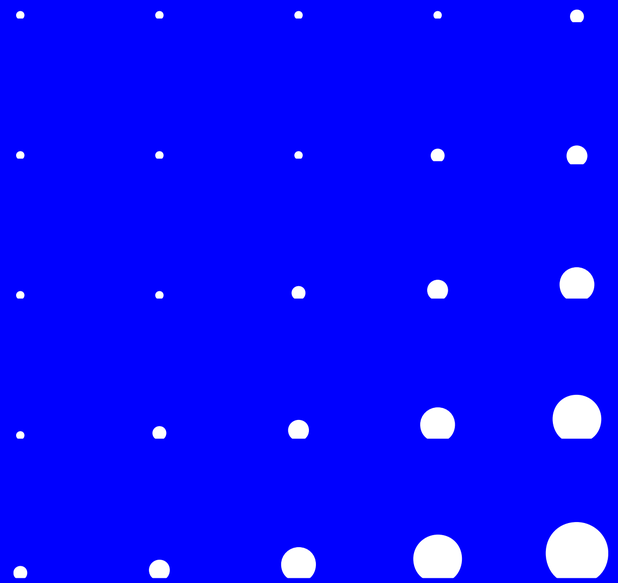
Technische Hochschule
mannheim

JavaScript
Objekte

Technische Hochschule Mannheim | Prof. Dr. Frank Dopatka



Objektbasierend…

Technische Hochschule mannheim

- JavaScript ist in erster Linie objektbasierend.
  - Funktionen sind in JavaScript auch Objekte.
  - Objekte können von Objekten erben: prototypische Vererbung.
- In Java...
  - legt die Klasse die Struktur der erzeugten Objekte fest.
  - Alle erzeugten Objekte haben denselben Aufbau.
- In JavaScript...
  - legt eine Konstruktorfunktion die Struktur der erzeugten Objekte fest.
  - Erzeugte Objekte können unterschiedlichen Aufbau haben.

Erzeugen von Objekten

Technische Hochschule mannheim

- Objekte können erzeugt werden über
  - ein Objekt-Literal via JSON.
  - eine Konstruktor-Funktion, die mit new aufgerufen wird.
  - eine Factory-Funktion, die ohne new aufgerufen wird.
- Konvention:
  - Konstruktor-Funktionen fangen mit Großbuchstaben an.
  - Factory-Funtionen fangen mit Kleinbuchstaben an.

Erzeugung via JSON Objekt-Literal

Technische Hochschule mannheim

```
// println() ist vorher definiert
var schulze = {
  name: "Schulze, Herbert",
  geboren: "4.5.1954",
  details: function() {
    return "Name: " + this.name
           + ", geboren: " + this.geboren;
    }
};

println(schulze.details());
```

Name: Schulze, Herbert, geboren: 4.5.1954

Erzeugung via Konstruktor-Funktion

Technische Hochschule mannheim

```
// println() ist vorher definiert
function Mitarbeiter(name, geboren) {
    this.name = name;
    this.geboren = geboren;
    this.details = function() {
        return "Name: " + this.name
             + ", geboren: " + this.geboren;
    };
}
var meier = new Mitarbeiter("Meier, Franz", "1.5.1972");
var mueller = new Mitarbeiter("Müller, Peter", "8.11.1981");
println(meier.details());
println(mueller.details());
```

Name: Meier, Franz, geboren: 1.5.1972
Name: Müller, Peter, geboren: 8.11.1981

Erzeugung via Factory-Funktion

Technische Hochschule mannheim

```
// println() ist vorher definiert
function makeMitarbeiter(name, geboren) {
    var that = new Object();
    that.name = name;
    that.geboren = geboren;
    that.details = function() {
        return "Name: " + this.name
            + ", geboren: " + this.geboren; };
    return that;
}
var meier = makeMitarbeiter("Meier, Franz", "1.5.1972");
var mueller = makeMitarbeiter("Müller, Peter", "8.11.1981");
println(meier.details());
println(mueller.details());
```

