



# Webapplicaties II

## Hoofdstuk 07 – Document Object Model

# Inhoud

- Inleiding
- Kennismaking met de applicatie - de repository
- DOM manipulatie
- Interactiviteit via events
- Extra methodes om de DOM te bevragen
- *repository: 07thDOM*

# **07 Document Object Model**

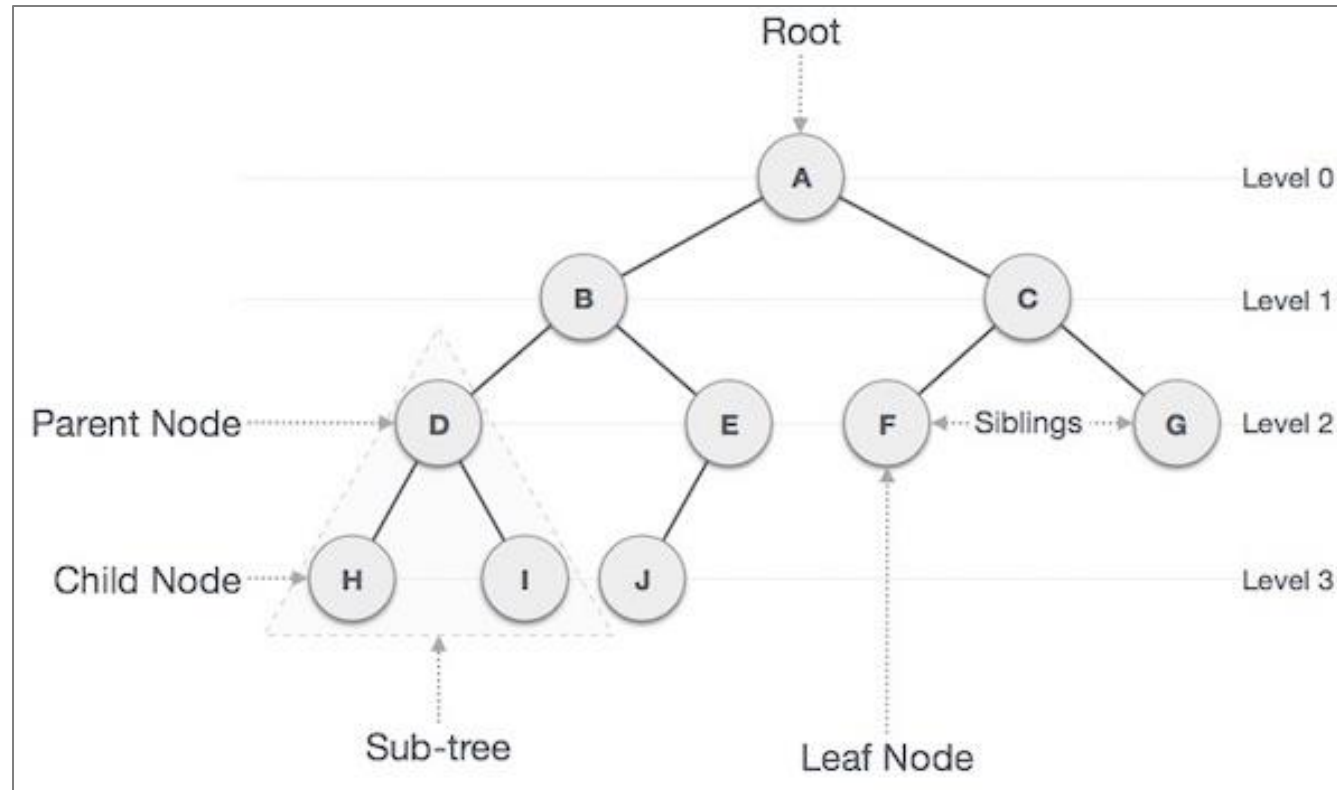
**Inleiding**

# Wat is DOM?

- Wat is DOM?
  - **Web API**
    - standard: <https://dom.spec.whatwg.org/>
  - Vanuit een programma kan je een HTML document als volledig object benaderen
    - Bouwt een **boomvoorstelling** van het HTML document in het geheugen
    - Biedt klassen en methodes (**tree-gebaseerde API**) aan om via code door de boom te navigeren en bewerkingen uit te voeren.
  - Platform- en taalafhankelijke API
  - Ontworpen voor HTML en XML

