JavaScript
Advanced Course
Part 1

jan.schulz@devugees.org

Agenda

1. Let, Var and Const
2. Default Arguments
3. Unary Operators
4. Primitives and Objects
5. Object Oriented Paradigm
6. Contructors and Instances
7. Inheritance
8. Prototype
9. Prototype-Chain
10. Call and Apply
11. Inheritance
12. Multi-Level Inheritance
13. Class Keyword

1. Let, Var and Const

- All of them do the same: create a placeholder for a value
- Difference between var -> let, const
  - Var is valid inside the function scope
  - Let, const is valid inside the block scope
- Difference between const -> let, var
  - Const does not allow to reassign the placeholder

2. Default Arguments

```
function multiply(a, b=1) {
    return a*b;
}

console.log(multiply(5, 2));
console.log(multiply(5));
```

3. Unary Operators

Operator	Explanation
+	Tries to convert operand into a number
-	Tries to convert the operand into a number and negates after
!	Converts to boolean value and negates it
++	Adds one to its operand
typeof	Returns a string which is the type of the operand
delete	Deletes specific index of an array or specific property of an object

4. Primitives and Objects

PRIMITIVES

Numbers
Strings
Booleans
Undefined
Null

OBJECTS

Arrays
Functions
Objects
Dates
Wrappers for Numbers, Strings, Booleans

... IS AN OBJECT

5. Object Oriented Paradigm

OOP (Object Oriented Programming)

- Object interacting with one another through methods and properties
- Used to store data, structure applications into modules and keeping clode clean

6. Constructors and Instances

```
var john = {
                Name: ‘John‘,
                yearOfBirth: 1990,
                isMarried: false
}
```

```
var jane = {
                Name: ‘Jane‘,
                yearOfBirth: 1991,
                isMarried: true
}
```

```
var mark = {
                Name: ‘Mark‘,
                yearOfBirth: 1948,
                isMarried: true
}
```

