels[i]);

}

for (var i = 0; i < propelers.length; i++) {

propulsionUnits.push(propelers[i]);

}

Vehicle.call(this, speed, propulsionUnits);

}

Amphibious.inherit(Vehicle);

var MovementMode = {

Water: 0,

Land: 1

}

Amphibious.prototype = {

changeMode: function (mode) {

this.mode = mode;

this.speed = 0;

},

accelerate: function () {

for (var i = 0; i < this.propulsionUnits.length; i++) {

if (this.mode === MovementMode.Land && this.propulsionUnits[i] instanceof Wheel) {

this.speed += this.propulsionUnits[i].produceAcceleration();

}

else if (this.mode === MovementMode.Water && this.propulsionUnits[i] instanceof Propeller) {

this.speed += this.propulsionUnits[i].produceAcceleration();

}

}

}

}

// propulsion unit

function PropulsionUnit() {

}

PropulsionUnit.prototype.produceAcceleration = function () {

throw new Error("Must be override from its children");

}

function Wheel(radius) {

this.radius = radius;

PropulsionUnit.apply(this, arguments);

}

Wheel.inherit(PropulsionUnit);

Wheel.prototype.produceAcceleration = function () {

var acceleration = 2 \* Math.PI \* this.radius;

return acceleration;

}

function Nozzle(power, isAfterburnerSwich) {

this.power = power;

this.isAfrerburnerSwich = isAfterburnerSwich;

PropulsionUnit.apply(this, arguments);

}

Nozzle.inherit(PropulsionUnit);

Nozzle.prototype = {

produceAcceleration: function () {

var acceleration = this.power;

if (this.isAfrerburnerSwich) {

acceleration \*= 2;

}

return acceleration;

},

changeAfterburnerState: function (isAfterburnerSwichOn) {

this.isAfrerburnerSwich = isAfterburnerSwichOn;

}

}

function Propeller(finCount, spinDirection) {

this.finCount = finCount;

this.spinDirection = spinDirection;

PropulsionUnit.apply(this, arguments);

}

var DirecnionEnum = {

clockwise: 0,

counterClockwise: 1

}

Propeller.inherit(PropulsionUnit);

Propeller.prototype = {

produceAcceleration: function () {

var accelerationa;

if (this.spinDirection == DirecnionEnum.clockwise) {

acceleration = this.finCount;

}

else if (this.spinDirection == DirecnionEnum.counterClockwise) {

acceleration = -1 \* this.finCount;

}

return acceleration;

},

changeSpinDirection: function (spinDirection) {

this.spinDirection = spinDirection;

}

}

return {

DirecnionEnum: DirecnionEnum,

Wheel: Wheel,

Propeller: Propeller,

Nozzle: Nozzle,

LandVehicle: LandVehicle,

AirCraft: AirCraft,

WaterCraft: WaterCraft,

MovementMode: MovementMode,

Amphibious: Amphibious

}

})();

//test

var wheels = [

            new vehicleNS.Wheel(5),

            new vehicleNS.Wheel(5),

            new vehicleNS.Wheel(5),

            new vehicleNS.Wheel(5)

]

var car = new vehicleNS.LandVehicle(50, wheels);

console.log("Car");

console.log("Speed: " + car.speed);

car.accelerate();

console.log("Speed after acceleration: " + car.speed);

var nozzle = [new vehicleNS.Nozzle(500, false)];

var airCraft = new vehicleNS.AirCraft(2000, nozzle);

console.log("AirCraft");

console.log("Speed: " + airCraft.speed);

airCraft.accelerate();

console.log("Speed after acceleration: " + airCraft.speed);

airCraft.swichAfterburner(true);

airCraft.accelerate();

console.log("Speed after acceleration and Afterburner swich on: " + airCraft.speed);

var propeller = [

            new vehicleNS.Propeller(6,vehicleNS.DirecnionEnum.clockwise),

            new vehicleNS.Propeller(6,vehicleNS.DirecnionEnum.clockwise),

            new vehicleNS.Propeller(6,vehicleNS.DirecnionEnum.clockwise)

]

var waterCraft = new vehicleNS.WaterCraft(30, propeller);

console.log("WaterCraft");

console.log("Speed: " + waterCraft.speed);

waterCraft.accelerate();

console.log("Speed after acceleration: " + waterCraft.speed);

waterCraft.changeSpinDirection(vehicleNS.DirecnionEnum.counterClockwise);

waterCraft.accelerate();

console.log("Speed after acceleration and spin direction change: " + waterCraft.speed);

var amphibiousPropeller = [

            new vehicleNS.Propeller(6, vehicleNS.DirecnionEnum.clockwise),

            new vehicleNS.Propeller(6, vehicleNS.DirecnionEnum.clockwise),

            new vehicleNS.Propeller(6, vehicleNS.DirecnionEnum.clockwise),

            new vehicleNS.Propeller(6, vehicleNS.DirecnionEnum.clockwise)

