**Week 3**

Lasse Benninga & Andre Nanninga

**Opdracht 1: JavaScript patterns**

**1. Waarom is JavaScript zo'n populaire taal?**  
Iedereen kan JavaScript draaien. Het enige wat je nodig hebt om te beginnen is een webbrowser. Samen met nieuwe technologieen zoals Node.js en Nodewebkit is JavaScript een taal waarbij je maar 1 keer code hoeft te schrijven dat vervolgens overal kan draaien.

**2. Wat is een prototype in JavaScript? Geef een voorbeeld.**

**3. Wat is een *anonymous function*****en hoe verhoudt zich dit tot *closure?* Geef een voorbeeld.**Een *anonymous function* is een function zonder naam. Deze worden in javascript voornamelijke gebruikt in *callbacks*. Bijvoobeeld:

setTimeout(function() {  
 alert('Hello, world!');  
}, 1000);

De *function* die als eerste parameter wordt meegegeven is een *anonymous function*.

*Anonymous functions* worden vaak gebruikt om closures te maken zoals hieronder getoond.

**4. Hoe kan *closure* in JavaScript ingezet worden om de methoden en variablen van een klasse *private* te maken?**  
In het onderstaande voorbeeld is te zien hoe een *closure* gemaakt kan worden. In dit voorbeeld is de variabele *privateCounter* private omdat deze in de scope van de *anonymous* *function* is. Wanneer bijvoorbeeld de methode *counter.increment* worden aangeroepen kan deze wel de variable *privateCounter* bereiken en aanpassen omdat ze in dezelfde functie scope zitten.

var counter = (function() {  
 var privateCounter = 0;  
  
 return {  
 increment: function() { privateCounter++; },  
 decrement: function() { privateCounter--; },  
 valud: function() { return privateCounter; }  
 };  
})();

**5. Hoe geef je in JavaScript een attribuut aan een functie en wat zou een praktische toepassing daarvan kunnen zijn?**  
Bijvoorbeeld:

var User = function(name, age) {  
 this.name = name;  
 this.age = age;  
};

User.prototype.greeting = function() {  
 return 'Hello ' + this.name + '.';  
};

var user = new User('Piet', 20);

In een functie kan met behulp van het *this* keyword attibuten aan functie toegewezen worden. Hiermee wordt in principe van een functie een object gemaakt en zo kan object georienteerd geprogrammeerd worden.

**6. Waarom is het beter om bij het programmeren te werken met interfaces?**  
Interfaces zijn zelfdocumenterend en zorgen er voor dat code beter herbruikbaar is. De interfaces geven informatie aan de programmeur wat hij kan doen met een object. Wanneer je eenmaal een interface kent kan je deze ook meteen gebruiken bij andere objecten.

**7. Wat is *Duck Typing* en op welke manier kan die techniek gebruikt worden om het gebrek aan interfaces in JavaScript op te lossen?**  
Met *Duck Typing*kan worden gekeken of de attributen van een object functies zijn. Je kan dus een interface defineren met een aantal methodes en voorvolgens kijken op een object of deze de juiste attributen bevat en dat deze functies zijn.

// define the comparable interface  
var Comparable = new Interface('Comparable', ['compare']);

// define a Person object  
var Person = function(name) {  
 this.name = name;  
  
 this.compare = function() {  
 ...  
 }  
};

// method to check if the definition of an object matches an interface  
Interface.checkImplementation = function(object, interface) {  
 foreach(interfaces.methods, function(method) {  
 assert(typeof object[method] === 'function');  
 });  
}

**8. Wat is *encapsulation* in de context van Objectgeorienteerd programmeren?**Encapsulation is het strict defineren van de scope van informatie binnen in Object. In Java wordt bijvoobeeld gebruik gemaakt van de Public, Private en Protected keywords om vast te stellen wat de scope is van variabelen of functies.