# Wat is DOM?

- Boomstructuur - terminologie



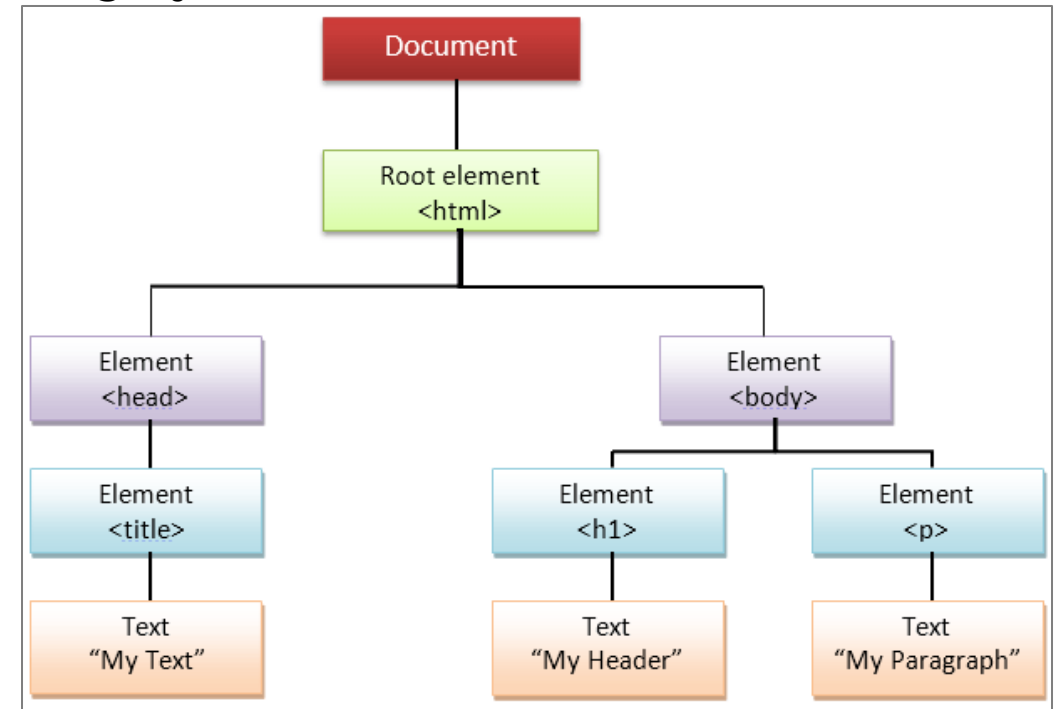
# Wat is DOM?

- Een node kan zijn:
  - **Document**
    - een **DOCUMENT\_NODE** bevat de toegangspoort tot de DOM, het stelt het volledige document voor.
    - bevat een verwijzing naar het <html>-element.
  - **Element:**
    - elke html-element komt overeen met een **ELEMENT\_NODE** in de boom
  - **Text:**
    - de tekst-inhoud van een element is een **TEXT\_NODE**, voorgesteld als child van het element-node  
*Let op: witruimte tussen elementen wordt ook voorgesteld als een text-node.*
  - ...

Node type constants		
Constant	Value	Description
Node.ELEMENT_NODE	1	An <b>Element</b> node like <p> or <div>.
Node.TEXT_NODE	3	The actual <b>Text</b> inside an <b>Element</b> or <b>Attr</b> .
Node.CDATA_SECTION_NODE	4	A <b>CDATASection</b> , such as <![CDATA[[ ... ]]]>.
Node.PROCESSING_INSTRUCTION_NODE	7	A <b>ProcessingInstruction</b> of an XML document, such as <?xml-stylesheet ... ?>.
Node.COMMENT_NODE	8	A <b>Comment</b> node, such as <!-- ... -->.
Node.DOCUMENT_NODE	9	A <b>Document</b> node.
Node.DOCUMENT_TYPE_NODE	10	A <b>DocumentType</b> node, such as <!DOCTYPE html>.
Node.DOCUMENT_FRAGMENT_NODE	11	A <b>DocumentFragment</b> node.

# Wat is DOM?

- DOM stelt het HTML Document voor **als een boom met nodes**.
  - De boom bevat bv. element-nodes die een voorstelling zijn van de elementen uit het HTML document
- **DOM + javascript**
  - DOM manipulatie, bv. opvragen, wijzigen, verwijderen, creëren van nodes
  - wijzigingen in de DOM worden gereflecteerd op de webpagina