]

var amphibious = new vehicleNS.Amphibious(50, wheels, amphibiousPropeller, vehicleNS.MovementMode.Land);

console.log("Amphibious");

console.log("Speed: " + amphibious.speed);

amphibious.accelerate();

console.log("Speed after acceleration: " + amphibious.speed);

amphibious.changeMode(vehicleNS.MovementMode.Water);

amphibious.accelerate();

console.log("Speed after acceleration and moving mode changed to water: " + amphibious.speed);

</script>

</body>

</html>

# Example four

## VechicleNS

var vehiclesNS = (function () {

    var LAND\_VEHICLE\_WHEELS\_COUNT = 4;

    var AfterburnerState = {

        "OFF": "OFF",

        "ON": "ON"

    };

    var SpinDirection = {

        "CLOCKWISE": "CLOCKWISE",

        "COUNTERCLOCKWISE": "COUNTERCLOCKWISE"

    };

    var AmphibiaMode = {

        "LAND": "LAND",

        "WATER": "WATER"

    };

    Function.prototype.inherit = function (parent) {

        this.prototype = new parent();

        this.prototype.constructor = parent;

    }

    Function.prototype.extend = function (parent) {

        for (var i = 1; i < arguments.length; i += 1) {

            var property = arguments[i];

            this.prototype[property] = parent.prototype[property];

        }

        return this;

    }

    // Propulsion unit class and methods

    function PropulsionUnit() {

    }

    PropulsionUnit.prototype.getAcceleration = function () {

        throw new Error("Cannot get acceleration without specifying the propulsion unit type!");

    }

    // Wheel class and methods

    function Wheel(radiusInInches) {

        if (radiusInInches <= 0) {

            throw new RangeError("Wheel's radius must be a positive number!");

        }

        this.radius = radiusInInches;

    }

    Wheel.inherit(PropulsionUnit);

    Wheel.prototype.getAcceleration = function () {

        var acceleration = 2 \* Math.PI \* this.radius;

        return acceleration;

    }

    // Propelling nozzle class and methods

    function PropellingNozzle(powerInHp, afterburnerState) {

        if (powerInHp <= 0) {

            throw new RangeError("Propelling nozzle's power must be a positive number!");

        }

        if (afterburnerState != AfterburnerState.OFF && afterburnerState != AfterburnerState.ON) {

            throw new Error("Provided afterburner state is invalid!");

        }

        this.power = powerInHp;

        this.afterburnerState = afterburnerState;

    }

    PropellingNozzle.inherit(PropulsionUnit);

    PropellingNozzle.prototype.getAcceleration = function () {

        var acceleration;

        if (this.afterburnerState == AfterburnerState.OFF) {

            acceleration = this.power;

        } else {

            acceleration = 2 \* this.power;

        }

        return acceleration;

    }

    // Propeller class and methods

    function Propeller(finsCount, spinDirection) {

        if (finsCount <= 0) {

            throw new RangeError("Propeller's fins count must be a positive number!");

        }

        if (spinDirection != SpinDirection.CLOCKWISE && spinDirection != SpinDirection.COUNTERCLOCKWISE) {

            throw new Error("Provided spin direction is invalid!");

        }

        this.finsCount = finsCount;

        this.spinDirection = spinDirection;

    }

    Propeller.inherit(PropulsionUnit);

    Propeller.prototype.getAcceleration = function () {

        var acceleration;

        if (this.spinDirection == SpinDirection.CLOCKWISE) {

            acceleration = this.finsCount;

        } else {

            acceleration = -1 \* this.finsCount;

        }

        return acceleration;

    }

    // Vehicle class and methods

    function Vehicle(speedInKmH, propulsionUnits) {

        this.speed = speedInKmH;

        this.propulsionUnits = propulsionUnits;

    }

    Vehicle.prototype.accelerate = function () {

        var propulsionUnitsCount = this.propulsionUnits.length;

        for (var i = 0; i < propulsionUnitsCount; i++) {

            this.speed += this.propulsionUnits[i].getAcceleration();

        }

    }

    // Land vehicle class and methods

    function LandVehicle(speedInKmH, wheels) {

        if (wheels.length != LAND\_VEHICLE\_WHEELS\_COUNT) {

            throw new Error("Invalid number of wheels for land vehicle!");

