Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Object Oriented JavaScript

Bangalore
19 October 2016
Thennarasan Shanmugam

Agenda

- Object Notations
- Object Properties
  - Data Properties
  - Accessor Properties
  - Defining Multiple Properties
  - Reading Property Attributes
- Object Creation
  - Factory Pattern
  - Constructor Pattern
- Prototypes
  - Prototype Pattern
  - How Prototypes Work
  - Combination Constructor Prototype Pattern

Object Notations

Object Notations

- Concept of "Class" not available in JavaScript (up to ES5)
- Objects are little different in JavaScript than Class based Object Oriented Languages such as C#, Java and etc.
- defn of Object:
  - *unordered collection of properties each of which contains a primitive value, object or function*
  - *it is like a grouping of name-value pairs where the value may be a data or a function*
  - *each property or method is identified by a name that is mapped to a value like a HashTable*

Object Notations (Contd.)

- Types of Object Notation
  - Object Constructor
    - old way of defining objects
  - Object Literal
    - preferred pattern and more concise way of defining objects

Object Notations (Contd.)

- Object Constructor

```
// Creating Object - Object Constructor

var sweet = new Object();

sweet.name = "Gulab Jamun";
sweet.price = 100;
sweet.expiry_days = 7;

sweet.eat = function(){
    document.writeln(this.name + " is Delicious !<BR/><BR/>");
}

sweet.eat();

document.writeln("Sweet Name : " + sweet.name + "<BR/><BR/>");
```

- Output

Gulab Jamun is Delicious !

Sweet Name : Gulab Jamun

Object Notations (Contd.)

- Object Literal

```
// Creating Object - Object Literal

var fruit = {
    name: "Apple",
    price: 200,
    expiry_days: 15,
    eat : function() {
        document.writeln("I Love " + this.name + "<BR/><BR/>");
    }
};

fruit.eat();

document.writeln("Fruit Name : " + fruit.name + "<BR/><BR/>");
```

- Output

I Love Apple

Fruit Name : Apple

Object Properties

Object Properties

- characteristics of properties can be controlled through the use of internal-only attributes
- internal-only attributes are not directly accessible in JavaScript
- To indicate that an attribute is internal, the attribute name is surrounded by a pair of []
  - Ex: [[Writable]]

Object Properties (Contd.)

- Data Properties
  - [[Configurable]]
    - indicates if the property may be redefined by removing the property via delete, changing the property's attributes, or changing the property into an accessor property
  - [[Enumerable]]
    - indicates if the property will be returned in a for-in loop
  - [[Writable]]
    - indicates if the property's value can be changed
  - [[Value]]
    - contains the actual data value for the property

Object Properties (Contd.)

- Data Properties
  - Object.defineProperty(object, "property_name", descriptorObject);
    - By default, data properties except Value are set to false

```
//Types of Properties - Data Properties

        var software = {};

        Object.defineProperty(software, "name", {
            configurable: false,
            writable: true,
            value: "SQUISH"
        });

        document.writeln(software.name + "<BR/><BR/>");

        delete software.name;

        document.writeln(software.name+ "<BR/><BR/>");

        software.name = "squeeze";

        document.writeln("Writable Property to True : " + software.name + "<BR/><BR/>");

        document.writeln("Going to set Writable property to False " + "<BR/><BR/>");

        Object.defineProperty(software, "name", {
            writable: false
        });

        software.name = "Squash";

        document.writeln("New software name is 'Squash', Result is : " + software.name+ "
        <BR/><BR/>");
```

Output

SQUISH

SQUISH

Writable Property to True : squeeze

Going to set Writable property to False

New software name is 'Squash', Result is : squeeze

Object Properties (Contd.)

- Accessor Properties
  - [[Configurable]]
    - indicates if the property may be redefined by removing the property via delete, changing the property's attributes, or changing the property into a data property
  - [[Enumerable]]
    - indicates if the property will be returned in a for-in loop
  - [[Get]]
    - function to call when the property is read from, default value is undefined
  - [[Set]]
    - function to call when the property is written from, default value is undefined

Object Properties (Contd.)