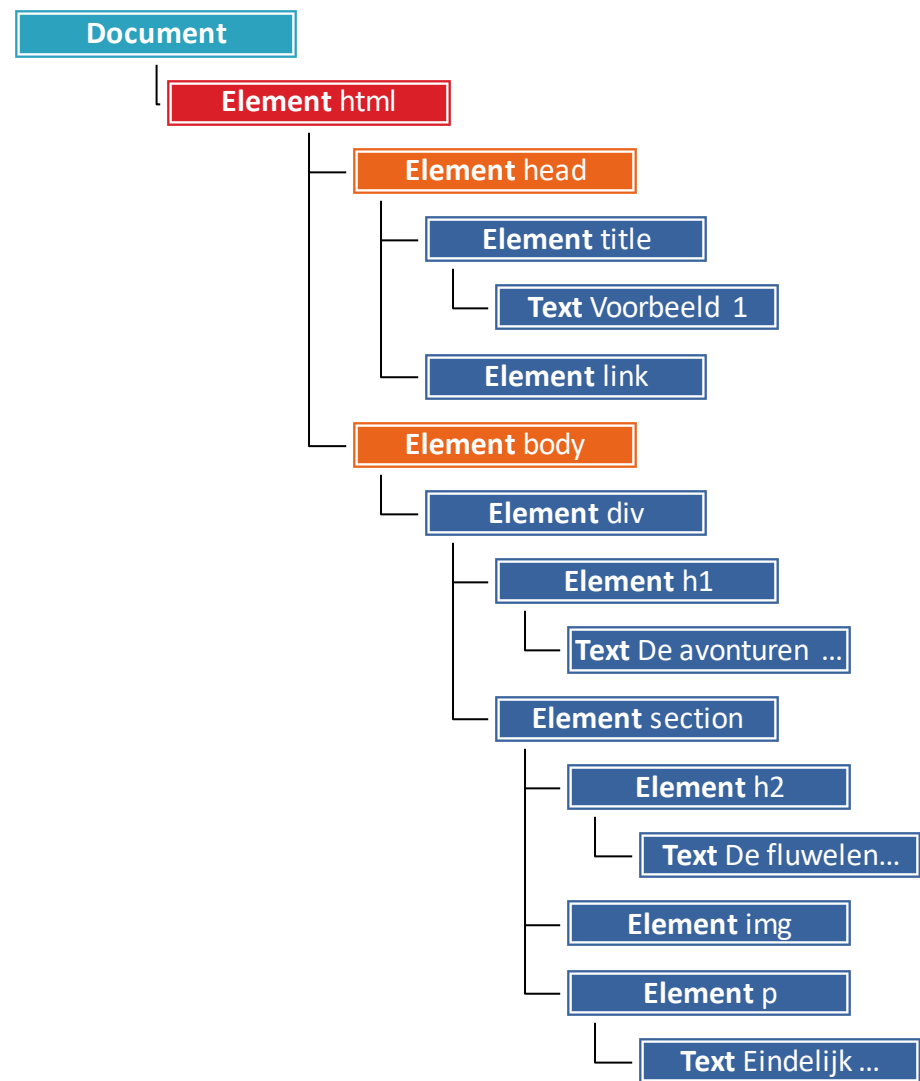


# Voorbeeld

.html

```
<html>
  <head>
    <title>Voorbeeld 1</title>
    <link rel="stylesheet"
      href="css/urbanus.css" />
  </head>
  <body>
    <div class="main">
      <h1>De avonturen van Urbanus</h1>
      <section id="urbanus_140">
        <h2>De Fluwelen Grapjas</h2>
        
        <p>Eindelijk erkenning! ...</p>
      </section>
    </div>
  </body>
</html>
```

DOM





# 07 Document Object Model

## Javascript en de DOM

Overzicht van het voorbeeld 2deHands

# Voorbeeld

- In de volgende slides wordt het onderstaande voorbeeld uitgewerkt
  - De gebruiker selecteert een categorie
  - Het aantal zoekertjes uit die categorie verschijnt  
De zoekertjes uit die categorie worden getoond aan de linkerkant
  - Als de gebruiker klikt op een zoekertje aan de linkerkant, verschijnen de details aan de rechterkant.  
De tekst van het zoekertje waarop werd geklikt, komt in vet
  - Als de gebruiker klikt op een thumbnail aan de rechterkant, wordt de grote afbeelding aangepast

# Voorbeeld



Kies een categorie: **Keuken** ▼

Aantal zoekresultaten: 3



**Tajine te koop**



**Super de luxe grill**



**Pottenset**

## Tajine te koop

Tajine - 27 cm - 2 liter van LE CREUSET

**Prijs: €100**



© 2dehands.be

**HO  
GENT**

# Voorbeeld – HTML code

- Hieronder staat de HTML code. Let op de id's

```
<body>
  <header>
    <div id="logo"></div>
  </header>
  <aside>
    <div id="zoekpaneel">
      <label for="categorie">Kies een categorie:</label>
      <select name="categorie" id="categorie">
        <option value="Alles">Alles</option>
      </select>
    </div>
  </aside>
  <div id="wrapper">
    <div id="aantalProducten"></div>
    <div id="overzichtProducten"></div>
    <div id="productDetails"></div>
  </div>
  <footer>&copy; 2dehands.be</footer>
  <script src="js/2dehands.js"></script>
</body>
```

# De klasse **Product**

- **properties:**
  - id
  - eigenaar
  - postcode
  - gemeente
  - titel
  - omschrijving
  - prijs
  - categorie
  - afbeeldingen

```
class Product {  
    #id;    #eigenaar;    #postcode;    #gemeente;    #titel;  
    #omschrijving;    #prijs;    #categorie;    #afbeeldingen;  
  
    constructor(id, eigenaar, postcode, gemeente, titel,  
        omschrijving, prijs, categorie, afbeeldingen) {  
        this.#id = id;  
        this.#eigenaar = eigenaar;  
        this.#postcode = postcode;  
        this.#gemeente = gemeente;  
        this.#titel = titel;  
        this.#omschrijving = omschrijving;  
        this.#prijs = prijs;  
        this.#categorie = categorie;  
        this.#afbeeldingen = afbeeldingen;  
    }  
    // getters  
}
```

# De klasse **ProductenRepository**

- In dit voorbeeld worden alle producten opgeslagen in de ProductenRepository.
- Een **repository** is een centrale plaats waar data opgeslagen wordt en die een aantal eenvoudige bewerkingen rond deze data implementeert, bijvoorbeeld
  - Alle data opvragen
  - Specifieke data opvragen adhv bepaalde voorwaarden
  - Data toevoegen
  - Data verwijderen
  - ...
- In ons voorbeeld is data hard – coded
  - In een verdere les zal behandeld worden hoe je data kan toevoegen aan de repository vertrekkend van een JSON bestand of van een webservice.