        }

        Vehicle.call(this, speedInKmH, wheels);

    }

    LandVehicle.inherit(Vehicle);

    // Air vehicle class and methods

    function AirVehicle(speedInKmH, propellingNozzle) {

        Vehicle.call(this, speedInKmH, [propellingNozzle]);

    }

    AirVehicle.inherit(Vehicle);

    AirVehicle.prototype.setAfterburnerState = function (afterBurnerState) {

        var propulsionUnitsCount = this.propulsionUnits.length;

        for (var i = 0; i < propulsionUnitsCount; i++) {

            if (this.propulsionUnits[i] instanceof PropellingNozzle) {

                this.propulsionUnits[i].afterburnerState = afterBurnerState;

            }

        }

    }

    // Water vehicle class and methods

    function WaterVehicle(speedInKmH, propellers) {

        Vehicle.call(this, speedInKmH, propellers);

    }

    WaterVehicle.inherit(Vehicle);

    WaterVehicle.prototype.setPropellersSpinDirection = function (spinDirection) {

        var propulsionUnitsCount = this.propulsionUnits.length;

        for (var i = 0; i < propulsionUnitsCount; i++) {

            if (this.propulsionUnits[i] instanceof Propeller) {

                this.propulsionUnits[i].spinDirection = spinDirection;

            }

        }

    }

    // Amphibious vehicle class and methods

    function Amphibia(speedInKmH, propeller, wheels, mode) {

        var propulsionUnits = [];

        propulsionUnits.push(propeller);

        for (var i = 0, wheelsCount = wheels.length; i < wheelsCount; i++) {

            propulsionUnits.push(wheels[i]);

        }

        Vehicle.call(this, speedInKmH, propulsionUnits);

        this.mode = mode;

    }

    Amphibia.inherit(Vehicle);

    Amphibia.extend(WaterVehicle, "setPropellersSpinDirection");

    Amphibia.prototype.setMode = function (mode) {

        this.mode = mode;

    }

    Amphibia.prototype.accelerate = function () {

        if (this.mode == AmphibiaMode.LAND) {

            var propulsionUnitsCount = this.propulsionUnits.length;

            for (var i = 1; i < propulsionUnitsCount; i++) {

                if (this.propulsionUnits[i] instanceof Wheel) {

                    this.speed += this.propulsionUnits[i].getAcceleration();

                }

            }

        } else {

            this.speed += this.propulsionUnits[0].getAcceleration();

        }

    }

    // Public classes for the outher world

    return {

        Amphibia: Amphibia,

        AirVehicle: AirVehicle,

        LandVehicle: LandVehicle,

        WaterVehicle: WaterVehicle,

        AfterburnerState: AfterburnerState,

        SpinDirection: SpinDirection,

        AmphibiaMode: AmphibiaMode,

        Wheel: Wheel,

        PropellingNozzle: PropellingNozzle,

        Propeller: Propeller

    }

})();

## Vechicles

<html>

<head>

    <meta charset="utf-8" />

    <title>Vehicles class hierarchy</title>

    <script src="scripts/VehiclesNS.js"></script>

    <link href="styles/js-console.css" rel="stylesheet" />

    <style>

        h1 {

            color: #00ff21;

            text-align: center;

        }

    </style>

</head>

<body>

    <h1>Presenting the functionality of the Vehicles class hierarchy</h1>

    <div id="js-console"></div>

    <script src="scripts/js-console.js"></script>

    <script>

        var wheels = [

            new vehiclesNS.Wheel(5),

            new vehiclesNS.Wheel(5),

            new vehiclesNS.Wheel(5),

            new vehiclesNS.Wheel(5)

        ];

        var landVehicle = new vehiclesNS.LandVehicle(30, wheels);

        jsConsole.writeLine("The initial speed of a land vehicle: " + landVehicle.speed);

        jsConsole.writeLine("The radius of the wheels of the land vehicle: " + landVehicle.propulsionUnits[0].radius);

        landVehicle.accelerate();

        jsConsole.writeLine("The speed of the land vehicle after acceleration: " + landVehicle.speed);

        jsConsole.writeLine();

        var propellingNozzle = new vehiclesNS.PropellingNozzle(500, vehiclesNS.AfterburnerState.ON);

        var airVehicle = new vehiclesNS.AirVehicle(150, propellingNozzle);

        jsConsole.writeLine("The initial speed of an air vehicle: " + airVehicle.speed);

        jsConsole.writeLine("The power of the propelling nozzle of the air vehicle: " + airVehicle.propulsionUnits[0].power);

        jsConsole.writeLine("The afterburner state of the propelling nozzle of the air vehicle: "

