Linnéuniversitetet
Institutionen för medieteknik
Interaktiva medier och webbteknologier

http://www.lnu.se/
Växjö
Kandidatprogram 180 hp

Henrik Andersen Ph.M.
henrik.andersen@lnu.se

1ME325 Webbteknik 5, 7,5hp

Föreläsning 4a; Objektorienterad programmering med JavaScript (Del 2)

Innehåll

- Objektorienterad programmering
    - Klass- och prototypbaserad
- Prototype
    - Funktionalitet
    - Omfångshantering
    - Prototypkedja
    - Syntaxer
    - Flexibilitet
    - Inbyggda objekt-prototyper
    - Problematik
    - Konstruktor-stöld

Syfte

- Föreläsningen behandlar avancerade objektstrukturer via prototyp-kedjan

Läsanvisningar

- *Object-Oriented JavaScript: Create scalable, reusable high-quality JavaScript applications, and libraries, Kap. 5-6*

Laboration(er)

- Denna veckan motsvarar följande laborationer i laborationshandboken:
  - Laboration 8; *Prototype*
  - Laboration 9; *Prototypbaserat arvsled*
  - Laboration 10; *JsOOP*

Objektorienterad programmering

- Två objektorienterade modeller:
  - Klass-baserad; *kodåteranvändning via arvsled*
  - Prototyp-baserad; *kodåtervinning via delegering*

Klass-baserad; *överblick*

- Karaktäristiska drag för klass-baserade programmeringsspråk:

  - Klasser delar metoder och definerar gemensamma egenskaper
  - Arvsled via klass-kedjan
  - Objekt är identiska instanser av sin klass
  - Då objekt är instanser av klasstrukturer, kan strukturen i dessa objekt vanligtvis inte förändras under körning

Prototyp-baserad; *överblick*

- Karaktäristiska drag för prototyp-baserade programmeringsspråk:

  - Det förekommer inga klasser, bara objekt
  - Objekt har möjlighet att definiera egna egenskaper och metoder
  - Objekt delegerar dataåtkomst till sin prototyp
  - Prototypreferenser kan överskrivas till referens till valfritt objekt; *ett sätt att imitera arvstrukturer*

Objektorienterad programmering; *reflektionsfrågor*

- Relevanta frågeställningar:
  - *Är PHP ett klass- eller prototyp-baserat objektorienterat skriptspråk?*
  - *Är JavaScript ett klass- eller prototyp-baserat objektorienterat skriptspråk?*

Prototype

- Prototyp-relaterad information att behandla:
  - Funktionalitet
  - Omfångshantering
  - Prototypkedja
  - Syntaxer
  - Flexibilitet
  - Inbyggda objekt-prototyper
  - Problematik
  - Konstruktor-stöld

Funktionalitet; *teori*

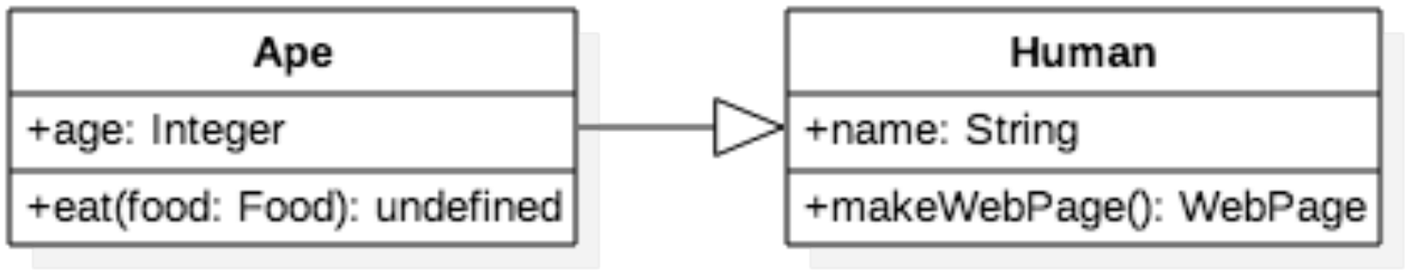
- Med ett prototyp-baserat programmeringsspråk är det möjligt att delegera data- och funktionalitetsåtkomst; *syftet är att sprida ut funktionaliteten över flera objekt*