# De klasse **ProductenRepository**

```
class ProductenRepository {  
    #producten = [];  
    constructor() {this.haalProductenOp(); }  
    get producten() {      return this.#producten;    }  
  
    // Voegt het product achteraan toe aan de array _producten  
    voegProductToe(product) { ... }  
  
    // Retourneert het product met opgegeven id  
    geefProduct(id) { ... }  
  
    // Retourneert een array met producten behorend tot de opgegeven categorie  
    geefProductenUitCategorie(categorie) { ... }  
  
    // Retourneert een array van strings met unieke categorieën  
    geefAlleCategorieen() { ... }  
  
    // Vul de repository met producten (hard gecodeerd)  
    haalProductenOp() { ... }  
}
```

- Tip: klap in VS Code de code van haalProductenOp toe

# De klasse **ProductenComponent**

- Deze klasse vormt de **lijm tussen ons domein** (Product & ProductenRepository) en de **HTML pagina**
  - als de gebruiker iets wijzigt op de html pagina gaat het domein aangesproken worden en de web pagina aangepast worden
  - **voorbeeld:** de gebruiker kiest een categorie:
    - de ProductenComponent gaat het domein aanspreken om de producten van die categorie op te halen
    - de ProductenComponent gaat de DOM (~web pagina) updaten met die producten



# De klasse **ProductenComponent**

- Om zijn werk te kunnen doen heeft de ProductenComponent nood aan
  - domeinobject(en), in ons geval de ProductenRepository
  - DOM, sowieso toegankelijk via de document property van het global window-object

```
class ProductenComponent {  
    #productenRepository;  
    constructor() { this.#productenRepository = new ProductenRepository(); }  
  
    // Initialiseer de pagina  
    initialiseerHtml() {  
  
        // Vult de select-list met alle categorieën  
        categorieenToHtml(categorieen) { ... }  
  
        // Toont overzicht van alle producten voor de opgegeven categorie  
        productenToHtml(producten) { ... }  
  
        // Toont de details van het product met opgegeven id  
        productDetailsToHtml(product) { ... }  
    }  
}
```

# De function **init()**

- In de function **init()** wordt een object van de klasse ProductenComponent geïntantieerd en wordt de pagina geïntialiseerd.

```
function init() {  
    const productenComponent = new ProductenComponent();  
    productenComponent.initialiseerHtml();  
}
```

- De function init() wordt aangeroepen op het moment dat de DOM volledig is opgebouwd

```
window.onload = init;
```

## 2deHands

vervolledig de implementatie van de methodes in de  
klasse **ProductenRepository**

## ProductenRepository – voegProductToe(product)

```
// Voegt het product achteraan toe aan de array #producten
voegProductToe(product) {
  this.#producten.push(product);
}
```

- Bekijk het resultaat, gebruik breakpoints
  - of schrijf de inhoud van de repository naar de console

## ProductenRepository – geefProduct(id)

```
// Retourneert het product met opgegeven id
geefProduct(id) {
    return this.#producten.find(product => product.id === id);
}
```

# ProductenRepository – geefAlleCategorieen()

```
// Retourneert een alfabetisch gesorteerde array van strings
// die de unieke categorieën van producten bevat
geefAlleCategorieen() {
  return this.#producten
    .reduce((acc, product) => {
      if (!acc.includes(product.categorie))
        acc.push(product.categorie);
      return acc;
    }, [])
    .sort();
}
```

```
// Alternatief:
geefAlleCategorieen() {
  // Mbv map wordt de array van producten overlopen en wordt een array
  // geretourneerd met per product de categorie
  // Deze array wordt gebruikt bij de creatie van de set. Als de categorie al bestaat in
  // de set, zal die niet opnieuw worden toegevoegd. Dat is juist de essentie van een set
  // Mbv de spread - operator wordt de set omgevormd naar een
  // array. Daarop kan dan gemakkelijk de sort - bewerking toegepast worden
  return [...new Set(this.#producten.map(product => product.categorie))].sort();
}
```

# ProductenRepository - geefProductenUitCategorie

```
// Retourneert een array met producten behorend tot de opgegeven categorie (case-insensitive)
geefProductenUitCategorie(categorie) {
  categorie = categorie.toUpperCase();
  return categorie === 'ALLES' ?
    this.#producten :
    this.#producten.filter(p => p.categorie.toUpperCase() === categorie);
}
```

# 07 Document Object Model

## Javascript en de DOM

voorbeeld 2deHands – dynamisch updaten van de pagina



# Wijzigen van de DOM

- We zagen reeds hoe we een element-node uit de DOM-tree kunnen **ophalen via het id-attribuut** van dat element, bv. ophalen van het element met id `productDetails`

```
const element = document.getElementById('productDetails');
```