- Accessor Properties

```
//Types of Properties - Accessor Properties
        var book = {
            _year: 2014,
            edition: 1
        };

        Object.defineProperty(book, "year", {
            get: function(){
                return this._year;
            },
            set: function(newValue){

                if (newValue > 2014) {
                    this._year = newValue;
                    this.edition = newValue - 2014;
                    this.edition++;

                }
            }
        });

        //Getting Value
        document.writeln("Publication Year : " + book.year +"<BR/><BR/>");
        document.writeln("Edition : " + book.edition +"<BR/><BR/>");

        //Setting Value
        book.year = 2015;
```

Output

Publication Year : 2014

Edition : 1

Publication Year : 2015

Edition : 2

Publication Year : 2016

Edition : 3

Publication Year : 2017

Edition : 4

Object Properties (Contd.)

- Defining Multiple Properties
  - Object.defineProperties(object, descriptorObject);

```
// Defining Multiple Properties

    var book = {};

    Object.defineProperties(book, {
        _year: {
            writable: true,
            value: 2014
        },
        edition: {
            writable: true,
            value: 1

        },

        year: {
            get: function(){
                return this._year;
            },

            set: function(newValue){
                if (newValue > 2014) {
                    this._year = newValue;
                    this.edition = newValue - 2014;
                    this.edition++;
                }
            }
        }
    });
```

Output

```
Publication Year : 2014

Edition : 1

Publication Year : 2015

Edition : 2

Publication Year : 2016

Edition : 3

Publication Year : 2017

Edition : 4
```

Object Properties (Contd.)

- Reading Property Attributes
  - Object.getOwnPropertyDescriptor(object, property_name);

```
var descriptor = Object.getOwnPropertyDescriptor(book, "_year");

document.writeln(descriptor.value + "<BR/><BR/>");          //2016
document.writeln(descriptor.configurable + "<BR/><BR/>");   //false
document.writeln(typeof descriptor.get + "<BR/><BR/>");     //"undefined"

var descriptor = Object.getOwnPropertyDescriptor(book, "year");
document.writeln(descriptor.value+ "<BR/><BR/>");          //undefined
document.writeln(descriptor.enumerable+ "<BR/><BR/>");     //false
document.writeln(typeof descriptor.get+ "<BR/><BR/>");     //"function"
```

Output

2016

false

undefined

undefined

false

function

Object Creation

Object Creation

- Factory Pattern
  - well-known design pattern used in software engineering to abstract away the process of creating objects
  - functions are created to encapsulate the creation of objects with specific interfaces
  - solves the problem of creating multiple similar objects
- Disadvantage
  - didn't address the issue of object identification

Object Creation (Contd.)

- Factory Pattern

```
//Factory Pattern

    function createSweet(name, price, expiry_days){
        var o = new Object();
        o.name = name;
        o.price = price;
        o.expiry_days = expiry_days;
        o.eat = function(){
            document.writeln(this.name + " is delicious :) :)<BR/><BR/>");
        };
        return o;
    }

    var sweet1 = createSweet("Laddu", 35, 7);
    var sweet2 = createSweet("Halwa", 50, 3);
    var sweet3 = createSweet("Rasagulla", 60, 4);

    sweet1.eat();
    sweet2.eat();
    sweet3.eat();
```

Output

Laddu is delicious :) :)

Halwa is delicious :) :)

Rasagulla is delicious :) :)

Object Creation (Contd.)

- Constructor Pattern
  - Creates a New Object
  - Assign the "this" value of the constructor to the new object (so this sets the context to the new object)
  - Execute the code inside the constructor (adds properties to the new object)
  - Returns the new object
- Advantages
  - No Object being created explicitly
  - Properties and Methods are assigned directly onto the "this" object
  - No return statement

Object Creation (Contd.)