- Konceptet fungerar väl då objekt representerar generell funktionalitet och data; *specifika objekt är svåra att återanvända då de är utformade för att göra en specifik sak*

Funktionalitet; *teori*

- Klass-baserade programmeringsspråk använder arvsled för att simulera evolution inom det egna arvsledet; *ett objekt ligger till grund för ett mer avancerat objekt*

Funktionalitet; *teori*



Figur. Beskriver en arvsrelation mellan klasserna Ape (super) och Human (sub)

Funktionalitet; *teori*

- Prototyp-baserade programmeringsspråk använder delegation för att överlämna arbetsuppgifter till ett annat objekt; *objekt samarbetar men vidareutvecklas inte*

Funktionalitet; *teori*

Figur. Beskriver relationen mellan klasserna Ape och Human

Funktionalitet; *teori*

- Relevant frågeställning:

  - *Vad spelar detta för roll då slutresultatet är densamma?*

Funktionalitet; *teori*

- Delegationer skapas via JavaScripts inbyggda prototype-stöd där varje data-/objekttyp tilldelas en egen prototyp

- Prototypen gör tillgänglig både via statisk och dynamisk referens:

  - __proto__; *dynamisk referens som går att återfinna i varje enskild objektinstans*

  - Datatype.prototype; *statisk referens som går att återfinna i varje enskild datatyp*

Funktionalitet; *teori*

- Saker att betrakta:

  - `__proto__` är en intern referens och menad att användas av webbläsaren - inte av utvecklare

  - Prototype-objekt innehåller egenskapen `constructor` som hänvisar tillbaka till den aktuella instansen

  - Instanser innehåller en länk mellan instansen och konstruktorns prototyp - inte mellan instansen och konstruktorn

  - Prototype-objekt bör betraktas som statiska

Omfångshantering

- När en egenskap eller metod efterfrågas, påbörjas sökningen i den egna objektinstansen

- Om egenskapen eller metoden inte går att finna, delegeras sökningen vidare till objektets prototyp

Omfångshantering


```
var kalle = new Human();
kalle.name = "Kalle";
kalle.eat("Hamburger"); // Humans can not eat, delegate
request to Ape
```

Prototypkedja

- ECMA-262 beskriver en prototypkedja, som en alternativ metod att uppnå arvsled inom ett prototyp-baserat programspråk

- Konceptet går ut på att skapa relationer mellan flera objekttyper via deras prototyp-objekt

Syntaxer

- Objektorienterad mjukvarustruktur kan skapas på många olika sätt med JavaScript; *föreläsningen innehåller exempel som brukar skriptspråkets inbyggda funktionalitet (med viss modifikation)*

Syntaxer; *__proto__ (object literal)*

- Använder objekt-literal för att definiera objektstruktur; *inget resulterar i en egen datatyp, allt är Object*

- Referensen __proto__ överskrivs och refererar till en fördefinerad objektstruktur; *denna arbetsmetod kan liknas med ”late static bindings” i PHP*

- Arbetsmetoden är inte tillåten i samtliga webbläsare

Syntaxer; *__proto__ (object literal)*

```
var Ape = {
  age : 0,
  eat: function(food) {
    console.log("Yum yum!");
  }
};

var Human = {
  name: "",
  makeWebPage: function() {
    return "<html>...</html>";
  }
}

Human.__proto__ = Ape;
Human.eat("Banana"); // "Yum yum!"
```


Syntaxer; *__proto__ (constructor)*

- Använder konstruktorer för att skapa dynamiska objektreferenser; *objekt skapas på begäran och betraktas inte som statiska*