- Er zijn nu verschillende manieren om de DOM-tree te **wijzigen**, vertrekkend van een element
  - `insertAdjacentHTML(..., ...)`
  - `innerHTML(...)`
  - `createElement(...) + setAttribute(...) + appendChild(...)`

# Wijzigen van de DOM – `insertAdjacentHTML()`

- De functie `insertAdjacentHTML()` parset de meegegeven **HTML-tekst** en voegt de resulterende nodes toe aan de DOM structuur op de meegegeven **positie**

```
element.insertAdjacentHTML(position, text);
```

## Parameters

### `position`

A `DOMString` representing the position relative to the `element`; must be one of the following strings:

- `'beforebegin'`: Before the `element` itself.
- `'afterbegin'`: Just inside the `element`, before its first child.
- `'beforeend'`: Just inside the `element`, after its last child.
- `'afterend'`: After the `element` itself.

### `text`

The string to be parsed as HTML or XML and inserted into the tree.

```
<!-- beforebegin -->
<p>
  <!-- afterbegin -->
  foo
  <!-- beforeend -->
</p>
<!-- afterend -->
```

voorbeeld positionering tov het  
<p>-element

# Wijzigen van de DOM – insertAdjacentHTML()

- voorbeeld

```
<p id="pId">Dit is de paragraaf</p>
```

```
const element = document.getElementById('pId');  
element.insertAdjacentHTML('afterbegin', '<div>Dit is de toegevoegde div</div>');
```

```
<p id="pId">  
  <div>Dit is de toegevoegde div</div>  
  "Dit is de paragraaf"  
</p>
```

## 2deHands

we vervolledigen de implementatie van de methodes  
in de klasse **ProductenComponent**

## voorbeeld 2dehands – categorieenToHtml(categorieen)

- De select lijst in de index.html

```
<select name="categorie" id="categorie">  
  <option value="Alles">Alles</option>  
</select>
```

Kies een categorie: Alles ▼

Alles

TODO:

alle categorieën overlopen,  
voor elke categorie een <option> toevoegen aan de select lijst

```
<select name="categorie" id="categorie">  
  <option value="Alles">Alles</option>  
  <option value="Keuken">Keuken</option>  
  <option value="Meubelen">Meubelen</option>  
  <option value="Tuin">Tuin</option>  
</select>
```

Kies een categorie: Alles ▼

Alles

Keuken

Meubelen

Tuin

## voorbeeld 2dehands – categorieenToHtml(categorieen)

- De select lijst in de index.html

```
<select name="categorie" id="categorie">
  <option value="Alles">Alles</option>
</select>
```

Kies een categorie: Alles ▼

Alles

```
// Vult de select-list met alle categorieën
categorieen.forEach((categorie) =>
  document.getElementById('categorie')
    .insertAdjacentHTML('beforeend', `<option value="${categorie}" >${categorie}</option>`));
```

```
<select name="categorie" id="categorie">
  <option value="Alles">Alles</option>
  <option value="Keuken">Keuken</option>
  <option value="Meubelen">Meubelen</option>
  <option value="Tuin">Tuin</option>
</select>
```

Kies een categorie: Alles ▼

Alles

Keuken

Meubelen

Tuin

# Wijzigen van de DOM – **innerHTML**

- via de **innerHTML-property** van een element kan je de **HTML** die zich binnen dat element bevindt opvragen/wijzigen.
- merk op dat als je de innerHTML wijzigt je alles die binnen het element aanwezig was overschrijft...

```
const content = element.innerHTML;  
  
element.innerHTML = htmlString;
```

# Wijzigen van de DOM – **innerHTML**

- voorbeeld

```
<p id="pId">Dit is de paragraaf</p>
```

```
const element = document.getElementById('pId');  
element.innerHTML = '<div>Dit is de div! Waar is de rest?</div>'
```

```
<p id="pId">  
  <div>Dit is de div! Waar is de rest?</div>  
</p>
```

```
element.innerHTML = '';
```

```
<p id="pId"></p>
```



## 2deHands

we vervolledigen de implementatie van de methodes  
in de klasse **ProductenComponent**

## voorbeeld 2dehands – productenToHtml(producten)

- In de linkerkolom moeten de producten van de gekozen categorie verschijnen. Daartoe zullen we de functie productenToHtml implementeren. In de volgende slides wordt dit stap voor stap uitgewerkt.
- In deze functie
  - **wordt het aantal gevonden producten uitgeschreven**
  - wordt het element met id overzichtProducten leeg gemaakt
  - wordt voor elk product van de gegeven producten dynamisch de HTML code gegeneerd

# voorbeeld 2dehands – productenToHtml(producten)

- Stap 1 – uitschrijven van het aantal producten

Kies een categorie:

© 2dehands.be

TODO:  
aantal producten bepalen  
h4 genereren

Kies een categorie:

**Aantal producten: 16**

© 2dehands.be

## voorbeeld 2dehands – productenToHtml(producten)

- Stap 1 – uitschrijven van het aantal producten

```
<div id="aantalProducten"></div>
```

TODO:

aantal producten bepalen  
binnen de div #aantalproducten een h4-element zetten

```
▼ <div id="aantalProducten">  
  <h4>Aantal producten: 16 </h4>  
</div>
```

## voorbeeld 2dehands – productenToHtml(producten)

- Stap 1 – uitschrijven van het aantal producten

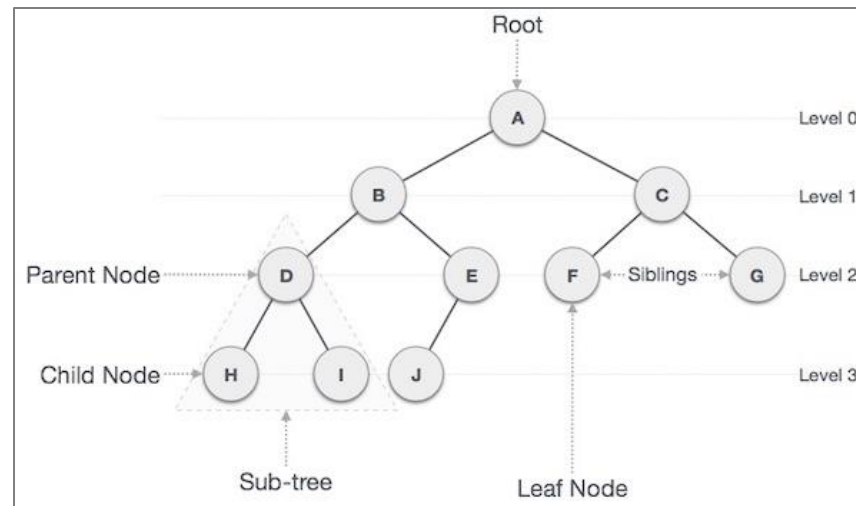
```
<div id="aantalProducten"></div>
```

```
document.getElementById('aantalProducten').innerHTML =  
  `<h4>Aantal producten: ${producten.length} </h4>`;
```

```
▼ <div id="aantalProducten">  
  <h4>Aantal producten: 16 </h4>  
</div>
```

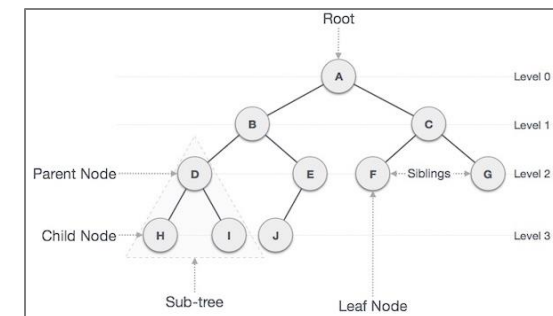
# Wijzigen van de DOM – werken met nodes

- je kan ook zelf **expliciet nodes maken en toevoegen** aan de DOM-tree
- **creatie** van nodes
  - `const anElementNode = createElement(tagName)`
  - `const aTextNode = createTextNode(text)`
- **toevoegen** van een node
  - `node.appendChild(newChild)`
    - voegt een node 'newChild' toe na de laatste child-node van node. De nieuw toegevoegde node wordt geretourneerd.



# Wijzigen van de DOM – werken met nodes

- nog enkele methodes...
  - **removeChild(removechild)** verwijdert een child node van de document tree. De verwijderde node wordt geretourneerd (en kan dus eventueel nog verder gebruikt worden)
  - **replaceChild(newchild, oldchild)** vervangt de *oldchild* node door de *newchild* node. De verwijderde oldchild node wordt geretourneerd (en kan dus eventueel nog verder gebruikt worden)
  - **cloneNode(deepcopy)** creëert en retourneert een exacte kloon. Als de boolean parameter *deepcopy* gelijk is aan true wordt een kloon gemaakt van de node zelf en de volledige subtree, anders wordt enkel een kloon gemaakt van de node zelf.



# Wijzigen van de DOM – werken met nodes

- werken met attributen
  - `setAttribute(attrName, attrValue)`
  - `getAttribute(attrName)`
  - `removeAttribute(attrName)`
- merk op: voor standaard attributen (~attributen herkend oor de browser) kan je ook werken met de **.-notatie**
  - daar class een gereserveerd woord is dien je `.className` te gebruiken ipv `.class`



# Wijzigen van de DOM – werken met nodes

- voorbeeld

```
<p id="pId">Dit is de paragraaf</p>
```

```
const divElement = document.createElement("div");  
divElement.setAttribute("id", "divId");  
const divTekst = document.createTextNode("Hier is de tekst voor de div...");  
divElement.appendChild(divTekst);  
element.appendChild(divElement);
```

```
<p id="pId">  
  "Dit is de paragraaf"  
  <div id="divId">Hier is de tekst voor de div...</div>  
</p>
```

## 2deHands

we vervolledigen de implementatie van de methodes  
in de klasse **ProductenComponent**