6. Constructors and Instances

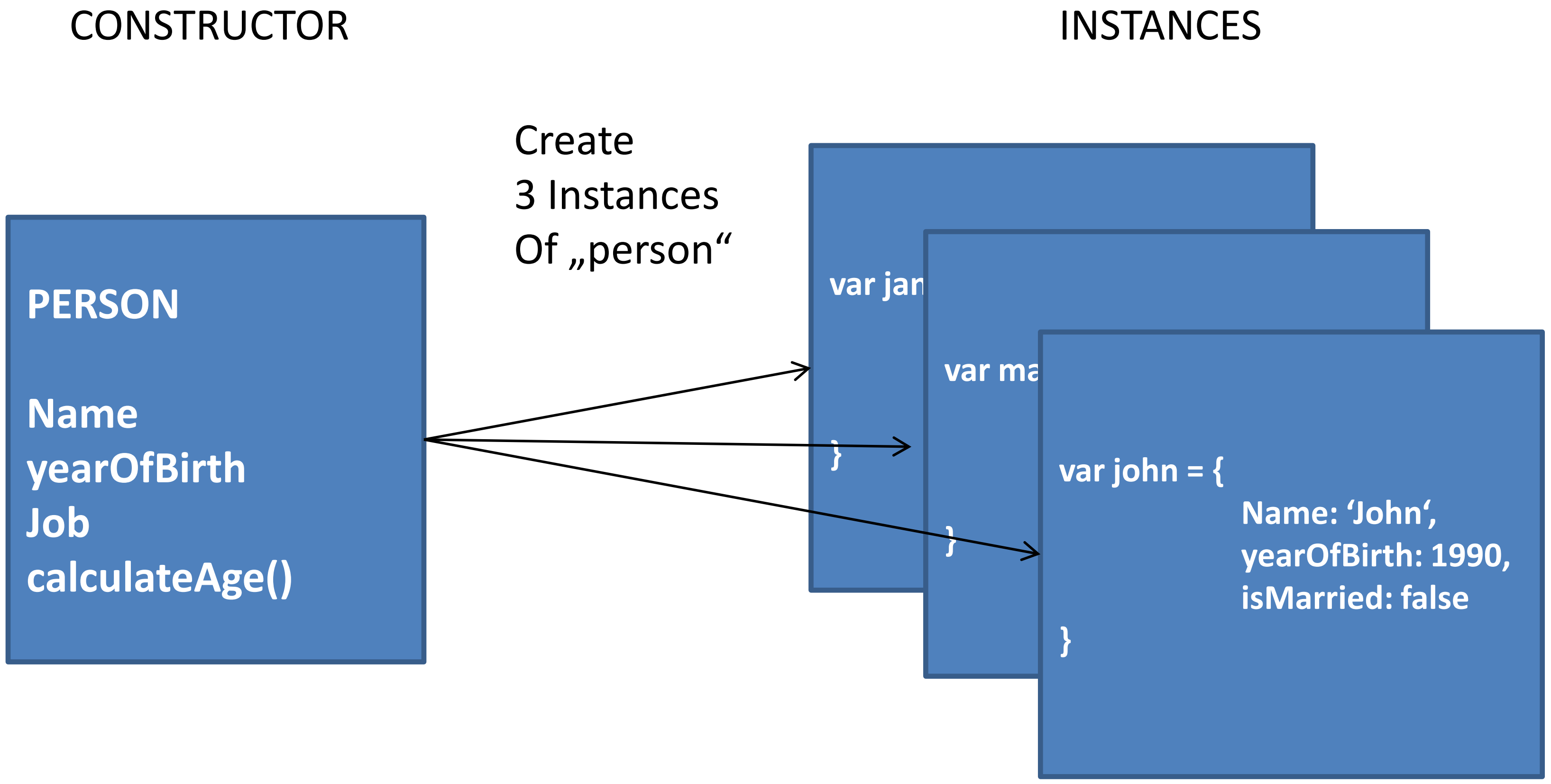
```
var john = {
		Name: ‘John‘,
		yearOfBirth: 1990,
		isMarried: false
}
```

```
var jane = {
		Name: ‘Jane‘,
		yearOfBirth: 1991,
		isMarried: true
}
```

```
var mark = {
		Name: ‘Mark‘,
		yearOfBirth: 1948,
		isMarried: true
}
```

3 Objects = A lot of typing

6. Constructors and Instances



7. Inheritance

PERSON

Name
yearOfBirth
Job
calculateAge()

7. Inheritance

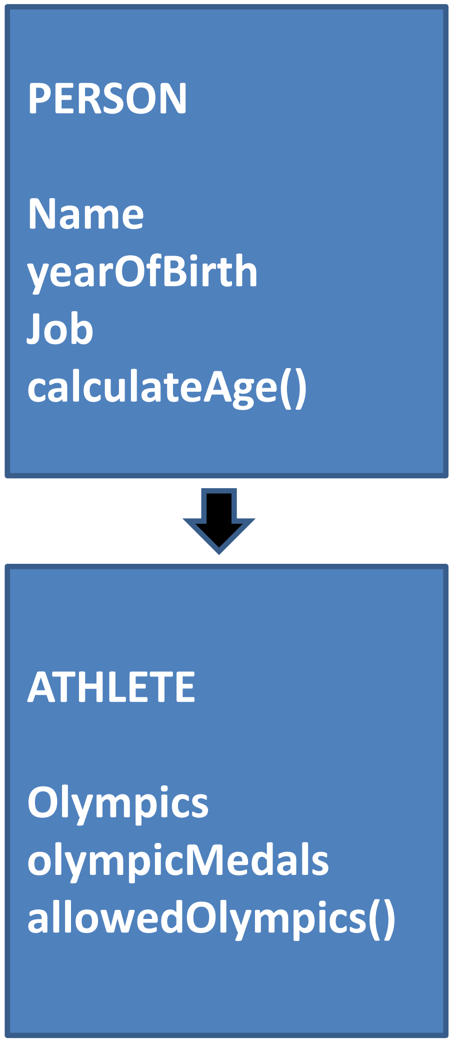
PERSON

Name
yearOfBirth
Job
calculateAge()