- Medför att nya delegationsobjekt skapas för varje underklass

- Arbetsmetoden är inte tillåten i samtliga webbläsare

Syntaxer; *__proto__ (constructor)*

```
var Ape = function(){
  this.age = 0;
  this.eat = function(food){
    console.log("Yum yum!");
  };
};

var Human = function() {
  this.__proto__ = new Ape();
  this.name = "";
  this.makeWebPage = function() {
    return "<html>...</html>";
  };
};

var kalle = new Human();
kalle.eat("Hamburger");
```


Syntaxer; *prototype (static)*

- Överskriver ett dynamiskt objekts prototyp med ett nytt ”statiskt” objekt; *kombinerar konstruktor och objekt literal*

- Prototyp-objektet måste innehålla egenskapen constructor, den skall hänvisa tillbaka till konstruktorn

Syntaxer; *prototype (static)*

```
var Human = function() {
  this.name = "";
  this.makeWebPage = function() {
    return "<html>...</html>";
  };
};

Human.prototype = {
   constructor: Human,
   age: 0,
   eat: function(food){
      console.log("Yum yum!");
   }
}

var kalle = new Human();
kalle.eat("Hamburger");
```


Syntaxer; *prototype (dynamic)*

- Arbetsmetod för att addera data och funktionalitet till en konstruktors prototyp; *bör betraktas som statisk*
- Kopplingarna skapas efter att konstruktorn definierats
- Delegationsobjekt skapas som en ny instans och överskriver sin supertyps prototyp

Syntaxer; *prototype (dynamic)*

```
var Ape = function(){
  this.age = 0;
};

Ape.prototype.eat = function(food) {
   console.log("Yum yum!");
};
```

```
var Human = function() {
  this.name = "";
  this.makeWebPage = function() {
    return "<html>...</html>";
  };
};

Human.prototype = new Ape();

var kalle = new Human();
kalle.eat("Hamburger");
```


Flexibilitet

- Då instansers prototyp-egenskap enbart representerar en referens, kan den överskrivas under körning och därmed förändras

Flexibilitet

```
var Ape = function() {};

var george = new Ape();
george.eat("banana"); // Error (Expected)
```