## voorbeeld 2dehands – productenToHtml(producten)

- In de linkerkolom moeten de producten van de gekozen categorie verschijnen. Daartoe zullen we de functie productenToHtml implementeren. In de volgende slides wordt dit stap voor stap uitgewerkt.
- In deze functie
  - wordt het aantal gevonden producten uitgeschreven
  - **wordt het element met id overzichtProducten leeg gemaakt**
  - **wordt voor elk producten van de gegeven producten dynamisch de HTML code gegenereerd**

# voorbeeld 2dehands – productenToHtml(producten)

- Stap 2 – overzicht van de producten genereren

Kies een categorie: Alles ▼

Aantal producten: 16

TODO:

alle producten overlopen  
voor elk product een div genereren

Kies een categorie: Alles ▼

Aantal producten: 16



Prachtige eetkamer



Eetkamer met 6 stoelen



Eetkamer landelijke stijl

# voorbeeld 2dehands – productenToHtml(producten)

- Stap 2 – overzicht van de producten genereren

```
<div id="overzichtProducten"></div>
```

TODO:

alle producten overlopen  
voor elk product een div met daarin een img en een p genereren

```
▼ <div id="overzichtProducten">  
  ▼ <div id="P000">  
      
    <p>Prachtige eetkamer</p>  
  </div>  
  ▶ <div id="P001">...</div>  
  ▶ <div id="P002">...</div>  
  ▶ <div id="P003">...</div>
```

# voorbeeld 2dehands – productenToHtml(producten)

- Stap 2 – overzicht van de producten genereren

```
<div id="overzichtProducten"></div>
```

```
producten.forEach((product, index) => {  
  const divElement = document.createElement('div');  
  divElement.id = product.id;  
  divElement.insertAdjacentHTML(  
    'afterbegin',  
    `![${product.titel}><br/](images/${product.id}/thumbs/thumb_${product.afbeeldingen[0]}.jpg)    <p> ${product.titel} </p>`  
  );  
  document.getElementById('overzichtProducten').appendChild(divElement);  
});
```

```
▼ <div id="overzichtProducten">  
  ▼ <div id="P000">  
      
    <p> Prachtige eetkamer </p>  
  </div>  
  ▶ <div id="P001">...</div>  
  ▶ <div id="P002">...</div>  
  ▶ <div id="P003">...</div>
```

## voorbeeld 2dehands – `productenToHtml(producten)`

- don't forget
  - als de categorie wijzigt en andere producten moeten getoond worden gaan we eerst de div moet leegmaken, we starten dus best met

```
document.getElementById('overzichtProducten').innerHTML = '';
```

# voorbeeld 2dehands – productenToHtml(producten)

- merk op
  - de randen en andere opmaak van deze div's is gedefinieerd in de css

```
#overzichtProducten {  
    width: 200px;  
    float: left;  
}  
  
#overzichtProducten div {  
    text-align: center;  
    border: 1px outset #848583;  
    border-radius: 5px;  
    margin: 5px;  
    cursor: pointer;  
    overflow: auto;  
}  
  
#overzichtProducten div img {  
    float: left;  
    margin: 5px;  
}  
  
#overzichtProducten div p {  
    margin: 3px;  
    padding-top: 3px;  
}
```

```
<div id="overzichtProducten">  
  <div id="P000">  
      
    <p> Prachtige eetkamer </p>  
  </div>  
  <div id="P001">...</div>  
  <div id="P002">...</div>  
  <div id="P003">...</div>
```





## voorbeeld 2dehands – productenToHtml(producten)

- extra: achtergrond van producten moet alterneren van kleur, we maken gebruik van de css class wit

```
producten.forEach((product, index) => {
    const divElement = document.createElement('div');
    divElement.id = product.id;
    if (index % 2 === 0) {
        divElement.className= 'wit';
    }
    divElement.insertAdjacentHTML(
        'afterbegin',
        `<img src=images/${product.id}/thumbs/thumb_${product.afbeeldingen[0]}.jpg alt=${product.titel}
        <p> ${product.titel} </p>`
    );
    document.getElementById('overzichtProducten').appendChild(divElement);
});
```



# 07 Document Object Model

## Javascript en de DOM

Interactiviteit via events

# Events – interactiviteit

- als een **andere categorie** wordt geselecteerd dan moet het **overzicht van de producten aangepast** worden, lees:
- wanneer het **change-event** wordt afgevuurd op de select #categorie dan moeten we
  - weten welke categorie werd gekozen, dit kan via het **value-attribuut** van het select-element
  - de juiste producten ophalen via de repository: **geefProductenUitCategorie(categorie)**
  - het producten overzicht aanpassen: **productenToHtml(producten)**
  - er is op dat moment geen product geselecteerd en dus moet de div #productDetails niet getoond worden, dit kan via het **style.display-attribuut** op “none” te zetten

# Events – interactiviteit

```
document.getElementById('categorie').onchange = () => {  
  
    productenComponent.productenToHtml(  
        productenComponent.#productenRepository.geefProductenUitCategorie(  
            document.getElementById('categorie').value));  
  
    document.getElementById('productDetails').style.display = 'none';  
  
}
```