Name: Meier, Franz, geboren: 1.5.1972
Name: Müller, Peter, geboren: 8.11.1981

Erzeugung via Factory-Funktion und JSON

Technische Hochschule mannheim

```
// println() ist vorher definiert
function makeMitarbeiter(name, geboren) {
    return {
        name: name,
        geboren: geboren,
        details: function() {
            return "Name: " + this.name
                + ", geboren: " + this.geboren;
        }
    };
}
var meier = makeMitarbeiter("Meier, Franz", "1.5.1972");
var mueller = makeMitarbeiter("Müller, Peter", "8.11.1981");
println(meier.details());
println(mueller.details());
```

Name: Meier, Franz, geboren: 1.5.1972
Name: Müller, Peter, geboren: 8.11.1981

Vererbung: Java vs. JavaScript

Technische Hochschule mannheim

```java
// Von Oberklasse Mitarbeiter ableiten
public class Manager extends Mitarbeiter {
    public String abteilung;

    public Manager(String name, int alter, String abteilung) {
        // Oberklassenkonstruktur benachrichtigen
        super(name, alter);
        this.abteilung = abteilung;
    }
}

Manager ma = new Manager("Franz", 37, "Webentwicklung");
```

Vererbung: Java vs. JavaScript

Technische Hochschule mannheim

```
function Manager(name, alter, abteilung) {
    // In JavaScript:
    Mitarbeiter.call(this, name, alter);
    this.abteilung = abteilung;
}

var ma = new Manager("Franz", 37, "Webentwicklung");
```

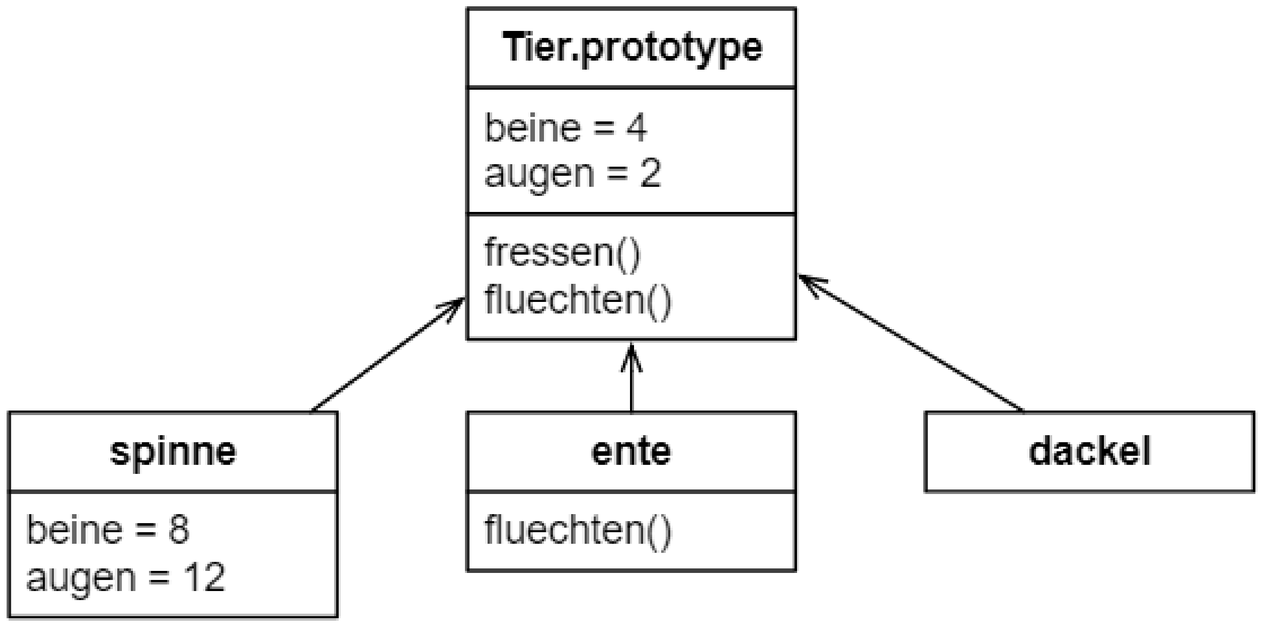
Prototypen in JavaScript

Technische Hochschule mannheim

- Zu jeder Funktion gehört ein Prototyp, das sogenannte prototype object.
- Der Zugriff erfolgt über Funktionsname.prototype
- Wird die Funktion als Konstruktor mit new verwendet, dann hat das erzeugte Objekt eine Referenz zum Prototypen.
- Sind Attribute und Funktionen nicht im Objekt definiert, dann werden sie im Prototypen gesucht.