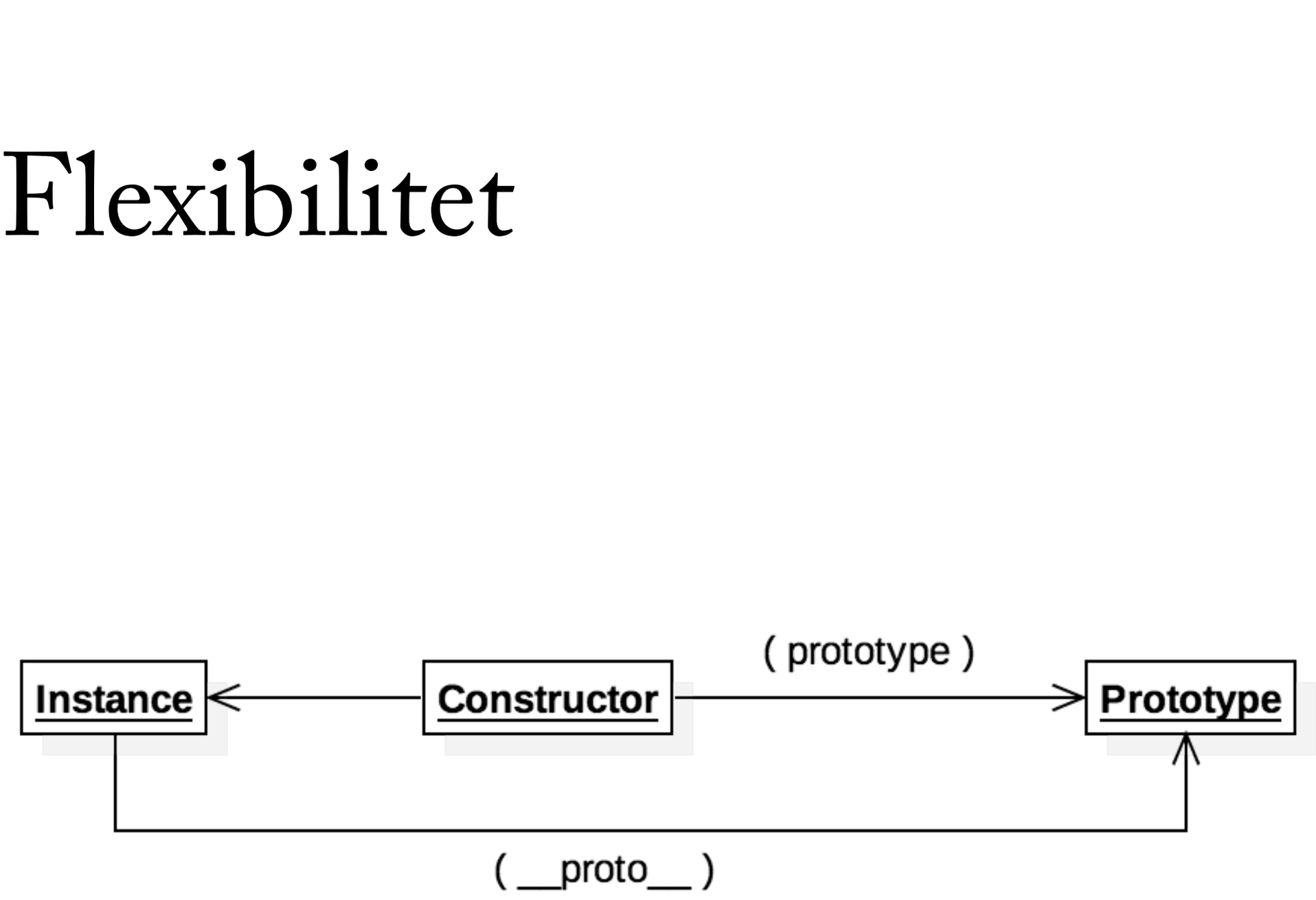
```
Ape.prototype = {
   constructor: Ape,
   eat: function(food) {
      console.log("Yum yum!");
   }
};

george.eat("banana"); // Error (Unexpeced)
```


Flexibilitet

- Saker att betrakta:
  - Instanser har en länk till prototypen, inte konstruktorn
  - Instans-referensen (__proto__) sätts i samband med instansiering via konstruktorn

Flexibilitet



Figur. Beskriver relationen mellan instanser, konstruktorer och prototyper. Beskrivningen är inte exakt men ger en övergripande beskrivning till hur och var viss information finns tillgänglig

Flexibilitet; *sammanfattning*

- Även om JavaScripts dynamiska natur kan tolkas som en fördel, finns det risker med att manipulera skriptspråkets grundläggande funktionalitet
- Undvik därför:
  - prototyp-manipulering efter instansiering av eftertraktade objekt
  - användning av instans-referensen

Inbyggda objekt-typer

- Samtliga av JavaScripts datatyper implementerar sina inbyggda metoder via prototyp-objekt:

```
console.log(typeof Array.prototype.push); // "function"
console.log(typeof String.prototype.substring); // "function"
```

- Detta i kombination med JavaScripts dynamiska struktur ger möjlighet att överskriva inbyggd funktionalitet; *kan exempelvis användas för att skapa polyfill*

Inbyggda objekt-typer

```
Array.prototype.push = function(element) {
  var l = this.length + 1;
  this.length = 0;
  for (var i = 0; i < l; i++) {
    this[i] = "Nuno";
  }
};

var names = new Array("Kalle", "David", "Rune");
names.push("Nuno");

console.log(names); // "Nuno", "Nuno", "Nuno", "Nuno"
```