- Constructor Pattern

```
// Constructor Pattern

    function Sweet(name, price, expiry_days){
        this.name = name;
        this.price = price;
        this.expiry_days = expiry_days;
        this.eat = function(){
            document.writeln(this.name + " is Delicious! <BR/><BR/>");
        };

    }

    var sweet1 = new Sweet("Laddu", 35, 7);
    var sweet2 = new Sweet("Halwa", 50, 3);

    sweet1.eat();
    sweet2.eat();
```

Output

Laddu is Delicious!

Halwa is Delicious!

Object Creation (Contd.)

- Constructor as Functions

```
// Constructor as Functions

    function Fruit(name, price, expiry_days){
        this.name = name;
        this.price = price;
        this.expiry_days = expiry_days;
        this.eat = function(){
            document.writeln("I am eating "+this.name + "<BR/><BR/>");
        };

        // this.eat = new Function(document.writeln("I am eating "+ this.name + "<BR/>
        <BR/>"));
        // logical equivalent
    }
    // use as a constructor
    var fruit1 = new Fruit("Apple", 120, 5);
    fruit1.eat();
    // call as a function
    Fruit("Orange", 100, 7);
    window.eat();
    // call in the scope of another object
    var fruit2 = new Object();
    Fruit.call(fruit2, "Pomegranate", 160, 6);
    fruit2.eat();
```

Output

I am eating Apple

I am eating Orange

I am eating Pomegranate

Object Creation (Contd.)

- Problem with Constructors
  - downside of constructor's is that methods are created once for each instance
  - hence, functions of same name on different instances are not equivalent
  - it doesn't make sense to have two instances of Function that do the same thing

```
this.eat = function(){
    document.writeln("I am eating "+this.name + "<BR/><BR/>");
};
// logical equivalent
// this.eat = new Function(document.writeln("I am eating "+ this.name + "<BR/>
<BR/>"));
```

Object Creation (Contd.)

- Problem with Constructors

```
var kiwi = new Fruit("Kiwi", 120, 5);
var strawberry = new Fruit("Strawberry", 100, 2);

kiwi.eat();
strawberry.eat();
```

```
document.writeln("kiwi.eat and strawberry.eat refer same function : ");
document.writeln(kiwi.eat == strawberry.eat);  //false
document.writeln("<BR/><BR/>");
```

Output

kiwi.eat and strawberry.eat refer same function : false

Object Creation (Contd.)

- Problem with Constructors – Solution!
  - to resolve the duplicate functions, define the function outside the constructor
  - now eat property contains just a pointer to the global eat() function
  - hence, all instances of Fruit end up sharing the same eat() function

```
// Problem with Constructors - Solution

    function Fruit(name, price, expiry_days){
        this.name = name;
        this.price = price;
        this.expiry_days = expiry_days;
        this.eat = eat;
    }

    function eat(){
         document.writeln("I am eating "+this.name + "<BR/><BR/>");
    }

    var kiwi = new Fruit("Kiwi", 120, 5);
    var strawberry = new Fruit("Strawberry", 100, 2);

    kiwi.eat();
    strawberry.eat();
```

Object Creation (Contd.)

- Problem with Constructors – Solution!

```
document.writeln("kiwi.constructor : Fruit");
document.writeln(kiwi.constructor == Fruit);  //true
document.writeln("<BR/><BR/>");

document.writeln("strawberry.constructor : Fruit");
document.writeln(strawberry.constructor == Fruit);  //true
document.writeln("<BR/><BR/>");

document.writeln("kiwi.eat and strawberry.eat refer same function : ");
document.writeln(kiwi.eat == strawberry.eat);  //false
document.writeln("<BR/><BR/>");
```

Output

I am eating Kiwi

I am eating Strawberry

kiwi.constructor : Fruit true

strawberry.constructor : Fruit true

kiwi.eat and strawberry.eat refer same function : true

Object Creation (Contd.)

- Prototype Pattern

  - even though constructor pattern resolves the duplicate function referencing issue, it creates some clutter in the global scope by introducing a function that can realistically be used in relation to an object

  - if an object needed multiple methods, that would mean multiple global functions, all of a sudden custom reference type is no longer nicely grouped in the code.