Prototypen in JavaScript: Ein Beispiel

Technische Hochschule mannheim



Prototypen in JavaScript: Ein Beispiel

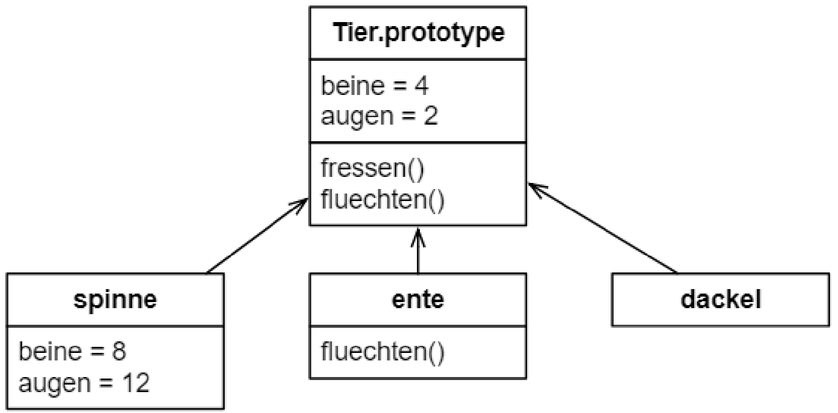
Technische Hochschule mannheim

```
function Tier(name) {
    this.name = name;
}

Tier.prototype.fressen = function() {
    println("Mampf"); };

Tier.prototype.fluechten = function() {
    println("Lauf weg"); };

Tier.prototype.beine = 4;
Tier.prototype.augen = 2;
```



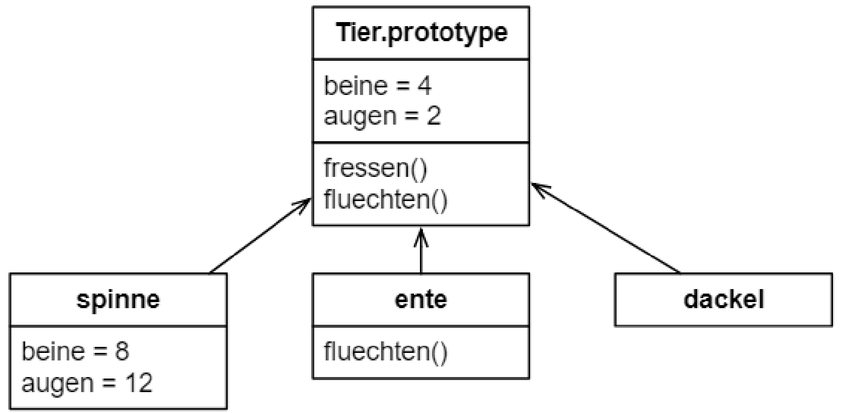
Prototypen in JavaScript: Ein Beispiel

Technische Hochschule mannheim

```
var spinne = new Tier("Spinne");
spinne.beine = 8;
spinne.augen = 12;

spinne.fressen();
spinne.fluechten();

println("Beine = " + spinne.beine);
println("Augen = " + spinne.augen);
```



