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 3a; Objektorienterad programmering med JavaScript (Del 1)

Innehåll

- Objektorienterad programmering (summering från tidigare kurser i webbteknik)

- Objektorienterad programmering med ECMAScript
  - Factory pattern
  - Constructor pattern
  - Prototype pattern
- Object literal VS JSON

Syfte

- Att presentera en plattform för att konstruera objektorienterad programkod med ECMAScript; *översätta objektorienterad teori till praktik med ECMAScript*

Läsanvisningar

- Veckans obligatoriska läsning:

  - *Object-Oriented JavaScript: Create scalable, reusable high-quality JavaScript applications, and libraries, Kap. 4(-5)*

  - *Weisfeld, M. (2013), "The Object-Oriented thought Process" 4. uppl., Addison-Wesley Professional. Kap 3-7*

Objektorienterad programmering; *översiktlig beskrivning*

- Objektorienterad programmering kan beskrivas som en designfilosofi där programkod struktureras och organiseras enligt enskilda objekt vars syfte är att återspegla vardagliga ting

Objektorienterad programmering; *terminologi*

- Objektorienterad programmering består av följande grundläggande termologi:

  - Objekt; *en entitet, sak eller plats*
  - Metod; *en åtgärd som objektet utför*
  - Egenskap; *beskrivande drag hos objektet*
  - Klass; *mall som ligger till grund för objekt*

Objektorienterad programmering; *utökad terminologi*

- Objektorienterad programmering består även av följande termologi:

  - Inkapsling
  - Arv
  - Abstraction
  - Polymorphism
  - Gränssnitt

Objektorienterad programmering; *JavaScript*

- Relevanta frågeställningar:
  - *Är JavaScript (ECMAScript) ett objektorienterat skriptspråk?*

Objektorienterad programmering; *JavaScript*

ECMAScript	Classes	Objects	Extends	Static	access modifiers
ES5	No	Yes	No	No	No
ES6	Yes	Yes	Yes	Yes	No
ES7	Yes	Yes	Yes	Yes	No

Figur. Kompabilitetstabell för ECMAScript-262 utgåvor, kolumnerna representerar funktionalitet som är "nödvändig" för ett objektorienterat programmeringsspråk

Objektorienterad programmering; *JavaScript*

- ECMAScript kan användas för att utforma objektorienterad programkod; *standarden erbjuder inget inbyggt stöd men objektorienterade riktlinjer kan användas för att skapa en objektorienterad struktur*

- Objektorienterad JavaScript är ett känsloladdat ämne; *varför bruka ett språk på ett sätt som det inte är tänkt?*

Objektorienterad programmering; *JavaScript*

- krav och förväntningar på webbapplikationer ökar;
  detta innebär att utvecklingsmetoderna måste utvecklas i syfte att möta förväntningar

Objektorienterad programmering; *JavaScript*

- Relevanta frågeställningar:
  - *Innebär det att ”moderna” applikationer inte kan skapas om de inte applicerar objektorienterade riktlinjer?*
  - *Vilken ECMAScript-utgåva förespråkas av kursen?*

ECMA-262, 5th Edition

- Fördelar med att använda ECMA-262, 5th Edition:

  - bakåtkompatibilitet; *fungerar i äldre webbläsare och är därför inte begränsade till HTML5-teknik*

  - inlärningssyfte; *förutsätter en bättre förståelse för skriftspråket och hur det tolkas*

  - funktionalitet; *avsaktat av inbyggt OOP-stöd vägs upp av möjligheterna att skapa eget*

ECMA-262, 5th Edition

- Då objektorienterad programmering i grund och botten är en designfilosofi, kan objektorienterad programkod uppnås på flera olika sätt; *detta görs möjligt tack vare JavaScripts lösa struktur*

ECMA-262, 5th Edition

- Nackdelar med att använda ECMA-262, 5th Edition:
  - komplexitet; *arbetsprocessen förutsätter kunskap om ECMAScript samt god förståelse för OOP*
  - avsaknad av klasser; *avsaknad av klasser förutsätter kreativa arbetsmetoder*

ECMA-262, 5th Edition; *återblick*

”As we have seen, most of the concepts that we recognise as making up object-orientation came from the class concept of Simula. Nevertheless, as both Dahl and Nygaard would be quick to point out, it’s the dynamic objects, not the classes, that form the system model, and modelling was the *raison d’être* of Simula.”