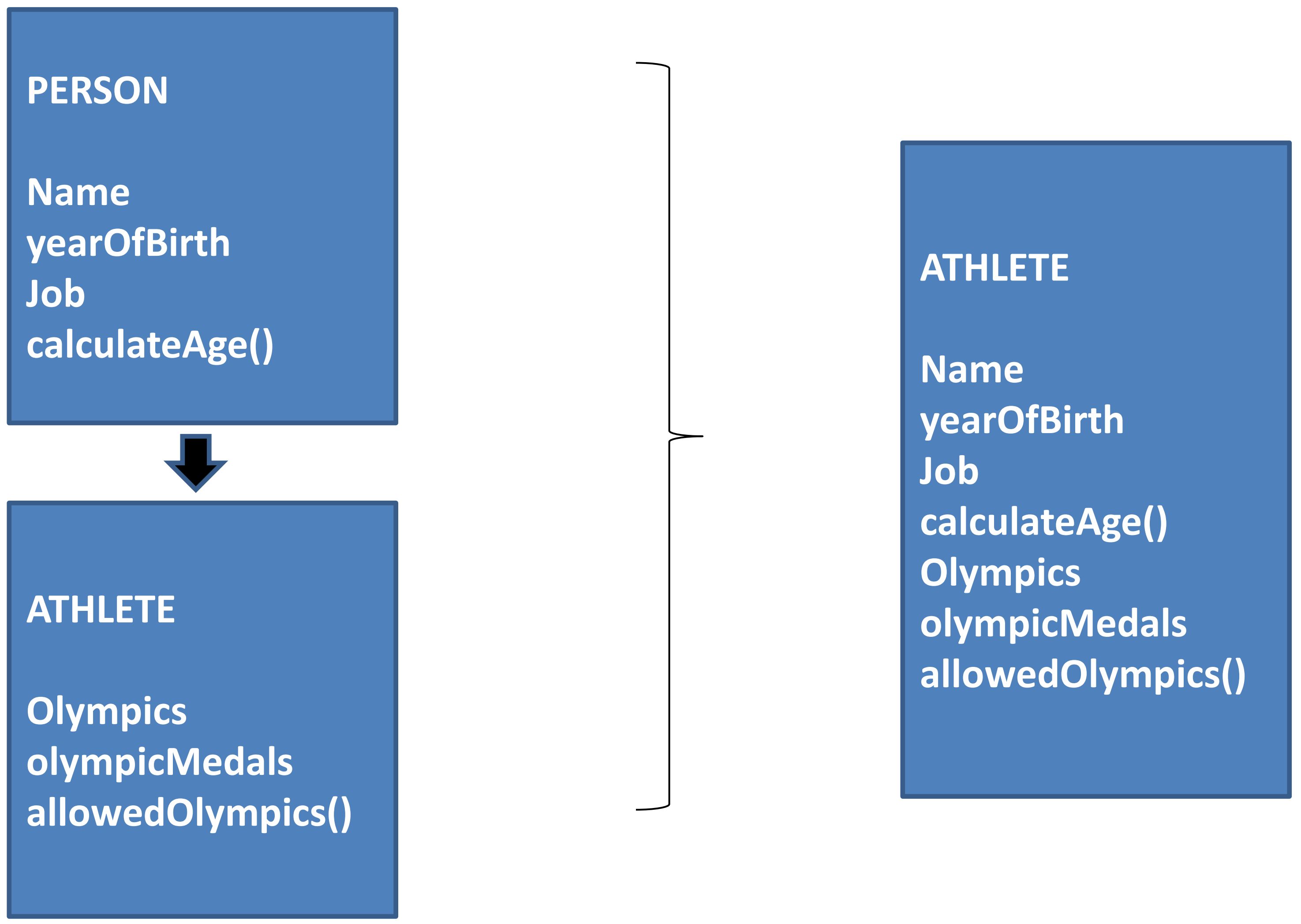
ATHLETE

Olympics
olympicMedals
allowedOlympics()

