

# Bachelor Project

## Scripting Objects for the Web

Hasso-Plattner-Institut Potsdam

Software Architecture Group

Prof. Dr. Robert Hirschfeld

BP2010H1: Stephan Eckardt, Anton Gulenko, Marcus Hoffmann, Hauke Klement, Robert Strobl, Lauritz Thamsen, Lars Wassermann, Sebastian Woinar

<http://www.hpi.uni-potsdam.de/swa/>

2011/18/01

# SqueakyJS

- Comparison between Squeak and JavaScript
- SqueakyJS class model
- Method name exceptions
- Inheritance and access to super methods
- Metaprogramming
- Block closures and non-local return
- Access to super methods

# Comparison between Squeak and JS

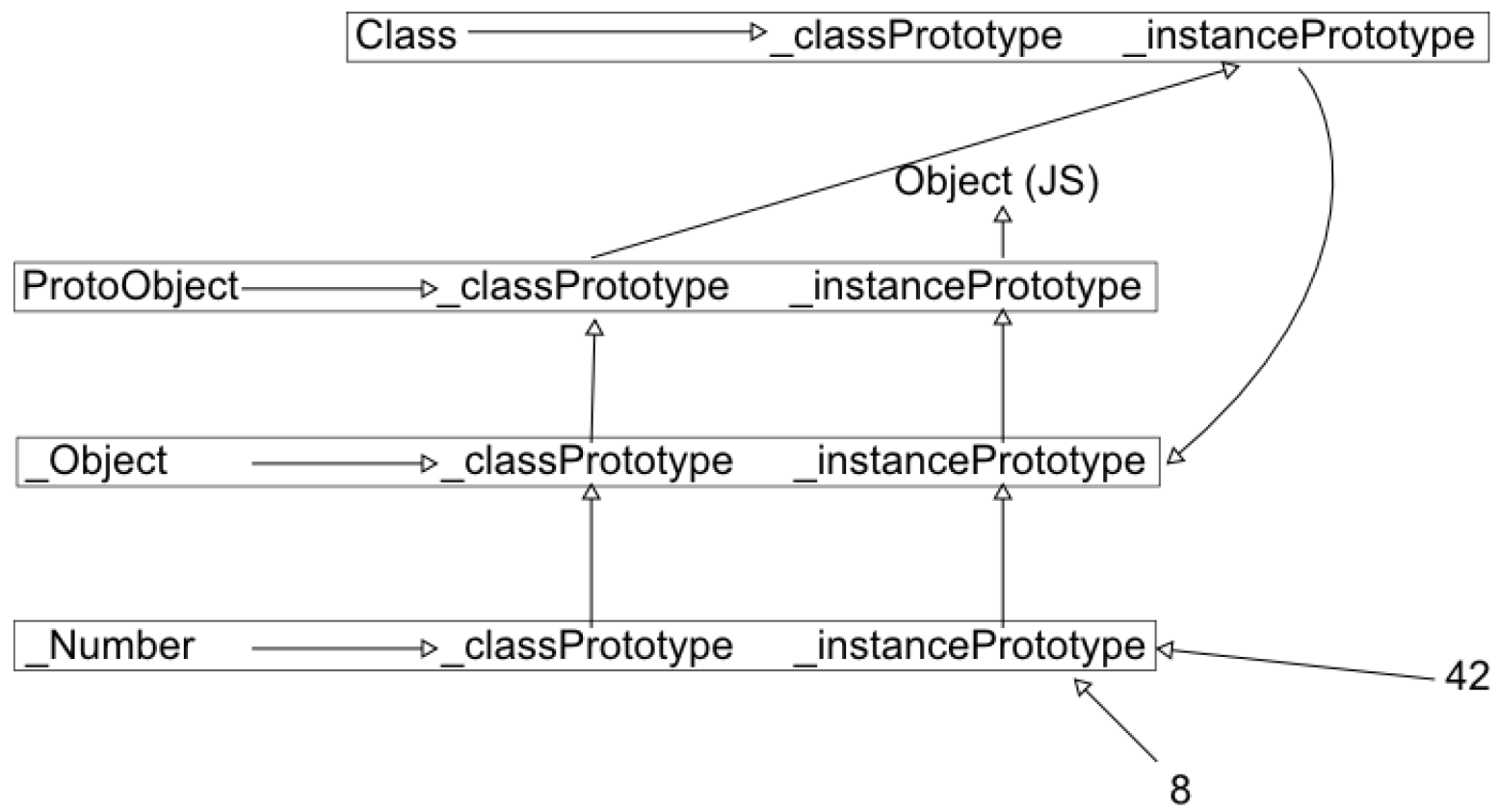
<b>Squeak</b>	<b>JavaScript</b>
Class driven, class methods	Class free
Single inheritance from superclass	Prototype inheritance
Access superclass by using super	Apply can call functions in another context
Block closures with non-local return	No non-local return