Prototypen in JavaScript: Ein Beispiel

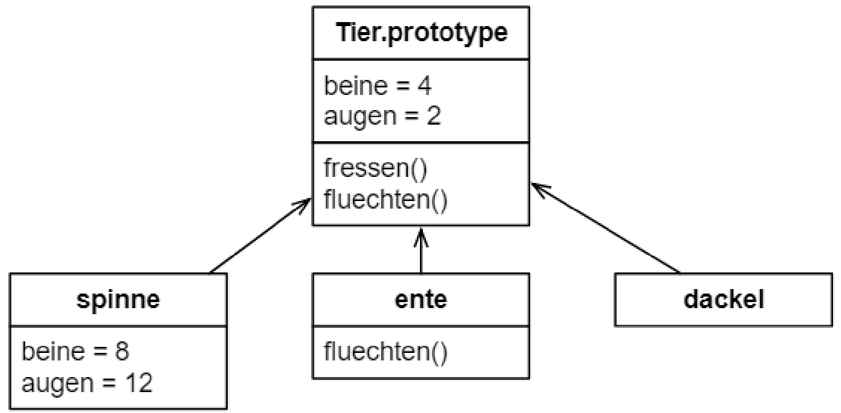
Technische Hochschule mannheim

```
var ente = new Tier("Ente");
ente.fluechten = function() {
    println("Flieg weg");
};

Tier.prototype.beine = 2;

ente.fressen();
ente.fluechten();

println("Beine = " + ente.beine);
println("Augen = " + ente.augen);
```



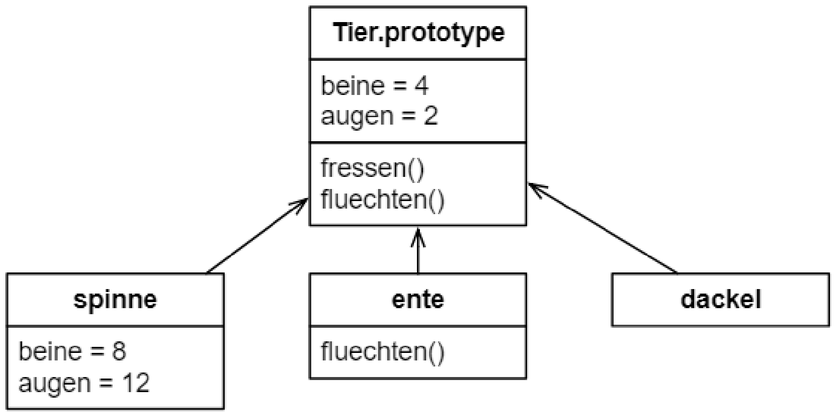
Prototypen in JavaScript: Ein Beispiel

Technische Hochschule mannheim

```
var dackel = new Tier("Dackel");

dackel.fressen();
dackel.fluechten();

println("Beine = " + dackel.beine);
println("Augen = " + dackel.augen);
```



Prototypische Vererbung in JavaScript

Technische Hochschule mannheim

- Objekte können als Vorlage für neue Objekte dienen.
- Mit Object.create(object) kann man ein Objekt erzeugen, das das gegebene Objekt als Vorlage benutzt.
- Das erzeugte Objekt kann man anpassen:
  - erweitern,
  - Methoden verändern,
  - etc.
- Von dem neuen Objekt kann man wieder weitere Objekte erzeugen.

Prototypische Vererbung in JavaScript: Ein Beispiel

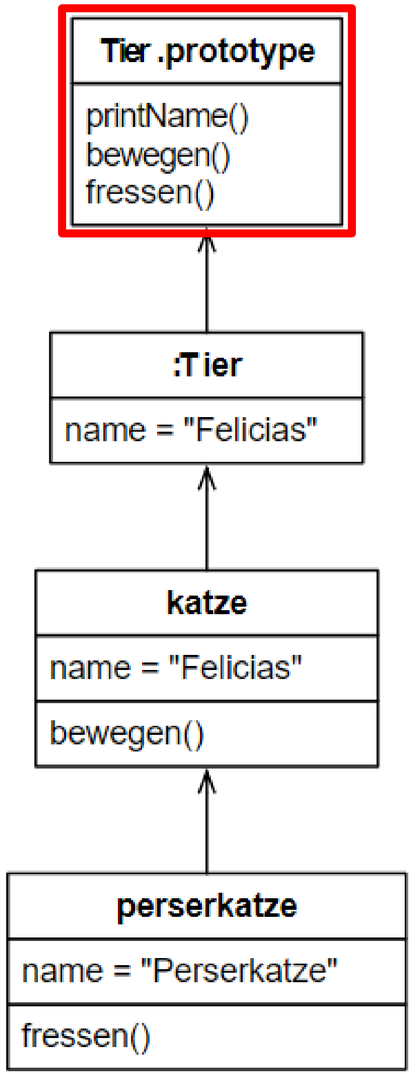
```
function Tier(name) {
    this.name = name;
}

Tier.prototype.printName =
    function() { println(this.name); };

Tier.prototype.bewegen =
    function() { println("bewege mich"); };

Tier.prototype.fressen =
    function() { println("fressen"); };
```