7. Inheritance



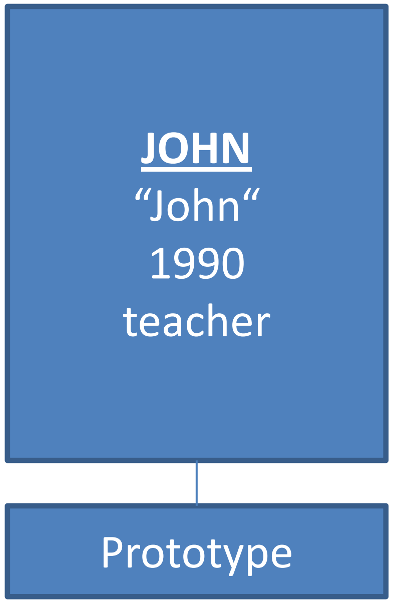
7. Inheritance



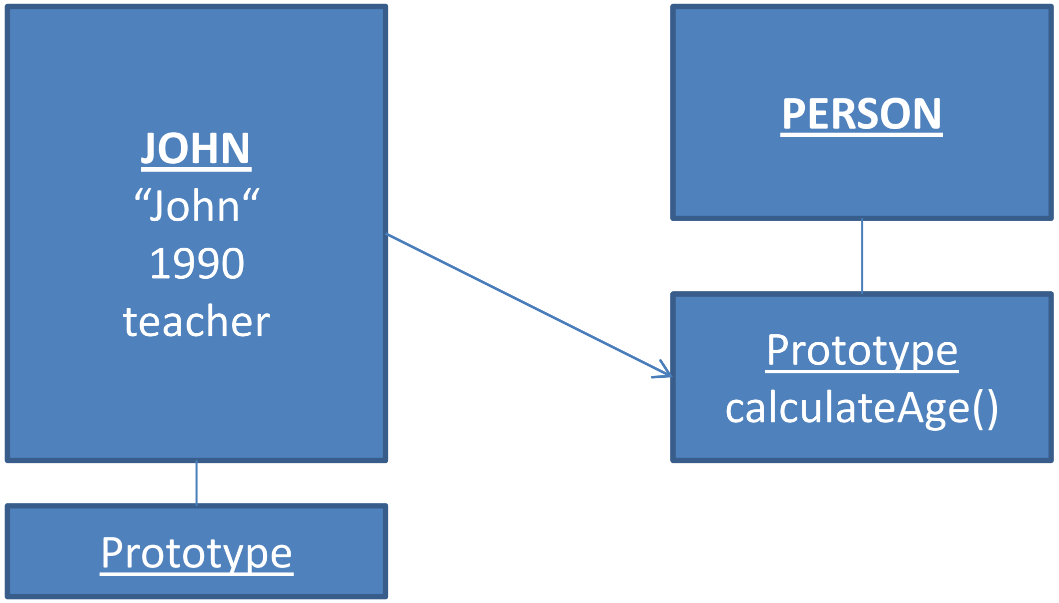
8. Prototype

- Every object in JavaScript has an attribute called **prototype**
- Each prototype has an attribute, which itself a prototype
- This goes on, **until prototype is null**