Problematik

- Att kombinera konstruktor- och prototyp-mönstren kan vara fördelaktigt, men det medför viss problematik:
  - Avsaknad av instnsieringsparametrar; *information kan inte förmedlas ned i prototypkedjan i samband med konstruktor-anrop*
  - Statiskt innehåll; *information som placeras i prototyp-objekt ses som referensvärden och gör det omöjligt att lagra instansspecifik information*
- Relevant frågeställning:
  - *Går problematiken att kringå?*

Problematik; *lösningsförslag – statiskt innehåll*

- Kombinera statisk och dynamisk information för att skapa objektstrukturer:

  - konstruktorn; *erbjuder möjlighet att deklarera dynamiskt innehåll, dvs innehåll som skall vara unikt för objektet – exempelvis egenskaper*

  - prototype; *representerar en statisk plats innehållande gemensam information – exempelvis metoder*

Problematik; *lösningsförslag – avsaknad av instnsieringsparametrar*

- problematiken kan kringgås via en arbetsmetod som kallas konstruktor-stöld…

Konstruktor-stöld

- Ett koncept som går ut på att aktivera en super-typs konstruktor i kontexten av dess sub-typ

- Konceptet fungerar då en funktion är ett objekt som exekverar programkod inom en begränsad kontext

- JavaScript erbjuder två inbyggda funktioner för att specificera kontexten av ett funktionsanrop; *apply och call*

Konstruktor-stöld

```
var Ape = function(age) {
   this.age = age || 0;
   this.favoriteFood = ['Banana', 'Nuts'];
};
```

```
var Human = function(name, age) {
   Ape.call(this, age);
   this.name = name || "John";
};

var rune = new Human("Rune", 55);
console.log(rune.name); // "Rune"
console.log(rune.age); // 55
console.log(rune.favoriteFood); // "Banana, Nuts"
```

Combination Inheritance

- Om prototyp-, konstruktor- och konstruktor-stöld-mönstret kombineras, kan följande objektstruktur uppnås…

- Denna kombination av mönster kallas *Combination Inheritance*

Combination Inheritance

```
var Ape = function(age) {
    this.age = age || 0;
    this.favoriteFood = ['Banana', 'Nuts'];
};

Ape.prototype.eat = function(food) {
    if (this.favoriteFood.indexOf(food) !== -1) {
        console.log("Yum yum!");
    } else {
        console.log("Yuk!");
    }
};

var Human = function(name, age) {
    Ape.call(this, age);
    this.name = name || "John";
};

Human.prototype = new Ape();

var rune = new Human("Rune", 55);
rune.eat("Banana"); // "Yum yum!"
console.log(rune.name); // "Rune"
```


Combination Inheritance

- Denna arbetsmetod är fördelaktig då den erbjuder en struktur som är snarlik andra klass-baserade programmeringsspråk

- Metodens största nackdel är att sub-typer instansieras två gånger; *först för att skapa sub-typens prototyp och sedan i sub-typens konstruktor*

- Relevanta frågeställningar:

  - *Är detta ett relevant problem?*

  - *Kan problemet lösas?*

Parastatic Combination Inheritance

```
var Ape = function(age) {
    this.age = age || 0;
    this.favoriteFood = ['Banana', 'Nuts'];
};

Ape.prototype.eat = function(food) {
    if (this.favoriteFood.indexOf(food) !== -1) {
        console.log("Yum yum!");
    } else {
        console.log("Yuk!");
    }
};

var Human = function(name, age) {
    Ape.call(this, age);
    this.name = name || "John";
};

Human.prototype = Object.create(Ape.prototype);
Human.prototype.constructor = Human;

var rune = new Human("Rune", 55);
rune.eat("Banana"); // "Yum yum!"
console.log(rune.name); // "Rune"
```


Sammanfattning

- JavaScript innefattar ett prototyp-baserat system för objektorienterad programmering

- Samtliga objekttyper har tillgång till en gemensam objekt-prototyp

Frågor