Technische Hochschule mannheim

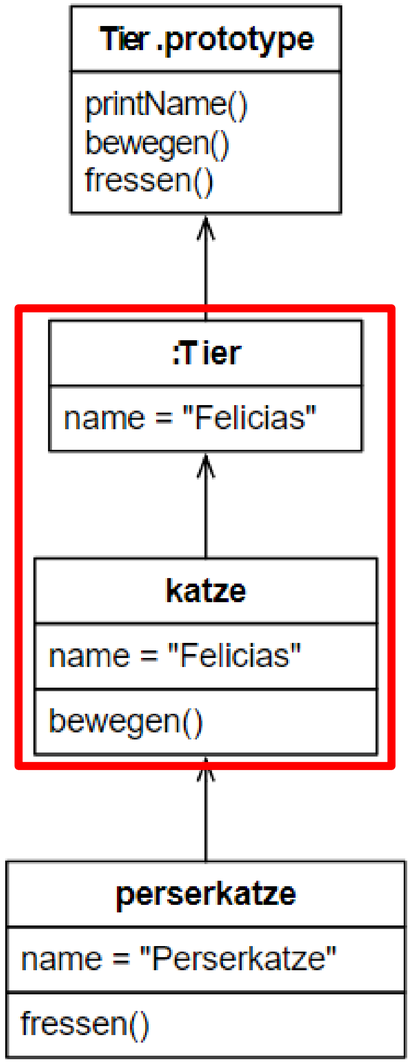


Prototypische Vererbung in JavaScript: Ein Beispiel

Technische Hochschule mannheim

```
var katze = Object.create(new Tier("Felizitas"));
katze.bewegen = function() {
    println("leise schleichen");
};

katze.bewegen();
katze.fressen();
katze.printName();
```

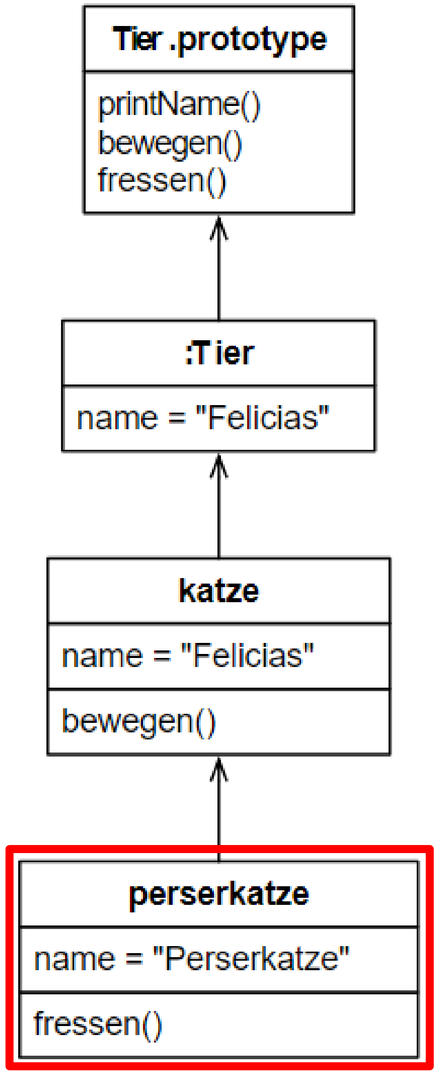


Prototypische Vererbung in JavaScript: Ein Beispiel

Technische Hochschule mannheim

```
var perserkatze = Object.create(katze);
perserkatze.name = "Perserkatze";
perserkatze.fressen = function() {
    println("Sheba geniessen");
};

perserkatze.bewegen();
perserkatze.fressen();
perserkatze.printName();
```