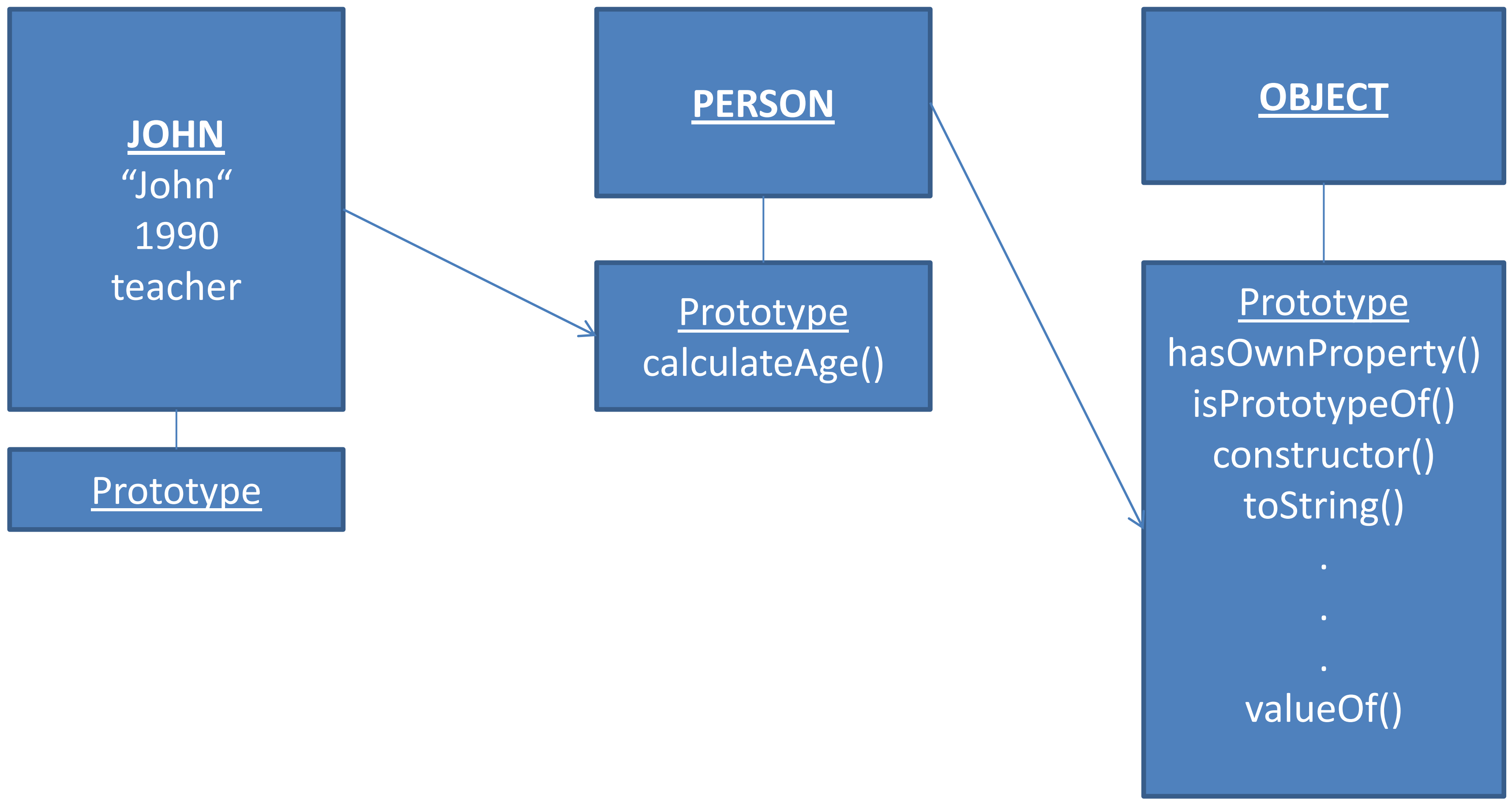
9. Prototype-Chain



9. Prototype-Chain



9. Prototype-Chain



9. Prototype-Chain

- All JavaScript objects inherit properties and methods from a prototype.
- Date objects inherit from Date.prototype. Array objects inherit from Array.prototype. Person objects inherit from Person.prototype.
- The Object.prototype is on the top of the prototype inheritance chain:
- Date objects, Array objects, and Person objects inherit from Object.prototype.