  - these problems are addressed using the prototype pattern

  - each function is created with a prototype property which is an object containing properties and methods that should be available to instances of a particular reference type

Object Creation (Contd.)

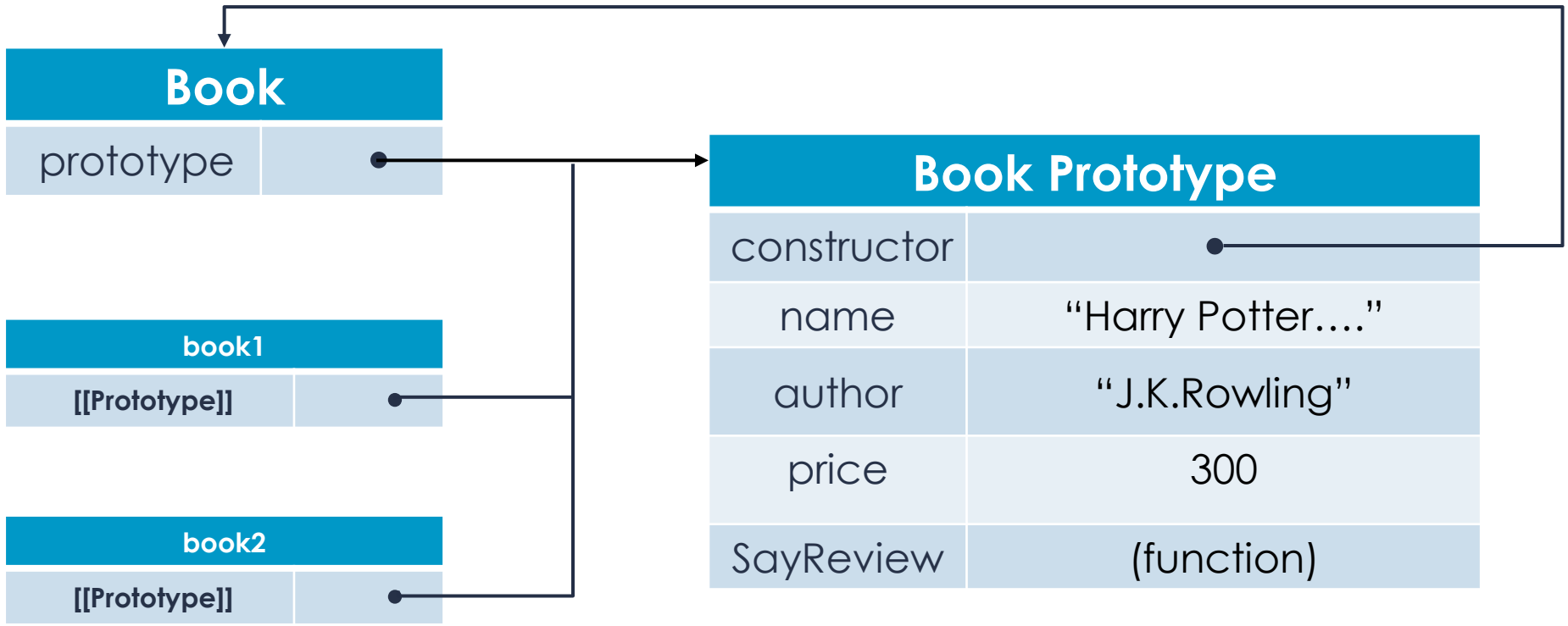
- Prototype Pattern
  - benefit of using the prototype is all of its properties and methods are shared among object instances
  - instead of assigning object information in the constructor, they can be assigned directly to the prototype as below:

```
// Prototype Pattern

    function Book(){
    }

    Book.prototype.name ="Harry Potter and The Sorcerer's Stone";
    Book.prototype.author ="J.K.Rowling";
    Book.prototype.price = 300;
    Book.prototype.sayReview = function(){
        document.writeln("The book " + this.name + " written by " + this.author + " is
        good <BR/><BR/>");
    }

    var book1 = new Book();
    book1.sayReview();

    var book2 = new Book();
    book2.sayReview();

    document.writeln("book1.sayReview and book2.sayReview refer the same function :
    ");
```

Prototypes in JavaScript

Prototypes in JavaScript

- How Prototypes Work



Prototypes in JavaScript (Contd.)