(Black, 2013, s. 13)

Black, A. (2013).”*Object-oriented programming: Some history, and challenges for the next fifty years*”, Information And Computation, 231, pp. 3-20. Academic Press Inc. Country of Publication: USA

Objekt

- ECMA-262 definerar objekt som en osorterad samling egenskaper bestående av primitiva värden, objekt eller funktioner; *ett objekt kan ses som en hashtabell innehållande namn-värde-par av varierande datatyp*

- Objekt skapas enligt en förbestämd datatyp eller deklareras som en egen; *då ECMAScript är löst typat kan en objektstruktur förändras under exekvering*

Objekt; *exempel*

```
var user = new Object();
user.name = "Rune Körnefors";
user.age = 55;
user.employment = "Lecturer";

user.sayHello = function() {
  alert(this.name);
};

user.sayHello(); // "Rune Körnefors"
```


Objekt

- Relevanta frågeställningar:
  - *Vad finns det för för- och nackdelar med föregående exempelkod?*

Factory Pattern

- Objektorienterat designmönster som implementerar konceptet med abstrakta fabriker för instansiering av likasinnade objekt, allt sker utan kännedom om dess innehåll

- Vanligt förekommande då det finns ett behov att ”instansiera” ett eller flera skräddarsydda objekt; *vanligtvis vid tillfällen där arvsled ses som en begränsning på konfigurationsmöjligheter*

- Designmönstret kan används av webbutvecklare för att kringgå problematiken med avsaknaden av klasser; *förenklad användning av mönstret*

Factory Pattern; *exempel*

```
function createUser(name, age, job) {
   var user = new Object();
      user.name = name || "John Doe";
      user.age  = age  || 0;
      user.job  = job  || "unemployed";

      user.sayHello = function() {
         alert("Hi, my name is "+this.name);
      };

      return user;
   }

   var u1 = createUser("Rune Körnefors", 55, "Lecturer");
   var u2 = createUser();
```


Factory Pattern; *reflektion*

- Fabriksmönstret erbjuder mekanik för att skapa förkonfigurerade datastrukturer i form av objekt; *valfritt antal objektinstanser kan skapas med enkelhet*

```
for (var i = 0; i < 10; i++) {
  createUser();
}
```

- Relevanta frågeställningar:

  - *Finns det någon nackdel med denna metod?*

Factory Pattern; *reflektion*

- Samtliga objektstrukturer som genereras av fabriksmönstret är ursprungligen ett tomt objekt; *objektstrukturen blir därför aldrig förknippad med en egen datatyp*

- Ett objekt kan därför enbart identifieras utefter sitt innehåll; *vilket nödvändigtvis inte är en garanti för identifiering*

Factory Pattern; *reflektion*

```
function createPerson(name) {
  var person = new Object();
    person.name = name;
    return person;
}

function createPet(name) {
  var pet = new Object();
    pet.name = name;

  return pet;
}
```


Constructor Pattern

- Inom objektorienterad programmering är en konstruktor en speciell metod som per automatik aktiveras då ett objekt instansieras; *syftet är att förbereda objektet för användning, exempelvis genom populering av objektstrukturer*

- Inom JavaScript används objekt-konstruktorer för att skapa olika typer av objekt

Constructor Pattern; *exempel*

```
function User(name, age, job) {
   this.name = name || "John Doe";
   this.age  = age  || 0;
   this.job  = job  || "unemployed";

   this.sayHello = function() {
      alert("Hi, my name is "+this.name);
   };
}

var u1 = new User("Rune Körnefors", 55, "Lecturer");
var u2 = new User();
```


Constructor Pattern; *reflektion*

- Relevanta frågeställningar:

  - *Vad innebär this?*
  - *Varför används this i samband med metod- och egenskapsdeklaration?*
  - *På vilket sätt kan detta ses som en förbättring gentemot tidigare exempel?*

Omfång; *this*

Anrop	This
funktion	window || undefined
metod	objektet
konstruktor	det nya objektet
apply- eller call-anrop	parameter

Figur. Betydelse av nyckelordet this beroende på omfång (tabellen inkluderar inte händelsebaserade situationer)

This; *funktion*

```
function sayHello() {
  this.alert("Hello World!");
}

sayHello(); // "Hello World!"
```

This; *konstruktor*

```
function Person() {
  this.name = "Kalle";
}

var p = new Person();
console.log(p.name); // "Kalle"
```


This; *metod*

```
function Person() {
  this.name = "Kalle";
  this.sayHello = function() {
    console.log(this.name);
  };
}

var p = new Person();
console.log(p.sayHello()); // "Kalle"
```


Constructor Pattern; *reflektion*

- Då objekt skapas via konstruktormönstret tillfaller objektet en egen datatyp:

```
var u1 = createUser();
var u2 = new User();

console.log(u1 instanceof Object); // true
console.log(u1 instanceof User);   // false
console.log(u2 instanceof Object); // true
console.log(u2 instanceof User);   // true
```