10. Call and Apply

- Each function in JavaScript is an object, too.
- Each function has methods, like other objects, as well.
- As we learned before: **call()** and **apply()** are methods of functions
  - **call(object, argument1, argument2, ...)** calls a function and injects another **this**-variable as **object.**
  - **apply(object, [argument1, argument2, ...])** does the same as **call()** except it accepts an array of arguments instead of an argument list.

10. Call and Apply: Task 1

1. Compile this code and analyze the object **gonzo** in the console. What did **call()** do?
2. Create a function in the object alfred setLastName(lastname) that attaches an attribute **lastname** to alfred and sets it to the parameter lastname.
3. Use **call()** again to borrow setLastName() on **gonzo** with the parameter „Gonzales“**.**

```
var alfred = {
    name: 'Alfred',
    count: 0,
    sayYourName: function() {
        if(this.count === undefined)
            this.count = 0;

        console.log( 'My name is ' + this.myName );
        this.count++;
    }
}

var gonzo = {
    myName: 'Gonzo'
}

alfred.sayYourName.call(gonzo);
```

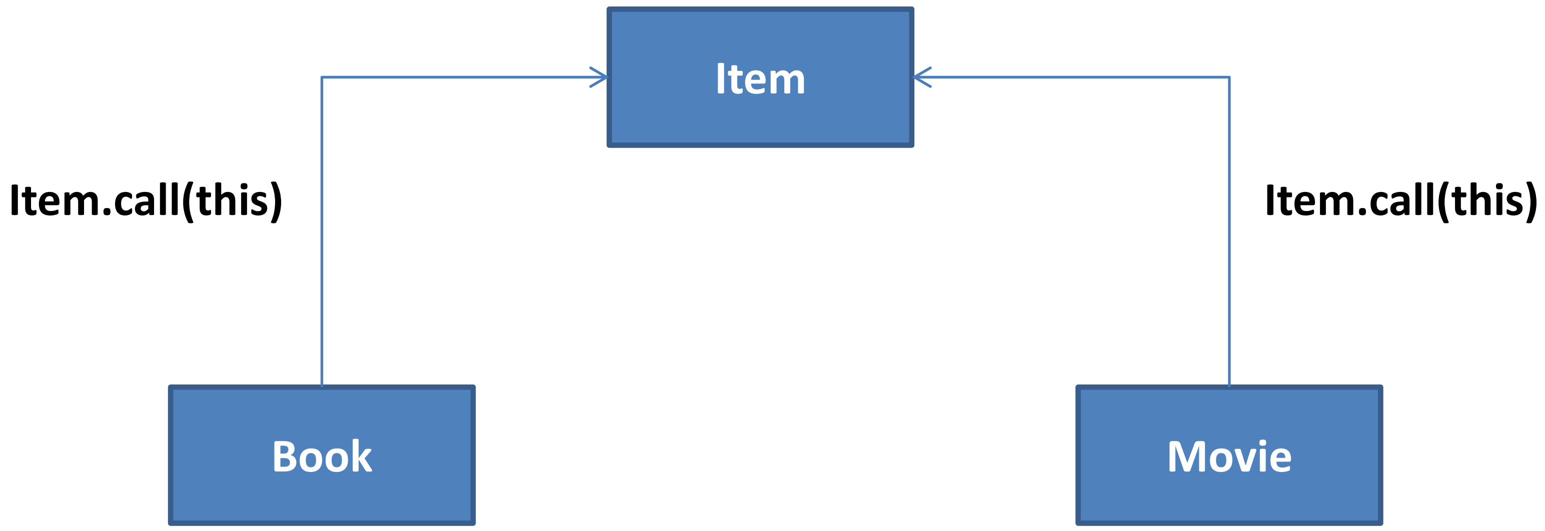
10. Call and Apply: Task 2

```
var john = {
	name: 'john',
	age: 26,
	job: 'teacher',
	presentation: function(style, timeOfDay) {
		if(style === 'formal') {
			console.log('Good ' + timeOfDay
				+ ' Ladies and Gentlemen I am '
				+ this.name + ', I am a '
				+ this.job + ' and I am '
				+ this.age + ' years old.');
		}
		else if(style === 'friendly') {
			console.log('Hey whatsup.'
				+ 'I am '
				+ this.name + ', I am a '
				+ this.job + ' and I am '
				+ this.age + ' years old.'
				+ 'Have a nice ' + timeOfDay);
		}
	}
};

john.presentation('formal', 'morning');

var emily = {
	name: 'Emily',
	age: 35,
	job: 'designer'
};
```