- Object.prototype
- isPrototypeOf()
- Object.getPrototypeOf()
- Object.__proto__

Prototypes in JavaScript (Contd.)

- Object.prototype

```
document.writeln("The Result of Book.prototype is : ")
document.writeln(Book.prototype);
document.writeln("<BR/><BR/>");
```

The Result of Book.prototype is : [object Object]

- isPrototypeOf()

```
document.writeln("The Result of Book.prototype.isPrototypeOf(book1) is : ");
document.writeln(Book.prototype.isPrototypeOf(book1));
document.writeln("<BR/><BR/>");

document.writeln("The Result of Book.prototype.isPrototypeOf(book2) is : ");
document.writeln(Book.prototype.isPrototypeOf(book2));
document.writeln("<BR/><BR/>");
```

The Result of Book.prototype.isPrototypeOf(book1) is : true

The Result of Book.prototype.isPrototypeOf(book2) is : true

Prototypes in JavaScript (Contd.)

- Object.getPrototypeOf()

```
document.writeln("The Result of Object.getPrototypeOf(book1) is : ")
document.writeln(Object.getPrototypeOf(book1));
document.writeln("<BR/><BR/>");
```

The Result of Object.getPrototypeOf(book1) is : [object Object]

```
document.writeln("The Result of Object.getPrototypeOf(book1).name is : ");
document.writeln(Object.getPrototypeOf(book1).name);  //true
document.writeln("<BR/><BR/>");

document.writeln("The Result of Object.getPrototypeOf(book1).sayReview() is :
");
document.writeln(Object.getPrototypeOf(book1).sayReview());
document.writeln("<BR/><BR/>");
```

The Result of Object.getPrototypeOf(book1).name is : Harry Potter and The Sorcerer's Stone

The Result of Object.getPrototypeOf(book1).sayReview() is : The book Harry Potter and The Sorcerer's Stone written by J.K.Rowling is good

Prototypes in JavaScript (Contd.)

- Object.__proto__

```
document.writeln("The Result of book1.__proto__ == Book.prototype is : ");
document.writeln(book1.__proto__ == Book.prototype);  //true
document.writeln("<BR/><BR/>");
```

The Result of book1.__proto__ == Book.prototype is : true

```
document.writeln("The Result of Object.getPrototypeOf(book1).sayReview() is :
");
document.writeln(Object.getPrototypeOf(book1).sayReview());
document.writeln("<BR/><BR/>");

document.writeln("The Result of book2.__proto__.sayReview() is : ");
document.writeln(book2.__proto__.sayReview());
document.writeln("<BR/><BR/>");
```

The Result of Object.getPrototypeOf(book1).sayReview() is : The book Harry Potter and The Sorcerer's Stone written by J.K.Rowling is good

The Result of book2.__proto__.sayReview() is : The book Harry Potter and The Sorcerer's Stone written by J.K.Rowling is good

Prototypes in JavaScript (Contd.)

