OOP in JS, Part 1 : Public/Private Variables and Methods

This page shows how to create private variables and methods in "classes" in Javascript through the rather simple example of a person. (This is really a very rough approximation of the classical OOP pattern, and not necessarily a "best-practice" for JavaScript programming.) Part 2 covers inheritance.

Summary

- **private variables** are declared with the 'var' keyword inside the object, and can only be accessed by private functions and privileged methods.
- **private functions** are declared inline inside the object's constructor (or alternatively may be defined via `var functionName=function(){...}`) and may only be called by privileged methods (including the object's constructor).
- **privileged methods** are declared with `this.methodName=function(){...}` and may invoked by code external to the object.
- **public properties** are declared with `this.variableName` and may be read/written from outside the object.
- **public methods** are defined by `Classname.prototype.methodName = function(){...}` and may be called from outside the object.
- **prototype properties** are defined by `Classname.prototype.propertyName = someValue`
- **static properties** are defined by `Classname.propertyName = someValue`

Example

In this example, a person's name and race are set at birth and may never be changed. When created, a person starts out at year 1 and a hidden maximum age is determined for that person. The person has a weight which is modified by eating (tripling their weight) or exercising (halfing it). Every time the person eats or exercises, they grow a year older. The person object has a publicly accessible 'clothing' property which anyone can modify, as well as a dirtFactor which can be modified manually (throwing dirt on or scrubbing it off), but which increases every time the person eats or exercises, and is reduced by the use of the shower() method.

Run Gavin's Life <- click to run the example code below (the code block being run appears after the Person constructor).

The Example Code

```
function Person(n,race){
        this.constructor.population++;

        // ************************************************************************
        // PRIVATE VARIABLES AND FUNCTIONS
        // ONLY PRIVELEGED METHODS MAY VIEW/EDIT/INVOKE
        // ************************************************************************
        var alive=true, age=1;
        var maxAge=70+Math.round(Math.random()*15)+Math.round(Math.random()*15);
        function makeOlder(){ return alive = (++age <= maxAge) }

        var myName=n?n:"John Doe";
        var weight=1;


        // ************************************************************************
        // PRIVILEGED METHODS
        // MAY BE INVOKED PUBLICLY AND MAY ACCESS PRIVATE ITEMS
        // MAY NOT BE CHANGED; MAY BE REPLACED WITH PUBLIC FLAVORS
        // ************************************************************************
        this.toString=this.getName=function(){ return myName }

        this.eat=function(){
                if (makeOlder()){
                        this.dirtFactor++;
                        return weight*=3;
                } else alert(myName+" can't eat, he's dead!");
        }
        this.exercise=function(){
                if (makeOlder()){
                        this.dirtFactor++;
                        return weight/=2;
                } else alert(myName+" can't exercise, he's dead!");
        }
        this.weigh=function(){ return weight }
        this.getRace=function(){ return race }
        this.getAge=function(){ return age }
        this.muchTimePasses=function(){ age+=50; this.dirtFactor=10; }


        // ************************************************************************
        // PUBLIC PROPERTIES -- ANYONE MAY READ/WRITE
        // ************************************************************************
        this.clothing="nothing/naked";
        this.dirtFactor=0;
}


// ************************************************************************
// PUBLIC METHODS -- ANYONE MAY READ/WRITE
// ************************************************************************
```

```
Person.prototype.beCool = function(){ this.clothing="khakis and black shirt" }
Person.prototype.shower = function(){ this.dirtFactor=2 }
Person.prototype.showLegs = function(){ alert(this+" has "+this.legs+" legs") }
Person.prototype.amputate = function(){ this.legs-- }


// ***********************************************************************
// PROTOTYOPE PROERTIES -- ANYONE MAY READ/WRITE (but may be overridden)
// ***********************************************************************
Person.prototype.legs=2;


// ***********************************************************************
// STATIC PROPERTIES -- ANYONE MAY READ/WRITE
// ***********************************************************************
Person.population = 0;



// Here is the code that uses the Person class
function RunGavinsLife(){
        var gk=new Person("Gavin","caucasian");       //New instance of the Person object created.
        var lk=new Person("Lisa","caucasian");        //New instance of the Person object created.
        alert("There are now "+Person.population+" people");

        gk.showLegs(); lk.showLegs();                 //Both share the common 'Person.prototype.legs' vari

        gk.race = "hispanic";                         //Sets a public variable, but does not overwrite pri
        alert(gk+"'s real race is "+gk.getRace());    //Returns 'caucasian' from private 'race' variable s
        gk.eat(); gk.eat(); gk.eat();                 //weight is 3...then 9...then 27
        alert(gk+" weighs "+gk.weigh()+" pounds and has a dirt factor of "+gk.dirtFactor);

        gk.exercise();                                //weight is now 13.5
        gk.beCool();                                  //clothing has been update to current fashionable le
        gk.clothing="Pimp Outfit";                    //clothing is a public variable that can be updated
        gk.shower();
        alert("Existing shower technology has gotten "+gk+" to a dirt factor of "+gk.dirtFactor);

        gk.muchTimePasses();                          //50 Years Pass
        Person.prototype.shower=function(){           //Shower technology improves for everyone
                this.dirtFactor=0;
        }
        gk.beCool=function(){                         //Gavin alone gets new fashion ideas
                this.clothing="tinfoil";
        };

        gk.beCool(); gk.shower();
        alert("Fashionable "+gk+" at "
                +gk.getAge()+" years old is now wearing "
                +gk.clothing+" with dirt factor "
                +gk.dirtFactor);

        gk.amputate();                                //Uses the prototype property and makes a public pro
        gk.showLegs(); lk.showLegs();                 //Lisa still has the prototype property

        gk.muchTimePasses();                          //50 Years Pass...Gavin is now over 100 years old.
        gk.eat();                                     //Complains about extreme age, death, and inability
}
```