1. Analyze this code and describe briefly what it does.
2. Use **call()** to use the function **presentation()** from the john-object on the emily-object with the parameters “friendly“ and “evening“.
3. Do (2) again with **apply()**.

11. Inheritance

call() and **apply()** can be used to borrow constructor functions from other objects, which can be used as parent-objects. We will use **call()** only.



11. Inheritance

PARENT

```
function Item( name, price ) {
    this.name = name;
    this.price = price;
    this.sold = false;
}

Item.prototype.sell = function() {
    this.sold = true;
}
```

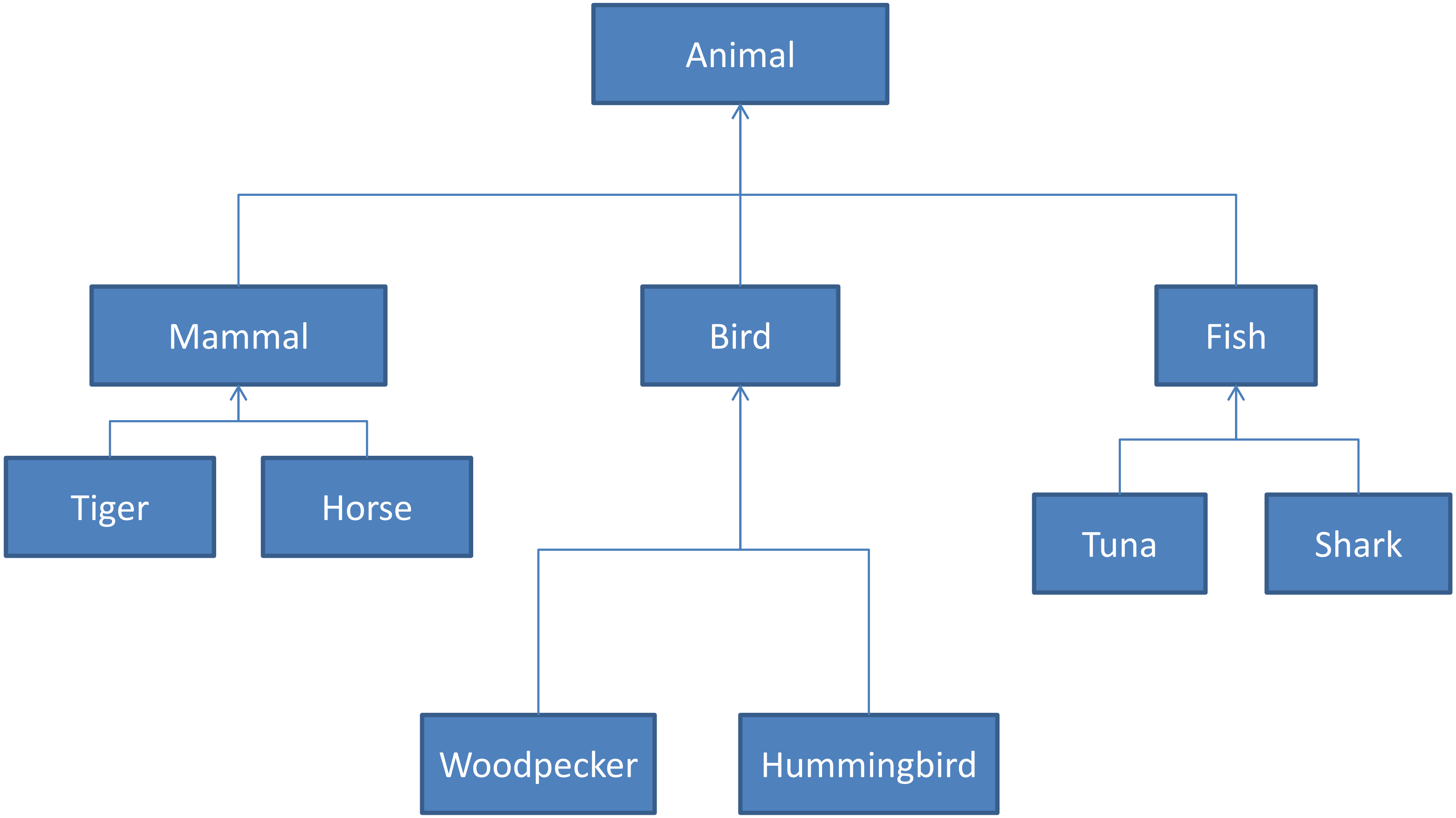
CHILDREN

```
function Book( name, price, author ) {
        Item.call(this, name, price);
        this.author = author;
        this.category = 'book';
}

Book.prototype =
Object.create(Item.prototype);

function Movie( name, price, director ) {
        Item.call(this, name, price);
        this.director = director;
        this.category = 'movie';
}

Movie.prototype =
Object.create(Item.prototype);
```

