Javascript Concepts for OO-Programmers

Javascript is not Java

- **Javascript is just a name close to Java**
  - It is not a descendant of the C familly
  - It is more of a functional language (like Scheme, Self or Python).
  - Its syntax is unfortunately close to Java
- **It is close to Java**
  - For the syntax of loops (`while`, `for`) or conditional branching (`if`, `switch`).
- **It is far away from Java**
  - For object and classes design

Main Javascript Particularities

- **Variables are loosely typed (like PHP)**
  - Variables do not receive a type at programming,
  - The Values carry out the typing
  - `v=1;` implies that `v` contains an integer
  - `v="test";` implies that `v` contains a string
- **Code is interpreted locally**
  - The source code is transfered to the browser, read and interpreted
  - There is no bytecode generation (unlike java applets)
- **Javascript Functions are first class objects**
  - In Java everything is an *Object* (or a class)
  - In Javascript everything is a *Function*

Objects in Javascript

- **An Object in javascript is more like a "hash-table"**
  - It contains fields that can be added dynamically and initialized or removed programatically.
- **Creation of an Object:** Creates an empty container

  var myObject = new Object();
- **Can contain fields:**

```
myObject.shoeSize="42";
myObject['shoeSize']="39";
```


Add a function to the Object (not a class)

- **We can define a new function**

```
myObject.speakYourshoesSize=function(){
  alert("shoe size : "+this.shoeSize);
}
```

- **or use a predifined one**

```
function sayHello(){
  alert ('hello, my shoeSize is '+this.shoeSize);
}
...
myObject.sayHello=sayHello; // WITHOUT ↘
→PARENTHESIS !!!!
```


Complex Objects

- **We can attach objects inside other objects**

```
var myLibrary=new Object();
myLibrary.books=new Array();
myLibrary.books[0]=new Object();
myLibrary.books[0].title="Ajax in Action";
myLibrary.books[0].authors=new Array();
var dave=new Array();
dave.age=45;
dave.name="Dave Crane";
myLibrary.books[0].authors[0]=dave;
...
```


Use JSON

- **JSON=JavaScript Object Notation**
  - Standard notation is not easy to create large objects
- **Data in an Array indexed with numbers**
  list of objects enclosed in `[]`

```
myLibrary.books=[predefinedBook1, predefinedBook2,
                 predefinedBook3];
```

- **Construct a new JavaScript object**
  List of `key:value` pairs enclosed in {}

```
myLibrary.books={
   bestSeller : predefinedBook1,
   cookbook : predefinedBook2,
   spaceFiller : predefinedBook3
```

JSON (Cont.)

- **We can define more complicated objects by merging the syntaxes** (or use functions to fill the content),

```
var myLibrary={
    location : "my office",
    keywords : ["AJAX", "PHP", "JSP", "Servlets"],
    books : [
        { title : "Ajax in Action",
            authors : [
                { name : "Dave Crane", age=45 },
                { name : "Eric Pascarello", age="41" }
             ],
             publicationDate : new Date(2006,04,01)
        },
        { ... }
}
```


JSON (Cont.)

- **We can define and use member functions**

```
function giveDate(){
    return new Date(2005,10,5);
}
var harryPotter={
    title : "Harry Potter and the Half Blood Prince",
    authors : [ {name: "J. K. Rowling", age : 42}],
    publicationDate : giveDate();
    summerize : function(){
        var summary = this.title+" by "
            +this.authors[0].name
            +" was published in "+ this.publicationDate ;
        alert summary;
        }
    }

...
harryPotter.summerize();
```


JSON (Cont.)

- **We can mix JSON and Standard Javascript Notation**

```
var numbers={ one : 1 , two:2, three:3};
numbers.five=5;
```


JSON for contacting many servers

- **We can use JSON to contact another server.**
  - We add some code in the page requesting a javascript file
  - But this file is generated by a program
  - One parameter is the function to be executed on the return value
- **Example**

```
var str='<script language="JavaScript" type="text/javascript"↘
→';
str+=' src="myfile.php?var1=toto&var2=foo&loopback=↘
→myFunction"></script>
document.write(str);
```

JSON for contacting many servers

- **Example of a value generated by the PHP programm**

```
myFunction(
{
    location : "my office",
    keywords : ["AJAX", "PHP", "JSP", "Servlets"],
    books : [
        { title : "Ajax in Action",
            authors : [
                { name : "Dave Crane", age=45 },
                { name : "Eric Pascarello", age="41" }
             ],
             publicationDate : new Date(2006,04,01)
        }]});
```

This code is executed when the javascript is loaded.

Objects, Classes and Prototypes

- **Java is fully object-oriented**
  - Everything is an Object (`java.lang.Object`)
  - An object has a Type and a Class

    MyType myObject = new MyClass();
  - Classes can extend an existing Class and implement interfaces
- **Javascript has another type of Objects**
  - Each Object belong to one unique class
  - It has capabilities to link new member functions and member variables dynamically.
  - We can use Prototypes to instantiate objects with a default set of functions and variables (like a new in Java).

Constructor

- **We can create a new object**

  ```
  var myObj=new MyObject();
  ```
- **using a constructor**- which is not a class but rather a *"function"*

  ```
  function MyObject(name,size){
    this.name=name;
    this.size=size;
  }
  var myObj=new MyObject("laptop","35cm");
  alert("size of ''+myObj.name+" is "+myObj.size);
  ```


Constructor (Cont.)

- **We can also define a member function**

  ```
  function MyObject(name,size){
    this.name=name;
    this.size=size;
    this.tellSize=function(){
      alert("size of "+this.name+" is "+this.size);
    }
  }
  var myObj=new MyObject("laptop","35cm");
  myObj.tellSize();
  ```

Constructor (Cont.)