- In de init function

# Events – interactiviteit

- als op een product wordt geklikt dan moeten de details van het product getoond worden lees:
- wanneer het **click-event** wordt afgevuurd op de div #productId dan moeten we
  - de details van het product ophalen via de repository `geefProduct(id)`
  - de div #productDetails moet worden aangepast via `productDetailsToHtml(product);`
    - deze methode is reeds geïmplementeerd
- merk op: de lijst van producten wordt dynamisch aangemaakt (~productenToHtml), het is dus daar dat we de event handlers gaan definiëren

# Events - interactiviteit

```
productenToHtml(producten) {  
  ...  
  producten.forEach((product, index) => {  
    const divElement = document.createElement('div');  
    divElement.id = product.id;  
    ...  
    divElement.onclick = () => {  
      this.productDetailsToHtml(product);  
    }  
    ...  
  });  
}
```

# 07 Document Object Model

## Javascript en de DOM

Flexibel nodes selecteren & cheatsheet

# Één of meerdere nodes selecteren

- **getElementById(*id*)**
  - geeft het eerste element met de opgegeven *id*
  - de zoekbewerking start meestal vanaf het document element, maar je kan evengoed van elk ander element starten
- **getElementsByTagName(*tagname*)**
  - retourneert een **live HTML collection** met alle elementen (0 of meer) met de opgegeven *tagname*
  - een HTML collection is een array-like object van elementen en voorziet properties en functies om elementen te selecteren in de lijst
- **getElementsByClassName(*value*)**
  - retourneert een **live HTML collection** met alle elementen met de opgegeven *value* als waarde voor het class attribuut.
- **querySelector(*CSS selector*)**
  - retourneert het eerste element dat voldoet aan de opgegeven CSS selector.
- **querySelectorAll(*CSS selector*)**
  - retourneert een **non live (static)** nodelist met alle elementen die voldoen aan de opgegeven CSS selector.



# Één of meerdere nodes selecteren

- voorbeeld: als een nieuw product wordt gekozen dan gaan we in de methode productToHtml
  - de class tekstVet weghalen van het huidig gekozen product
    - let op: mogelijks is er geen product met class tekstVet, bv. net na het veranderen van de categorie
  - de class tekstVet toevoegen aan het nieuwe gekozen product

```
// zet, in het overzicht van de producten, het gekozen product in het vetjes
if (document.querySelector(`#overzichtProducten .tekstVet`))
    document.querySelector(`#overzichtProducten .tekstVet`).classList.remove('class');
document.querySelector(`#${product.id} p`).classList.add('tekstVet');
```

# The most common DOM methods at a glance

## Reaching Elements in a Document

`document.getElementById('id')`: Retrieves the element with the given `id` as an object

`document.getElementsByTagName('tagname')`: Retrieves all elements with the tag name `tagname` and stores them in an array-like list

## Reading Element Attributes, Node Values and Other Data

`node.getAttribute('attribute')`: Retrieves the value of the attribute with the name `attribute`

`node.setAttribute('attribute', 'value')`: Sets the value of the attribute with the name `attribute` to `value`

`node.nodeType`: Reads the type of the node (1 = element, 3 = text node)

`node.nodeName`: Reads the name of the node (either element name or `#textNode`)

`node.nodeValue`: Reads or sets the value of the node (the text content in the case of text nodes)

## Navigating Between Nodes

`node.previousSibling`: Retrieves the previous sibling node and stores it as an object.

`node.nextSibling`: Retrieves the next sibling node and stores it as an object.

`node.childNodes`: Retrieves all child nodes of the object and stores them in an list. here are shortcuts for the first and last child node, named `node.firstChild` and `node.lastChild`.

`node.parentNode`: Retrieves the node containing `node`.

## Creating New Nodes

`document.createElement(element)`: Creates a new element node with the name `element`. You provide the name as a string.

`document.createTextNode(string)`: Creates a new text node with the node value of `string`.

`newNode = node.cloneNode(bool)`: Creates `newNode` as a copy (clone) of `node`. If `bool` is `true`, the clone includes clones of all the child nodes of the original.

`node.appendChild(newNode)`: Adds `newNode` as a new (last) child node to `node`.

`node.insertBefore(newNode,oldNode)`: Inserts `newNode` as a new child node of `node` before `oldNode`.

`node.removeChild(oldNode)`: Removes the child `oldNode` from `node`.

`node.replaceChild(newNode, oldNode)`: Replaces the child node `oldNode` of `node` with `newNode`.

`element.innerHTML`: Reads or writes the HTML content of the given element as a string—including all child nodes with their attributes and text content.

## Known browser quirks:

`getAttribute` and `setAttribute` are not reliable. Instead, assign the property of the element object directly: `obj.property = value`. Furthermore, some attributes are actually reserved words, so instead of `class` use `className` and instead of `for` use `HTMLfor`.

Real DOM compliant browsers will return linebreaks as text nodes in the `childNodes` collection, make sure to either remove them or test for the `nodeType`.

10  
GENT