NEW CLASS SYNTAX IN JAVASCRIPT

By Brian, Megan, and Nic

Our paper and presentation slides are on github

- **https://github.com/Meglet/CSCI3155Paper**

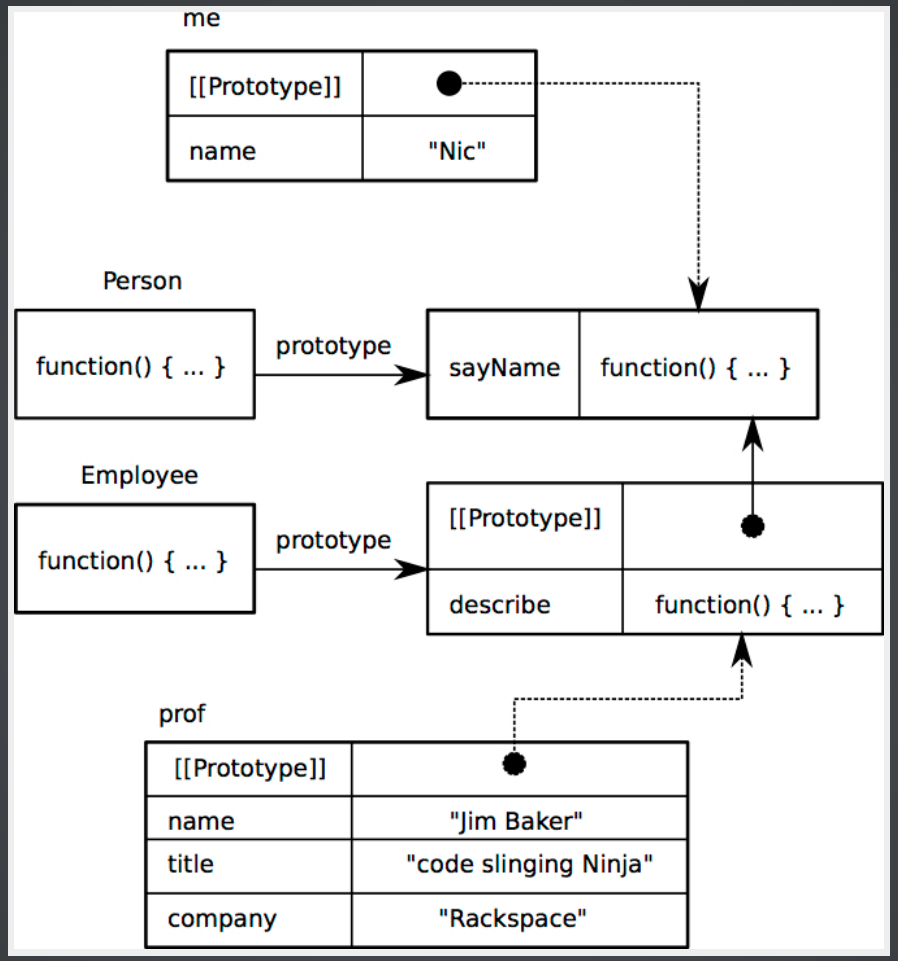
OUTLINE

1. Background on our topic
2. Old vs. New Syntax
3. Pros and Cons

THE JAVASCRIPT STANDARD

1. JS derived from the ECMAScript specifiaction
   http://www.ecma-international.org/ecma-262/5.1/
2. TC39: group in charge of writing ECMAScript6
   http://www.ecma-international.org/memento/TC39.htm
3. The new spec will be out in late 2013

PROTOTYPE CHAIN DIAGRAM



CONSTRUCTOR FUNCTIONS

```
> function Person(name) {
        this.name = name;
}
> Person.hasOwnProperty('name') // => true
> Person.prototype              // => {}
> Person.prototype.sayName = function() {
    return "Hi I am a Person and my name is: " + this.name;
  };
> Person.prototype              // => { sayName: [Function] }
> function Employee(name,title,company) {
        Person.call(this,name);
        //Person.apply(this,['name']);
        this.title = title;
        this.company = company;
}
```