            + airVehicle.propulsionUnits[0].afterburnerState);

        airVehicle.accelerate();

        jsConsole.writeLine("The speed of the air vehicle after acceleration: " + airVehicle.speed);

        jsConsole.writeLine("Switching off the afterburner...")

        airVehicle.setAfterburnerState(vehiclesNS.AfterburnerState.OFF);

        jsConsole.writeLine("The afterburner state of the propelling nozzle of the air vehicle: "

            + airVehicle.propulsionUnits[0].afterburnerState);

        airVehicle.accelerate();

        jsConsole.writeLine("The speed of the air vehicle after acceleration: " + airVehicle.speed);

        jsConsole.writeLine();

        var propellers = [

            new vehiclesNS.Propeller(30, vehiclesNS.SpinDirection.CLOCKWISE),

            new vehiclesNS.Propeller(30, vehiclesNS.SpinDirection.CLOCKWISE),

            new vehiclesNS.Propeller(30, vehiclesNS.SpinDirection.CLOCKWISE),

            new vehiclesNS.Propeller(30, vehiclesNS.SpinDirection.CLOCKWISE),

            new vehiclesNS.Propeller(30, vehiclesNS.SpinDirection.CLOCKWISE)

        ];

        var waterVehicle = new vehiclesNS.WaterVehicle(25, propellers);

        jsConsole.writeLine("The initial speed of an water vehicle: " + waterVehicle.speed);

        jsConsole.writeLine("The fins count of a propeller of the water vehicle: " + waterVehicle.propulsionUnits[0].finsCount);

        jsConsole.writeLine("The spin direction of the propellers of the water vehicle: "

            + waterVehicle.propulsionUnits[0].spinDirection);

        waterVehicle.accelerate();

        jsConsole.writeLine("The speed of the water vehicle after acceleration: " + waterVehicle.speed);

        jsConsole.writeLine("Changing the spin direction of the propellers to counter-clockwise...");

        waterVehicle.setPropellersSpinDirection(vehiclesNS.SpinDirection.COUNTERCLOCKWISE);

        jsConsole.writeLine("The spin direction of the propellers of the water vehicle: "

            + waterVehicle.propulsionUnits[0].spinDirection);

        waterVehicle.accelerate();

        jsConsole.writeLine("The speed of the water vehicle after acceleration: " + waterVehicle.speed);

        jsConsole.writeLine();

        var amphibia = new vehiclesNS.Amphibia(20, propellers[0], wheels, vehiclesNS.AmphibiaMode.WATER);

        jsConsole.writeLine("The mode of the amphibian vehicle: " + amphibia.mode);

        jsConsole.writeLine("The initial speed of an amphibian vehicle: " + amphibia.speed);

        jsConsole.writeLine("The fins count of the propeller of the amphibian vehicle: " + amphibia.propulsionUnits[0].finsCount);

        amphibia.propulsionUnits[0].spinDirection = vehiclesNS.SpinDirection.CLOCKWISE;

        jsConsole.writeLine("The spin direction of the propeller of the amphibia vehicle: "

            + amphibia.propulsionUnits[0].spinDirection);

        amphibia.accelerate();

        jsConsole.writeLine("The speed of the amphibian vehicle after acceleration: " + amphibia.speed);

        jsConsole.writeLine("Changing the spin direction of the propeller to counter-clockwise...");

        amphibia.propulsionUnits[0].spinDirection = vehiclesNS.SpinDirection.COUNTERCLOCKWISE;

        jsConsole.writeLine("The spin direction of the propeller of the amphibia vehicle: "

            + amphibia.propulsionUnits[0].spinDirection);

        amphibia.accelerate();

        jsConsole.writeLine("The speed of the water vehicle after acceleration: " + amphibia.speed);

        jsConsole.writeLine();

        jsConsole.writeLine("Changing the mode of the amphibian vehicle...");

        amphibia.setMode(vehiclesNS.AmphibiaMode.LAND);

        jsConsole.writeLine("The mode of the amphibian vehicle: " + amphibia.mode);

        jsConsole.writeLine("The initial speed of the amphibian vehicle: " + amphibia.speed);

        jsConsole.writeLine("The number of wheels of the amphibian vehicle: " + (amphibia.propulsionUnits.length - 1));

        jsConsole.writeLine("The radius of the wheels of the amphibian vehicle: " + amphibia.propulsionUnits[1].radius);

        amphibia.accelerate();

        jsConsole.writeLine("The speed of the amphibian vehicle after acceleration: " + amphibia.speed);

    </script>

</body>

</html>