- **It works, but is problematic**
  - We create the same function for each instance of `MyObject`. Risks of Memory Leaks if the number of instances becomes large.
  - We created a *closing* which is harmless but can be dangerous if included in the DOM.
- **We use a better solution: The Prototype**

Prototype

- **Functions and properties can be tied to the prototype of a constructor**
  - Each time the constructor function is executed with a `new`,
  - the properties and functions of the prototype are attached to the new object.

```
function MyObject(name,size){
  this.name=name;
  this.size=size;
}
MyObject.prototype.tellSize=function(){
  alert("size of "+this.name+" is "+this.size);
}
var myObj=new MyObject("laptop","32cm");
myObj.tellSize();
```


Extending existing classes

- **In JavaScript, you can incorporate native objects in your programms**
  - They are written in C++ or Java
- **Let us consider the `Array` class**

```
Array.prototype.indexOf=function(obj){
  var result=-1;
  for (var i=0;i<this.length;i++){
    if(this[i]==obj){
      result=i;
      break;
    }
  }
  return result;
}
```


Extending existing classes (Cont.)

- **We can also add other methods**

```
Array.prototype.contains=function(obj){
  return (this.indexOf(obj)>=0);
}
Array.prototype.append=function(obj,nodup){
  if(!(nodup && this.contains(obj))){
    this[this.length]=obj;
  }
}
var numbers=[1,2,3,4,5];
var got8=numbers.contains(8);
numbers.append("cheese",true);
```

Reflexion

- **It is possible to discover the type and functionalities of unknown objects**
- **We can test if an object supports a given method or has a given property**

```
if(MyObject.someProperty){
...
}
```

  - It does not work it the value of `someProperty` is `false` (or simply `0` or `null`).

- **we can do this more properly (much like the php `isset` function)**

```
if(typeof(MyObject.someProperty) != "undefined"){
```


Reflexion (Cont.)

- **If we want to test the type of an object**

```
if(myObj instanceof Array){
  ...
} else if (myObj instanceof Object){
  ...
}
```

- **Or test our self-defined classes**

```
if(myObj instanceof MyObject){
  ...
}
```

- **Restrictions**
  - JSON can only create `Arrays` and `Objects`
  - Any `Array` is also an `Object` (take care to the order of the test)

Reflection (Cont.)

- **Iterator over the properties and functions of an object**

```
function MyObject(){
  this.color='red';
  this.flavor='strawberry';
  this.azimuth='45 degrees';
  this.favoriteDog='collie';
}
var myObj=new MyObject();
var debug="discovering ....\n";
for(var i in myObj){
  debug+=i+'' -> ''+myObj[i]+"\n";
}
alert(debug);
```


Encapsulation

- **It is not possible to extend classes or have interfaces. We have to use Prototypes instead.**
- **Encapsulation is also part of any OO-framework, that does not exist in JavaScript**
  - It can be "emulated" using Duck Typing:
    "If it walks like a duck and sing like a duck, then it is a duck"
  - It is heavy testing the input and you have to rely on the quality of your team
  - But it is the only thing we have

Function as First Class Objects

- **In Java:**
  - Functions belong to a class and/or an object and can not live without it
- **In JavaScript**
  - Functions are floating entities
  - They have an existance outside the objects (can be transfered for instance)

Attach functions to an Object

- **We can simply define a function**

  function doSomething(x,y,z){...}
- **Or using the syntax of the definition of a variable**

  var doSomething=**function**(x,y,z){ ...}
- **And we can attache this to an object**

  myObj.doSomethingNew=doSomething;
  myObj.doSomethingNew(x,y,z);

Calling a function from another object

- **The `call` function**
  - Start a method with another object

```
function Tree(name){ this.name = name;}
Tree.prototype.describe=function(){
  alert this.name;
}
function Dog(name){ this.name = name; }

myTree=newTree("Oak")
myDog=new Dog("Blacky");
var tmpFunc=myTree.describe;
tmpFunc.call(myDog);
```


The Function Objects

- **Each function is a `Function`**
  - It extends the `Object` class.
  - It can handle properties and contain functions itself.
- **A function can be executed using its `call()` method** (or its cousin `apply()`

```
function multiply(){ return this.y * this.x;}
myObject=new Object();
myObject.x=3;
myObject.y=4;
myOtherObject=new Object();
myOtherObject.x=5;
myOtherObject.y=4;
MyObject.operation=multiply;
var res=myObject.operation(); // returns 12
res=MyObject.operation.call(myOtherObject); // returns 20
```

Events Handling and Function Context in AJAX

- **We can define event handling inside HTML taging**

- **Or in the JavaScript programm**

  function clickHandler(){ alert(**this**.id);}
  myDiv.onclick=clickHandler;

- **Or with an anonymous function**

  myDiv.onclick=**function**(){ alert(**this**.id); }

Assign an Handler to a tag in JavaScript

- **In a javascript controler**

  function MyObj(id,div){
  this.id=id;
  this.div=div;
  this.div.onclick=**this**.clickHandler;
  }
  MyObj.prototype.clickHandler=**function**(event){
  alert(**this**.id);
  }

- **Problems**
  - The method does not return the `id` of the `MyObj`
  - It returns the `id` of the `div`

Browser implement the "Sandbox" Principle

- **The javascript originating from one server can only connect this server**
- **AJAX program can not be used to merge many source of information**
- **It makes sens: harder to write a Distributed Deny of Service**

Conclusion

- **JavaScript is not Java ;-)**
- **You can easyly make a mess**
- **Work very properly to prevent massive errors.**

References

- **Ajax in Action**, Crane et al.