# SqueakyJS class model

```
// class example
Class('Person', {
  instanceVariables: ['name'],
  instanceMethods: {
    setName: function(aName) {
      this.name = aName;
    },
    getName: function() {
      return this.name;
    },
    makeNoise: function() {
      return this.getName() + " says: ";
    }
  },
});

// create an instance
aPerson = Person._newInstance();
```

# SqueakyJS class model



# Method name exceptions (1)

Exact mapping from Squeak to JS methods not possible  
-> we need to escape methods

**Goal: avoid name conflicts!!**

# Method name exceptions (2)

## Issue:

special characters can not be used as method names in JavaScript (e.g. +,\*)

## Solution:

provide dictionary for compiler, convert to **\_plus**, **\_times** – leading underscore assures, that there is no conflict with a possible **plus** or **times** Squeak method

# Method name exceptions (3)

## Issue:

accessor and variables have the same name, but  
accessor must be bound to *this* (*self*) -> conflict

## Solution:

all instance variables have a leading \$ which is not a  
valid character in Squeak -> avoid conflicts



# Method name exceptions (4)

## Issue:

getters and setters only differ in trailing colon

e.g. `foobar`, `foobar:` -> colon not valid in JavaScript

## Solution:

setters have trailing underscores (`foobar_`)

# Method name exceptions (5)

## Issue:

multiple method parameters in Squeak

Squeak:

```
OrderedCollection>>at: anInteger put: anObject
```

## Solution:

concatenation of method name partials

JavaScript:

```
at_put_(anInteger, anObject)
```

# Method name exceptions (6)

## Issue:

reserved keywords in JavaScript, (*new*, *class*)

## Solution:

escape method names with leading underscore  
(*\_new*, *\_class*)

# Inheritance and access to super methods

```
Class('Pirate', {  
  // inherits from Person  
  superclass: Person,  
  instanceMethods: {  
    makeNoise: function() {  
      return this.__super.makeNoise() +  
             this.__class.noise();  
    }  
  },  
  classMethods: {  
    noise: function() {  
      return 'Arrrrrrr!!!';  
    }  
  }  
});  
  
aPirate = Pirate._newInstance();
```

# Metaprogramming

```
Pirate.__addClassMethods({  
    printFoobar: function() {  
        alert("foobar");  
    }  
});  
  
// prints foobar  
aPirate.__class.printFoobar();
```

# Block closures and non-local return

```
aPirate._addClassMethods({
  evaluateBlock: function(aBlock) {
    aBlock.value("yiii").value("haaaa");
  },
  // overwrite noise method
  noise: function() {
    aBlock = block(function(a) {
      return block(function(b) {
        nonLocalReturn(a+b);
      });
    });

    this.evaluateBlock(aBlock);

    // should never be called
    return 'Arrrrrrr!!!';
  });
});
```

# Access to super methods (1)

**Optimal:** super as global object

```
super.foo(a, b);
```

**Issue:**

no object context!

**Next idea:** super as global function

```
super().foo(a, b);
```

**Issue:**

could get callers context, but can't provide method slot for this context

## Access to super methods (2)

**Current solution:** bind super slot to object context

```
this._super.foobar(a, b);
```

**Benefit:** comfortable use for programmer

**Issue:**

methods have to be copied into the slot on class creation -> no prototypical inheritance

**Alternative:** super as function with method parameter

```
this._super(' f oobar ' ) (a, b);
```

**Benefit:** use of prototypical inheritance

**Issue:** uncomfortable for the JavaScript programmer



# Access to super methods (3)

What would you prefer?

Comfortability  
vs.  
Implementation