Notes

- `maxAge` is a private variable with no privileged accessor method; as such, there is no way to publicly get or set it.
- `race` is a private variable defined only as an argument to the contructor. Variables passed into the constructor are available to the object as private variables.
- The 'tinfoil' `beCool()` fashion method was applied only to the `gk` object, not the entire `Person` class. Other people created and set to `beCool()` would still use the original 'khakis and black shirt' clothing that Gavin eschewed later in life.
- Note the implicit call to the `gk.toString()` method when using string concatenation. It is this which allows the code `alert(gk+' is so cool.')` to put the word 'Gavin' in there, and is equivalent to `alert(gk.toString()+' is so cool.')`. Every object of every type in JS has a `.toString()` method, but you can override it with your own.
- You cannot (to my knowledge) assign public methods of a class inside the main object constructor...you must use the `prototype` property externally, as above with the `beCool()` and `shower()` methods.
- As I attempted to show with the `Person.prototype.legs` property and the `amputate()` function, prototype properties are shared by all object instances. Asking for `lk.legs` yields '2' by looking at the single prototype property. However, attempting to *change* this value using either `gk.legs=1` or (in the Person object) `this.legs=1` ends up making a new public property of the object specific to that instance. (This is why calling `gk.amputate()` only removed a leg from Gavin, but not Lisa.) To modify a prototype property, you must use `Person.prototype.legs=1` or something like `this.constructor.prototype.legs=1`. (I say 'something like' because I discovered that `this.constructor` is not available inside private functions of the object, since `this` refers to the window object in that scope.)
- Wherever an anonymous function is declared inline with
  `foo = function(p1,p2){ some code }`
  the `new Function()` constructor is **NOT** equivalent, e.g.
  `foo = new Function('p1','p2','code');`
  since the latter runs in the global scope--instead of inheriting the scope of the constructor function--thus preventing it from accessing the private

variables.

- As noted above in the code comments, the act of setting `gk.race` to some value did NOT overwrite the private `race` variable. Although it would be a dumb idea, you can have both private and public variables with the same name. For example, the `yell()` method in the following class will yield different values for `foo` and `this.foo`:

```
function StupidClass(){
  var foo = "internal";
  this.foo = "external";
  this.yell=function(){ alert("Internal foo is "+foo+"\nExternal foo is "+this.foo) }
}
```

- Private functions and privileged methods, like private variables and public properties, are instantiated with each new object created. So each time `new Person()` is called, new copies of `makeOlder()`, `toString()`, `getName()`, `eat()`, `exercise()`, `weigh()`, `getRace()`, `getAge()`, and `muchTimePasses()` are created. For **every** Person, **each** time. Contrast this with public methods (only one copy of `beCool()` and `shower()` exist no matter how many Person objects are created) and you can see that for memory/performance reasons it can be preferable to give up some degree of object protection and instead use only public methods.

  Note that doing so requires making private variables public (since without privileged accessor methods there would be no way to use them) so the public methods can get at them...and which also allows external code to see/destroy these variables. The memory/performance optimization of using only public properties and methods has consequences which may make your code less robust.

  For example, in the above `age` and `maxAge` are private variables; `age` can only be accessed externally through `getAge()` (it cannot be set) and `maxAge` cannot be read or set externally. Changing those to be public properties would allow any code to do something like `gk.maxAge=1; gk.age=200;` which not only does it not make sense (you shouldn't be able to manipulate someone's age or lifespan directly), but by setting those values directly the `alive` variable wouldn't properly be updated, leaving your Person object in a broken state.