instanceof

Technische Hochschule mannheim

- Mittels des Operators instanceof wird überprüft, ob die Konstruktor.prototyp-Eigenschaft in der Prototypkette des überprüften Objektes vorkommt.
  - println({} instanceof Object); true
  - println(Object.create({}) instanceof Object); true
  - println(true instanceof Boolean); false
  - println(new Boolean(true) instanceof Boolean); true
  - println("Hallo" instanceof String); false
  - println(new String("Hallo") instanceof String); true
- true als Wahrheitswert und auch “Hallo“ als String-Primitiv sind in JavaScript primitive Typen!

Entfernen von Eigenschaften und Methoden

Technische Hochschule mannheim

```
// println() ist vorher definiert
var hund = {
    name: 'Hasso',
    alter: 7,
    bellen: function() {
        return 'wau';
    }
}

println(hund.alter);
delete hund.alter;
delete hund.bellen;
println(hund.alter);
```

7
undefined

Kapselung und Information-Hiding

Technische Hochschule mannheim

- JavaScript kennt zunächst keine Zugriffsbeschränkungen.
  - Alle Attribute eines Objektes sind immer erreichbar und änderbar.
- Information-Hiding kann trotzdem erreicht werden:
  - Man verwendet eine Factory-Methode.
  - Die Methode erzeugt ein Objekt für die privaten Variablen.
  - Die Methode erzeugt ein Objekt für die öffentlichen Funktionen.
  - Dem privaten Objekt werden alle privaten Variablen und privaten Funktionen hinzugefügt.
  - Das öffentliche Objekt greift auf die privaten Daten zu, was es wegen der Sichtbarkeit der Daten innerhalb der Funktion darf.
  - Das öffentliche Objekt wird zurückgegeben.

Kapselung und Information-Hiding: Ein Beispiel

Technische Hochschule mannheim

```
// println() ist vorher definiert
function erzeugeTier(name, my) {
    var that = {}; // leeres Objekt für die Methoden
    var my = {};   // leeres Objekt für die Eigenschaften
    my.name = name;

    that.getName = function() { return my.name; };
    that.setName = function(name) { my.name = name; };
    return that;
}

var hund = erzeugeTier("Hasso");
println(hund.getName());
hund.setName("Archibald");
println(hund.getName()); // my.name ist nicht sichtbar
```

Kapselung und Information-Hiding: Ein Beispiel mit Vererbung

Technische Hochschule mannheim

```
// println() ist vorher definiert
function erzeugeKatze(name) {
    var that = erzeugeTier(name);
    that.schnurre = function() { println("Brrrrrrr"); };
    return that;
}

var katze = erzeugeKatze("Felizitas");
println(katze.getName());
katze.schnurre();
```

Ausnahmebehandlung

Technische Hochschule mannheim

- Es existiert ein try, catch, finally ähnlich wie in Java.
  - Ausnahmen sind jedoch leider nicht typisiert.
  - Daher gibt es nur einen catch-Block pro try.
  - finally wird immer ausgeführt.

```
try {
    println("Vor dem throw...");
    throw { name: "Fehlermeldung" };
     println("Nach dem throw...");
}
catch (e) {
    // Fehlerbehandlung
    println("Ooops: " + e.name);
}
finally {
    // wird immer ausgeführt
    println("finally");
}
```

Doch Klassen?

Technische Hochschule mannheim

- Seit ES2015 besitzt JavaScript eine class-Syntax.
- Dies ist jedoch nur Kosmetik...

```
class Auto2 {
    constructor (marke, model, jahr) {
        this.marke = marke;
        this.model = model;
        this.jahr = jahr;
    }
    methode () {
        return 'Methode';
    }
}

const auto3 = new Auto2('Marke1', 'Model1', 1991);
console.log(auto3.jahr, auto3.marke, auto3.model, auto3.methode());

// Konsole: 1991 Marke1 Model1 Methode
```