- Objekt kan därmed identifieras enligt sin data-/objekttyp; *resultatet är enligt klass-objekt-relation*

Constructor Pattern; *reflektion*

- Mönstret utgör en god grund i strävan efter ett objektorienterat ECMAScript, men viss problematik kvarstår:

  - *instansierade objekt innehåller identiskt men unikt innehåll*

Constructor Pattern; *reflektion*

- Då objektstrukturer dupliceras för varje objektinstans medföljer följande problematik:

  - längre instansieringstid; *innehåll och funktionalitet måste konstruerar från grunden för varje objektinstans*

  - ökad minnesförbrukning; *utöver objektets innehåll, allokerar funktionalitet så som metoder minne*

  - duplicerad programkod; *arkitekturen kan inte tolkas som elegant då den innehåller duplicerad programkod*

Constructor Pattern;
reflektion

- Relevanta frågeställningar?

  - *Är beskriven problematik ett påtagligt problem?*

Prototype pattern

- Ett mönster vars syfte är att skapa objekt baserade på en existerande struktur via kloning

- ECMAScript har inbyggt stöd för prototyp-mönstret där det exempelvis används för att skapa arvsstrukturer; *mer om detta under kommande föreläsning*

- Mönstret kan innebära en prestandaförbättring; *om används korrekt*

Prototype pattern; *exempel*

```
function User() {

}

User.prototype.name = "John Doe";
User.prototype.age  = 0;
User.prototype.job  = "unemployed";
User.prototype.sayHello = function() {
   alert("Hi, my name is "+this.name);
};

var u1 = new User();
var u2 = new User();
   u2.name = "Rune Körnefors";
   u2.age = 55;
   u2.job = "Lecturer";
```

Prototype pattern; *reflektion*

- Mönstret utgör en möjlighet att återanvända funktionalitet som finns tillgänglig inom den gemensamma prototypen; *objekt av samma datatyp använder samma objektreferenser för på så sätt hushålla med resurser*

```
console.log(u1.sayHello == u2.sayHello); // true
```

- Då sayHello-metoden deklarerats som en del av User-prototypen är den identisk i samtliga instanser

Kombination av Constructor och Prototype

- Då inget av tidigare beskrivna designmönster resulterar i en önskvärd effekt kan konstruktor- och prototyp-mönstret kombineras

- Denna kombination resulterar i en snarlik objektorienterad miljö för mjukvaruutveckling

Kombination av Constructor och Prototype

```
function User(name, age, job) {
   this.name = name || "John Doe";
   this.age  = age  || 0;
   this.job  = job  || "unemployed";
}

User.prototype = {
   constructor: User,
   sayHello: function() {
      alert("Hi, my name is "+this.name);
   }
};

var u1 = new User("Rune Körnefors", 55, "Lecturer");
var u2 = new User();
```


Objekt-literal

- Objekt kan även skapas via så kallad objekt-literal; *initierar en lista av noll till flera namn-värde-par som omsluts av klammerparenteser*

Objekt-literal; *exempel*

```
var Main = {
  id : "se.lnu.mediatech",
  version : 1.0,

  getVersion : function() {
    return Main.version;
  }
};

console.log(Main.getVersion()); //"1.0"
```


Objekt-literal

- Saker att tänka på när man arbetar med objekt-literaler:

  - Automatisk instansiering; *objektet instansieras i samband med deklaration och behöver därför inte instansieras innan användning*

  - Statisk referens; *hantera klasstrukturen som statisk och använd ClassName i stället för this*

  - kommatecken; *kom ihåg att använda kommatecken för att indikera att innehållet fortsätter*

  - Publikt innehåll; *information som deklareras i objektet blir per automatik publikt*

Objekt-literal

- Relevant frågeställningar:
  - *När är det fördelaktigt att använda objekt-literaler?*
  - *Är objekt-literal och JSON samma sak?*

Objekt-literal VS JSON

- Objekt-literal och JSON ser likvärdiga ut men är inte samma; *här följer en lista med olikheter:*

  - JSON godkänner enbart egenskapsdeklaration via citattecken

  - JSON tillåter enbart information av datatyperna string, array, boolean, null eller JSON; *metoder är inte tillåtna*

  - Objekt som exempelvis Date konverteras till strängar då det tolkas som JSON

Sammanfattning

- ECMAScript 5th Edition innehåller inte ett komplett objektorienterat stöd, men programspråket kan brukas på ett sätt som gör objektorienterad programmering möjlig

- Föreläsningen har introducerat tre designmönster för objectorienterad ECMAScript; Factory pattern, Constructor pattern & Prototype pattern

- Objektlateraler och JSON-objekt har en liknande struktur men är inte samma sak

Frågor