PROTOTYPE INHERITANCE

```
> function Employee(name,title,company) {
        Person.call(this,name);
        //Person.apply(this,['name']);
        this.title = title;
        this.company = company;
}
> Employee.prototype = new Person('John Doe','TBD','Unemployed');
> Employee.prototype.constructor = Employee;
> Employee.prototype.describe = function() {
        return Person.prototype.sayName.call(this,this.name)
               + ", I am a " + this.title + " at "
               + this.company + "!";
}
```

NOW LET'S MAKE SOME OBJECTS

```
> var me = new Person('Nic');
> var prof = new Employee('Jim Baker','code slinging Ninja','Rackspace');
> console.log(me.sayName());
  Hi I am a Person and my name is: Nic
> console.log(prof.sayName());
  Hi I am a Person and my name is: Jim Baker
> console.log(prof.describe());
  Hi I am a Person and my name is: Jim Baker,
  I am a code slinging Ninja at Rackspace!
```

MAXIMALLY MINIMAL CLASSES

- A Class keyword
- A body that includes the constructor function and instance methods
- Cand declare a class as a subclass of another class using extends
- Super is available from any of the methods or the constructor

CLASS BNF

```
ClassDeclaration: class BindingIdentifier ClassTail
ClassExpression: class BindingIdentifieropt ClassTail
ClassTail: ClassHeritageopt{ ClassBodyopt}
ClassHeritage : extends AssignmentExpression
ClassBody: ClassElementList
ClassElementList:
    ClassElement
    ClassElementList ClassElement
ClassElement:
    MethodDefinition;
MethodDefinition :
    PropertyName(FormalParameterList) {FunctionBody}
    *PropertyName(FormalParameterList){FunctionBody}
    get PropertyName ( ){FunctionBody}
    set PropertyName(PropertySetParameterList) {FunctionBody}
```

PARSING OF JUDGEMENT FORMS

```
Static Semantics: ConstructorMethod
ClassBody : ClassElementList
  1. Let list be PrototypeMethodDefinitions of ClassElementList.
  2. For each MethodDefinition m in list, do
    a. If PropName of m is "constructor", return m.
  3. Return empty.

Static Semantics: Contains
ClassTail : ClassHeritageopt { ClassBody }
1. If symbol is ClassBody, return true.
2. If ClassHeritage is not present, return false.
3. If symbol is ClassHeritage, return true.
4. Return the result of Contains for ClassHeritage
   with argument symbol.
```

FROM PROTOTYPE TO CLASSES

```
// Supertype
    class Person {
        constructor(name) { this.name = name; }
        describe() { return "Person called "+this.name; }
    }
// Subtype
    class Employee extends Person {
        constructor(name, title) {
            super.constructor(name);
            this.title = title;
        }
        describe() { return super.describe() + this.title; }
    }
```

HOW CLASSES WILL BE USED

```
> let prof = new Employee("Jim Baker", "code slinging Ninja");
> prof instanceof Person
  true
> prof instanceof Employee
  true
> prof.describe()
  Hi I am a Person and my name is: Jim Baker,
  I am a code slinging Ninja!
```

MAIN ARGUMENTS AGAINST

1. The new syntax will add complexity
2. It is just sugar
3. Minimal classes are redundant

CLASSES MAY ADD COMPLEXITY

- Adding common features of classes (inheritance, subclasses, etc) will obscure the inner workings of JS

PROTOTYPE CLASSES WORK

You can always write your own JS function to extend CF

```
function extend(Child, Parent) {
     var F = function(){};
     F.prototype = Parent.prototype;
     Child.prototype = new F();
     Child.prototype.constructor = Child;
     Child.uber = Parent.prototype;
}
```

or use Proto.js

https://github.com/rauschma/proto-js

WHAT'S THE POINT OF MINIMAL CLASSES

- There will be TWO ways of creating classes and that may lead to errors later on

THE NEED FOR CLASSES IN JS

“ When they (newcomers) arrive to JavaScript their concepts of how an object-oriented language works no longer apply. They learn that their big OOP investment was not complete. -Peter Michaux ”

PROFESSIONAL LEARNING RESOURCES

Amazon.com results that refer to:

1. "prototype inheritance javascript": `N(14)`
2. "class inheritance": `N(3001)`

CONCLUSION

- Its happening!
- Its just syntactic sugar for prototypical classes

THANK YOU FOR YOUR TIME