11. Inheritance: Task

1. Compile the code and analyze it. Note that **call()** is used to call the constructor of another object and note that Object.create() is used to create the object‘s prototype based of another object‘s prototype.
2. Create one movie “Casino“ from
   “Martin Scorsese“ and one book “IT“ from “Stephen King“.
2. Sell them both.
3. Create a new function constructo**r ComicBook** that inherits from **Book** and introduces a new attribute minAge which will set to 6 if it is undefined or less than 6.
4. Create the comic book “Jessica Jones“ from “Marvel“ with minAge 12.
5. Sell it.

12. Multi-Level Inheritance



12. Multi-Level Inheritance: Task

1. With your knowledge from (2) Inheritance, please create the Function constructors according to the Animal diagram and consider the following rules:
    1. Each animal has a **name** that is set when it is constructed.
    2. All animals can **sleep**, **eat** and **die** (use functions for this, e.g. **sleep()**)
    3. Mammals and birds can **breathe**.
    4. Fishes can **swim**.
    5. Birds can **fly**.
    6. Tigers and Sharks can kill, whereas **kill()** expects one parameter **otherAnimal.** Kill() calls the die() function o**f otherAnimal**.
2. Create one tiger with name „Vitaly“, one Shark with name „Nemo“, one horse with name „Fury“.
3. Nemo is hungry and kills Fury and Vitaly. Then Nemo eats.
4. Nemo dies.

13. Class Keyword

- The class keyword is **syntactic sugar** for defining prototypes

```
class Person {
    constructor(name, age, job) {
            this.name = name;
            this.age = age;
            this.job = job;
    }

    calculateAge() {
            return 2018 – this.age;
    }
}
```

Task

1. Build a function constructor called Question to describe a question. A question should include:a) question itselfb) the answers from which the player can choose the correct one (choose an adequate data structure here, array, object, etc.)c) correct answer (I would use a number for this)
2. Create a couple of questions using the constructor
3. Store them all inside an array
4. Select one random question and log it on the console, together with the possible answers (each question should have a number) (Hint: write a method for the Question objects for this task).
5. Use the 'prompt' function to ask the user for the correct answer. The user should input the number of the correct answer such as you displayed it on Task 4.
6. Check if the answer is correct and print to the console whether the answer is correct ot nor (Hint: write another method for this).
7. Suppose this code would be a plugin for other programmers to use in their code. So make sure that all your code is private and doesn't interfere with the other programmers code (Hint: we learned a special technique to do exactly that).