- Shadowing prototype Property

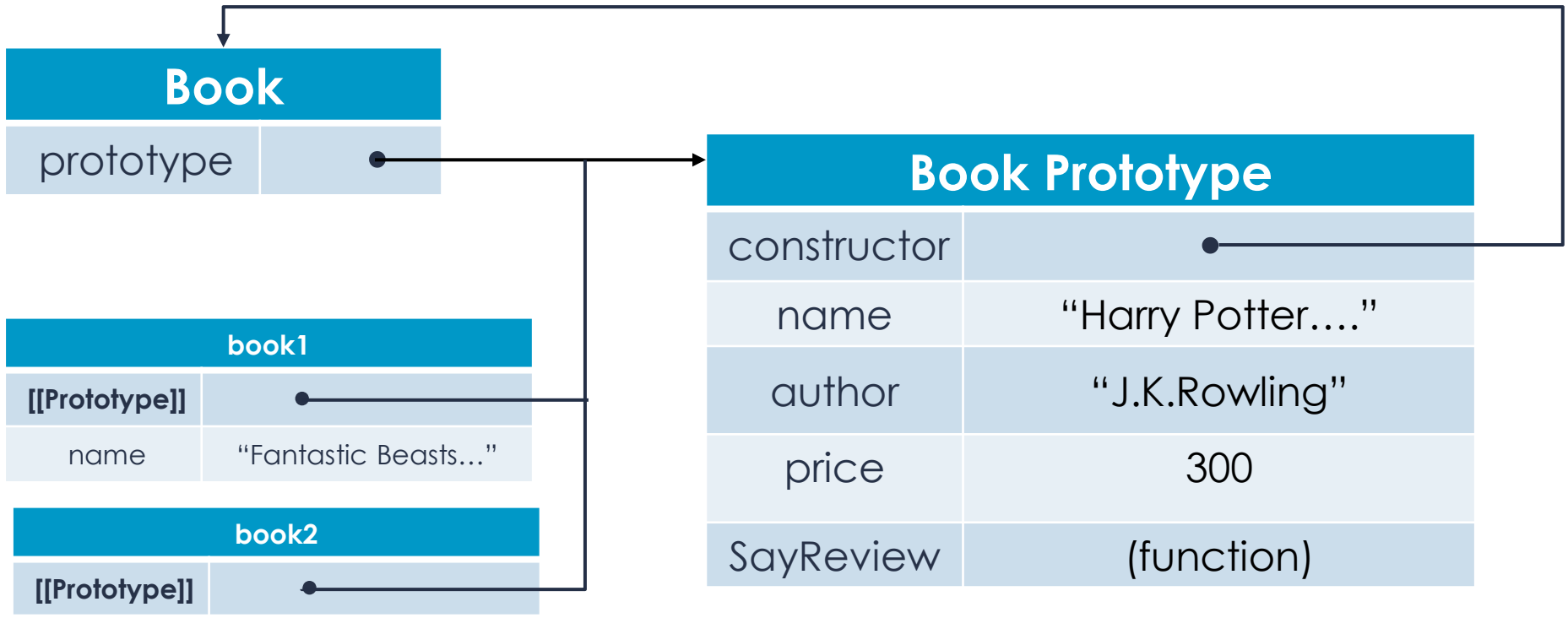
```
// Shadowing Prototype Properties with Instance Properties

    function Book(){
    }

    Book.prototype.name ="Harry Potter and The Sorcerer's Stone";
    Book.prototype.author ="J.K.Rowling";
    Book.prototype.price = 300;
    Book.prototype.sayReview = function(){
        document.writeln("The book " + this.name + " written by " + this.author + " is
        good <BR/><BR/>");
    }

    var book1 = new Book();
    var book2 = new Book();

    book1.name = "Fantastic Beasts and Where to Find them";

    document.writeln('Book1 " ' +book1.name + ' " written by ' + book1.author + "<BR/>
    <BR/>"); // - from instance
    document.writeln('Book2 " ' +book2.name + ' " written by ' + book2.author + "<BR/>
    <BR/>");  // - from prototype
```

Book1 " Fantastic Beasts and Where to Find them " written by J.K.Rowling

Book2 " Harry Potter and The Sorcerer's Stone " written by J.K.Rowling

Prototypes in JavaScript

- Shadowing prototype Property with Instance Property



Prototypes in JavaScript (Contd.)

- Shadowing prototype Property

```
delete book1.name;
document.writeln("book1.name instance property is deleted now!<BR/><BR/>");

document.writeln('Book1 " ' +book1.name + ' " written by ' + book1.author + "<BR/>
<BR/>"); // - from prototype
document.writeln('Book2 " ' +book2.name + ' " written by ' + book2.author + "<BR/>
<BR/>");  // - from prototype
```

book1.name instance property is deleted now!

Book1 " Harry Potter and The Sorcerer's Stone " written by J.K.Rowling

Book2 " Harry Potter and The Sorcerer's Stone " written by J.K.Rowling